Apache OFBiz Developer Manual

The Apache OFBiz Project

Version unspecified, La traduction en français est en cours, donc de nombreuse partie sont encore en anglais, elle est basé sur la version 1850155 de la version anglaise.

Table of Contents

1. Introduction

Welcome to the Apache OFBiz developer manual. This manual provides information to help with customizing and developing OFBiz. If you are new to OFBiz and interested in learning how to use it, you may want to start with the "Apache OFBiz User Manual".

OFBiz is a large system composed of multiple subsystems. This manual attempts to introduce the overall architecture and high level concepts, followed by a detailed description of each subsystem. In addition, the manual will cover topics necessary for developers including the development environment, APIs, deployment, security, and so on.

1.1. Main systems

OFBiz at its core is a collection of systems:

- A web server (Apache Tomcat)
- A web MVC framework for routing and handling requests.
- An entity engine to define, load and manipulate data.
- A service engine to define and control business logic.
- A widget system to draw and interact with a user interface.

On top of the above mentioned core systems, OFBiz provides:

- A data model shared across most businesses defining things like orders, invoices, general ledgers, customers and so on.
- A library of services that operate on the above mentioned data model such as "createBillingAccount" or "updateInvoice" and so on.
- A collection of applications that provide a user interface to allow users to interact with the system. These applications usually operate on the existing data model and service library. Examples include the "Accounting Manager" and "Order Manager".
- A collection of optional applications called "plugins" that extend basic functionality and is the main way to add custom logic to OFBiz.

1.2. Components

The basic unit in OFBiz is called "component". A component is at a minimum a folder with a file inside of it called "ofbiz-component.xml"

Every application in OFBiz is a component. For example, the order manager is a component, the accounting manager is also a component, and so on.

By convention, OFBiz components have the following main directory structure:

```
component-name-here/
 ├──── config/              - Properties and translation labels (i18n)
 ├──── data/                - XML data to load into the database
 ├──── entitydef/           - Defined database entities
 ├──── groovyScripts/       - A collection of scripts written in Groovy
 ├──── minilang/            - A collection of scripts written in minilang (deprecated)
 ├──── ofbiz-component.xml  - The OFBiz main component configuration file
 ├──── servicedef           - Defined services.
 ├──── src/
     ├──── docs/            - component documentation source
     └──── main/java/       - java source code
     └──── test/java/       - java unit-tests
 ├──── testdef              - Defined integration-tests
 ├──── webapp               - One or more Java webapps including the control servlet
 └──── widget               - Screens, forms, menus and other widgets
```

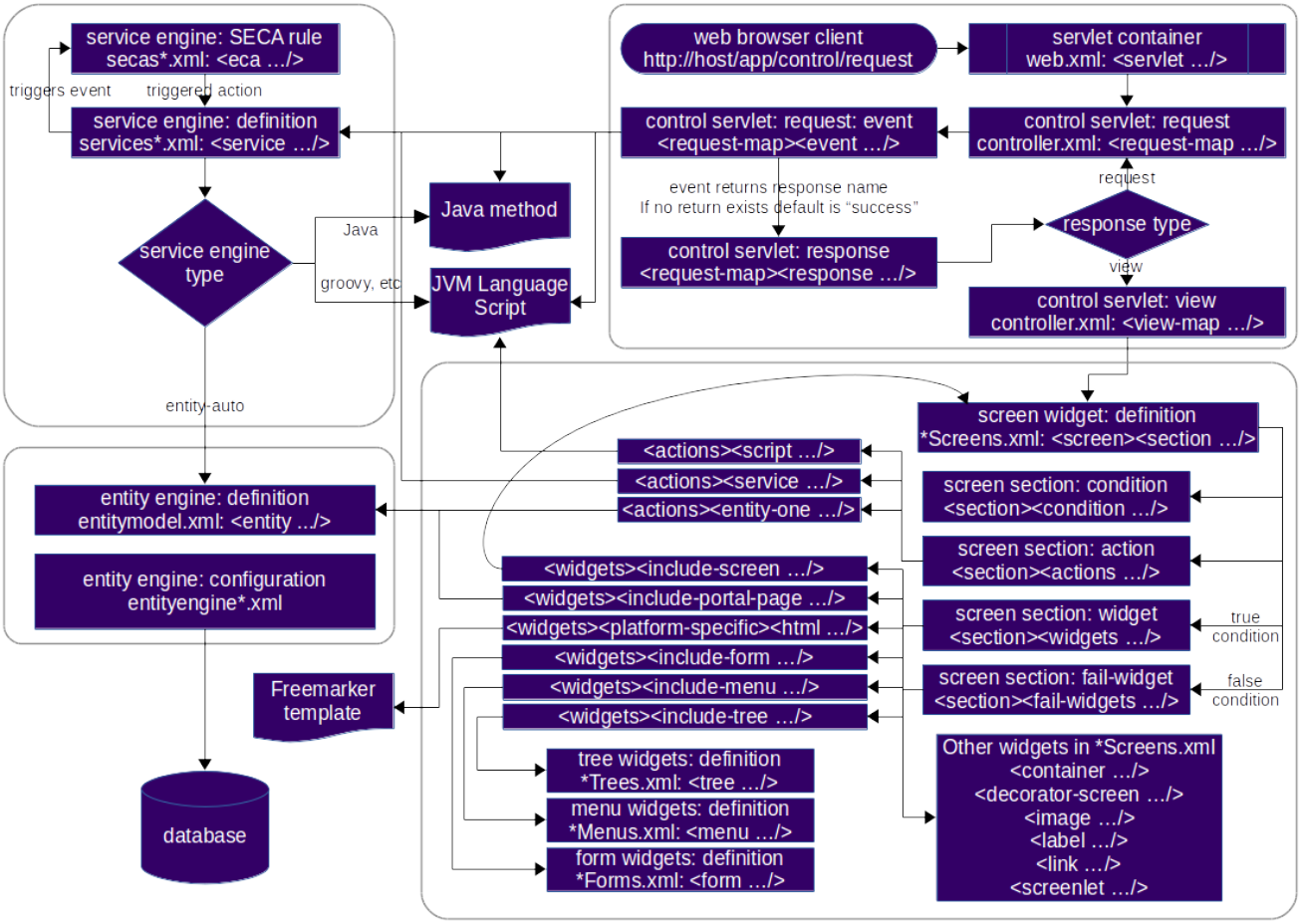
It is apparent from the above directory structure that each OFBiz component is in fact a full application as it contains entities, data, services, user interface, routing, tests, and business logic.

Both core OFBiz applications as well as plugins are nothing more than components. The only difference is that core applications reside in the "applications" folder whereas plugins reside in the "plugins" folder; also OFBiz does not ship with plugins by default.

1.3. Example workflow

Many basic concepts were explained so far. An example would help in putting all of these concepts together to understand the bigger picture. Let us take an example where a user opens a web browser and enters a certain URL and hits the enter key. What happens? It turns out answering this question is not quite simple because lots of things occur the moment the user hits "enter".

To try to explain what happens, take a look at the below diagram. Do not worry if it is not fully understandable, we will go through most of it in our example.



1.3.1. User enters URL

In the first step in our example, the user enters the following URL:

https://localhost:8443/accounting/control/findInvoices

If we break down this URL, we identify the following parts:

- localhost: Name of the server in which OFBiz is running
- 8443: Default https port for OFBiz
- accounting: web application name. A web application is something which is defined *inside* a component
- control: Tells OFBiz to transfer routing to the control servlet
- findInvoices: request name inside the control servlet

1.3.2. Control servlet takes over

The Java Servlet Container (tomcat) re-routes incoming requests through web.xml to a special OFBiz servlet called the control servlet. The control servlet for each OFBiz component is defined in controller.xml under the webapp folder.

The main configuration for routing happens in controller.xml. The purpose of this file is to map requests to responses.

Request Map

A request in the control servlet might contain the following information:

- Define communication protocol (http or https) as well as whether authentication is required.
- Fire up an event which could be either a piece of code (like a script) or a service.
- Define a response to the request. A response could either be another request or a view map.

So in this example, the findInvoices request is mapped to a findInvoices view.

View Map

A view map maps a view name to a certain view-type and a certain location.

View types can be one of:

- screen: A screen widget which translates to normal HTML.
- screenfop: A PDF screen designed with Apache FOP based constructs.
- screencsv: A comma separated value output report.
- screenxml: An XML document.
- simple-content; A special MIME content type (like binary files).
- ftl: An HTML document generated directly from a FreeMarker template.
- screenxls: An Excel spreadsheet.

In the findInvoices example, the view-map type is a normal screen which is mapped to the screen: component://accounting/widget/InvoiceScreens.xml#FindInvoices

1.3.3. Widget rendered

Once the screen location is identified and retrieved from the previous step, the OFBiz widget system starts to translate the XML definition of the screen to actual HTML output.

A screen is a collection of many different things and can include:

- Other screens
- Decorator screens
- Conditional logic for hiding / showing parts of the screen
- data preparation directives in the <action> tag
- Forms
- Menus
- Trees
- Platform specific code (like FreeMarker for HTML output)
- Others (portals, images labels etc ...)

Continuing the example, the FindInvoices screen contains many details including two forms. One form is for entering invoice search fields and the other form displays search results.

1.4. Gestion de la documentation

Comment la documentation est organisé, avec quels outils est-elle réalisé, quels sont les conventions et toutes les autres questions concernant la gestion de la document et son élaboration sont traité dans un document dédié.

cf documentation_guidelines

Il existe plusieurs documentations, chacune avec un objectif ou un lectorat cible, pour l'instant chacune est représenté par un fichier dans le répertoire racine .

2. Web Framework

3. Web Applications

The OFBiz webapp is one of the core framework components. It is tightly integrated with other framework components.

3.1. Cross-domains Single Sign On (SSO)

In some cases you need to split the OFBiz applications on different servers, and possibly in production on different domains. This can happen for different reasons, most often for performance reason.

As it's annoying to give each time a credential when changing from an OFBiz application to another on the same server, the same applies when changing from an OFBiz application to another on another domain.

To prevent that on the same server, the ExternalLoginKey mechanism is used. The cross-domains SSO feature allows to navigate from a domain to another with automated SSO.

It based on 3 technologies:

JWT

JWT Official site - Wikipedia for JWT

CORS

CORS (Mozilla doc) - Wikipedia for CORS

Ajax

Ajax, now well known I guess, in OFBiz we use jQuery for that.

The mechanism is simple.

