DOCUMENTAZIONE PLUGIN TEST ROUNDCUBE

Obiettivo

Sviluppo di un plugin di test volto a documentare i passaggi fondamentali per la creazione di un plugin custom.

File System

Tutti i plugin di roundcube devono avere una struttura ben definita. A partire dalla cartella plugins, il file system è così strutturato:

```
plugins/
  nome_plugin/
    composer.json
    skins/
    file.js  **opzionale
    config.inc.php.dist
    nome_plugin.php
    localization/
    media/
```

è fondamentale che il file .php abbia lo stesso nome della cartella che lo contiene.

Il file nome_plugin.php contiene una classe a sua volta chiamata nome_plugin che estende la classe padre rcube_plugin. Al suo interno sarà necessario creare una funzione init() per l'inizializzazione del plugin.

```
Class nome_plugin extends rcube_plugin
{
        function init()
        {

        }
}
```

All'interno della cartella media/ andranno inserite le immagini utilizzate per la creazione degli elementi di UI del plugin (qualora sia necessario)

Nella cartella skins/ saranno inseriti eventuali override delle skin di default di roundcube le skins interamente sviluppate dall'utente.
N.B. è possibile anche creare un semplice file .css per dettare solo le regole di stile necessarie, lasciando il comando alla skin attualmente in uso.

All'interno della cartella localization/ vi saranno i file di localizzazione. I plugin che aggiungono elementi dell'interfaccia utente solitamente includono i propri testi per nominare e descrivere le varie funzionalità. Di seguito un esempio:

In **nome_plugin.php** (all’interno della funzione init())

$this→add_texts(‘localization/’, true)

In **localization/en_US.inc** (unica lingua **obbligatoria**)

```
$labels = array(
   'input1' => 'Label for input 1',
   'title' => 'This is the title',
   'errormsg' => 'This is an error msg',
);
```

Inizializzazione del plugin

Ogni plugin ha bisogno di una funzione init(). Al suo interno devono essere caricati i file di configurazione, stile e logica. Sarà necessario definire gli Hook e le azioni al suo interno.
Di seguito un esempio:

```
function init()
{
        $this->rc = rcmail::get_instance();

        //load configuration and localization
        $this->load_config();
        $this->add_texts('localization/', true);

        $this->add_hook('settings_actions', [$this, 'settings_actions']);
        $this->register_action('plugin.test', array($this, 'display'));
        $this->register_action('plugin.test_submit', array($this, 'test_submit'));

        //includes
        $this->include_script('client.js');
        $this->include_stylesheet('test.css');
}
```

Se il tuo plugin deve attivarsi solo in un compito specifico (come mail, rubrica o impostazioni), imposta la proprietà pubblica $task. Così sarà attivo solo in quel contesto, ottimizzando memoria e prestazioni.

Gestione degli Hooks

I plugin di Roundcube funzionano tramite "hook", che controllano se ci sono funzioni registrate da eseguire in determinati momenti. Queste funzioni possono modificare o estendere il comportamento predefinito di Roundcube. Di seguito un esempio.

```
$this->add_hook('nome-hook', $callback);
```

dove il secondo argomento è un callback PHP che può riferirsi a una funzione semplice o a un metodo di un oggetto. **Lista completa degli hooks disponibili**:
https://github.com/roundcube/roundcubemail/wiki/Plugin-Hooks

Gestione delle azioni

Le azioni servono a mettere in comunicazione il frontend con il backend. Lo script client può inviare richieste AJAX GET o POST al server utilizzando:

rcmail.http_get('plugin.someaction', ...)
rcmail.http_post('plugin.someaction', ...).

Per indirizzare queste richieste alla funzione giusta del tuo plugin, registra un'azione personalizzata (ad esempio, 'plugin.someaction') nel metodo init() della classe del plugin:

```
$this->register_action('plugin.someaction', array($this, 'request_handler'));
```

Le richieste HTTP del tipo ./?_task=mail&_action=plugin.someaction attiveranno la funzione di callback registrata, che deve elaborare la richiesta e inviare una risposta valida al client. Chiamare rcmail::get_instance()->output->send('plugin') alla fine della funzione completerà l'operazione.

