



vtenext

VTE SDK 2 - Manuale

Ultimo aggiornamento: 11/03/2025 - 15:33

Introduzione

Questo manuale descrive i metodi SDK che permettono la personalizzazione di VTECRM.

Dalla versione 4 alla 5 di VTECRM è cambiato molto il core dell'applicazione (es. la gestione dei popup per le related list) quindi vanno presi alcuni accorgimenti perché i vostri sviluppi realizzati per la versione 4 siano compatibili con la 5.

Di seguito con il tag `#COMP-5` saranno segnalate le indicazioni, dove necessarie, per rendere compatibili i vostri sviluppi SDK anche nell'ultima versione.

Indice

Indice.....	3
Indice delle funzioni.....	4
Inserimento di file php/js/css personalizzati.....	5
Override ed estensioni Javascript.....	5
Sostituzione php standard.....	6
Inclusione di altri files.....	7
Uitypes personalizzati.....	7
Template personalizzati (Smarty).....	10
Gestione dei popup.....	11
Esempi.....	12
Presave.....	12
Advanced query.....	13
Advanced Permissions.....	14
Estensioni di classi.....	14
Header delle pagine.....	15
Traduzioni.....	16
Modifica della visibilità dei campi.....	16
Gestione blocchi Home.....	18
Bottoni personalizzati.....	18
Gestore stati.....	19
Funzioni Custom PDFMaker.....	20
Reports personalizzati.....	20
Contatore Turbolift.....	24
Processi.....	25
Tracciamento files caricati.....	27
Esportazione di moduli.....	27
Esempi.....	28
Uitypes social links.....	28
Picklist collegate.....	29
Business unit.....	30
Viste.....	30
File Customisation.....	32
Esempi.....	32

Indice delle funzioni

SDK::setUtil(\$src).....	5
SDK::unsetUtil(\$src).....	5
Vtiger_Link::addLink(\$id, \$tipo, 'SDKScript', \$file);.....	5
SDK::setFile(\$module, \$file, \$newfile).....	6
SDK::unsetFile(\$module, \$file).....	6
SDK::setExtraSrc(\$module, \$src).....	7
SDK::unsetExtraSrc(\$module, \$src).....	7
SDK::setUitype(\$uitype, \$src_php, \$src_tpl, \$src_js, \$type="", \$params="").....	7
SDK::unsetUitype(\$uitype).....	7
SDK::setSmartyTemplate(\$params, \$src).....	10
SDK::unsetSmartyTemplate(\$params, \$src = NULL).....	10
SDK::setPopupQuery(\$type, \$module, \$param, \$src, \$hidden_rel_fields = "").....	11
SDK::unsetPopupQuery(\$type, \$module, \$param, \$src).....	11
SDK::setPopupReturnFunction(\$module, \$fieldname, \$src).....	11
SDK::unsetPopupReturnFunction(\$module, \$fieldname = NULL, \$src = NULL).....	11
SDK::setPreSave(\$module, \$src).....	12
SDK::unsetPreSave(\$module, \$src = NULL).....	12
SDK::setAdvancedQuery(\$module, \$func, \$src).....	13
SDK::unsetAdvancedQuery(\$module).....	13
function f(\$module).....	13
SDK::setAdvancedPermissionFunction(\$module, \$func, \$src).....	14
SDK::unsetAdvancedPermissionFunction(\$module).....	14
function f(\$module, \$actionname, \$record_id = "").....	14
SDK::setClass(\$extends, \$module, \$src).....	14
SDK::unsetClass(\$extends).....	14
SDK::setLanguageEntry(\$module, \$langid, \$label, \$newlabel).....	16
SDK::setLanguageEntries(\$module, \$label, \$strings).....	16
SDK::deleteLanguageEntry(\$module, \$langid, \$label = NULL).....	16
SDK::addView(\$module, \$src, \$mode, \$success).....	16
SDK::deleteView(\$module, \$src).....	17
SDK::setHomelframe(\$size, \$url, \$title, \$userid = null, \$useframe = true).....	18
SDK::unsetHomelframe(\$stuffid).....	18
SDK::unsetHomelframeByUrl(\$url).....	18
SDK::setMenuButton(\$type, \$title, \$onclick, \$image="", \$module="", \$action="", \$condition = "").....	18
SDK::unsetMenuButton(\$type, \$id).....	19
SDK::setTransition(\$module, \$fieldname, \$file, \$function).....	19
SDK::unsetTransition(\$module, \$fieldname).....	19
function (\$module, \$fieldname, \$record, \$status, \$values).....	19
SDK::setPDFCustomFunction(\$label, \$name, \$params).....	20
SDK::unsetPDFCustomFunction(\$name).....	20
SDK::setReportFolder(\$name, \$description).....	20
SDK::unsetReportFolder(\$name, \$delreports = true).....	20
SDK::setReport(\$name, \$description, \$foldername, \$reportrun, \$class, \$jsfunction = "")....	20
SDK::unsetReport(\$name).....	20
SDK::setTurboliftCount(\$relation_id, \$method).....	24
SDK::unsetTurboliftCount(\$relation_id).....	24
SDK::getAllCustomizations(\$readLinks = false).....	27
linkedListAddLink(\$src, \$dest, \$srcval, \$destval).....	29
linkedListDeleteLink(\$src, \$dest = NULL, \$srcid = NULL).....	29

Inserimento di file php/js/css personalizzati

Per inserire del codice php personalizzato (che verrà incluso all'inizio di ogni pagina) basta registrare il nuovo file tramite il metodo:

```
SDK::setUtil($src)
```

\$src : percorso del file php da includere

Per rimuoverlo usare la funzione:

```
SDK::unsetUtil($src)
```

\$src : percorso del file php da rimuovere (il file non verrà cancellato). Deve essere lo stesso percorso specificato in setUtil.

Se invece si vogliono includere file js o css in ogni pagina, è disponibile la funzione:

```
Vtiger_Link::addLink($id, $tipo, 'SDKScript', $file);
```

\$id : id del modulo che registra il file (in questo caso quello di SDK, 31)

\$tipo : può essere "HEADERCSS" o "HEADERSCRIPT"

\$file : il percorso del file da includere

In generale quando si vogliono includere i file php personalizzati si può chiamare semplicemente SDK::getUtils().

Hooks:

```
include/Webservices/Utils.php  
include/squirrelmail/src/redirect.php  
install/PopulateSeedData.php  
index.php
```

Override ed estensioni Javascript

È possibile sostituire o estendere alcune funzioni javascript di utilizzo comune per modificarne il comportamento. Per far ciò è sufficiente creare una funzione che ha lo stesso nome della funzione da modificare con l'aggiunta di "_override" o "_extension" e gli stessi parametri ed includerla in un file Javascript aggiuntivo, caricato nel modo spiegato nel precedente paragrafo. Il comportamento delle due estensioni è il seguente:

FUNZIONE_override()	Se presente, viene chiamata questa funzione invece di quella originale. Il valore di ritorno di questa funzione è quello restituito.
FUNZIONE_extension()	Se presente, viene chiamata questa funzione e se restituisce false o un valore equivalente a false, la funzione originale termina restituendo false, mentre se restituisce true o un valore equivalente, l'esecuzione prosegue nella funzione originale.