On the source side:

1. When an user log in in an application (webApp) a webappName.securedLoginId cookie is created. This cookie will be used by the mechanism to know the current logged in user. *Note that all webappName.securedLoginId cookies are deleted when the user session is closed or time out. Hence (apart also using an intrinsically secured cookie) the mechanim is secured, even on shared machines. Of course if people are sharing a machine during their sessions, things could get complicated. This unlikely later case is not taken in account.*
2. The user is given a JavaScript link which passes the URL to reach and the calling webapp name to the sendJWT() Ajax function.
3. The sendJWT() Ajax function calls the loadJWT() Ajax function which in turn calls the CommonEvents::loadJWT method through the common controller.
4. The CommonEvents::loadJWT method uses the calling webapp name to retrieve the userLoginId from the secured webappName.securedLoginId cookie, creates a JWT containing the userLoginId, and returns it to the loadJWT() Ajax function.
5. Then the sendJWT() Ajax function sends an Authorization header containing the JWT to the URL

to reach. At this stage, if all things are correct, the flow leaves the source side.

On the server side:

1. A CORS policy is needed. *Without it, the Authorization token containing the JWT will be rejected. It's a simple policy but you need to strictly define the authorized domains. Never use the lazy "*" for domains (ie all domains), else the preflight request will not work.* Here is an example for Apache HTTPD (domain value is "https://localhost:8443" for official OFBiz demo):

```
Header set Access-Control-Allow-Origin domain
Header set Access-Control-Allow-Headers "Authorization"
Header set Access-Control-Allow-Credentials "true"
```

1. The checkJWTLogin preprocessor, similar to the checkExternalLoginKey, intercepts the JWT, checks it and if all is OK signs the user on. That's it !

In the example component, the FormWidgetExamples screen contains 2 new fields in the LinksExampleForm which demonstrate the use from a local instance to the trunk demo instance.

If you are interested in more details you may refer to https://issues.apache.org/jira/browse/OFBIZ-10307

3.2. Control Servlet

3.2.1. Requests

3.2.2. Views

4. Entity Engine

4.1. Entities

4.1.1. Standard Entities

4.1.2. View Entities

4.1.3. Extended Entities

4.1.4. Dynamic View Entities

4.2. XML Data

4.3. Entity engine configuration

4.4. Supported databases

5. Service Engine

5.1. Declaration and Implementation

5.2. Supported languages

5.3. Transaction management

5.4. Web services

6. Widget System

6.1. Screen Widget

6.1.1. Decoration Pattern

6.2. Form Widget 2

6.3. Menu Widget

6.4. Tree Widget

6.5. Portal Widget

6.6. Platform Specific Code

7. FrontJs Portal

Le composant frontJsPortal, inclus dans le plugin du même nom, a pour objectif de fournir un moyen de gérer l'interface utilisateur avec une technologie de type FrontJs, c'est à dire VueJs, ou React ou Angular.

Il utilise, d'une part le systéme de portlet / portal de ofbiz, défini en XML, et un applicatif développé en javascript.

Pour l'instant ce composant est à l'état de POC (Proof Of Concept), c'est à dire que c'est un concrétisation de différentes idées afin de pouvoir en discuter plus facilement.
Il est développé dans un état d'esprit "Agile" c'est à dire que les résultats concrets sont privilégiés en acceptant de générer une dette technique.

La documentation qui suit a pour objectif,

- d'une part d'expliquer où en est le composant et comment il fonctionne et quels sont les points choisis et ceux laissés de coté
- d'autre part de préparer la futur documentation en détaillant les éléments à destination des futur utilisateurs

L'objectif majeur d'utiliser un framework javascript pour l'interface utilisateur est d'augmenter le nombre d'élément interactif dans les écrans, même quand les écrans sont construit / paramétré à partir de "module" existant en standard :

- mettre à jours une partie de l'écran en fonction d'une action ou d'une mise à jours de donnée;
- modification de formulaire en cours de saisi en fonction de la saisi des premmier champ;
- paramétrage simple d'un écran (page portal) sans avoir à se préoccuper de l'interaction entre portlet.

7.1. POC Vuejs Portal

Actuellement le premier applicatif javascript permettant d'utiliser les portlets - portal Apache OFBiz est écrit avec le framework VueJs.

Il a été choisi pour sa simplicité d'apprentissage et en raison du pilotage de son développement par une communauté plutôt que par une société.

7.1.1. Installation POC VuejsPortal

1. à partir d'un ofbiz-framework (trunk) téléchargé à partir du svn (documentation standard SourceRepository) une version svn supérieur à 1837232
2. créer le répertoire plugins dans ofbiz-framework
3. télécharger, avec git, dans le répertoire plugins :
    a. le plugin vuejsportal à partir du https://gitlab.ofbizextra.org/ofbizextra/ofbizplugins/vuejsPortal

b. le plugin example (de Apache OFBiz) modifié pour les besoins du POC à partir de https://gitlab.ofbizextra.org/ofbizextra/ofbizplugins/example et en se mettant sur la branch vuejsPortal

4. Il faut ensuite modifier quelques fichiers dans ofbiz-framework.
   Pour se faire, executer la commande suivante en étant positionné dans le repertoire 'plugins/VuejsPortal' :
   `tools/applyOfbizFilesPatchs.sh`
   puis ajouter quelques fichiers
   `rsync -a --exclude-from=./ofbizFiles/rsyncExcludes ./ofbizFiles/ ../../`
5. Pour le build de l'applicatif vuejs il faut avoir nodejs installé sur son environement
   a. Pour installer node.js dans un environnement debian, executer le commandes suivantes :
      `curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -`
      `sudo apt-get install -y nodejs`
      Dans le cas d'un autre environnement veuiller consulter la page https://nodejs.org/en/download/package-manager/#debian-and-ubuntu-based-linux-distributions
   b. en attendant une commande intégré à gradle
      i. il faut se positionner à la racine de la webapp vuejs 'plugins/vuejsPortal/webapp/vuejsPortal'
         - `npm i` ⇐ pour charger toutes les dépendances
         - `npm run build` ⇐ pour builder
6. il est maintenant possible de lancer ofbiz classiquement :
   `./gradlew cleanAll loadAll ofbiz`
   (à partir de la **racine de ofbiz-framework**)
7. aller se connecter à l'application portal https://localhost:8443/exampleapi/control/main

7.1.2. Choix pour le POC

Usage de la **branche trunk** de Apache OFBiz.

Usage des principes et architecture de **Portal Page et Portlet** de Apache OFBiz.

L'ensemble des exemples d'usage sont **avec le composant example** et pendant la durée du POC même des nouveaux fichiers qui devrait être placé dans common sont mis dans example afin de simplifier l'installation et mise à jours.

Une **webapp supplémentaire exampleapi** a été créé pour centraliser les uri autorisé pour le composant vuejsPortal, c'est paramétré dans le fichier constantes.js.
Le composant vuejsPortal n'est pas spécifique à un composant, il necessite juste une uri de base pour ses requêtes. Cette uri est paramètré dans le fichier constantes.js pour l'instant, mais dans le futur le paramétrage sera plus souple afin de pouvoir utiliser vuejsPortal avec plusieurs composants, y compris si cela doit apparaitre comme plusieurs "webapp" dans le menu utilisateur.

Le composant portal (https://localhost:8443/exampleapi/control/login) utilise le méchanisme **ofbiz standard de login via cookies**, il pourra par la suite utiliser le login par token de ofbiz (cf https://issues.apache.org/jira/browse/OFBIZ-9833).

Les **screens, forms, menus** utilisés pour les portlets sont définis **en xml**, et dans des fichiers dédiés pour plus de lisibilité.

Il y a **un composant (vuejs) par élément screen ofbiz** (SingleItemRow ⇒ vue-single-item-row), qui sont défini via les renderer au niveau screen, form, menu (et pour les renderer html qui correspondent globalement au macro.ftl). Pour l'instant, pour le POC, pour avoir la liste, il faut faire une recherche sur les appels `ouput.*Screen(` .

Balise XML utilisé

Pendant le POC, trés peu de nouvelle property ou balise XML n'est ajouté mais certaine balise ou property sont détournés pour des usages VueJs exclusif.

- screen
  - dans la balise `<container`, `auto-update-target` est utilisé pour le nom du watcher associé à ce container
  - dans la balise `<container`, `id` est utilisé comme identifiant de ce container permettant de lui injecter (par la suite) du contenu via un "setArea"
- form
  - dans la balise `<on-event-update-area` (il peux y en avoir plusieurs, toutes seront executer dans l'ordre de présence dans le xml
    - `event-type`
      pour les 3 valeurs suivante, event-type permet de déterminer l'action a exécuter, lors du submit de la form (clic sur le button "submit")
      - `setArea` Mettre à jour l'area ayant l'identifiant `areaId` avec le retour (un get) de l'uri `area-target` et avec les éléments `parameter` inclus en tant que paramètre.
      - `setWatcher` Mettre à jour le watcher `areaId` avec le/s `parameter` inclus en tant que paramètre/s. Si aucun paramètre n'est renseigné, le watcher est mis à jour avec l'integralité des champs contenus dans la form.
      - `submit` il utilise la property `area-target` en tant que uri pour le post avec l'ensemble des champs de la form en tant que paramètre. <A-VERIFIER>Si area-target est vide alors il utilise la `target` de la form. Le retour de la requête est attendu avant d'enchainer sur le `<on-event-update-area` suivant si le retour est success, sinon le/s message/s d'erreur/s sont affiché/s.</A-VERIFIER>
  - dans la balise `<on-field-event-update-area` (il peux y en avoir plusieurs, toutes seront executer dans l'ordre de présence dans le xml
    - `event-type` permet de déterminer l'action a exécuter, lors du submit de la form (clic sur le button "submit")
      - `post` il utilisera la property `area-target` en tant que uri pour faire un post avec les éléments `parameter` inclus en tant que paramètre. <A-VERIFIER>Si area-target est vide alors il utilise la `target` de la form. Le retour de la requête est attendu avant d'enchainer sur le `<on-event-update-area` suivant si le retour est success, sinon le/s message/s d'erreur/s sont affiché/s.</A-VERIFIER>
      - `setArea` Mettre à jour l'area ayant l'identifiant `areaId` avec le retour (un get) de l'uri

area-target et avec les éléments parameter inclus en tant que paramètre.

- setWatcher Mettre à jour le watcher areaId avec le/s parameter inclus en tant que paramètre/s.
- submit il utilise la property area-target en tant que uri pour faire un submit avec l'ensemble des champs de la form en tant que paramètre. <A-VERIFIER>Si area-target est vide alors il utilise la target de la form. </A-VERIFIER>
- setFieldInForm il utilise la property area-target en tant que field au sein de la form area-id afin d'y attribuer la valeur du champs qui comporte l'instruction.

◦ dans la balise ···-event-update-area

- auto-parameters-portlet permet de générer l'ensemble des paramètres nécessaire aux portlet, c'est à dire
  - portalPageId
  - portalPortletId
  - portletSeqId
  - currentAreaId

◦ dans la balise <hyperlink

- link-type="anchor" est utilisé pour signifier que c'est un appel interne FrontJs
- target-window est utilisé pour désigner le nom du watcher qui doit être mis à jours

- menu
  ◦ dans la balise <link
  - link-type="anchor" est utilisé pour signifier que c'est un appel interne FrontJs
  - target-window est utilisé pour désigner le nom du watcher qui doit être mis à jours

Renderer FrontJs

Un **nouveau viewHandler** ainsi qu'un nouvel ensemble de **renderer** a été créé.
Un nouveau package à été créé dans org.apache.ofbiz.widget.renderer frontJs qui contient l'ensemble des nouveaux renderer.

