



Come pacchettizzare un modulo custom

Guida per pacchettizzare un modulo o plugin e renderlo installabile da Impostazioni > Gestore Moduli senza dover eseguire ulteriori operazioni manuali.

Step 1

Creare un modulo tramite i normali script SDK oppure da interfaccia (Impostazioni > Creazione moduli)

Eventuali file di cron o template smarty vanno già predisposti nelle cartelle:

- cron/modules/MODULE_NAME
- Smarty/templates/modules/MODULE_NAME

Step 2

Esportare il modulo da Impostazioni > Gestore Moduli

Viene generato un pacchetto zip contenente:

- una cartella modules, contenente i file che saranno copiati in fase di installazione in modules
- una cartella templates, contenente i file che saranno copiati in Smarty/templates/modules/MODULE_NAME
- una cartella cron, contenente i file che saranno copiati in cron/modules/MODULE_NAME
- un file xml con lo schema del modulo (campi, filtri, relazioni, ecc.

In modules/language ho dei file php con le traduzioni relative a quel modulo. Posso aggiungere lì tutte le traduzioni di cui ho bisogno e richiamarle successivamente nelle mie personalizzazioni nei seguenti modi:

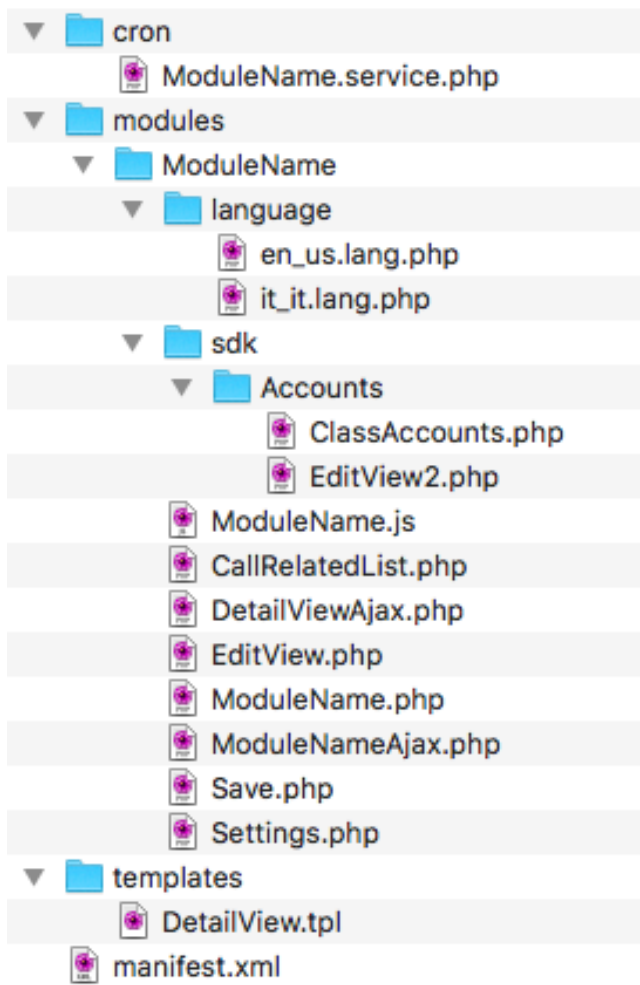
In php: `SDK::getTranslatedString('LABEL','MODULE_NAME')`

In tpl: `{'LABEL'|getTranslatedString:'MODULE_NAME'}`

Step 3

Personalizzare l'installazione del modulo

Solitamente oltre al modulo vanno eseguite delle query, degli script php, SDK oppure serve copiare/spostare file. Queste operazioni vanno inserite nel caso 'module.postinstall' del metodo `vtlib_handler` nel file `modules/MODULE_NAME/MODULE_NAME.php`



Nel caso in cui si debba registrare una `SDK::setFile` di un altro modulo conviene avere il file all'interno dello zip nella cartella `modules`, meglio ancora in una sottocartella `sdk`, e spostarlo durante l'installazione con un comando php di copia in modo da automatizzare anche questa operazione).