La differenza è quindi che nel primo caso, la funzione originale viene interamente ignorata, mentre nel secondo, si può decidere se continuare l'esecuzione standard o

no. Questo è molto comodo nel caso delle funzioni di validazione, solitamente molto lunghe, in cui si voglia semplicemente aggiungere un controllo, senza ricopiare l'intera funzione per piccoli cambiamenti.

Le funzioni che supportano queste estensioni sono indicate di seguito:

File	Funzioni	
include/js/general.js	doformValidation startCall	getFormValidate
include/js/Inventory.js	setttotalnoofrows deleteRow calcTotal calcProductTotal calcGrandTotal validateInventory FindDuplicate validateNewTaxType validateTaxes setDiscount callTaxCalc	calcCurrentTax calcGroupTax calcSHTax validateProductDiscounts updatePrices updatePriceValues resetSHandAdjValues moveUpDown InventorySelectAll fnAddProductOrServiceRowNew

Sostituzione php standard

Si possono sostituire i file php standard dei moduli, come DetailView.php, EditView.php... , tramite il metodo:

```
SDK::setFile($module, $file, $newfile)
```

\$module : il nome del modulo

\$file : il valore del parametro "action" da confrontare

\$newfile: il nuovo sorgente php, senza estensione e senza percorso

Il nuovo file deve essere nella stessa cartella del modulo e deve essere specificato senza estensione e senza percorso.

Nota: se si vuole sostituire la ListView, si deve chiamare setFile due volte, una con \$file="ListView" e una con \$file="index".

Per de-registrare il file usare:

```
SDK::unsetFile($module, $file)
```

\$module : il nome del modulo

\$file : il nome del parametro "action" che era stato ridefinito

Hooks:

```
include/Ajax/CommonAjax.php
```

```
index.php
```

NOTA: In alternativa alla sostituzione dei file è possibile intervenire alla fine di alcuni file php. Vedi la sezione File Customisations

Inclusione di altri files

Per associare ad un modulo dei files o cartelle qualunque, in modo che vengano esportati o importati in modo automatico, sono disponibili i seguenti metodi:

```
SDK::setExtraSrc($module, $src)
```

\$module : il nome del modulo

\$src : il percorso del file o della cartella da associare

Per eliminare l'associazione (ma non i file stessi) usare:

```
SDK::unsetExtraSrc($module, $src)
```

\$module : il nome del modulo

\$src : il percorso del file

Uitypes personalizzati

Si possono aggiungere dei nuovi tipi a quelli già esistenti e gestirli completamente senza modificare altro codice. Per crearne la procedura è:

1. Creare un nuovo campo personalizzato (tramite vtlib o interfaccia web di VTE)
2. Cambiare manualmente il tipo del nuovo campo nel database (tabella *vtiger_field*, colonna *uitype*) con un valore non utilizzato (nnn) (nel file *modules/SDK/examples/fieldCreate.php* e *script.php* è mostrato come creare i campi con il uitype desiderato senza modificare a mano il database)
3. Creare i file:
 - a. *nnn.php* in *modules/SDK/examples*
 - b. *nnn.js* in *modules/SDK/examples*
 - c. *nnn.tpl* in *Smarty/templates/modules/SDK/examples*Questi file gestiscono il comportamento del nuovo campo a seconda del contesto (list, detail...)
4. Registrare il nuovo tipo con una chiamata a *SDK::setUitype*.

```
SDK::setUitype($uitype, $src_php, $src_tpl, $src_js, $type='', $params='')
```

\$uitype : il numero del nuovo tipo; deve essere nnn (nome dei files)

\$src_php: percorso di *nnn.php*

\$src_tpl: percorso di *nnn.tpl* (senza *Smarty/templates/* all'inizio)

\$src_js : percorso di *nnn.js*

\$type : tipo in formato webservice ('text', 'boolean', ...)

\$params : non usato

Per de-registrare un uitype è disponibile il metodo:

```
SDK::unsetUitype($uitype)
```

\$uitype : è il numero del uitype da rimuovere (non verranno eliminati i files a esso associati)

Si consiglia di usare uitype con valore maggiore di 2000, per evitare conflitti con futuri rilasci del CRM.

All'ingresso degli script php sono disponibili varie variabili, la prima è:

\$sdk_mode : le viste e funzioni che posso personalizzare per il nuovo uitype

("insert", "detail", "edit", "relatedlist", "list", "pdfmaker", "report", ecc.)

In base al tipo di \$sdk_mode ho a disposizione diverse variabili che posso leggere e modificare.

detail

per gestire la visualizzazione del campo in DetailView

```
INPUT
$module      : modulo corrente
$fieldlabel  : etichetta del campo
$fieldname   : nome del campo
$col_fields : (array) valori dei campi
OUTPUT
$label_fld[] : etichetta tradotta
$label_fld[] : valore da visualizzare
```

edit

per gestire la visualizzazione del campo in EditView

```
INPUT
$module_name : modulo corrente
$fieldlabel   : etichetta del campo
$value       : valore del campo
OUTPUT
$editview_label[] : etichetta tradotta
$fieldvalue[] : valore da visualizzare
```

relatedlist, list, pdfmaker

per gestire la visualizzazione del campo in ListView, RelatedList e PDFMaker

```
INPUT
$sdk_value : valore del campo
OUTPUT
$value     : valore da visualizzare
```

report

per gestire la visualizzazione del campo nei Report

```
INPUT
$sdk_value : valore del campo
OUTPUT
$fieldvalue : valore da visualizzare
```

Se il valore salvato nel database è diverso da quello mostrato da interfaccia (es. numero 1.000,25 che deve essere salvato come 1000.25) allora vanno gestite anche le seguenti modalità per salvare il valore nel formato corretto e cercarlo.

insert

per convertire il valore nel formato da salvare nel database

```
INPUT
$this->column_fields : (array) valori dei campi
$fieldname : nome del campo
OUTPUT
$fldvalue : valore da salvare nel database
```

formatvalue

per convertire il valore che arriva dalla \$_REQUEST nel formato salvato nel database (usato nella nuova gestione dei Campi Condizionali)

```
INPUT
$value : valore del campo
OUTPUT
$value : valore convertito nel formato database
```

querygeneratorsearch

per convertire il valore cercato dall'utente in lista e filtri

```
INPUT E OUTPUT
$fieldname : nome del campo
$operator  : operatore di confronto
$value     : valore cercato
```

customviewsearch

per gestire la conversione del valore nei filtri dei popup per i campi reference

INPUT E OUTPUT

\$tablename : tabella del campo
\$fieldname : nome del campo
\$comparator : operatore di confronto
\$value : valore cercato

popupbasicsearch

per gestire la conversione del valore nella ricerca dei popup per i campi reference

INPUT

\$table_name : tabella del campo
\$column_name : colonna del campo
\$search_string : valore cercato

OUTPUT

\$where : condizione della query
es. \$where = "\$table_name.\$column_name =
"".convertToDBFunction(\$search_string)."";

popupadvancedsearch

per gestire la conversione del valore nella ricerca avanzata dei popup per i campi reference

INPUT E OUTPUT

\$tab_col : tabella e colonna del campo
\$srch_cond : operatore di confronto
\$srch_val : valore cercato

reportsearch

per convertire il valore cercato dall'utente nei report

INPUT E OUTPUT

\$table : tabella del campo
\$column : colonna del campo
\$fieldname : nome del campo
\$comparator : operatore di confronto
\$value : valore cercato

Hooks

```
data/CRMEntity.php
include/ListView/ListViewController.php
include/utils/crmv_utils.php
include/utils/EditViewUtils.php
include/utils/DetailViewUtils.php
include/utils/ListViewUtils.php
include/utils/SearchUtils.php
include/QueryGenerator/QueryGenerator.php
modules/PDFMaker/InventoryPDF.php
modules/Reports/ReportRun.php
modules/Users/Users.php
modules/CustomView/Save.php
modules/CustomView/CustomView.php
Smarty/templates/DisplayFieldsReadonly.tpl
Smarty/templates/DisplayFieldsHidden.tpl
Smarty/templates/DetailViewFields.tpl
Smarty/templates/EditViewUI.tpl
Smarty/templates/DetailViewUI.tpl
```

Template personalizzati (Smarty)

È possibile creare dei template personalizzati, che si sostituiscono a quelli standard (come EditView.tpl ...).