Dans ce package, il y a une class FrontJsOutput qui permet de construire les éléments nécessaires au format de sortie souhaité. Un objet de cette class est instancié en début de traitement du viewHandler, puis est compléter par les appels a chacun des objet renderer.

L'uri **showPortletFj** du composant exampleApi utilise le nouveau viewHandler pour retourner le resultat du rendu du screen, showPortlet.

cette uri est utilisé par l'applicatif vueJs pour toutes les requêtes de récupération des informations d'affichage.
Le viewHandler retourne, au format json deux ensemble (map) d'élément (viewScreen et viewEntities), la première le/s affichage/s et la seconde pour les données.

Dans les futures versions, il sera surement faisable de ne souhaiter recevoir que la map de données.

ShowPortlet, ShowPortal

Le screen **showPortlet est redéfini** dans le POC (dans example/widget/example) par simplification, pour faire ce que l'on souhaite et seulement ça.

Il y a une uri showPortal dans l'exampleApi qui est dédié FrontJs.

Champ complémentaires

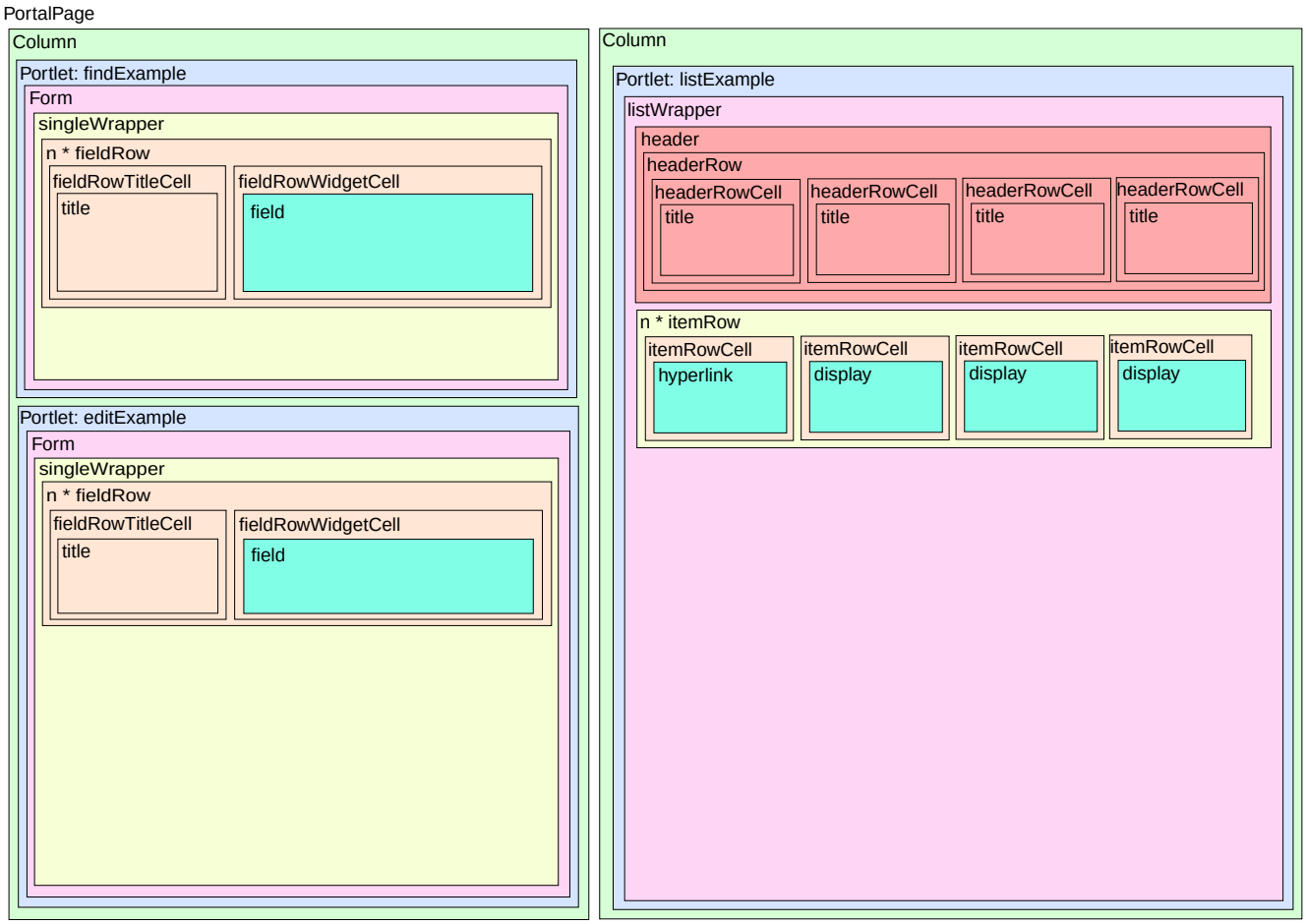
Ajout d'un champ dans l'entity PortalPagePortlet **watcherName** qui correspond au nom du watcher qui doit déclancher la mise à jour de la portlet

7.1.3. Situation actuel du POC

L'objectif est de faire fonctionner des pages portails correspondant à différent useCase.
Au niveau du menu, il y a les différentes page, pour l'instant le nom de la page portail est ExampleFrontJs et ExampleFrontJs2.
La première page contient 2 colonnes avec dans la première 2 portlets, le findExample puis le editExample et le listExample dans la seconde colonne.
La seconde page se rapproche de la page Mgmnt avec 2 portlets par colonne.

Pour chacune de ces portlet un composant portlet est créé dans le front avec le nom de la portlet et celui-ci s'initialise avec le showPortlet.

Le showPortletFj communique au composant la grappe de composants fils à créer (ViewScreen) et eventuellement le jeu de données nécessaire à son utilisation (ViewEntities).
Actuellement le format utilisé est json.



ViewScreen et ViewEntities

Actuellement tous les refresh d'écran (ou de portlet) font appel au FrontJsViewHandler qui renvoi la map ViewScreen et la map ViewEntities.
Dans une logique client FrontJs, la map ViewScreen devrait être reçu uniquement la première fois, contrairement à la map ViewEntities qui contient les données.

Actuellement, la gestion des "use-when", quelque soit le niveau (screen, form, field, menu) est réalisé (dans OFBiz) au niveau de la gestion du modèle donc cette information n'est pas accessible dans les rendrerer.
Dans le futur, il faudra transmettre au rendrer l'ensemble des éléments screens mais s'il y avait un use-when ajouter une données complémentaire boolean pour signaler qu'il faut l'afficher ou non. Ainsi, il sera possible lors des prochains appels à l'envoi de donnée de transmettre le/s boolean en question avec la/es valeurs correspondantes, afin de pouvoir mettre à jours correctement l'écran final.

7.1.4. Les TODO restant du POC

Mineurs

Ces TODO sont plutot dans l'ordre de priorité. Les premièrs étant les plus prioritaire et sont sensés ne concerner que l'étapes majeur en cours (c'est à dire pas la suivante ☺)

Le screen showPortlet doit vérifier que les paramètre suivant sont bien présent, c'est nécessaire pour la lecture des attribues :

- portalPageId
- portalPortletId
- portletSeqId

Le **showPortlet** doit vérifier le respect de la **sécurité** par rapport au deux champs SecurityServiceName et SecurityMainAction.

Majeurs

Normalement, il ne faut pas démarrer deux évolutions majeurs en paralléle, mais l'une après l'autre
histoire d'avoir le temps de tester ☺.

Dans la suite, les Notes correspondent à des évolutions qui sont encore en discussion.

Pouvoir choisir facilement si une portlet s'étend en hauteur ou en largeur. Cas d'usage: portlet search, soit sur une colonne soit en ligne.

A discuter, si la form doit inclure / géré les positions.

7.1.5. Les Cas d'Utilisation du POC

Opérationnels

Se logger sur l'application portal et afficher la page portail d'acceuil (ExampleFrontJs) Celle ci affiche les trois portlet findExample, listExample et editExample.

A mettre en oeuvre

Usage de la fonction recherche example
Test portlets FindExample and ListExample :

1. search by name with contain (data-obj testContain), result: 11 lines (assert with message "testContain 12")
2. search by name with BeginWith (data-obj testBegin), result: 35 lines (test with FindAndListExamples.checkTotalFound), so 2 pages test presence of nav-next (assert with message "Pb next button")
3. search by id (data-obj testId), result: only one line and id is on list (assert with message "testId link")
4. search by exampleType (data-obj testType), result: 25 lines and a next button (assert with

message "Pb next button2") click on next-page and check there is 5 line print (assert with message "testType 5")

5. search by name with contain (data-obj testContain1), result: less than 20 lines, so nav-next disappear, count number of line
6. sort by description (data sortList.sort_0) via FindAndListExamples.sortListBy
7. sort by name (data sortList.sort_1) via FindAndListExamples.sortListBy
8. check number of line has not change and first line name equal sortList.result_1
9. sort by id (data sortList.sort_2) via FindAndListExamples.sortListBy
10. check number of line has not change and first line name equal sortList.result_2

7.1.6. Details sur Vue.js

Vue.js general

todo

Composants

Un composant est défini par 3 block distinct (template, script, style) qui sont rassemblés au sein d'un fichier '.vue'

1. Le template :
   - Le template doit être contenu dans un élément unique (div, span, table, ect…)
   - Dans le template on peut faire reference à des éléments de la partie script (variables, fonctions) grâce à l'aide de doubles accolades: **{{}}**
   - Les attributs html classiques (id, type, class, style, ect…) peuvent être précédé de '**:**' pour être lié aux données/fonctions du script (ex: :class="data.class.alert")
   - Des directives sont mises à disposition par Vue.js afin de faciliter la logique dans le template (v-for, v-if, v-on:click, ect…). Ces directives propres à Vue.js sont liées par default au context du script et n'ont pas besoin d'être précédées par '**:**'
2. Le script :
   - La section script est principalement constituer d'un 'export default {}' qui va contenir l'ensemble des éléments du script.
     TIP: Cet export peut être précédé par des import du paquets node.js ou de fichiers (image, script)
   - Cet export est un table de hash pouvant contenir les clefs data/method/computed/props
     - data() est un fonction qui contient les variables du composant.
       Toute variables declarées dans data() avant la création du composant ou rajouter grâce à la méthode Vue.set() sont réactives.
       C'est à dire que Vue.js va suivre les évolution de ses valeur et les répercuter partout où elle sont utilisées.
     - computed et une table de hash contenant uniquement des fonctions dont le resultat est évalué en fonction d'au moins une propriété réactive.