Per eseguire delle query all'interno del metodo servono gli oggetti `$adb` e `$table_prefix` che devono essere dichiarati come global all'inizio del metodo. Per compatibilità con VTE migrati da precedenti crm le query vanno scritte usando la variabile `$table_prefix` al posto del prefisso 'vte' che hanno le tabelle.

Step 4

Aggiornamento

Come prima cosa per permettere l'aggiornamento del modulo è necessario incrementare il numero di versione nel manifest.xml (tag 'version'), a questo punto il pacchetto può essere usato per aggiornare un modulo già installato.

L'aggiornamento di un modulo sovrascrive i file, processa il manifest.xml ed esegue i casi 'module.preupdate' e 'module.postupdate' del metodo vtlb_handler.

Se in installazione vengono creati tutti i blocchi, campi, relazioni, ecc. definiti nel manifest, in aggiornamento non vengono duplicate queste proprietà ma viene fatto un confronto tra quelle definite nel manifest e quelle nel sistema e create solamente quelle nuove e aggiornate alcune proprietà.

In dettaglio le funzioni automatizzate sono queste:

CAMPI: aggiunta nuovi campi e update solo delle informazioni helpinfo e masseditable

BLOCCHI: aggiunta nuovi blocchi

PANNELLI (tab in detail e editview): aggiunta nuovi pannelli

RELATED: aggiunta nuove relazioni e update solo della colonna actions

CUSTOMVIEW: aggiunta nuovi filtri

Per l'update vengono usate le seguenti chiavi:

CAMPI: modulo + fieldname

BLOCCHI: modulo + label

PANNELLI: modulo + label

CUSTOMVIEW: modulo + viewname

RELATED: modulo + name

Per altre personalizzazioni in aggiornamento (codice php, sdk, query, ecc.) possiamo sfruttare il caso 'module.postupdate' del metodo vtlb_handler.

Nell'esempio sottostante sfrutto la variabile \$old_version (versione del modulo precedentemente installato) per gestire l'aggiornamento di diverse versioni del modulo.

Questo mi permette di mantenere sempre solo un modulo/plugin e non averne uno per ogni versione.

```
<?php
function vtlb_handler($modulename, $event_type) {
    global $adb,$table_prefix;
    if($event_type == 'module.postinstall') {

        // codice standard per inizializzare ModComments, ModNotification, ecc.

        // queries
        /*
        * creo una nuova tabella vte_test_table e faccio delle insert
        * uso la classe adoSchema per avere compatibilit  su tutti i database
supportati
        */
        $name = "{$table_prefix}_test_table"; // uso $table_prefix al posto di
'vte'

        $schema_table = '<?xml version="1.0"?>
<schema version="0.3">
```

```

        <table name="'. $name.'">
        <opt platform="mysql">ENGINE=InnoDB</opt>
        <field name="id" type="I" size="19">
            <KEY/>
        </field>
        <field name="descr" type="C" size="255"/>
        </table>
    </schema>';
    if(!Vtiger_Utills::CheckTable($name)) {
        $schema_obj = new adoSchema($adb->database);
        $schema_obj->ExecuteSchema($schema_obj-
>ParseSchemaString($schema_table));
    }
    /*
        * il metodo pquery vuole come primo parametro la query con '?' al posto
        dei valori e si occuperà lui di formattarli in base al tipo di colonna e come secondo
        parametro l'array dei valori da sostituire ai '?'
        * la colonna id contiene un campo numerico autoincrementante, usiamo il
        metodo getUniqueID al posto di usare la proprietà auto_increment di MySQL (sempre per
        compatibilità con altri db)
        */
        $result = $adb->pquery("select * from {$table_prefix}_test_table where
descr = ?", array('test1'));
        if ($result && $adb->num_rows($result) == 0) {
            $adb->pquery("insert into {$table_prefix}_test_table(id,descr) val-
ues(?,?)", array($adb->getUniqueID($table_prefix."_crmentity"), 'test1'));
        }
        $result = $adb->pquery("select * from {$table_prefix}_test_table where
descr = ?", array('test2'));
        if ($result && $adb->num_rows($result) == 0) {
            $adb->pquery("insert into {$table_prefix}_test_table(id,descr) val-
ues(?,?)", array($adb->getUniqueID($table_prefix."_crmentity"), 'test2'));
        }

        // sdk
        /*
        * posso aver bisogno di registrare delle eccezioni per il modulo corrente
        oppure per altri (es. setLanguageEntries, setFile, setClass)
        */
        SDK::setLanguageEntries('Accounts', 'LBL_TEST_LABEL', array(
            'it_it'=>'Etichetta di esempio',
            'en_us'=>'Test label',
        ));
        SDK::setClass('Accounts', 'Accounts2', "modules/{$modulename}/sdk/Ac-
counts/ClassAccounts.php");
        /*
        * devo registrare una setFile sul file EditView del modulo Accounts, la
        setFile prevede che il nuovo file sia all'interno del proprio modulo
        * quindi copio il nuovo file dalla cartella di supporto interna a mod-
        ules/MODULE_NAME in modules/Accounts e poi registro la setFile
        */
        copy("modules/
{$modulename}/sdk/Accounts/EditView2.php", 'modules/Accounts/EditView2.php');
        SDK::setFile('Accounts', 'EditView', 'EditView2');

        // cron
        /*