Il nuovo template viene utilizzato se i valori di `$_REQUEST` della pagina soddisfano i requisiti. La registrazione di un nuovo template viene fatta tramite il metodo

```
SDK::setSmartyTemplate($params, $src)
```

`$params` : array associativo con i requisiti (vedere sotto)

`$src` : percorso del nuovo template

Per `$params` è possibile specificare un valore speciale (“\$NOTNULL\$”) per indicare che tale parametro deve esistere, con qualsiasi valore. I parametri non specificati vengono ignorati.

Se la regola da inserire esiste già o non è compatibile con quelle esistenti (cioè potrebbe causare ambiguità per alcune `$_REQUEST`), l’inserimento fallisce (e viene salvato nel log un messaggio esplicativo).

Per de-registrare un template personalizzato chiamare il metodo:

```
SDK::unsetSmartyTemplate($params, $src = NULL)
```

`$params` : array con i requisiti

`$src` : percorso del template (se NULL, include tutti i files)

Per sostituire completamente tutti i tipi di vista di un modulo servono almeno 7 regole:

\$params	Note
<code>array('module'=>'Leads', 'action'=>'ListView')</code>	ListView
<code>array('module'=>'Leads', 'action'=>'index')</code>	
<code>array('module'=>'Leads', 'action'=>'DetailView' , 'record'=>'\$NOTNULL\$')</code>	DetailView
<code>array('module'=>'Leads', 'action'=>'EditView' , 'record'=>'\$NOTNULL\$')</code>	EditView (deve esserci record nella request)
<code>array('module'=>'Leads', 'action'=>'EditView')</code>	Creazione nuovo elemento
<code>array('module'=>'Leads', 'action'=>'EditView', 'record'=>'\$NOTNULL\$', "isDuplicate"=>"true")</code>	Duplicazione
<code>array('module'=>'Leads', 'action'=>'LeadsAjax', 'record'=>'\$NOTNULL\$', 'ajaxaction'=>'LOADRELATEDLIST', 'header'=>'Products')</code>	Related List dei Leads che mostra I Prodotti

NOTA: Se più regole corrispondono, verrà utilizzata la più specifica; ad esempio, se ci sono 2 regole, una con “\$NOTNULL\$” e una con il valore “Leads” e la request è “Leads”, verrà usata la seconda regola.

Hooks

Smarty_setup.php

Gestione dei popup

Per la gestione delle finestre popup sono disponibili due azioni. Si può inserire uno script php prima che venga fatta la query per caricare i dati, in modo da poterne selezionare diversi da quelli standard. Inoltre è possibile inserire un altro script php prima che i dati vengano mostrati, in modo da poter modificare tali dati o il risultato alla sua chiusura.

Nel primo caso sono disponibili i 2 metodi:

```
SDK::setPopupQuery($type, $module, $param, $src, $hidden_rel_fields = '')

$type : "field" o "related" per indicare un campo standard o il popup aperto da
una related list
$module: il modulo in cui si apre il popup
$param : il nome del campo che apre il popup (deve essere uitype 10) nel caso
type = "field", altrimenti il nome del modulo collegato.
$src : il percorso del file php
$hidden_rel_fields : array del tipo array($urlvalue => $jscode)
    $urlvalue : parametro da aggiungere all'url quando si apre un popup
    $jscode : codice javascript eseguito quando si apre il popup, il cui
    valore viene assegnato a $urlvalue
```

E per de-registrarlo:

```
SDK::unsetPopupQuery($type, $module, $param, $src)
```

Stessi parametri di prima

All'interno dello script php sono disponibili le seguenti variabili:

```
$query : la query che prende i valori da mostrare
$sdk_show_all_button : se true mostra il pulsante per annullare le restrizioni
SDK e mostrare tutti i record
```

#COMP-5

Solamente nel caso *\$type = 'related'* è cambiato il punto di hook. Questo comporta che se nel vostro script php usavate dei parametri della `$_REQUEST` questi potrebbero non essere più disponibili o avere un nome diverso.

es. non è più disponibile `$_REQUEST['recordid']` ma lo stesso valore è contenuto in `$_REQUEST['from_crmid']`

Quindi la soluzione più veloce è quella di stampare il contenuto della `$_REQUEST` dall'interno dello script php per vedere i nuovi parametri disponibili.

Nel secondo caso invece ci sono i seguenti metodi:

```
SDK::setPopupReturnFunction($module, $fieldname, $src)

$module : il modulo che contiene il campo che apre il popup
$fieldname : il nome del campo che apre il popup (solo uitype 10)
$src : il file php
```

Per de-registrare la funzione di popup usare il metodo:

```
SDK::unsetPopupReturnFunction($module, $fieldname = NULL, $src = NULL)
```

Per ora gli unici campi supportati sono quelli con uitype 10 (a parte per le related list)

Hooks

```
Popup.php
include/Utils/ListViewUtils.php
include/ListView/SimpleListView.php
```

Esempi

```
SDK::setPopupQuery('field','Contacts','account_name','modules/SDK/examples/PopupQuery1.php');

SDK::setPopupQuery('related','Contacts','Products','modules/SDK/examples/contacts/PopupRelQuery.php');

SDK::setPopupReturnFunction('Contacts','vendor_id','modules/SDK/examples/ReturnVendorToContact.php');
```

Presave

Si può inserire uno script personalizzato anche quando si preme il pulsante “Salva” in modalità EditView. Per registrare uno script di questo tipo usare il metodo:

SDK::setPreSave(\$module, \$src)

\$module : il nome del modulo
\$src : il percorso dello script php

Per de-registrarlo usare il metodo:

SDK::unsetPreSave(\$module, \$src = NULL)

\$module: il nome del modulo
\$src : il percorso dello script (se NULL, include tutti gli script registrati per quel modulo)

All'interno dello script sono disponibili le seguenti variabili:

\$type : tipo di salvataggio ("MassEditSave", "DetailView", "EditView", "createTODO", "QcEditView", "ConvertLead", "createQuickTODO", "Kanban")
\$values : (array) nuovi valori
\$values['massedit_recordids'] : (string, solo per MassEdit) gli id da modificare, separati da ";"

Nota: nel caso *createQuickTODO* i campi disponibili in *\$values* sono i seguenti:

Eventi	TODO
Module => 'Calendar', CalendarTitle, CalendarStartTime, CalendarEndTime, IsAllDayEvent, timezone, EventType, Description, Location	Module = 'Calendar', activity_mode => Task, hour, day, month, year, task_time_start, task_subject, task_description, taskstatus, taskpriority, task_assigntype, task_assigned_user_id, task_assigned_group_id, starthr, startmin, startfmt, task_date_start, task_due_date

E possono essere restituite le seguenti:

\$status : (bool) se il salvataggio ha avuto successo o no
\$message: (string) se presente viene mostrato un popup con il messaggio
\$confirm: (bool) se vero viene mostrato un popup Javascript che chiede conferma per proseguire, mostrando \$message. In tal caso non si deve impostare \$status.
\$focus : (string) in caso di errore, l'elemento che prende il focus (solo se \$status = false)
\$changes: (array) valori da assegnare ai campi (solo se \$status = false)

Le variabili \$focus e \$changes sono disponibili solo quando \$status è false e \$type è uno tra 'EditView', 'createTodo', 'QcEditView', 'ConvertLead'.

