1 - Qu'est ce que roundcube ?

Webmail développé par Thomas Bruederli (http://roundcube.net/)

- Gestion des email via une librairie imap en PHP
- Gestion des contacts
- Framework de développement PHP/Moteur de templates
- Gestionnaire de plugins
- Produit modulaire (Contrainte d'un accès IMAP)

Langages

- PHP5 (Orienté objet)
- Javascript → librairie jquery / jquery-ui via plugin
- HTML / CSS
- SQL
- memcached disponible pour la gestion des sessions

2 – Comment développer un plugin ?

Voir http://trac.roundcube.net/wiki/Doc_Plugins

Mise en place du plugin

- Créer un répertoire dans le dossier plugins de roundcube
- Créer un fichier php dans ce répertoire avec le même nom
- La classe php principale doit étendre la classe « rcube_plugin »
- Ajouter le plugin à la configuration de roundcube

Méthode init

- La seule méthode obligatoire d'un plugin est la méthode init()
- Elle va permettre de définir l'initialisation du plugin, que ce soit les différents appels, les actions, et l'affichage de l'interface
- Comme Roundcube utilise des frame pour son interface, cette méthode peut être appelée plusieurs fois par page, il faut donc vérifier que l'appel est toujours pertinent
- Un élément important à récupérer dans le plugin est l'instance de rcmail, il s'agit de l'instance générique de roundcube qui va permettre d'effectuer toutes les actions d'interface et toutes les interactions avec le framework : `$this->rc = rcmail::get_instance();`

Gestion du « task »

- Permet de définir à quel niveau intervient de plugin
- Le task est passé dans l'url de roundcube, via le champ get « _task », ex : « mail », « addressbook », « settings »
- Possibilité de définir plusieurs tasks « mail|settings » ou toutes « .* »
- Pour n'exécuter du code que dans certaines tasks, on peut connaître à tout moment dans quelle task on se situe. On utilise pour cela l'instance de rcmail.
- Ex : `if ($this->rc->task == 'settings')`
- Dans ce cas le code dans le if ne sera exécuté que lorsque l'utilisateur clic sur les préférences

Définition d'un « hook »

Voir http://trac.roundcube.net/wiki/Plugin_Hooks

- Des hooks sont définis dans le code de roundcube ou dans les autres plugins
- Ils permettent de personnaliser des actions précises et peuvent du coup remplacer le bout de code par défaut (ou le compléter)
- L'ajout d'un appel à un hook : `$this->add_hook('authenticate', array($this, 'authenticate'));`
- Le premier paramètre correspond au nom du hook, le seconde est un tableau qui défini l'instance de la classe qui contient la méthode à appeler et le nom de la méthode à appeler. Le nom du hook et le nom de la méthode peuvent être différents.
- Pour des plugins plus complexes on peut donc séparer les traitements par classe et définir le hook de cette façon : `$this->add_hook('authenticate', array(new class_authenticate(), 'handle'));`
- La méthode handle de la classe classe_authenticate sera appelée
- Les paramètres sont toujours passés en tableau. Ils seront donc différents selon les méthodes. Ces mêmes paramètres sont ensuite à retourner par la méthode hook après traitement. La plupart du temps, il est possible de mettre la valeur 'abort' à true pour terminer le traitement directement après l'appel de la méthode (le traitement par défaut ne s'exécute donc pas).