Chacune de ces fonctions va être réévalué dès que une des propriété dont elle dépend est mise à jour.
Ces fonction sont donc réactives et Vue.js va également répercuter ses changements à tout les endroits où elle est utilisée.

- methods est une table de hash contenant des fonctions 'helper'.
  Elles peuvent être utilsées aussi bien dans le template que dans le script.
  WARNING: Ces fonctions ne seront jamais réévaluer automatiquement par Vue.js
- props est un tableau de string qui defini quelles sont les paramètre que le composant peut recevoir de son parent.
  Elle sont utilisées dans le code sous la forme: 'this.props.nomDeLaProps'.

◦ Cette section peut aussi contenir des **hook** existants dans le cycle de vie du composant (created(), mounted(), beforeUpdate(), ect…).
Voir ci-dessous :
[Component lifecycle]
WARNING: Lors du **hook** created() les valeurs computed et les methodes du composant ne sont pas encore créer.

3. Le style
   Cette section permet de definir le style du composant en css.
   Cette section sera prioritaire sur le style general du projet.

Vuex

Vuex est le systeme de centralisation de l'état de l'application. Il permet de créer des **stores** qui nous serviront à stocker des données accessible et modifiable à tout niveau de l'application.

Un store contient 4 élément :

1. Le State
   Le state est une table de hash contenant les informations.
   Ce state ne peut être modifier que par des mutations.
   Les données présentes dans le state à l'initialisation de celui-ci seront réactive.
   Les données rajoutées après l'initialisation du store devront être crées à l'aide de la methode Vue.set() afin d'activer le suivi des modifications et de la rendre réactive.
2. Les Mutations
   Les mutations sont une table de hash contenant les fonction capable de changer le **state**.
   Par convention la clef de ces fonction ne doit contenir que des majuscules.
   Les mutations ne peuvent pas être déclenchées directement à partir du code, elle doivent être appelées par une action.
   Les mutations **doivent être synchrones**.
3. Les Actions
   Les actions servent à déclencher les mutations.
   A la manière d'un setter elles controlent que les nouvelles valeurs correspondent à ce que l'on attends.
   Une action **peut être asynchrone**, dans ce cas elle retourne une promesse.
4. Les Getters
   Les getters permettent de consulter une/des valeur(s) du state.

Les getters sont réactif.
Les getters dans certain cas peuvent être paramétrés, dans ce cas ils retournent une fonction qui prendra des paramétres mais perdront leur faculté d'être réactifs. Ils seront réévalué seulement lorsque l'on rééxecuteras celui-ci ou qu'on en changera les paramétres.

Le store peut être composant de modules.
Dans ce cas à la racine du repertoire store on créé un fichier index.js que se charge d'importer Vuex et les stores que l'on placera dans un sous-repertoire modules.

Reactivity

todo

7.2. UI générique et modulaire

Afin d'avoir un ERP modulaire et ouvert, il est important que la gestion de l'interface Utilisateur permette, à partir d'un ensemble de "module" écran, de gérer de multiple écran pour gérer le même type d'objet métier dans différent type de contexte de l'entreprise.

Les chapitres suivants décrivent un POC pour un système de gestion de l'interface utilisateur afin de vérifier qu'il répond à ce besoin de modularité.

To be able to have a ERP modular and open, it's imprtant that User Interface must allow, using a set of screen "block", to manage multiple screen to manage the **same business object** type in **differents business context**.

Chapter following, describes a POC for checking if a User Interface Management system meets this need for modularity.

7.2.1. Use cases for POC-UI

Theses use cases are to be used for new UI POC, documentation associated and selenium unit task test.

All these use cases should be done with existing entities and services, if it's necessary to develop one, simplify the use case, the goal is UI, not service or entity.

These use case description are done in a agile philosophy, only main point is present and during realization details choices will be discuss and done.

Preliminary remarks :

1. In this document, the term "application" corresponding to "plugin component" in the ofbiz terminology which is not same as a "applications/trunk component" in ofbiz terminology. An application is dedicated for a business purpose for a user type, it's build by assembling piece of ofbiz components, sometimes without any specifics entities, services and screen (ex: CRM-B2C, CRM-B2B, SFA are 3 applications uses by sales men)
2. Each use case is on an "application" and is part of one of the menu of this application.

Of course, this document describe only menu-option needed by the use-case. As it's difficult to do a clear definition of "web-page" because it's depending of theme/template, use case is for me at two level :

a. screen, which can be well define
b. page, which depend on default theme

Of course, some of use case (screen or page) will be done by a previous one (ex : sometime edit is done by the same screen as add). It's even, one of the re-usable important point (UI, Doc and Selenium)

3. Each time a line (or point) start by a "?" the question is "is this point is needed ?"

7.2.2. Release and goal

Goal is the POC realization, not doing multiple super applications. POC realization should generate discussion and decision, each time be careful to mainly discuss on UI, Doc or Selenium and not on use case business justification. Use case are to be realistic but mainly as a support for a specifics UI, Doc or Selenium needed.

V1

Main Goal is to proof the technical points. Of course UI re-organization is the driver for this phase, documentation start to work in parallel but will wait first realization to test integration for help. For selenium unit task, it will be a simple usage of the new UI with the default theme, the sole purpose being to check that test with default theme is simple.

So never mind if all cases is not done, the list exist to give different classic case what we will need to know how to process it. Some case which seem to be duplicate from other can be use for beginner who want to help and check if the concept is understanding.

During V1 realization, UseCase list will be updated

V2

Main Goal is to check if the solution is **useful for the different type of contributors**

- experiment developer
- beginner developer
- functional consultant for parameters
- functional consultant for personalization
- ? end user - for parameters ? (if it's still possible with the new architecture)

Use Case list will be use as a deployment plan. The list contains similar case to these which are realize in V1, so those on V2 can be achieve by all types of contributors.

Documentation Goal

- Apache OFBiz User documentation (asscidoc)
- Apache OFBiz web site wiki
- OFBiz Help

Selenium test

1. demo data for each use case and scenario
2. selenium scenario test for each page (or page group)
3. selenium unit test for each screen

7.2.3. Simple Party Management

Which Sub-Application

1. in HR application, simple employee management
2. in CRM B2C application, simple customer (person) management
3. in eCommerce, simple profile page
4. in HR application, simple organization/place (group) management
5. in CRM B2B application, simple customer (company) management
6. in Facility application, simple worker management

Which Party sub-component

1. Party - Person - PartyGroup
2. Contact Mech,
    - with postal address, phone and mail;
    - one or two fixes purpose (ex: phone fix number and mobile phone number)
3. Role
4. Party Identification
5. Party association
6. Not ~~**UserLogin**~~ because all Security entities should be use and it will generate a too large domain for this POC

Which Screen

1. Party
    - find, list
    - A person
        - add / edit, show
    - A group

        - add / edit, show
    - A company
        - add / edit, show
    - show a synthesis view (party/person/company, contact informations, roles, Identification)
        - Person
        - Company
        - PartyGroup
2. Contact information
    - all contact informations (for one party / facility)
        - with and without purpose
        - with and without history
        - deactivated
    - add / edit postal address
    - add / edit mail
    - add / edit phone
3. Role
    - list for a party
    - add a role (for a parent RoleType)
    - add a role in two step :
4. select parent RoleType
5. select the role
    - remove a role
6. Party Identifications
    - list, add, remove

7.2.4. HR Employee management

In HR Component, starting person management with the more complete form about person.

- Menu option to manage employee
    - find, list, show, add, edit and manage his
        - contact information
        - identification (3 idTypes, one mandatory, two optionals)
- template page with a header (or sidebar or ...) to show on which employee we are

Use Case Screen :

1. find Person

    - simple form (only on party or person)
    - with an add button (which can be show or not depending on parameter or authorization)
2. Person list with an add button (which can be show or not depending on parameter or authorization)
3. add a Person
4. show a Person
5. show a Person with sub-menu with two options : contact informations and Identifications
6. edit a Person
7. List of all contact informations for a person, with an add button (which can be show or not depending on parameter or authorization)
8. add a postal address
9. add a mail
10. add a phone number (to go step by step, without purpose management, will be done in next Use Case group)
11. edit a postal address
12. edit a mail
13. edit a phone number
14. List of all identification number for a person, with an add button (which can be show or not depending on parameter or authorization)
15. add a identification number with choice of identification type
16. edit a identification number with choice of identification type
17. add a identification number with a fix identification type
18. edit a identification number with a fix identification type

Use Case Page :

1. create a person
2. search a person
3. visualize a person
4. manage informations about a person
5. template page with a header (or sidebar or ...) to show on which employee we are, (for example to show all his knowledges, or his skills, or his positions, or his ...)
6. manage informations about a person on one page, and with access at this page directly by a field (auto-completion on id, first, last name)

7.2.5. CRM B2C, customer mgnt

In a CRM B2C application, the customer (so, in this context, a person) management.
The difference from previous use case group is :

1. person form is more simple than in HR
2. role will be used to characterize customer position (suspect, prospect, with_Quote, customer)

Menu option to manage employee

- find (with role field), list, show, add, edit and manage his
  - contact informations
  - identification (3idTypes, one mandatory, two optionals)
- template page with a header (or sidebar or ...) to show on which customer we are

Use Case Screen :

1. find Person with an add button (which can be show or not depending on parameter or authorization)
   - search field same as in HR find person
   - role field which can appear or not, when not appear a fix value has been put as parameters.
   - contact information field, phone, mail, town. These fields can be show or not by the user with a "deploy" button
2. Person list with an add button (which can be show or not depending on parameter or authorization)
   - role field appear or not, when not appear a fix value has been put as parameters, so only person with this role appear
3. add a Person, all main informations in the form
   - role
   - less field about person than in HR form
   - 1 postal address
   - 2 phone number
   - 1 identification number
4. show a Person, all main informations in the screen with indicator for contact information and identification when there are more data that what it's show.
5. show a Person with sub-menu with options :
   - contact informations
   - Identifications
   - role history
   - change role : a direct action button
6. edit a Person, only "Person" field
7. a button bar to change role (ex: for a suspect, there are the 3 options), this use case is for having a action bar, in this business process case it's maybe not a need, but for more complex object like order or task, it's a classical need.

8. List of all contact informations for a person, with one or multiple add buttons (which can be show or not depending on parameter or authorization) and purpose are show, it's the second step, with purpose management.
9. add a postal address (or just a purpose)
10. add a mail
11. add a phone number
12. edit a postal address
13. edit a mail
14. edit a phone number
15. List of all identification number for a person, with an add button (which can be show or not depending on parameter or authorization)
16. add a identification number with choice of identification type
17. edit a identification number with choice of identification type

Use Case Page

1. create a new entry in CRM (role is choose during creation)
2. search a "customer" (or suspect, prospect, ...)
3. visualize a "customer"
4. manage informations about a "customer"
5. template page with a header (or sidebar or ...) to show on which "customer" we are, (for example to show all his quotes, or his orders, or ...)
6. manage informations about a person on one page, and with access at this page directly by a field (auto-completion on id, first, last name)

7.2.6. eCommerce, profile page

A simple profile page.
The difference from previous use case will be mainly on Use Case Page because eCommerce theme could be more original and public user interface should be, most of the time, more simple.