Hooks

```
include/js/general.js
include/js/KanbanView.js
modules/Calendar/script.js
modules/Calendar/wdCalendar/sample.php
modules/Leads/Leads.js
modules/Users/Forms.php
modules/VteCore/KanbanAjax.php
Smarty/templates/Header.tpl
Smarty/templates/ComposeEmail.tpl
Smarty/templates/Popup.tpl
```

Advanced query

Si può modificare la query eseguita per caricare i dati in modalità ListView, RelatedList e Popup in modo da rendere accessibili o meno alcuni dati. Questo non influenza gli utenti di tipo Administrator, che hanno accesso a tutti i dati; inoltre il modulo deve essere impostato come Privato.

La modifica della query viene fatta tramite una funzione php personalizzata (vedere sotto). In ogni modulo è possibile utilizzare solo una funzione di questo tipo. Per registrare la funzione usare:

SDK::setAdvancedQuery(\$module, \$func, \$src)

\$module : il modulo in cui applicare la funzione (Se per il modulo è già registrata una funzione, non viene inserita la nuova)
\$func : il nome della funzione php
\$src : il file php in cui è contenuta la funzione

Per de-registrare usare il metodo:

SDK::unsetAdvancedQuery(\$module)

\$module : il modulo con cui era stata registrata la funzione

La funzione \$func deve essere definita nel seguente modo:

function f(\$module)

\$module : il modulo che chiama la funzione

E restituisce una stringa:

"" : (stringa vuota) la query non subisce modifiche
? : (stringa non vuota) questa stringa viene aggiunta alla query

Hooks

```
data/CRMEntity.php
```

Advanced Permissions

Unitamente alla modifica della query per il recupero dei dati, si può inserire un controllo dei permessi personalizzato registrando un'ulteriore funzione:

```
SDK::setAdvancedPermissionFunction($module, $func, $src)
```

```
$module : modulo a cui associare la funzione  
$func   : nome della funzione da chiamare (da isPermitted())  
$src    : file php in cui è definita la funzione
```

Per de-registrarla c'è il metodo:

```
SDK::unsetAdvancedPermissionFunction($module)
```

```
$module : il modulo a cui è associata la funzione
```

La funzione deve essere definita in questo modo:

```
function f($module, $actionname, $record_id = '')  
  
$module       : il modulo da cui è chiamata  
$actionname   : il nome dell'azione (ListView, DetailView...)  
$record_id    : l'ID del record che viene analizzato
```

E deve restituire una stringa i cui valori determinano l'esito del controllo:

```
"no" : l'accesso al record NON è consentito  
""   : (stringa vuota) accesso di default per quel campo  
?    : (qualsiasi stringa) l'accesso al record è consentito
```

Hooks

```
include/utils/UserInfoUtil.php
```

Estensioni di classi

Si possono estendere le classi esistenti al fine di ridefinire o aggiungere attributi e metodi. Si deve innanzi tutto creare un file in cui è definita la nuova classe (ad esempio: *class Accounts2 extends Accounts*) e registrarla poi con:

```
SDK::setClass($extends, $module, $src)
```

```
$extends : la classe che viene estesa  
$module  : il nome della nuova classe  
$src     : il file in cui $module è definita
```

Non è possibile estendere più di una volta la stessa classe. Alcune classi non possono essere estese (Users, Conditionals, Transitions, Calendar, Rss, Activity, Reports). Per de-registrare una sottoclasse:

```
SDK::unsetClass($extends)
```

```
$extends : la classe che è stata estesa (non la sottoclasse)
```

Chiamando *unsetClass* verranno rimosse a catena anche tutte le sottoclassi di *\$extends*.

È necessario che tutte le istanziazioni di oggetti avvengano tramite *CRMEntity::getInstance(\$nomemodulo)*, altrimenti non verranno caricate eventuali classi estese.

Hooks

```
include/utils/VtlibUtils.php  
data/CRMEntity.php
```

Header delle pagine

Si può personalizzare l'icona utente, l'icona delle impostazioni o le barre blu in testa alle pagine del VTE per incorporare nuove funzionalità nel VTE. Per ottenere ciò, è sufficiente estendere il metodo `VTEPageHeader::setCustomVars` nel seguente modo:

```
SDK::setClass('VTEPageHeader', 'NewPageHeader', 'modules/SDK/src/NewPageHeader.php');

// Il file NewPageHeader avrà questo contenuto:

require_once('include/utils/PageHeader.php');

class NewPageHeader extends VTEPageHeader {

    protected function setCustomVars(&$smarty, $options = array()) {
        $overrides = array(

            // HTML code to be put right after the menu bar
            'post_menu_bar' => null,

            // HTML code right after the second bar
            'post_primary_bar' => null,

            // HTML code after the third bar
            'post_secondary_bar' => null,

            // HTML code that replace the standard user icon
            'user_icon' => null,

            // HTML code that replace the standard settings icon
            'settings_icon' => null,

        );
        // assign these values to a smarty variable
        $smarty->assign("HEADER_OVERRIDE", $overrides);
    }
}
```

Traduzioni

Si possono personalizzare le stringhe per ogni lingua e modulo installato. Per modificare o inserire una nuova stringa usare il metodo:

```
SDK::setLanguageEntry($module, $langid, $label, $newlabel)
```

```
$module    : il modulo che contiene la stringa  
$langid    : il codice della lingua (es: "en_us", "it_it")  
$label     : l'etichetta della stringa (es: LBL_TASK_TITLE)  
$newlabel  : la nuova stringa
```

Se l'etichetta esiste già per il modulo e la lingua scelti, verrà sostituita. Come modulo si può specificare "APP_STRINGS" per inserire una traduzione globale o "ALERT_ARR" per rendere la traduzione disponibile nei file JavaScript. Per caricare una stringa contemporaneamente in più lingue è disponibile il metodo:

```
SDK::setLanguageEntries($module, $label, $strings)
```

```
$module    : il modulo  
$label     : l'etichetta della stringa  
$strings   : array associativo con le traduzioni ( array( "it_it"=>str1, ... ) )
```

Per cancellare una stringa usare:

```
SDK::deleteLanguageEntry($module, $langid, $label = NULL)
```

```
$module    : il modulo  
$langid    : il codice della lingua  
$label     : l'etichetta (se NULL, tutte le stringhe che corrispondono)
```