Buffer de sortie

- Roundcube utilise un output buffer pour la gestion de l'affichage (via la méthode ob_start() dans le index.php)
- L'écriture dans le buffer de sortie se fait via les méthodes disponible dans l'objet « output » de l'instance rcmail. Ex : `$this->rc->output->set_env("share_action", "agendas");`

Generation de l'interface

- Plusieurs façons existent pour générer une interface sur un plugin : utiliser des templates , des méthodes php permettant de générer l'interface HTML ou générée via du javascript
- Les deux premières méthodes ont le même résultat : une génération dynamique de l'interface côté serveur qui sont écrites via le buffer php, la dernière méthode va permettre la génération de l'interface côté client, avec la possibilité d'utiliser des appels ajax

Utilisation de templates

- Des templates peuvent être utilisés dans le plugin pour créer l'interface HTML
- Le moteur de templates est propre à Roundcube et reste assez simpliste
- Le template doit être ajouté dans le dossier « skins/<nom_skin>/templates » du plugin et est un fichier .html. Les noms de skins par défaut sont larry ou classic
- L'appel au template se fait via l'output : `$this->rc->output->send('melanie2.moncompte');`
- Le nom du template est précédé du nom du plugin (dans l'exemple il va donc récupérer le template moncompte.html du plugin melanie2)
- Les balises spécifiques à roundcube sont de type <roundcube:type />
- Les principaux types de balises sont : label, include, button, form, var, container, object
- label : permet d'ajouter du texte récupéré depuis le fichier d'internationalisation, ex : `<roundcube:label name="melanie2.options" />`
- include : permet d'inclure un autre fichier template html, ex : `<roundcube:include file="/includes/header.html" />`
- form : permet de générer un formulaire html, ex : `<roundcube:form name="form" method="post">`
- var : récupérer la valeur d'une variable (de configuration ou d'environnement), ex : `<roundcube:var name="config:product_name">`
- button : permet d'ajouter un bouton permettant d'effectuer une action traitée en javascript (voir la partie javascript ci-dessous), ex : `<roundcube:button command="calendar-edit" label="calendar.edit" classAct="active" />`
- container : permet d'ajouter dans un div du code html personnalisé, le container est utilisé dans roundcube et peut permettre de modifier l'interface directement depuis le plugin, ex : `<roundcube:container name="toolbar" id="calendartoolbar" />`
- object : permet d'ajouter un objet personnalisé pour un traitement particulier, se fait via l'appel d'un handler, ex : `<roundcube:object name="plugin.calendar_list" id="calendarslist" class="listing" />`
- Il est également possible de mettre des conditions dans le template
- ex :

```
<roundcube:if condition="env:calendar_driver == 'kolab'" />
            <li class="separator_above"><roundcube:button command="folders"
task="settings" type="link" label="managefolders" classAct="active" /></li>
```

```
<roundcube:endif />
```

Générer du code HTML en PHP

- La classe Roundcube statique « html » permet de générer de l'html depuis les méthodes PHP
- La méthode générique « tag » permet de générer une balise standard html, ex : `html::tag('div', array("class" => "folderlist-header-m2", "id" => "folderlist-header-m2")`
- Des méthodes pré-définies existent pour générer plusieurs balises : doctype, div, p, img, a, etc.
- Des classes dérivées de la classe « html » vont également permettre de générer des objets avancés, notamment pour les formulaires : html_select, html_intputfield, html_passwordfield, html_hiddenfield. Ex. d'utilisation :

```
// filter set name input
$input_name = new html_inputfield(array('name' => '_name', 'id' => '_name', 'size' => 30, 'class' => ($this->errors['name'] ? 'error' : '')));
```

- D'autres méthodes génériques existent pour générer dynamiquement du php : « rcube_table_output » pour générer une table html à partir d'un tableau
- Le code HTML ainsi généré peut être retourné par l'output, ajouté dans le traitement d'un template ou bien dans l'appel d'un hook

Ajouter des données dans un container

- Il est possible de modifier l'interface via un container
- L'ajout du code html se fait via la méthode add_content
- Ex :

```
$this->api->add_content(
                                    html::tag('div', array("class" => "folderlist-header-m2", "id" => "folderlist-header-m2"),
                                                html::tag('span', array("title" => $this->gettext('mailboxchangetext')), $infos['cn'][0])),
                        'folderlistheader');
```

- Permet d'ajouter un div dans le container « folderlistheader »