Use Case Screen :

1. show the person, all main informations in the screen with indicator for contact information and identification when there are more data that what it's show.
2. show the Person with sub-menu with options :
    - contact informations
    - Identifications
3. edit a Person, only "Person" field
4. List of all contact informations for a person, with an add button and purpose are show, purpose is need for invoice or shipping.

5. add a postal address (or just a purpose)
6. add a mail
7. add a phone number
8. edit a postal address
9. edit a mail
10. edit a phone number

Use Case Page :

1. visualize the profile (the person) with edit button
2. manage his contact informations
3. manage his identifications
4. All in one page, which can be look as a long page.

7.2.7. HR organization mgnt

In HR component, a simple organization/place (group) management.
Now PartyGroup management (very simple), but with complex screen to manage hierarchy. In this use case group we will use the word "group" for service or department, or subsiadiry.

- Menu option to manage the Company organization
  - manage group
  - associated employee in a group
  - manage a hierarchy of group

Use Case Screen :

1. find group (with a specific partyType)
   - simple form (only on party or partyGroup)
   - with an add button (which can be show or not depending on parameter orauthorization)
2. PartyGroup list with an add button (which can be show or not dependingon parameter or authorization)
3. add a group
4. show a Person, all informations in screen with sub-menu with two options : contact informations and Identifications
5. edit a Group
6. List all contact informations for a group, with an add button (which can be show or not depending on parameter or authorization)
7. add a postal address
8. add a phone number
9. edit a postal address

10. edit a phone number
11. List all identification number for a group, with an add button (which can be show or not depending on parameter or authorization)
12. add a identification number with choice of identification type
13. edit a identification number with choice of identification type
14. add a identification number with a fix identification type
15. edit a identification number with a fix identification type
16. List all person associated to the group with two add buttons (which can be, individually, show or not depending on parameter or authorization)
    - add a manager
    - add a member
17. List all group associated to the group (the child) with two add buttons (which can be, individually, show or not depending on parameter or authorization)
    - add an existing group as a child
    - create a new group and add it as a child
    - in the list, each group is a link to this screen, to be able to navigate top-down
    - a third button to go to the parent level, to be able to navigate bottom-up
    - the name of the group manager appear above the list
18. ? List all parent group for a group or for a person ?
19. show group hierarchy as a tree with action or detail at each level, top-down
20. show group hierarchy as a tree with action or detail at each level, bottom-up

Use Case Page :

1. search a group
2. manage a group
3. manage its contact informations
4. manage hierarchy step by step (parent to child or child to parent)
5. manage hierarchy with a tree view
6. in HR employee, show the tree, top-down or bottom-up with the template "for an employee"

7.2.8. CRM B2B customer mgnt

In a CRM B2B application, the customer (so, in this context, a company) management.
For clarification, in these Use Cases, B2B is an other application than B2C.
The "CRM B2C & B2B" will be a third, but not in this list because it contains no specificity on screen-page definition

The main difference between B2C is :

1. company versus person,
2. contact management with PartyAssociation
3. ? customer organization management ?

Use Case Screen :

1. find customer (a company (specific partyType)) with an add button (which can be show or not depending on parameter or authorization)
    - search field are on multiple entities with some part deploy or not
    - role field which can appear or not, when not appear a fix value has been put as parameters.
    - contact information field, phone, mail, town. These fields can be show or not by the user with a "deploy" button
2. Company list with an add button (which can be show or not depending on parameter or authorization)
    - role field appear or not, when not appear a fix value has been put as parameters, so only company with this role appear
3. add a Company, all main informations in the form
    - role
    - field from PartyGroup
    - 1 postal address
    - 2 phone number
    - 2 identification number
4. show a Company, all main informations in the screen with indicator for contact informations and identification when there are more data that what it’s show.
5. show a Company with sub-menu with options :
    - contact informations
    - Identifications
    - role history
    - change role : a direct action button
6. edit a Company, only "Company" field
7. a button bar to change role (ex: for a suspect, there are the 3 options), this use case is for having a action bar.
   In this business process case it’s maybe not a need, but for more complex object like order or task, it’s a classical need.
8. List of all contact informations for a company, with an add button (which can be show or not depending on parameter or authorization) and purpose are show, (so, with purpose management).
9. add a postal address (or just a purpose)
10. add a mail

11. add a phone number with purpose
12. edit a postal address
13. edit a mail
14. edit a phone number
15. List of all identification number for a company, with an add button (which can be show or not depending on parameter or authorization)
16. add a identification number with choice of identification type
17. edit a identification number with choice of identification type
18. list of contact (person) associated to this company with an add button (which can be show or not depending on parameter or authorization)
    - a contact is a person with contact information
    - list with only one line per contact
    - list of block with contact details for each
19. edit a contact or his contact information

Use Case Page :

Exactly the same as the CRMB2C

1. create a new entry in CRM (role is choose during creation)
2. search a "customer" (or suspect, prospect, ...)
3. visualize a "customer"
4. manage informations about a "customer"
5. template page with a header (or sidebar or ...) to show on which "customer" we are, (for example to show all his quotes, or his orders, or ...)
6. manage informations about a company on one page, and with access at this page directly by a field (auto-completion on id, first, last name).

7.2.9. Facility worker mgnt

In Facility application, simple facility's worker management.
For this last use case group, it's a simplification of the previous one.
Only a very simple and short process for adding people.

It's the last one, because the goal is to check if it's easy and rapid to create (or parametrize) a new small application from existing one.

In the Warehouse Management application (simple version OOTB)

- in the administration menu
    - the user menu to manage internal user per facility In the standard business process, it will be used mainly for login and authorization, in our case we will only manage person, his phone number and his facility (where he's authorized)

- the facility menu to manage contact informations and person authorized

Use Case Screen :

Already existing screen used

1. find Person
   - simple form (only on party or person)
   - with an add button
2. Person list with an add button
3. add a Person, simple form 3-6 fields
4. show a Person
5. show a Person with sub-menu with option to manage contact informations
6. edit a Person
7. List of all contact informations for a person, with one or multiple add button
8. add a mail
9. add a phone number
10. edit a mail
11. edit a phone number

New Screen

1. add a facility, simple form, if service exist, including some contact informations
2. List of all existing facility
3. List of all contact informations for a facility, with one or multiple add button
4. List of all persons associated to the facility, with two add button
   - add an existing person
   - create a new person and add it to the facility
5. List of all facility associated to a person, with one add button
   - add an existing facility

Use Case Page :

1. manage facilities
2. manage persons
3. visualize a facility details (info, contact informations, persons associated)

7.3. Portlet

Une portlet est une portion d’écran "autonome" c’est à dire qu’il peut y avoir des actions qui ne concerne que cette portion d’écran et qui sont déclanché par des éléments interne à elle.

Les portlets doivent permettre une grand modularité de l'interface utilisateur, aussi une action utilisateur (clic sur un lien, un bouton submit, ...) ne doit pas indiquer la portlet qui doit se mettre à jour mais un nom logique auquel pourront s'abonner une ou plusieurs portlet.

Ce nom logique auquel la portlet s'abonne est la watcherName, c'est un champ qui est dans la table d'association entre PortalPage te PortalPortlet

7.3.1. Mise à jour Portlet

C'est la mise à jours de donnée au niveau du store du client Js qui déclanche la mise à jours de la portlet

7.4. Portal Page

7.5. Dev and prod

7.5.1. message de retour

- *EVENT_MESSAGE_LIST* : pour obtenir un event message list il faut; faire un creatExample ayant comme statusId="EXST_COMPLETE".
- *EVENT_MESSAGE* : pour obtenir un EventMessage il faut; faire un creatExample ayant comme statusId="EXST_APPROVED".
- *ERROR_MESSAGE* : pour obtenir un ErrorMessage il faut; que le exampleTypeId et que le description soient null.
- *ERROR_MESSAGE_LIST* : pour obtenir un ErrorMessage list il faut; que le exampleTypeId soit null.

7.6. FrontJs Glossary

watchers

c'est nom du store VueJs utilisé pour stocker les différentes variable sur lequel les container ou portletsont abonné et donc se mette à jours quand celui-ci change.

watcherName

c'est le nom du champ (dans l'association Portlet - PortalPage) qui contient le nom de la variable dans watchers permettant de mettre à jours cette portlet dans cette PortalPage (cf *Portlet* .

Store entities

c'est le store qui stock l'ensemble des données utilisé sur la page portail, il est organisé par "record" de manière à pourvoir être utilisé aussi bien en tant que ligne de list que en tant que Screen Single.

8. Core APIs

9. Development environment

9.1. Setup your environment

9.1.1. Java SE

9.1.2. IDE

Eclipse

Intellij Idea

pour le hotswap, il faut importer (menu file project structure libraries clicker sur plus pour ajouter ofbiz-framework/build/libs/ofbiz.jar tous les modules (Ctrl A)

9.1.3. Database

Unresolved directive in developer-manual_fr.adoc - include::../../plugins/vuejsPortal/src/docs/asciidoc/webhelp_example_fr.adoc[leveloffset=+2]

9.2. Web tools

Unresolved directive in developer-manual_fr.adoc - include::../../plugins/example/src/docs/asciidoc/example_fr.adoc[leveloffset=+2]

10. Testing

10.1. Unit Tests

10.2. Integration Tests

10.3. UI Tests

Il existe un projet dédié "OFBiz® Selenium-WebDriver" (OfbSwd) dont le rôle est exclusivement les tests via l'interface utilisateur.

Cela couvre, aussi bien les tests unitaires de l'interface utilisateur que des tests de use case métier.

Pour plus de détail se référer à la documentation de ce projet

11. Deployment

12. Security

12.1. Passwords and JWT (JSON Web Tokens) usage

12.1.1. How are set and used passwords and JWT in Apache OFBiz

The Apache OFBiz Project Release 17.12

Passwords

Demo and seed passwords are stored in files loaded through security ofbiz-component.xml. To know more about that be sure to read:

- The technical production setup guide notably "Initial Data Loading" and "Security Settings" sections
- How to secure your deployment

These configuration steps are not to be neglected for the security of a **production environment**

JWT usage

As says Wikipedia:

> JSON Web Token (JWT) is an Internet standard for creating JSON-based access tokens that assert some number of claims.

We currently use JWT in 2 places:

1. To let users safely recreate passwords (in backend and frontend)
2. To allow SSO (Single Sign-on) jumpings from an OFBiz instance to another on another domain, by also using CORS (Cross-origin resource sharing) on the target server

How to secure JWT

When you use JWT, in order to sign your tokens, you have the choice of using a sole so called secret key or a pair of public/private keys: https://jwt.io/introduction/.