Modifica della visibilità dei campi

Si può modificare la visibilità dei vari campi (valore di \$readonly) e altre variabili nelle varie modalità (ListView, EditView...). Per registrare una nuova "Vista" usare il metodo:

```
SDK::addView($module, $src, $mode, $success)
```

```
$module    : il modulo in cui applicare la vista  
$src       : il file php  
$mode      : la modalità di applicazione della regola (vedere sotto)  
$success   : cosa fare dopo l'applicazione della regola (vedere sotto)
```

Le Viste definite per ogni modulo vengono applicate nell'ordine in cui sono registrate. Quando si registrano, vengono aggiunte in coda a quelle già definite per quel modulo. Il parametro *\$mode* ha senso solo se si modifica la variabile *\$readonly*, e ammette i seguenti valori:

```
"constrain" : forza il valore di $readonly ad assumere il nuovo valore dato  
              nello script.  
"restrict"  : cambia il valore di $readonly solo verso un valore più restrittivo  
              (da 1 a 99 o 100, da 99 a 100, non viceversa)
```

Il parametro *\$success* invece può essere:

```
"continue" : dopo l'applicazione della vista, continua con la successiva  
"stop"     : se la vista restituisce $success = true, non vengono eseguite altre  
              regole
```

All'interno degli script sono disponibili le seguenti variabili:

```
$sdk_mode : uno tra "" (create), "edit", "detail", "popup_query",  
"list_related_query", "popup", "related", "list", "mass_edit"  
$readonly : il valore di readonly per il campo che si sta per scrivere (1, 99,  
100)  
$col_fields: valori dei campi, solo per $sdk_mode = "edit", "detail" e ""  
$fieldname o $fieldName: nome del campo corrente
```

`$current_user`: l'utente corrente

Ed è possibile restituire la variabile `$success` con true o false.

A seconda della modalità (ListView, EditView, ...) ci sono diversi modi per modificare le query e diverse variabili disponibili.

Modalità	Valore di <code>\$sdk_mode</code>	Variabili disponibili	Note
CreateView	""	<code>\$col_fields</code>	1
EditView	"edit"	<code>\$current_user</code>	
DetailView	"detail"	<code>\$fieldname</code>	
MassEdit	"mass_edit"	<code>\$readonly</code>	
PopupQuery	"popup_query"	<code>\$success</code>	2
List/RelatedQuery	"list_related_query"	<code>\$sdk_columns</code>	
Popup	"popup"	<code>\$success</code>	
Related	"related"	<code>\$current_user</code>	3
		<code>\$fieldname</code>	4
		<code>\$sdk_columnnames</code>	
		<code>\$sdk_columnvalues</code>	
List	"list"	<code>\$readonly</code>	
		<code>\$success</code>	

Note:

1. In queste modalità i valori dei campi sono in `$col_fields[ nomi-dei-campi ]`
2. Queste modalità servono per modificare la query in modo da poter prelevare campi aggiuntivi. In `$sdk_columns` ci sono le colonne del db da aggiungere alla query. Includere poi il file "modules/SDK/AddColumnsToQueryView.php"
3. Per prelevare valori di altri campi, specificati nelle modalità PopupQuery e ListRelatedQuery, scriverli nell'array `$sdk_columnnames`, includere il file "modules/SDK/GetFieldsFromQueryView.php" e prenderli poi da `$sdk_columnvalues`
4. Nel caso della related "Storico attività" sono disponibili solo le variabili `$recordId` e `$readonly` e si applicano all'intera riga, non al singolo campo.

Per de-registrare la vista usare:

```
SDK::deleteView($module, $src)
```

```
$module : il modulo associato  
$src    : il file php
```

Hooks

```
include/ListView/ListViewController.php  
include/Utils/DetailViewUtils.php  
include/Utils/EditViewUtils.php  
include/Utils/ListViewUtils.php  
include/QueryGenerator/QueryGenerator.php  
Popup.php
```

Gestione blocchi Home

Si possono aggiungere nuovi blocchi alla home del VTE tramite SDK; i nuovi blocchi creati non saranno eliminabili tramite interfaccia. Il metodo per la creazione di un nuovo blocco è:

```
SDK::setHomeIframe($size, $url, $title, $userid = null, $useframe = true)
```

```
$size      : la dimensione orizzontale del blocco (da 1 a 4)
$url       : l'indirizzo da mostrare all'interno del blocco. Può avere anche un
protocollo all'inizio (es: http://www.sito.com/file )
$title     : etichetta per il titolo. Installare la relativa traduzione con
setLanguageEntry
$userid    : array contenente gli id degli utenti che vedono il blocco. Se si
lascia null, il blocco è visibile per tutti gli utenti.
$useframe  : se true il contenuto viene messo dentro ad un <iframe> altrimenti
viene incluso il file direttamente.
```

Gli utenti creati successivamente vedranno tutti i blocchi registrati in precedenza. La cancellazione del blocco è possibile tramite 2 metodi:

```
SDK::unsetHomeIframe($stuffid)
```

```
$stuffid : l'id del blocco
```

```
SDK::unsetHomeIframeByUrl($url)
```

```
$url : l'url specificato durante la registrazione
```

I blocchi vengono rimossi per tutti gli utenti.

Hooks

```
modules/Home/HomestuffAjax.php
modules/Home/HomeWidgetBlockList.php
modules/Home/HomeBlock.php
modules/Home/Homestuff.js
modules/Users/Save.php
Smarty/templates/Home/MainHomeBlock.tpl
```

Bottoni personalizzati

È possibile aggiungere bottoni alla pulsantiera sotto il menu principale. Per inserire un nuovo bottone utilizzare il seguente metodo:

```
SDK::setMenuButton($type, $title, $onclick, $image='', $module='', $action='', $condition = '')
```

```
$type : il tipo del pulsante, può essere 'fixed' o 'contestual'; nel primo caso
il bottone appare a sinistra ed è sempre visibile, nel secondo caso il bottone
viene inserito a destra e sarà visibile solo nel modulo e per l'azione scelta.
$title : il nome del pulsante
$onclick : codice javascript da eseguire. NON è possibile usare i doppi apici
(")
$image : l'immagine per il bottone. Deve essere specificata senza percorso e
risiedere in themes/softed/ anche nella versione più piccola (esempio:
immagine.png e immagine_min.png)
$module : se type = 'contestual', il modulo in cui il bottone è visibile
$action : se type = 'contestual', l'azione (parametro action) in cui il bottone
è visibile
```

\$condition : stringa del tipo Nomefunzione:Percorsophp che rappresenta una funzione (nel file Percorsophp) da chiamare prima di mostrare il bottone. Se restituisce false, il pulsante non viene mostrato. La funzione ha un solo parametro di tipo reference ad un array con le informazioni sul bottone.