Ajout d'un handler

- Quand un object est défini dans le template, il est possible d'ajouter un handler
- Les handlers peuvent être ajoutés un par un :
- `$this->rc->output->add_handler('melanie2_moncompte_options_list', array($moncompte, 'options_list'));`
- Ou bien par bloc :
-

```
$this->rc->output->add_handlers(array(
                            'melanie2_moncompte_options_list'    => array($moncompte, 'options_list'),
                            'melanie2_moncompte_option_frame'    =>
```

```
array($moncompte, 'option_frame'),
                  ));
```

- Les méthodes associées sont alors appelées au moment du traitement du template
- Il suffit alors de générer le code html associé à l'objet et de le retourner
- Les handlers sont à définir avant l'appel au « output->send »

Définir une action personnalisé

- Des actions personnalisées et propres au plugin sont possibles
- Elles sont appelées via l'url Roundcube dans le champ get « _action »
- Au chargement de la page, si une action est présente dans l'url, la méthode définie dans le plugin sera automatiquement appelée
- Pour ajouter une action : `$this->register_action('plugin.melanie2_shares', array($this, 'shares_init'));`
- Par convention, le nom d'une action de plugin sera de « plugin. » le nom du plugin « _ » le nom de l'action, le tout en minuscule.

Ajout d'une internationalisation

- Roundcube permet l'internationalisation de l'application et de ses plugins
- Pour internationaliser un plugin, il faut créer un répertoire « localization » qui va contenir des fichier .inc dans chaque langue souhaité. Ex : fr_FR.inc, en_US.inc
- Le fichier .inc devra définir un tableau et contenir la traduction de chaque label.
- Ex :

```
$labels = array();

// preferences
$labels['default_view'] = 'Vue par défaut';
```

- Les données d'internationalisation sont ensuite chargées via la méthode « add_text ». Ex : `$this->add_texts('localization/', true);`
- On peut ensuite récupérer les labels dans les différentes options d'interface, en php pour récupérer la valeur d'un label, on utilise gettext : `$this->gettext('mailboxchangetext')`

Ajouter du javascript

- L'ajout de javascript se fait via la méthode include_script dans la méthode init du plugin, ex : `$this->include_script('melanie2.js');`
- L'utilisation de la libriaire jquery se fait pas défaut, on peut donc utiliser ses api

Les événements en javascript

Voir http://trac.roundcube.net/wiki/Plugin_Events

- Possibilité d'appeler des function javascript lors de différents évènements

- Par exemple, pour l'évènement init, appelé au chargement du plugin, on appel le callback passé en paramètre de la function : `rcmail.addEventListener('init', function(evt) {});`
- Ceci va nous permettre, par exemple, de générer dynamiquement l'interface au chargement du plugin

Générer l'interface HTML en javascript

- La génération de l'interface se fait via les api de jquery. Ex pour générer un lien dans un span : `tab = $('<span>').attr('id', 'settings_tab_plugin_test_plugin_moncompte').addClass('tablink melanie2'),`

```
        button = $('<a>').attr('href',
rcmail.env.comm_path+'&_action=plugin.test_plugin_moncompte')
            .attr('title', rcmail.gettext('test_plugin.managemoncompte'))
            .html(rcmail.gettext('test_plugin.moncompte'))
            .appendTo(tab);
```

- La var tab est ensuite ajouté au container choisi via la méthode add_element. Ex ici pour ajouter au container tabs : `rcmail.add_element(tab, 'tabs');`

Appels AJAX

- Des appels AJAX sont possibles depuis le javascript
- L'appel javascript va exécuter une action enregistrée depuis le PHP
- L'appel POST se fait de cette façon : `rcmail.http_post('plugin.move2archive', rcmail.selection_post_data());`
- Les appels AJAX sont assez peu utilisés dans Roundcube, qui se base plutôt sur des frames