```



```
* se v® gi+ schedulato il file cron/RunCron.sh mi basta aggiungere una
riga alla tabella vte_cronjobs per configurare un nuovo cron
*/

require_once('include/utils/CronUtils.php');
$cj = CronJob::getByName($modulename); // to update if existing (nome
cron)
if (empty($cj)) {
    $CU = CronUtils::getInstance();
    $cj = new CronJob();
    $cj->name = $modulename; // nome cron
    $cj->active = 1;
    $cj->singleRun = false;
    $cj->fileName = 'cron/modules/'.$modulename.'/'.$modulename.'.ser-
vice.php'; // percorso al file
    $cj->timeout = 600; // tempo di timeout in secondi
    $cj->repeat = 60; // ripeti ogni x secondi
    $cj->maxAttempts = 10; // numero massimo di tentativi in caso di
errore
    $CU->insertCronJob($cj);
}

} else if($event_type == 'module.disabled') {
    // TODO Handle actions when this module is disabled.
} else if($event_type == 'module.enabled') {
    // TODO Handle actions when this module is enabled.
} else if($event_type == 'module.preuninstall') {
    // TODO Handle actions when this module is about to be deleted.
} else if($event_type == 'module.preupdate') {

    static $old_version = '';
    $result = $adb->pquery("select version from {$table_prefix}_tab where name
= ?", array($modulename));
    if ($result && $adb->num_rows($result) > 0) $old_version = $adb-
>query_result($result,0,'version');

    } else if($event_type == 'module.postupdate') {

    static $old_version = '';
    $result = $adb->pquery("select version from {$table_prefix}_tab where name
= ?", array($modulename));
    if ($result && $adb->num_rows($result) > 0) $new_version = $adb-
>query_result($result,0,'version');

    if ($old_version == '10') {

        // codice per aggiornare dalla versione 10 ...

    } elseif ($old_version == '11') {

        // codice per aggiornare dalla versione 11 ...

    }

}
}
?>
```

Step 5

Altri accorgimenti

Se la quantità di codice da inserire nel postinstall/postupdate è elevata, conviene mettere quel codice in un metodo separato, o in un file separato, in modo da avere il file principale del moduli il più leggibile possibile.

Tutte le modifiche che richiedono la modifica di file core, che non possono essere ottenute via sdk, non si possono fare. E' possibile fare richiesta a VTECRM per l'implementazione di hook aggiuntivi o estensioni dell'sdk per gestire i casi desiderati.