Per rimuovere il pulsante usare:

```
SDK::unsetMenuButton($type, $id)
```

\$type : il tipo del pulsante
\$id : l'ID del bottone

Hooks

Smarty/templates/Buttons_List.tpl

Gestore stati

È possibile modificare le opzioni di scelta per le picklist gestite dal gestore stati, nonché aggiungere messaggi al blocco “Gestore stati”, a destra del record. Per registrare tale funzionalità usare il metodo:

```
SDK::setTransition($module, $fieldname, $file, $function)
```

\$module : il nome del modulo da gestire
\$fieldname : il nome del campo gestito dal gestore stati
\$file : percorso del file php che contiene la funzione da chiamare
\$function : nome della funzione da chiamare

E per rimuoverlo:

```
SDK::unsetTransition($module, $fieldname)
```

La funzione richiamata dal gestore stati ha il seguente formato:

```
function ($module, $fieldname, $record, $status, $values)
```

\$module : il modulo corrente
\$fieldname : il nome del campo gestito dal gestore stati
\$record : id del record visualizzato
\$status : valore del campo usato per lo stato del record corrente
\$values : array di valori ammissibili per il suddetto campo

Deve restituire *null* nel caso non si voglia modificare il comportamento del gestore stati oppure un array con il seguente formato:

```
array(  
    'values' => array(..) // array con i valori ammissibili per lo stato  
    'message' => '' // codice html da includere sotto al blocco gestore stati  
);
```

Hooks

modules/Transitions/Transitions.php

modules/Transitions/Statusblock.php

Smarty/templates/modules/Transitions/StatusBlock.tpl

Funzioni Custom PDFMaker

È possibile aggiungere funzioni custom nel PDFMaker. Per inserirne una usare:

```
SDK::setPDFCustomFunction($label, $name, $params)
```

```
$label: etichetta per la funzione (viene tradotta all'interno del modulo  
PDFMaker)  
$name: nome della funzione  
$params: array con i nomi dei parametri della funzione
```

Le funzioni così registrate devono essere salvate all'interno di files php in modules/PDFMaker/functions/

Per rimuovere la funzione basta chiamare:

```
SDK::unsetPDFCustomFunction($name)
```

```
$name: il nome della funzione
```

Reports personalizzati

Si possono inserire cartelle e report personalizzati usando i seguenti metodi

```
SDK::setReportFolder($name, $description)
```

```
$name : nome della cartella  
$description : descrizione
```

Crea una nuova cartella nella pagina dei report. La cartella creata sarà visibile da tutti gli utenti e non può essere modificata. Il nome e la descrizione possono essere tradotti nel solito modo.

Le cartelle così create si possono eliminare con

```
SDK::unsetReportFolder($name, $delreports = true)
```

```
$name : il nome della cartella da cancellare  
$delreports : se true elimina anche tutti i reports (creati tramite SDK) in  
quella cartella (non vengono eliminati files)
```

All'interno delle cartelle create via SDK si possono inserire reports personalizzati con:

```
SDK::setReport($name, $description, $foldername, $reportrun, $class, $jsfunction =  
'' )
```

```
$name : il nome del report (che si può tradurre come sempre)  
$description : descrizione del report (traducibile)  
$foldername : il nome della cartella (SDK) in cui inserire il report  
$reportrun : il percorso del file php che contiene la classe che genera il  
report  
$class : il nome della classe che gestisce il report (dettagli dopo)  
$jsfunction : il nome di una funzione Javascript da avviare quando si preme  
"Genera report" (vedere dettagli dopo)
```

Analogamente si possono cancellare con

```
SDK::unsetReport($name)
```

```
$name : il nome del report da eliminare (non verranno eliminati i files)
```

La classe specificata in `$c/class` deve rispettare questa struttura:

```
class NewReportRun extends ReportRun {
    var $enableExportPdf = true;        // mostra pulsante "Esporta in PDF"
    var $enableExportXls = true;        // mostra pulsante "Esporta in Excel"
    var $enablePrint = true;            // mostra pulsante "Stampa"
    var $hideParamsBlock = true;        // nasconde blocco di opzioni del report

    function __construct($reportid) {
        $this->reportid = $reportid;
        $this->primarymodule = 'Accounts';    // modulo principale (opzionale)
        $this->reporttype = '';              // tipo report (opzionale)
        $this->reportname = Nuovo Report SDK; // nome report (opzionale)
    }
}
/*
 * Genera codice HTML che va inserito sotto ai filtri del reports
 */
function getSDKBlock() {
    $html = "";
    return $html;
}
/*
 * Genera il report (vedere i report standard per sapere cosa ritorna il metodo)
 * Vedere in modules/SDK/examples/Reports/ per un esempio di report SDK.
 */
function GenerateReport($outputformat, $filterlist = null, $directOutput=false) {
    switch ($outputformat) {
        default:
        case 'HTML':
        case 'PRINT':
            $return_data = ... // genera report per html/stampa
            break;
        case 'PDF':
        case 'XLS':
            $return_data = ... // genera report per pdf/xls
            break;
        case 'TOTALXLS':
        case 'TOTALHTML':
        case 'PRINT_TOTAL':
            $report_data = ... // genera totali
    }
    return $report_data;
}
} // end class
```

La funzione Javascript specificata in *\$jsfunction* deve già essere dichiarata e restituisce una stringa da aggiungere alla request

```
function preRunReport(id) {
    var params = "";
    var select = getObj('picklist1');

    if (select) {
        params += "&picklist1="+select.options[select.selectedIndex].value;
    }

    var selectuser = getObj('picklist2');
    if (selectuser) {
        params += "&picklist2="+selectuser.options[selectuser.selectedIndex].value;
    }

    return params;
}
```

Quando si aggiorna un VTE alla versione 16.09 (o successiva), alcune funzionalità dei report SDK vanno verificate in quanto sono cambiate alcune funzioni.

Se si aggiorna ad un VTE con revisione minore di 1576 è necessario riportare la patch identificata da crmv@140813 (files ReportRun.php e SDKSaveAndRun.php).

Una delle parti cambiate è la gestione dei filtri temporali, che in precedenza si poteva modificare estendendo le funzioni *getPrimaryStdFilterHTML* e *getSecondaryStdFilterHTML*, che ora non vengono più usate.

Per ottenere lo stesso risultato bisogna estendere la funzione *getStdFilterFields* che restituisce un array di campi disponibili, ad esempio:

```
function getStdFilterFields() {
    // See the method Reports::getStdFilterFields for the standard implementation

    // this example just loads standard fields for the Potential module
    $list = $this->reports->getStdFiltersFieldsListForChain(0, array('Potentials'));

    return $list;
}
```

Quando il report viene generato, non va più letta la variabile *\$filterlist*, bensì *\$this->stdfilters*, che contiene il filtro da usare.

Se la query del report è completamente personalizzata, va gestita manualmente anche la generazione del filtro temporale, ad esempio usando una funzione come questa:

```

function addStdFilters() {
    global $current_user;
    $sql = '';

    if (is_array($this->stdfilters)) {
        foreach ($this->stdfilters as $flt) {
            if ($flt['fieldid'] > 0) {
                // get field informations
                $finfo = $this->getFieldInfoById($flt['fieldid']);
                $table = $finfo['tablename'];
                $qgen = QueryGenerator::getInstance($finfo['module'], $current_user);
                $operator = 'BETWEEN';
                if ($flt['value'] == 'custom') {
                    $value = array($flt['startdate'], $flt['enddate']);
                } else {
                    $cv = CRMEntity::getInstance('CustomView');
                    $value = $cv->getDateforStdFilterBytype($flt['value']);
                }
                // adjust for timezone
                $value[0] = $this->fixDateTimeValue(
                    $qgen, $finfo['fieldname'], $value[0]
                );
                $value[1] = $this->fixDateTimeValue(
                    $qgen, $finfo['fieldname'], $value[1], false
                );
                // add the condition
                $sql .= ' AND '.$table.'.'.$finfo['columnname'].' '.$operator.' '.$value[0].' AND '.$value[1];
            }
        }
    }

    return $sql;
}

```

Hooks

```

modules/Reports/Listview.php
modules/SaveAndRun.php
modules/Reports/Reports.php
modules/Reports/CreatePDF.php
modules/Reports/CreateXL.php
modules/Reports/PrintReport.php
Smarty/templates/ReportContents.tpl
Smarty/templates/ReportRunContents.tpl
Smarty/templates/ReportRun.tpl

```

Contatore Turbolift

Quando si modificano i criteri di estrazione standard di una relatedlist, per esempio usando un altro metodo o ridefinendolo estendendo la classe che lo contiene, il contatore del numero di record collegati visibile nel turbolift (colonna a destra con la lista dei moduli in detailview) potrebbe non esser più valido. Va quindi definito un metodo che ritorna il conteggio corretto tramite le seguenti api.