Gestion de la configuration

- Il est possible de stocker des données de configuration pour le plugin dans un fichier
- Par défaut le fichier de configuration doit s'appeler « config.inc.php » et se situer à la racine
- Pour ensuite charger la configuration du plugin il faut appeler la méthode : `$this->load_config();`
- On peut ensuite accéder aux données stockées dans la configuration via l'api classique : `$this->rc->config->get('changepassword_ws_url')`

3 - Développer un plugin pas à pas : changement de mot de passe

Mise en place de l'architecture

- Créer un dossier du nom du plugin, ex « demo_plugin »
- Créer un fichier .php du même nom dans ce dossier, ex « demo_plugin.php »
- Développer un premier squelette de la classe de plugin, ex :

```php
/**
 * Classe de démontrations pour le développement
 * d'un plugin pour roundcube
 */
class demo_plugin extends rcube_plugin
{
    /**
     * Contient l'instance de rcmail
     * @var rcmail
     */
    private $rc;
    /**
     * Contient la task associé au plugin
     * @var string
     */
    public $task = '.*';
    /**
     * Méthode héritée de rcube_plugin
     * pour l'initialisation du plugin
     * @see rcube_plugin::init()
     */
    function  init()
    {
        // Récupération de l'instance de rcmail
        $this->rc = rcmail::get_instance();
    }
}
```

- Le task peut être défini suivant les tasks liés au plugin
- La fonction init va récupérer l'instance courante de rcmail (utilisé pour toutes les interactions avec roundcube)

Ajout d'un fichier de localization pour les labels

- Pour la gestion des labels on peut mettre en place un fichier de localization
- Il suffit de créer un répertoire « localization/ » à la racine du plugin, et d'ajouter un fichier php .inc du nom de la langue souhaité, dans notre cas « fr_FR.inc »
- On ajoute ensuite un tableau « $labels » et les différents labels peuvent être ajouté au tableau
- Exemple d'un label pour l'application mon compte

```php
// Tableau contenant les labels pour le plugins
$labels = array();
// Labels pour Mon Compte
$labels['moncompte'] = 'Mon compte';
$labels['managemoncompte'] = 'Gérer mon compte';
```

- Pour la prise en compte de la localization dans le plugin, il suffit d'ajouter la ligne suivant dans la méthode init :

```php
        // Prise en compte de la localization
        $this->add_texts('localization/', true);
```

Ajout du bouton pour accéder aux options mon compte

- On peut ensuite ajouter le bouton d'accès aux options mon compte
- Le bouton va alors appeler une action qui doit être traitée
- On ajoute ensuite le traitement de l'action par l'appel d'une méthode
- A ajouter à la suite dans la méthode « init » :

```
        // Est-ce que l'on se situe dans la tache settings
        if ($this->rc->task == 'settings') {
            // Ajout d'un bouton supplémentaire
            $this->api->add_content(
                    html::span(array('id' =>
'settingstabplugindemo_plugin_moncompte', 'class' => 'tablink demo_plugin'),
                            html::a(array('href' => $this->rc-
>url(array('_action' => 'plugin.demo_plugin_moncompte')), 'title' => $this-
>gettext('managemoncompte')), $this->gettext('moncompte')))
                    , 'tabs');
            // Enregistre l'action lors du clic sur le bouton
            $this->register_action('plugin.demo_plugin_moncompte', array($this,
'demo_plugin_moncompte'));
        }
```

- L'ajout du bouton se fait en PHP via la méthode « add_content ». Pour générer l'url du href on utilise la méthode url de l'instance rcmail
- L'appel à l'action va appeler la méthode « demo_plugin_moncompte » qui doit être ajoutée à la classe

```
    /**
     * Fonction appelé lors de l'utilisation de l'action
'plugin.demo_plugin_moncompte'
     * @param array $args Tableau de paramètres
     */
    function demo_plugin_moncompte($args)
    {
    }
```

- On retrouve ensuite le bouton dans l'interface de paramètres

- Lorsque l'on clic sur « Mon compte », le menu est sélectionné (automatiquement par roundcube grâce à l'id de l'objet) et l'url chargée contient l'action « plugin.demo_plugin_moncompte »

- Au chargement de cette url, la méthode « demo_plugin_moncompte » sera donc appelée suite au « register_action »
- Le « register_action » va permettre d'associer chaque nom d'action à une méthode. Lorsqu'au chargement de l'url, l'action est passée en paramètre, il va rechercher si l'action existe et est associée à une méthode. Si oui, il lance l'exécution de la méthode.