Le azioni personalizzate servono non solo per le richieste AJAX, ma possono anche estendere l'applicazione con schermate e passaggi personalizzati.

Configurazione plugin

Tutti i plugin hanno il proprio file di configurazione. Il nome di default è **config.inc.php** ma gli può essere dato anche il nome del plugin. Il caricamento avviene nel seguente modo, all'interno della funzione init():

```
$this->load_config('config.inc.php.dist');
$this→load_config('config.inc.php');
```

uesto caricherà un file di configurazione della distribuzione e poi unirà un file di configurazione locale, sovrascrivendo eventuali impostazioni.

Creazione UI

La creazione dell'interfaccia grafica può essere effettuata in 2 modi distinti:
Il primo, utilizzato nello sviluppo di questo plugin, prevede la creazione di un'azione che richiama una funzione di callback nella quale si trova un handler che stamperà il nostro html.
Utilizzando questo metodo, la struttura html sarà contenuta all'interno di una variabile $html e verrà costruita tramite concatenazione di stringhe (operatore .=). Di seguito un esempio:

```
function init()
{
	…
	$this->register_action('plugin.test', array($this, 'display'));
	...
}
```

```
function display()
{
	$rcmail = rcmail::get_instance();
	$this->register_handler('plugin.body', array($this, 'render_form'));
	$rcmail->output->send('plugin'); // Send the plugin's output
}

function render_form($args)
{
	$html = '';
	$html .= html::tag('h1', array('id' => 'main-title', 'class' => 'main-title'), $this->gettext('title'));
	return $html;
}
```

Il secondo metodo prevede l'utilizzo di file del tipo template.html. Se il modello vuoto del plugin non soddisfa le tue esigenze, puoi creare i tuoi modelli. Puoi inserirli all'interno di una sottodirectory skins/default/.

```
plugins/
  my_plugin/
    my_plugin.php
    localization/
    skins/default/templates/mytemplate.html
```

mytemplate.html può avere uno o più contenitori oggetto che possono essere riempiti dal tuo plugin. Puoi usare plugin.html dalla skin predefinita come esempio per iniziare.

```
<roundcube:object name="plugin.my_content" />
```

all'interno del file .php:

```
$this->register_handler('plugin.my_content', array($this, 'my_function'));

function my_function()
{
     $content = "Foo";
     return($content);
}

$rcmail = rcmail::get_instance();
$rcmail->output->set_pagetitle('my_title');
$rcmail->output->send('my_plugin.mytemplate');
```

Event listeners

Per quanto concerne il javascript, roundcube mette a disposizione una serie di event listeners. Tutto il codice contenuto all'interno dei file .js deve essere scritto all'interno di un event listener 'init' che sarà innescato all'avvio della pagina. Al suo interno potremo scrivere elementi di UI o comandi custom. Di seguito una porzione di codice significativa:

```
window.rcmail && rcmail.addEventListener('init', function(evt) {

	...

	if (window.rcmail) {
		// reload page after ca. 3 minutes
		rcmail.reload(3 * 60 * 1000 - 2000);
		return;
	}
	})
}
```

Di seguito sono riportati tutti gli event listeners disponibili:

init

Il primo evento ad essere attivato quando una pagina viene caricata. Questo è il luogo in cui i plugin possono aggiungere i loro elementi dell'interfaccia utente e registrare comandi personalizzati.
Args:

- task
- action

selectfolder

Attivato quando un utente seleziona una nuova cartella (sia nella posta che nella rubrica). Si nota che viene chiamato prima che la lista dei messaggi venga ricaricata.
Args:

- folder: Nuova cartella selezionata
- old: Cartella precedentemente selezionata

listupdate

Simile all'evento selectfolder, ma viene attivato dopo che la lista dei messaggi (o dei contatti) è stata aggiornata.
Args:

- folder
- rowcount

insertrow

Viene attivato dopo che è stata aggiunta una nuova riga alla lista dei messaggi o alla lista dei contatti, rispettivamente.
Args:

- uid: Row UID
- row:Riferimento al nodo del DOM

group_insert

Viene attivato dopo che un nuovo gruppo di contatti è stato aggiunto alla lista delle cartelle.
Args:

•id: ID gruppo
•name: Nome grouppo
•li: Riferimento al nodo dell’oggetto della lista

group_update

viene attivato dopo che il gruppo di contatti è stato aggiornato nella lista delle cartelle.
Args:

•id: ID gruppo
•name: Nome gruppo
•li: Riferimento al nodo dell’oggetto della lista

group_delete

viene attivato appena prima che un gruppo di contatti venga rimosso dalla lista delle cartelle.
Args:
•id: ID gruppo
•li: Riferimento al nodo dell’oggetto della lista da rimuovere

Linee guida per lo sviluppo

Nome file

Per i file di inclusione PHP che non sono destinati ad essere eseguiti direttamente, si utilizza l'estensione .inc. Questo previene che vengano eseguiti direttamente nel caso in cui le restrizioni di accesso non funzionino come previsto.
Tutti i file devono essere nominati con lettere minuscole e underscore (per separare le parole).
Poiché RoundCube utilizza la tecnologia di autoloading di PHP, le classi devono essere salvate in un file chiamato class_name.php all'interno di program/include/ per essere incluse su richiesta."

Classi, funzioni e variabili

Tutti i nomi di classi, funzioni e variabili dovrebbero contenere solo lettere minuscole e numeri, e usare underscore per separare le parole. La documentazione ufficiale raccomanda di non usare il CamelCase in nessun caso. Le funzioni che forniscono funzionalità specifiche di Roundcube dovrebbero iniziare con rcmail (funzioni specifiche per webmail) o rcube (funzioni globali/framework). I nomi delle classi dovrebbero iniziare con rcube.

Stile di scrittura del codice

Tutto il codice (PHP e JavaScript) deve rispettare le stesse linee guida di stile del codice:

- Le parentesi di apertura e chiusura per le definizioni di funzioni e classi devono trovarsi su una nuova riga.
- Non scrivere mai codice di controllo del flusso su una sola riga.
- Il codice non deve superare il limite di 80 caratteri per riga.
- Non omettere mai le parentesi per i blocchi che contengono solo un'istruzione.
- L'indentazione deve essere di 4 spazi per livello e non devono essere usato tab

- Gli operatori convenzionali devono essere circondati da uno spazio.
- Le virgole devono essere seguite da uno spazio.
- I punti e virgola nelle dichiarazioni for devono essere seguiti da uno spazio.
- Tutti i nomi devono essere scritti in inglese.
- Una funzione dovrebbe avere solo un'istruzione return alla fine.
- Le unità logiche all'interno di un blocco devono essere separate da almeno una riga vuota.
- Le variabili iteratore devono essere chiamate "i", "j", "k", ecc.
- Le variabili globali (PHP) dovrebbero essere scritte in maiuscolo, ma è meglio evitarle.
- Le variabili dovrebbero essere inizializzate dove sono dichiarate e dovrebbero essere dichiarate nello scope più ristretto possibile."

Di seguito un esempio di codice scritto in maniera corretta:

```
function foo_bar($aa, $bb)
{
    $out = '';

    if ($aa == 1 || $aa > 10) {
        $out .= "Case one\n";
        write_log("Foo: $aa");
    }
    else {
        alert('Bar');
    }

    for ($i=0; $i < $bb; $i++) {
        $out .= $i . 'a';
    }
    return $out;
}
```

Tutto il codice scritto dovrebbe essere funzionante dalla versione 5.3.X di PHP.

Link alla documentazione completa di roundcube:
https://github.com/roundcube/roundcubemail/wiki/Dev-Docs