You might prefer to use pair of public/private keys, for now by default OFBiz uses a simple secret key. Remains the way how to store this secret key. This is an interesting introduction about this question.

1. The first idea which comes to mind is to use a property in the security.properties file. It's safe as long as your file system is not compromised.
2. You may also pick a SystemProperty entity (overrides the file property). It's safe as long as your DB is not compromised.

3. We recommend to not use an environment variable as those can be considered weak:
    - http://movingfast.io/articles/environment-variables-considered-harmful
    - https://security.stackexchange.com/questions/49725/is-it-really-secure-to-store-api-keys-in-environment-variables
4. You may want to tie the encryption key to the logged in user. This is used by the password recreation feature. The JWT secret key is salted with a combination of the current logged in user and her/his password. This is a simple and effective safe way.
5. Use a JTI (JWT ID). A JTI prevents a JWT from being replayed. This auth0 blog article get deeper in that. The same is kinda achieved with the password recreation feature. When the user log in after the new password creation, the password has already been changed. So the link (in the sent email) containing the JWT for the creation of the new password can't be reused.
6. Tie the encryption key to the hardware. You can refer to this Wikipedia page for more information.
7. If you want to get deeper in this get to this OWASP documentation

Note: if you want to use a pair of public/private keys you might want to consider leveraging the Java Key Store that is also used by the "catalina" component to store certificates. Then don't miss to read:

- https://cryptosense.com/blog/mighty-aphrodite-dark-secrets-of-the-java-keystore/
- https://neilmadden.blog/2017/11/17/java-keystores-the-gory-details/

Also remember that like everything a JWT can be attacked and, though not used or tried in OFBiz yet, a good way is to mitigate an attack by using a KeyProvider. I have created OFBIZ-11187 for that.

Properties

The *security.properties* file contains five related properties:

```
# -- If false, then no externalLoginKey parameters will be added to cross-webapp urls
security.login.externalLoginKey.enabled=true
```

```
# -- Security key used to encrypt and decrypt the autogenerated password in forgot
password functionality.
#    Read Passwords and JWT (JSON Web Tokens) usage documentation to choose the way
you want to store this key
login.secret_key_string=login.secret_key_string
```

```
# -- Time To Live of the token send to the external server in seconds
security.jwt.token.expireTime=1800
```

```
# -- Enables the internal Single Sign On feature which allows a token based login
between OFBiz instances
# -- To make this work you also have to configure a secret key with security.token.key
security.internal.sso.enabled=false
```

```
# -- The secret key for the JWT token signature. Read Passwords and JWT (JSON Web
Tokens) usage documentation to choose the way you want to store this key
security.token.key=security.token.key
```

There are also SSO related SystemProperties in *SSOJWTDemoData.xml*:

```
    <SystemProperty systemResourceId="security"
systemPropertyId="security.internal.sso.enabled" systemPropertyValue="false"/>
    <SystemProperty systemResourceId="security" systemPropertyId="security.token.key"
systemPropertyValue="security.token.key"/>
    <SystemProperty systemResourceId="security"
systemPropertyId="SameSiteCookieAttribute" systemPropertyValue="strict"/>
```

Internal SSO

The introduction of the same-site attribute set to 'strict' for all cookies prevents the internal Single Sign On feature. Why is clearly explained here.

So same-site attribute set to 'none' is necessary for the internal SSO to work, 'lax' is not enough. So if someone wants to use the internal SSO feature s/he also needs to use the CSRF token defense. If s/he wants to be safe from CSRF attacks. Unfortunately, due backporting difficulties, this option is currently (2020-04-15) only available in trunk.

Fecth API

An alternative would be to use the Fetch Javascript API with the

```
credentials: "include"
```

option to enable CORS. Here is an example

For those interested, there are more information in https://issues.apache.org/jira/browse/OFBIZ-11594

Last but not least

Be sure to read Keeping OFBiz secure

12.2. Impersonation

12.2.1. What is Impersonation in Apache OFBiz

The Apache OFBiz Project Release 17.12

Introduction to User impersonation

User Impersonation is a feature that offer a way to select a user login and impersonate it, i.e. see what the user could see navigating through the application in his name.

How do this work ?

An authorized user *(see security and controls section for configuration)*, can select a user that will be impersonated.

The impersonation start, if everything is well configured, in current application (partymgr for the demo). Everything appears like if we were logged in with the userLoginId and the valid password (though we know nothing about it)

The only thing showing that we currently are impersonating a user is the little bottom-right image :

This icon indicates, when clicking on it, the user impersonated, and offer a way to depersonate.

The impersonate period is stored for audit purpose, and if the impersonator forgot to depersonate, the period is terminated *one hour* after impersonation start.

Security

This feature can draw some concerns about security aspect. This paragraph will introduce every controls and properties that have been implemented around the impersonation feature.

These configuration steps are not to be neglected for a **production environment** since this feature offer a way to act in place of another user.

Properties

The *security.properties* file introduce two properties that control impersonation feature :

```
security.disable.impersonation = true
```

This property, set by default to **true**, controls the activation of impersonation feature. If no configuration is done any user trying to use impersonation will face an error message, indicating that the feature is disabled.

To enable impersonation this property need to be set to **false**

```
security.login.authorised.during.impersonate = false
```

This property controls the way impersonation occurred to the impersonated user :

In default configuration, the impersonated user see nothing and can use the application without knowing that he is currently impersonated. Several authorized user can impersonate a same login without any issue.

This configuration is intended for testing/QA environment allowing any authorized user to impersonate a login to validate its configuration, test the application etc.

Set to **true**, this configuration improve the control of the data generated by the impersonated user. Indeed, Only one authorized user can impersonate a login at the same time, and during the impersonation process, the impersonated user is unable to act within the application.

Since the impersonation period is stored in database, the actions done by the authorized user can be identified if there is the need to do so.

This configuration is intended for production environment

Controls

The permission

First, to be able to use impersonation, a user need to possess *IMPERSONATE_ADMIN* permissions. Demo data offer *IMPERSONATION* security group for this purpose.
In demo data, *FULLADMIN* security group also possess the permission.

Permission based user restriction

An authorized user cannot impersonate any user. There are two main controls that will restrict the impersonation feature.

Cannot impersonate Admin user

It is impossible to impersonate a user that is granted any of the admin permission :

```
"IMPERSONATE_ADMIN"
"ARTIFACT_INFO_VIEW"
"SERVICE_MAINT"
"ENTITY_MAINT"
"UTIL_CACHE_VIEW"
"UTIL_DEBUG_VIEW"
```

Cannot impersonate more privileged user

It is impossible to impersonate a user that has more permission than your user. Even if the missing persmission is a minor one.

12.3. CSRF defense

12.3.1. How is done the CSRF defense in Apache OFBiz and how to adapt it if needed

The Apache OFBiz Project Release 17.12

The same-Site attribute

> The SameSite attribute is an effective counter measure to cross-site request forgery, cross-site script inclusion, and timing attacks.
>
> — According to OWASP ZAP

By default OOTB the SameSiteFilter property sets the same-site attribute value to 'strict. SameSiteFilter allows to change to 'lax' if needed. If you use 'lax' we recommend that you set the csrf.defense.strategy property to org.apache.ofbiz.security.CsrfDefenseStrategy in order to provide an effective defense against CSRF attacks.

Properties

The *security.properties* file contains related properties:

```
# -- By default the SameSite value in SameSiteFilter is 'strict'.
# -- This property allows to change to 'lax' if needed.
# -- If you use 'lax' we recommend that you set
# -- org.apache.ofbiz.security.CsrfDefenseStrategy
# -- for csrf.defense.strategy (see below)
SameSiteCookieAttribute=
```

```
# -- The cache size for the Tokens Maps that stores the CSRF tokens.
# -- RemoveEldestEntry is used when it's get above csrf.cache.size
# -- Default is 5000
# -- TODO: possibly separate tokenMap size from partyTokenMap size
csrf.cache.size=
```

```
# -- Parameter name for CSRF token. Default is "csrf" if not specified
csrf.tokenName.nonAjax=
```

```
# -- The csrf.entity.request.limit is used to show how to avoid cluttering the Tokens
Maps cache with URIs starting with "entity/"
# -- It can be useful with large Database contents, ie with a large numbers of tuples,
like "entity/edit/Agreement/10000, etc.
# -- The same principle can be extended to other cases similar to "entity/" URIs
(harcoded or using similar properties).
# -- Default is 3
csrf.entity.request.limit=
```

```
# -- CSRF defense strategy.
# -- Because OFBiz OOTB also sets the SameSite attribute to 'strict' for all cookies,
# -- which is an effective CSRF defense,
# -- default is org.apache.ofbiz.security.NoCsrfDefenseStrategy if not specified.
# -- Use org.apache.ofbiz.security.CsrfDefenseStrategy
# -- if you need to use a 'lax' for SameSiteCookieAttribute
csrf.defense.strategy=
```

There is also a SystemProperty in *SSOJWTDemoData.xml*:

```
<SystemProperty systemResourceId="security" systemPropertyId="SameSiteCookieAttribute"
systemPropertyValue="strict"/>
```

13. Appendices

14. From Mini Language to Groovy

This is a small guide for everybody involved in converting the Mini Language into Groovy.

Why is this important?
This tutorial is directly linked to the efforts of converting all scripts in Mini Language to newer Groovy Scripts. All of this is done, because Groovy is much more readable and easier to review, more up to date and many other reasons, which can be found here: Proposal for deprecating Mini Language

To contribute, or just be up to date with the current process, you can look at the existing JIRA issue OFBIZ-9350 - Deprecate Mini Lang

14.1. Groovy DSL (dynamic scripting library)

14.1.1. How to get Groovy support in your IDE

The following paragraph is for Eclipse users.

It is possible to get Groovy support in Eclipse by converting the loaded project to a Groovy Project. The project itself will work as before.

To do this just follow these few steps:

1. Right-click on the project that has to be converted
2. Click on "Configure"
3. Click on "Convert to Groovy Project"

Eclipse will automatically load the file OfbizDslDescriptorForEclipse.dsld , in which the known fields and methods used in Groovy Scripts are defined.

14.1.2. Known Fields

```
property name: 'parameters'
type : 'java.util.Map'
```
These are the parameters given to the Groovy Script, when it is called as a service. It is equivalent to `Map<String, Object>` context in the Java-Service-Definition.

```
property name: 'context'
type: 'java.util.Map'
```
More parameters, which are, for example, given through a screen or another Groovy Script. This is important when the script is called through an action segment of a screen.

```
property name: 'delegator'
type: 'org.apache.ofbiz.entity.Delegator'
```
Normal instance of the Delegator, which is used for special database access.

`property name: 'dispatcher'`
`type: 'org.apache.ofbiz.service.LocalDispatcher'`
Normal instance of the LocalDispatcher, which is used to call services and other service-like operations.