Afficher le template moncompte

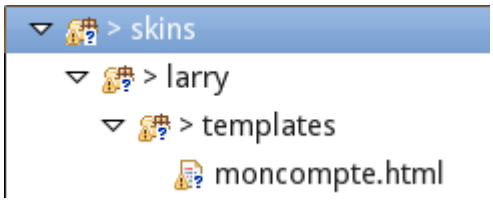
- L'affichage du template va se faire depuis la méthode appelée par l'action « plugin.demo_plugin_moncompte »

```
    /**
     * Fonction appelé lors de l'utilisation de l'action
'plugin.demo_plugin_moncompte'
     * Charge le template moncompte
     * @param array $args Tableau de paramètres
     */
    function demo_plugin_moncompte($attrib)
    {
        $this->rc->output->add_handlers(array(
            'demo_plugin_moncompte_options_list' => array($this,
'moncompte_options_list'),
            'demo_plugin_moncompte_option_frame' => array($this,
'moncompte_options_frame'),
        ));
        $this->rc->output->send('demo_plugin.moncompte');
    }
```

- L'ajout des handler va permettre de modifier directement des données dans le template (notamment en générant du code HTML)
- La méthode « send » va ensuite charger le template passé en paramètre. Le nom du template doit être précédé du nom du plugin

Mise en place du template mon compte

- On peut alors passer à la création du template pour le menu mon compte
- Ce template sera utilisé quand l'utilisateur cliquera sur le menu « Mon compte »
- Pour générer ce template on crée un dossier « skins/ » à la racine du plugin. Dans le dossier « skins/ » on crée un dossier « larry/ » (nom de la skin utilisée par défaut), et dans ce dossier un dossier « templates/ ». Dans ce dernier on crée ensuite un fichier « moncompte.html ».
- On aura donc l'arborescence



- Ce template va générer toute la page de roundcube, il a donc besoin des headers, que l'on ajoute au fichier html

```
<roundcube:object name="doctype" value="html5" />
<html>
<head>
<title><roundcube:object name="pagetitle" /></title>
<roundcube:include file="/includes/links.html" />
```

- On va ensuite lui ajouter un fichier css pour personnaliser l'affichage. Pour cela on ajoute la ligne suivante

```
<link rel="stylesheet" type="text/css" href="/this/moncompte.css" />
```

- Le chemin relatif vers notre css dans le plugin doit utiliser le mot clé « /this/ » qui sera automatiquement remplacé par le bon chemin
- On peut ensuite ajoute le reste des headers, les menu roundcube et la liste des paramètres

```
</head>
<body class="noscroll">
<roundcube:include file="/includes/header.html" />
<div id="mainscreen" class="offset">
<roundcube:include file="/includes/settingstabs.html" />
<div id="settings-right">
```

- On peut alors ajouter la liste contenant les options propres à mon compte. On utilise pour cela un une balise « roundcube:object » qui sera remplacée par du code HTML généré depuis le code PHP (voir plus loin)

```
<div id="demo_plugin_moncompte_options_listbox" class="uibox listbox">
<h2 class="boxtitle"><roundcube:label name="demo_plugin.options" /></h2>
<div class="scroller withfooter">
  <roundcube:object name="demo_plugin_moncompte_options_list"
id="demo_plugin_moncompte_options_list" class="listing" cellspacing="0"
summary="Options list" noheader="true" />
</div>
</div>
```