Definizione di

```
SDK::setTurboliftCount($relation_id, $method)
```

```
$relation_id : id relazione (vedi vte_relatedlists)
```

```
$method : metodo che ritorna il conteggio
```

```
I metodo va nella classe che contiene la relazione
```

```
(es. nella relazione Contatti collegati ad una Azienda si intende la classe  
Accounts o eventuali sue estensioni)
```

Rimozione:

```
SDK::unsetTurboliftCount($relation_id)
```

Il metodo può essere il metodo della relatedlist (colonna name in vte_relatedlists) oppure un metodo custom che ritorna un intero.

Processi

Log di Processo

* Per le installazioni dalla rev 1500 in Impostazioni → Business Process Manager → ProcessManager sarà disponibile un tasto 'LOG' una volta eseguito il seguente script:

```
require_once('include/utils/VTEProperties.php');  
$VP = VTEProperties::getInstance();  
$VP->set('settings.process_manager.show_logs_button', 1);
```

* Per le precedenti versioni invece, si può accedere ai log:

- Direttamente da browser (<https://yourAddress/logs/ProcessEngine/01.log>)
- Da filesystem. Questi log sono nella cartella logs/ProcessEngine/

N.B. Inizialmente viene creato il file 01.log. Quando questo supera una certa dimensione verrà creato il file 02.log, etc.

Importazione Processi da script

Dalla versione 18.05 (rev. 1696) è possibile importare da script php dei processi precedentemente esportati nei formati **vtebpmn** (diagramma + configurazione) o **bpmn** (solo diagramma) col metodo *importFile* della classe *ProcessMakerUtils*.

Il metodo *importFile* vuole come primo parametro il percorso del file da installare (.vtebpmn / .bpmn) e come secondo un valore boolean (true/false) a seconda se si vuole attivare automaticamente il processo o lasciarlo in stato disattivo.

Questo metodo risulta essere utile per installare dei processi al termine dell'installazione di un nuovo modulo: basta includere il file del processo nello zip di installazione ed eseguire l'importazione nel postinstall del metodo *vtlib_handler* della classe del modulo.

es.

```
require_once('modules/Settings/ProcessMaker/ProcessMakerUtils.php');  
$PMUtils = ProcessMakerUtils::getInstance();  
$PMUtils->importFile('PATH_FILE/Process1.vtebpmn', true);  
$PMUtils->importFile('PATH_FILE/DiagramProcess2.bpmn', false);
```

È possibile aggiungere funzioni custom per popolare campi di moduli, form dinamiche, mail, ecc con il seguente metodo:

```
SDK::setProcessMakerFieldAction($func, $src, $label, $parameters='');
```

```
$func   : nome della funzione  
$src    : percorso file contenente la funzione  
$label  : etichetta della funzione  
$parameters (optional) : parametri statici (es. riferimento campo fisso)
```

```
SDK::setProcessMakerFieldAction('vte_calculate_percentage','modules/SDK/src/ProcessMaker/  
Utils.php','Calculate percentage (percentage,total)');
```

N.B. All'interno delle funzioni registrate sono disponibili alcune variabili globali, le quali devono essere dichiarate: `global $engine, $current_process_actionid`.

Queste contengono rispettivamente l'oggetto contenente tutte le informazioni relative al processo che si sta eseguendo e l'id dell'azione corrente (es. Crea entità, Aggiorna entità, ...)

È possibile inoltre aggiungere funzioni custom per verificare condizioni complesse:

```
SDK::setProcessMakerTaskCondition($func, $src, $label);
```

```
$func   : nome della funzione  
$src    : percorso file contenente la funzione  
$label  : etichetta della funzione
```

```
SDK::setProcessMakerTaskCondition('vte_compare_account_bill_ship_street','modules/SDK/src/  
ProcessMaker/Utils.php','Indirizzi di spedizione e fatturazione uguali [e/n]');
```

Infine è possibile registrare nuove azioni con il seguente metodo:

```
SDK::setProcessMakerAction($func, $src, $label);
```

```
$func   : nome della funzione  
$src    : percorso file contenente la funzione  
$label  : etichetta della funzione
```

```
SDK::setProcessMakerAction('close_tickets','modules/SDK/src/ProcessMaker/Utils.php','Chiudi  
Ticket relazionati');
```

Tracciamento files caricati

Si può avere una lista di tutti i files registrati tramite SDK chiamando il metodo

```
SDK::getAllCustomizations($readLinks = false)
```

\$readLinks : se true restituisce anche i js e css caricati con
Vtiger_Link::addLink.

Viene restituito un array di files, come questo:

```
Array
(
    [0] => Smarty/templates/modules/CreditLines/DetailViewTab.tpl
    [1] => Smarty/templates/modules/Customized/Accounts/CustomView.tpl
    [2] => Smarty/templates/modules/Customized/Accounts/DetailViewTab.tpl
    [3] => Smarty/templates/modules/Customized/Accounts/ListView.tpl
    [4] => Smarty/templates/modules/Customized/Calendar/ActivityDetailView.tpl
    [5] => Smarty/templates/modules/Customized/Products/DetailViewTab.tpl
    [6] => Smarty/templates/modules/Customized/Products/EmailSend.tpl
    [7] => Smarty/templates/modules/Customized/Products/ListView.tpl
    [8] => Smarty/templates/modules/SampleRequests/DetailViewTab.tpl
    [9] => modules/Accounts/CustomView2.php
    [10] => modules/Accounts/DetailView2.php
    [11] => modules/Accounts/ListView2.php
    [12] => modules/Accounts/Save2.php
    [13] => modules/Calendar/DetailView2.php
    [14] => modules/Calendar/EditView2.php
    [15] => modules/Calendar/Save2.php
    [16] => modules/CustomView/Save2.php
);
```

Esportazione di moduli

Quando si esporta un modulo, verranno esportate anche tutte le modifiche fatte tramite SDK. Le uniche cose che non vengono esportate sono i link inseriti tramite Vtiger_Link::addLink e SDK::setUtil dato che i files inseriti con tali metodi non sono specifici di un modulo. Pertanto, se un modulo utilizza uitypes personalizzati che creano tabelle o utilizzano altri files oltre a quelli specificati tramite setUitype bisogna ricreare tale campo in vtlib_handler (type = postinstall).

Esempi

Il file `modules/SDK/examples/VTE-SDK-2.php` contiene esempi delle api descritte. Di seguito qualche altro esempio.