`property name: 'security'`
`type: 'org.apache.ofbiz.security.Security'`
Normal instance of the Security-Interface with which permission checks are done.

14.2. Known Methods

`method name: 'runService'`
`type: 'java.util.Map'`
`params: [serviceName: 'String', inputMap: 'java.util.Map']`
Helping method to call services instead of dispatcher.runSync(serviceName, inputMap). Also possible: run service: serviceName, with: inputMap

`method name: 'makeValue'`
`type: 'java.util.Map'`
`params: [entityName: 'String']`
Helping method to make a GenericValue instead of delegator.makeValue(entityName). Creates an empty GenericValue of the specific entity.

`method name: 'findOne'`
`type: 'java.util.Map'`
`params: [entityName: 'String', inputMap: 'java.util.Map']`
Helping method to find one GenericValue in the database. Used instead of delegator.findOne(entityName, inputMap)

`method name: 'findList'`
`type: 'java.util.List'`
`params: [entityName: 'String', inputMap: 'java.util.Map']`
Helping method to find many GenericValue in the database. Used instead of delegator.findList(entityName, inputMap, null, null, null, false)

`method name: 'select'`
`type: 'org.apache.ofbiz.entity.util.EntityQuery'`
`params: [entity: 'java.util.Set']`
Helping method used instead of EntityQuery.use(delegator).select(...)

`method name: 'select', type: 'org.apache.ofbiz.entity.util.EntityQuery', params: [entity: 'String…']`
As above.

`method name: 'from'`
`type: 'org.apache.ofbiz.entity.util.EntityQuery'`
`params: [entity: 'java.lang.Object']`
Helping method used instead of EntityQuery.use(delegator).from(...)

`method name: 'success'`
`type: 'def'`
`params: [message: 'String']`
Helping method used instead of ServiceUtil.returnSuccess(message)

`method name: 'failure'`
`type: 'java.util.Map'`
`params: [message: 'String']`
Helping method used instead of ServiceUtil.returnFailure(message)

`method name: 'error'`
`type: 'def'`
`params: [message: 'String']`
Helping method used instead of ServiceUtil.returnError(message)

`method name: 'logInfo'`
`type: 'void'`
`params: [message: 'String']`
Helping method used instead of Debug.logInfo(message, fileName)

`method name: 'logWarning'`
`type: 'void'`
`params: [message: 'String']`
Helping method used instead of Debug.logWarning(message, fileName)

`method name: 'logError'`
`type: 'void'`
`params: [message: 'String']`
Helping method used instead of Debug.logError(message, fileName)

`method name: 'logVerbose'`
`type: 'void'`
`params: [message: 'String']`
Helping method used instead of Debug.logVerbose(message, fileName)

The actual definition of the methods can be found in `/framework/service/src/main/java/org/apache/ofbiz/service/engine/GroovyBaseScript.groovy`, the variables `dctx`, `dispatcher` and `delegator` are set in the file `GroovyEngine.java` which can be found in the same location.

14.3. Services

14.3.1. From MiniLang to Groovy

To see additional examples and finished conversions, which may help with occurring questions, click: OFBIZ-9350 - Deprecate Mini Lang There is a chance that a similar case has already been converted.

When a simple-method ends, it will automatically at least return a success-map.

All the Groovy Services have to return success at least, too.

```
return success()
```

14.3.2. Getting started

MiniLang files consist of services, which, in most cases, implement services.

The get converted to Groovy like the following:

```
<!-- This is MiniLang -->
<simple-method method-name="createProductCategory" short-description="Create an
ProductCategory">
   <!-- Code -->
</simple-method>
```

```
// This is the converted Groovy equivalent
/**
 * Create an ProductCategory
 */
def createProductCategory() {
    // Code
}
```

It will be useful for future developers, and everybody who has to check something in the code, to put at least the short-description as the new Groovydoc. This will hopefully more or less explain, what the method should or shouldn't do. If the short-description isn't helpful enough, feel free complete it.

The structure of if and else in MiniLang is a little different than the one from Groovy or Java and can be a bit confusing when first seen, so here is an example:

```
<if-empty field="parameters.productCategoryId">
    <sequenced-id sequence-name="ProductCategory" field=
"newEntity.productCategoryId"/>
<else>
    <set field="newEntity.productCategoryId" from-field=
"parameters.productCategoryId"/>
    <check-id field="newEntity.productCategoryId"/>
    <check-errors/>
</else>
</if-empty>
```

Notice, that the else always starts before the if-tag is closed, but sometimes isn't indented as one would expect it.

When navigating through bigger `if`-phrases, the navigation itself will be much easier through just clicking in the opening or closing `if`-tag; Eclipse will automatically mark the matching opening or closing `if`-tag for you.

There are two possibilities to initialize a field/variable in Groovy.

1. To define a field/variable with its correct typing
   `String fieldName = "value"`\`
2. To just "define" a field/variable. The IDE you are working with may not recognize the typing, but OFBiz can work with it:
   `def fieldName = "value"`

14.4. Checking Fields

Minilang	Groovy
`<if-empty field="fieldName"></if-empty>`	`//checks if fieldName is existent and/or empty` `if (!fieldName) {}`

`<if-empty field="fieldName.property"></if-empty>`	`// fieldName has to be existent, property doesn't need to`
	`// if known, that property does exist, the ? can be left out`
	`if (!fieldName?.property) {}`
	`// CAUTION: every query like this in Groovy evaluates to a Boolean type`
	`// everything that is empty or false will turn into false:`
	`// null, [], [:], "", false -> false`
	
	`if (UtilValidate.isEmpty(fieldName)) {}`

Minilang	Groovy

<if>
  <condition>
  <or>
  <if-empty field="field1"/>
  <if-empty field="field2"/>
  </or>
  </condition>
  <then>
  <!-- code in if -->
  </then>
  <else>
  <!-- code in else -->
  </else>
</if>
	if (!field1 || !field2) {
	 // code in if
	} else {
	 // code in else
	}

<if-compare-field
field="product.primaryProductCategoryId"
to-field="parameters.productCategoryId"
operator="equals">
  <!-- code -->
</if-compare-field>
	// this will even work, if product is
	not existent or null
	if (UtilValidate.areEqual(product
	?.primaryProductCategoryId, parameters
	.productCategoryId)) {
	  // code
	}

<if-instance-of field="parameters.categories" class="java.util.List"></if-instance-of>	if (parameters.categories instanceof java.util.List) {}

14.5. Setting Fields

Minilang	Groovy

<set field="fieldName" value="value"/>	// if fieldName is not initialized
	String fieldName = "value"
	 // if fieldName is initialized
	fieldName = "value"

Minilang	Groovy

<set field="otherFieldName.property" value="value"/>
<set field="otherFieldName.otherProperty" value="true" type="Boolean"/>
<set field="otherFieldName.otherProperty" from-field="parameters.property/>
	// if otherFieldName is not yet initialized, you have to do it first
	// MiniLang does that automatically
	Map otherFieldName = [:] // empty Map
	// now put the values in
	otherFieldName = [
	property: "value",
	otherProperty: true
	]
	// or the less efficient way
	otherFieldName.property = "value"
	otherFieldName.otherProperty = true
	
	// it is possible to put different values in later:
	otherFieldName.property = parameters.property

<set field="thisFieldName" value="${groovy: []}" type="List"/>	// this is easier in Groovy List thisFieldName = []

<property-to-field resource="CommonUiLabels" property="CommonGenericPermissionError" field="failMessage"/>
<!-- there are different cases of this, which are not distinguished in MiniLang -->
<property-to-field resource="general.properties" property="currency.uom.id.default" field="parameters.rateCurrencyUomId"/>
	String failMessage = UtilProperties.getMessage("CommonUiLabels", "CommonGenericPermissionError", parameters.locale)
	// in Groovy there can be a difference for the second case
	parameters.rateCurrencyUomId = UtilProperties.getPropertyValue('general.properties', 'currency.uom.id.default')

<clear-field field="product.primaryProductCategoryId" />	product.primaryProductCategoryId = null

14.6. Starting Services

Minilang	Groovy
<set field="relatedCategoryContext.parentProductCategoryId" from-field="defaultTopCategoryId"/> <call-service service-name="getRelatedCategories" in-map-name="relatedCategoryContext"> <result-to-field result-name="categories" field="resCategories"/> </call-service>	def relatedCategoryContext = [parentProductCategoryId: defaultTopCategoryId] def serviceResult = run service: "getRelatedCategoryies", with: relatedCategoryContext def resCategories = serviceResult.categories // if it is not too confusing to read you can leave out the extra variable run service: "getRelatedCategoryies", with: [parentProductCategoryId: defaultTopCategoryId]
<set-service-fields service-name="productCategoryGenericPermission" map="parameters" to-map="productCategoryGenericPermissionMap"/> <call-service service-name="productCategoryGenericPermission" in-map-name="productCategoryGenericPermissionMap"> <results-to-map map-name="genericResult"/> </call-service>	// instead of setting the service fields from parameters, it is possible to run the service with the parameters map Map genericResult = run service: "productCategoryGenericPermission", with: parameters

14.7. Preparing Service Results

Minilang	Groovy
<field-to-result field="fieldBudgetId" result-name="budgetId"/>	// MiniLang knows this implicitly def result = success() result.budgetId = fieldBudgetId return result

14.8. Database Communication

Minilang	Groovy

<make-value entity-name="FinAccountTrans" value-field="newEntity"/>
<set-nonpk-fields map="parameters" value-field="newEntity"/>
<set-pk-fields map="parameters" value-field="newEntity"/>
	// this is the easy way
	GenericValue newEntity = makeValue ("FinAccountTrans", parameters)
	// this is also possible
	GenericValue newEntity = makeValue ("FinAccountTrans")
	newEntity.setPKFields(parameters)
	newEntity.setNonPKFields(parameters)

<entity-and entity-name="BudgetStatus" list="budgetStatuses"> <field-map field-name="budgetId" from-field="parameters.budgetId"/> <order-by field-name="-statusDate"/> </entity-and>	// this can also be done in one line, but it can easily become unreadable def budgetStatuses = from("BudgetStatus") .where("budgetId", paramters .budgetId) .orderBy("-statusDate") .queryList()

<entity-one entity-name="StatusValidChange" value-field="statusValidChange">
<field-map field-name="statusId" from-field="budgetStatus.statusId"/>
<field-map field-name="statusIdTo" from-field="parameters.statusId"/>
</entity-one>
<!-- entity-one can be called without child elements, too -->
<entity-one entity-name="Product" value-field="product" auto-field-map="true"/>
	// MiniLang has false set for useCache as the default value
	statusValidChange = findOne ("StatusValidChange", [statusId: budgetStatus.statusId, statusIdTo: parameters.statusId], false)
	// this is also possible
	statusValidChange = from ("StatusValidChange")
	.where("statusId", budgetStatus .statusId, "statusIdTo", parameters .statusId)
	.queryOne()
	// if there are no child elements, this can be used
	GenericValue product = from("Product").where(parameters).queryOne()