- L'affichage des formulaires se fera dans une frame. On ajoute donc également cette frame au template

```
<div id="demo_plugin_moncompte_option_box" class="uibox contentbox">
  <div class="iframebox">
    <roundcube:object name="demo_plugin_moncompte_option_frame"
id="demo_plugin_moncompte_option_frame" style="width:100%; height:100%"
frameborder="0" src="/watermark.html" />
  </div>
</div>
```

- On fini ensuite par les footers et des splitters permettant de redimensionner les différents menus

```
<roundcube:include file="/includes/footer.html" />
<script type="text/javascript">
 new rcube_splitter({ id:'demo_pluginsplitter', p1:'#settings-sections',
p2:'#settings-right',
      orientation:'v', relative:true, start:180, min:42, size:12 }).init();
```

```
  new rcube_splitter({ id:'demo_pluginmoncomptesplitter2',
p1:'#demo_plugin_moncompte_options_listbox',
p2:'#demo_plugin_moncompte_option_box',
          orientation:'v', relative:true, start:180, min:120,
size:12 }).init();
</script>
</body>
</html>
```

- Le template nécessite un nouveau label, à ajouter dans le fichier de localization fr_FR.inc

```
// Label pour le template mon compte
$labels['options'] = 'Options';
```

Mise en place d'un fichier de configuration

- On peut créer un fichier de configuration qui va nous permettre de lister les options disponibles pour mon compte
- Pour cela on crée un répertoire « config/ » dans l'arborescence, et dans ce dossier un fichier « config.inc.php »
- On ajoute ensuite la liste des options dans notre fichier de configuration

```
// Configuration de la liste des menu disponible pour mon compte
$config['moncompte_options_list'] = array(
        'acl' => array('id' => 'm2acl', 'label' => 'acl', 'class' => ''),
        'modifmdp' => array('id' => 'm2modifmdp', 'label' => 'modifmdp',
'class' => ''),
    );
```

- On peut alors charger la configuration dans notre plugin, en ajoutant dans la méthode init

```
        // Chargement de la configuration du plugin
        $this->load_config('config/config.inc.php');
```

- La valeur « label » du tableau sera récupérée directement depuis le fichier de localization. On doit donc ajouter les labels nécessaires dans notre fichier

```
// Labels pour l'affichage des options mon compte
$labels['modifmdp'] = 'Modifier votre mot de passe';
$labels['acl'] = 'Changer les droits';
```

Génération de la liste à partir de la configuration

- On peut alors directement générer une liste d'options depuis la configuration
- La méthode « moncompte_options_list » est appelé par le handler

```
    /**
     * Génération de la liste des options pour mon compte
     * @param array $args Tableau de paramètres
     */
    function moncompte_options_list($attrib)
    {
```

```
        $result = array();
        // parcour la liste des options de la configuration
        foreach($this->rc->config->get('moncompte_options_list') as $option) {
            // Ajoute l'option pour mon compte
            if (isset($option['label'])) {
                $option['name'] = $this->gettext($option['label']);
                unset($option['label']);
            }
            $result[] = $option;
        }
        // create XHTML table
        $out = rcube_table_output($attrib, $result, array('name'), 'id');
        // set client env
        $this->rc->output->add_gui_object('demo_plugin_moncompte_options_list',
$attrib['id']);
        $this->rc->output->include_script('list.js');
        return $out;
    }
```

- Le chargement d'une valeur depuis la configuration se fait via la méthode « $this->rc->config->get »
- L'appel au add_gui_object va permettre d'ajouter la liste dans une variable d'environnement. L'intérêt est d'ensuite pouvoir y accéder plus facilement en javascript
- Le « include_script » de « list.js » ajoute des méthodes javascript fournies par roundcube pour le traitement des listes
- La liste HTML sera générées depuis le tableau grâce à la méthode « rcube_table_output »