Uitypes social links

A titolo di esempio, ecco alcuni uitypes che rappresentano un link con alcuni tra i maggiori social networks.

8 textbox per alcuni Social Networks: LinkedIn (171), Facebook (170), YouTube(172), Google+ (173), Orkut (176), Twitter (174), VKontakte (175), QZone (177).

Per crearli, seguire il metodo descritto nel paragrafo Uitypes, registrarli con:

```
$basedir = 'modules/SDK/examples/';

$uitypes = array(170, 171, 172, 173, 174, 175, 176, 177);
foreach ($uitypes as $uit) {
    SDK::setUitype(
        $uit,
        $basedir.strval($uit).'.php',
        $basedir.strval($uit).'.tpl',
        $basedir.strval($uit).'.js'
    );
}
```

Vedere i files contenuti nella directory `examples`.

Picklist collegate

È disponibile un uitype (300) per collegare picklist esistenti affinché i loro valori filtrino i valori di altri picklist. Per utilizzarle:

1. Creare le picklist in modo standard, popolandole con i dati
2. Cambiare manualmente nel DB il tipo delle picklist (tabella *vtiger_field*, colonna *uitype*), da 15 a 300
3. Creare le connessioni tra gli elementi delle picklist usando la funzione

```
linkedListAddLink($src, $dest, $srcval, $destval)
```

```
$src      : nome della picklist primaria (es: "cf_738")
$dest     : nome della picklist secondaria (es: "cf_739")
$srcval   : valore nella picklist primaria a cui associare i valori (es: "Italia")
$destval  : array di valori della picklist secondaria associati a $srcval (es:
array("Roma", "Milano", "Firenze") )
```

Se la nuova connessione genera un percorso ciclico tra le picklist collegate, non verrà accettata.

Le connessioni si cancellano chiamando la funzione:

```
linkedListDeleteLink($src, $dest = NULL, $srcid = NULL)
```

```
$src      : nome della picklist primaria
$dest     : nome della picklist secondaria (se NULL, include tutte le picklist
collegate a $src)
$srcid    : valore nella picklist primaria (se NULL, li include tutti)
```

Entrambe le precedenti funzioni sono definite in *modules/SDK/examples/300Utils.php*

La tabella nel DB che contiene le connessioni è *vtiger_linkedlist*.

Ad esempio, dopo aver creato le seguenti picklist:

Picklist	Valori
cf_730	Italia, Francia, Germania
cf_731	Veneto, Lazio, Bretagna, Alsazia, Baviera, Sassonia
cf_732	Verona, Padova, Venezia, Roma, Rennes, Brest, Strasburgo, Monaco, Norimberga, Dresda

Si possono collegare con le seguenti chiamate:

```
require_once('modules/SDK/examples/300Utils.php');

linkedListAddLink("cf_730", "cf_731", "Italia", array("Veneto", "Lazio");
linkedListAddLink("cf_730", "cf_731", "Francia",
    array("Bretagna", "Alsazia")
);
...
...
linkedListAddLink("cf_731", "cf_732", "Baviera",
    array("Monaco", "Norimberga")
);
linkedListAddLink("cf_731", "cf_732", "Sassonia", array("Dresda"));
```

Business unit

Un'applicazione delle advanced query e advanced permission è il campo Business Unit, il cui obiettivo è di fornire privilegi particolari ad alcuni utenti. E' stato creato un nuovo campo per il modulo Utenti e Aziende chiamato appunto "Business Unit" (uitype 33, multilist) nel quale selezionare uno o più valori. Nella ListView delle Aziende, un'azienda è mostrata anche se condivide con l'utente tale valore, indipendentemente da altri criteri di visibilità.

Per ottenere ciò si è creato il nuovo campo nel modulo Users e Accounts tramite vtlib e in seguito si è registrata una funzione con *setAdvancedQuery()* che aggiunge alla query il controllo per il campo Business Unit in modo che questo venga visualizzato nella ListView. Poi si è registrata un'altra funzione con *setAdvancedPermissionFunction()* in modo da garantire l'accesso agli utenti desiderati, in modo da permettere loro il pieno controllo di tale azienda.

Vedere il file `modules/SDK/examples/testFilter.php` per l'implementazione.

Viste

Ecco un esempio di file che gestisce le viste.

```
<?php
global $sdk_mode;
switch($sdk_mode) {
    case '': //createview
        break;
    case 'edit':
    case 'detail':
        if ($col_fields['industry'] == 'Banking' &&
            $current_user->id == 6 &&
            $fieldname == 'website') {
            $readonly = 1;
            $success = true;
        }
        break;
    // nei seguenti casi aggiungo alla query la colonna industry
    case 'popup_query':
    case 'list_related_query':
        $sdk_columns = array('vtiger_account.industry');
        include('modules/SDK/AddColumnsToQueryView.php');
        break;
    // recupero il campo industry e se è Banking, imposto l'accesso
    case 'popup':
    case 'related':
    case 'list':
        $sdk_columnnames = array('industry');
        include('modules/SDK/GetFieldsFromQueryView.php');
```

```
        if ($sdk_columnvalues['industry'] == 'Banking') {  
            $readonly = 1;  
            $success = true;  
        }  
        break;  
    }  
?>
```

Che viene registrato con:

```
SDK::addView(  
    'Contacts',  
    'modules/SDK/examples/View3.php',  
    'constrain',  
    'continue'  
);
```

File Customisation

Alla fine dei file Delete.php, DetailView.php e ListView.php (prima del comando display di smarty) viene verificata la presenza di un file all'interno della cartella del modulo corrente ed eventualmente incluso.

Questo per esempio permette di sovrascrivere o aggiungere variabili smarty e anche di modificare il template evitando di usare le SDK::setFile e SDK::setSmarty.

Esempi

includere il file modules/Accounts/SDKDeleteCustomisations.php alla fine del file modules/[Accounts/VteCore]/Delete.php

```
SDK::setFile('Accounts', 'DeleteCustomisations', 'SDKDeleteCustomisations');
```

includere il file modules/Accounts/SDKDetailViewCustomisations.php alla fine del file modules/[Accounts/VteCore]/DetailView.php

```
SDK::setFile('Accounts', 'DetailViewCustomisations', 'SDKDetailViewCustomisations');
```

includere il file modules/Accounts/SDKListViewCustomisations.php alla fine del file modules/[Accounts/VteCore]/ListView.php

```
SDK::setFile('Accounts', 'ListViewCustomisations', 'SDKListViewCustomisations');
```

Nel file SDK creato va incluso anche l'eventuale file esistente in modo da essere compatibile con futuri aggiornamenti. Qui l'esempio per DetailView, negli altri casi basta includere DeleteCustomisations.php o ListViewCustomisations.php.

```
<?php
global $currentModule;
@include("modules/$currentModule/DetailViewCustomisations.php");

// tuo codice...
?>
```

Hooks

```
modules/VteCore/Delete.php
modules/MyNotes/Delete.php
modules/VteCore/DetailView.php
modules/VteCore/DetailViewBlocks.php
modules/Services/DetailView.php
modules/MyNotes/DetailView.php
modules/Charts/DetailView.php
modules/VteCore/ListView.php
modules/Calendar/ListView.php
modules/RecycleBin/ListView.php
modules/Users/ListView.php
```