Minilang	Groovy

<find-by-primary-key entity-name="ProductCategoryMember" map="lookupPKMap" value-field="lookedUpValue"/>	GenericValue lookedUpValue = findOne ("ProductCategoryMember", lookupPKMap, false)
	// this is also possible
	lookedUpValue = from ("ProductCategoryRole")
	.where(lookupPKMap)
	.queryOne()

<entity-condition entity-name="ProductCategoryContentAndInfo" list="productCategoryContentAndInfoList" filter-by-date="true" use-cache="true">
<condition-list combine="and">
<condition-expr field-name="productCategoryId" from-field="productCategoryList.productCategoryId"/>
<condition-expr field-name="prodCatContentTypeId" value="ALTERNATIVE_URL"/>
</condition-list>
<order-by field-name="-fromDate"/>
</entity-condition>
<!-- entity-condition can also be used with the "or" operator -->
<entity-condition entity-name="ProdCatalogCategory" list="prodCatalogCategoryList" filter-by-date="true">
<condition-list combine="and">
<condition-expr field-name="productCategoryId" from-field="parameters.productCategoryId"/>
<condition-list combine="or">
<condition-expr field-name="prodCatalogCategoryTypeId" value="PCCT_VIEW_ALLW"/>
<condition-expr field-name="prodCatalogCategoryTypeId" value="PCCT_PURCH_ALLW"/>
</condition-list>
</condition-list>
</entity-condition>
	// the Groovy methods use the "and" and "equals" operator as default values
	List productCategoryContentAndInfoList = from("ProductCategoryContentAndInfo")
	.where("productCategoryId", productCategoryList.productCategoryId, "prodCatContentTypeId", "ALTERNATIVE_URL")
	.cache().orderBy("-fromDate")
	.filterByDate()
	.queryList()
	// with the use of the "or" operator you have to build your condition like this
	EntityCondition condition = EntityCondition.makeCondition([
	EntityCondition.makeCondition([
	EntityCondition.makeCondition ("prodCatalogCategoryTypeId", "PCCT_VIEW_ALLW"),
	EntityCondition.makeCondition ("prodCatalogCategoryTypeId", "PCCT_PURCH_ALLW")
	], EntityOperator.OR),
	EntityCondition.makeCondition ("productCategoryId", parameters .productCategoryId)
	])
	List prodCatalogCategoryList = from ("ProdCatalogCategory").where(condition) .filterByDate().queryList()

Minilang	Groovy
<make-value entity-name="FinAccountTrans" value-field="newEntity"/> <set-nonpk-fields map="parameters" value-field="newEntity"/> <!-- In this case multiple fields of the GenericValue are set --> <make-value entity-name="ProductCategoryRollup" value-field="newLimitRollup"/> <set field="newLimitRollup.productCategoryId" from-field="newEntity.productCategoryId"/> <set field="newLimitRollup.parentProductCategoryId" from-field="productCategoryRole.productCategoryId"/> <set field="newLimitRollup.fromDate" from-field="nowTimestamp"/>	def newEntity = makeValue ("FinAccountTrans", parameters) // you can set multiple fields of a GenericValue like this def newLimitRollup = makeValue ("ProductCategoryRollup", [productCategoryId: newEntity.productCategoryId, parentProductCategoryId: productCategoryRole.productCategoryId, fromDate: nowTimestamp])
<set field="statusValidChange.prop" value="value"/>	statusValidChange.prop = "value"
<create-value value-field="newEntity"/>	newEntity.create()
<store-value value-field="newEntity"/> <store-list list="listToStore"/>	newEntity.store() delegator.storeAll(listToStore)
<clone-value value-field="productCategoryMember" new-value-field="newProductCategoryMember"/>	def newProductCategoryMember = productCategoryMember.clone()
<remove-value value-field="lookedUpValue"/>	lookedUpValue.remove()

Minilang	Groovy
`<sequenced-id sequence-name="ProductCategory" field="newEntity.productCategoryId"/>`	`newEntity.productCategoryId = delegator.getNextSeqId("ProductCategory")`
`<check-id field="newEntity.productCategoryId"/>`	`UtilValidate.checkValidDatabaseId(newEntity.productCategoryId)`

`<make-next-seq-id value-field="newEntity" seq-field-name="linkSeqId"/>`	`delegator.setNextSubSeqId(newEntity, "linkSeqId", 5, 1)`
	`// the numbers 5 and 1 are used in the Java implementation of the MiniLang method`
	`// and can also be found as the default values in the MiniLang documentation`

14.9. Permissions

To also check for admin-permissions, this method has to be used: `hasEntityPermission(permission, action, userLogin)`

If the method is used with wildcards, it is important to not forget the underscore, which comes before the parameter action!

Minilang	Groovy

`<check-permission permission="CATALOG" action="_CREATE">`
`<alt-permission permission="CATALOG_ROLE" action="_CREATE"/>`
`<fail-property resource="ProductUiLabels" property="ProductCatalogCreatePermissionError"/>`
`</check-permission>`
`<check-errors/>`
	`if (!(security.hasEntityPermission("CATALOG", "_CREATE", parameters.userLogin)`
	`|| security.hasEntityPermission("CATALOG_ROLE", "_CREATE", parameters.userLogin))) {`
	`return error(UtilProperties.getMessage("ProductUiLabels", "ProductCatalogCreatePermissionError", parameters.locale))`
	`}`

Minilang	Groovy
<set field="hasCreatePermission" value="false" type="Boolean"/> <if-has-permission permission="${primaryPermission}" action="${mainAction}"> <set field="hasCreatePermission" value="true" type="Boolean"/> </if-has-permission>	// this will automatically be set to false if the user doesn't have the permission def hasCreatePermission = security.hasEntityPermission(primaryPermission, "_${mainAction}", parameters.userLogin)

14.10. Timestamp And System Time

The first two simple-method are deprecated; the third method should have been used instead.

Minilang	Groovy
<now-timestamp field="nowTimestamp"/>	Timestamp nowTimestamp = UtilDateTime.nowTimestamp()
<now-date-to-env field="nowDate"/>	Timestamp nowDate = UtilDateTime.nowTimestamp()
<!-- this method also has the parameter "type", which is set to 'java.sql.timestamp' as default --> <now field="fooNow"/>	Timestamp fooNow = UtilDateTime.nowTimestamp()

<if-compare-field field="productCategoryMember.thruDate" to-field="expireTimestamp" operator="less" type="Timestamp">
    <!-- code -->
</if-compare-field>
	Timestamp thruDate = productCategoryMember.thruDate
	if (thruDate && thruDate.before(expireTimestamp)) {
	    // code
	}

14.11. Logging

Since all of the log methods are know to the Groovy Language, it is possible to just nearly use them as they are in MiniLang.
For further explanation, here are some examples:

Minilang	Groovy
<log level="verbose" message="Permission check failed, user does not have permission"/>	logVerbose("Permission check failed, user does not have the correct permission.")
<log level="info" message="Applying feature [${productFeatureId}] of type [${productFeatureTypeId}] to product [${productId}]"/>	logInfo("Applying feature [${productFeatureId}] of type [${productFeatureTypeId}] to product [${productId}]")

14.12. General

Minilang	Groovy

<call-simple-method method-name="checkCategoryRelatedPermission"/>
<check-errors/>
	// simple-methods inside of classes, as long as they are not services, will be called like normal methods
	Map res = checkCategoryRelatedPermission("updateProductCategory", "UPDATE", null, null)
	if (!ServiceUtil.isSuccess(res)) {
	    return res
	}

<iterate list="subCategories" entry="subCategory">
    <!-- code -->
</iterate>
	for (def subCategory : subCategories) {
	    // code
	}
	subCategories.each { subCategory ->
	    // code
	}

Minilang	Groovy

<iterate-map map="parameters.productFeatureIdByType" key="productFeatureTypeId" value="productFeatureId">
  <!-- in here something should happen with value and key -->
</iterate-map>
	for (Map entry : parameters.productFeatureIdByType.entrySet()) {
	  def productFeatureTypeId = entry.getKey()
	  def productFeatureId = entry.getValue()
	  // in here something should happen with value and key
	}

<if>
  <condition>
  <not>
  <or>
  <if-has-permission permission="CATALOG" action="_${checkAction}"/>
  <and>
  <if-has-permission permission="CATALOG_ROLE" action="_${checkAction}"/>
  <not><if-empty field="roleCategories"/></not>
  </and>
  </or>
  </not>
  </condition>
  <then>
  <!-- code -->
  </then>
</if>
	if (!security.hasEntityPermission("CATALOG", "_${checkAction}", parameters.userLogin)
	  && !(security.hasEntityPermission("CATALOG_ROLE", "_${checkAction}", parameters.userLogin)
	  && roleCategories)) {
	  // code
	}

<set field="validDate" from-field="parameters.validDate"/>
<if-not-empty field="validDate">
  <filter-list-by-date list="productCategoryMembers" valid-date="validDate"/>
</if-not-empty>
	def query = from("ProductCategoryMember").where("productCategoryId", parameters.productCategoryId)
	if (parameters.validDate) {
	  query.filterByDate()
	}
	List productCategoryMembers = query.queryList()

Minilang	Groovy

`<order-map-list list="productsList">`
`  <order-by field-name="sequenceNum"/>`
`</order-map-list>`
	`productsList = EntityUtil.orderBy`
	`(productsList, ["sequenceNum"])`

14.13. Where to find MiniLang implementation

If you find yourself in a position, where you don't know how to convert a certain tag from MiniLang to Groovy, you can always check the Java implementation of the MiniLang method.
All of the methods have an existing Java implementation and you can find all of them in this folder: `/ofbiz/trunk/framework/minilang/src/main/java/org/apache/ofbiz/minilang/method`

The interesting part of this implementation is the method `exec()`, which actually runs the MiniLang tag.
The tag `<remove-by-and>` for example is realized using this part of code here:

```
@Override

public boolean exec(MethodContext methodContext) throws MiniLangException {
    @Deprecated
    String entityName = entityNameFse.expandString(methodContext.getEnvMap());
    if (entityName.isEmpty()) {
        throw new MiniLangRuntimeException("Entity name not found.", this);
    }
    try {
        Delegator delegator = getDelegator(methodContext);
        delegator.removeByAnd(entityName, mapFma.get(methodContext.getEnvMap()));
    } catch (GenericEntityException e) {
        String errMsg = "Exception thrown while removing entities: " + e.getMessage();
        Debug.logWarning(e, errMsg, module);
        simpleMethod.addErrorMessage(methodContext, errMsg);
        return false;
    }
    return true;
}
```

In this you can find one important part of code, which is:

```
delegator.removeByAnd(entityName, mapFma.get(methodContext.getEnvMap()));
```

This tells you, that, if you're trying to convert the tag `<remove-by-and>`, you can use `delegator.removeByAnd()` in Groovy.