SDK 2

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

Per rimuoverlo usare la funzione:

Vtiger_Link::deleteLink($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

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	settotalnoofrows 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

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 indicando il nuovo uitype con un valore non utilizzato (nnn).
2. 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...)
3. Registrare il nuovo tipo con una chiamata a SDK::setUitype.

Nel file modules/SDK/examples/VTE-SDK-2.php ci sono vari esempi di creazione campi e registrazione uitype.
In modules/SDK/doc/VTE-SDK-2.pdf alla voce **Uitypes List** è presente la lista dei principali uitype standard.

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 ricercarlo.

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:

$params	**Note**
array('module'=>'Leads', 'action'=>'ListView')	ListView
array('module'=>'Leads', 'action'=>'index')	ListView
array('module'=>'Leads', 'action'=>'DetailView' , 'record'=>'$NOTNULL$')	DetailView
array('module'=>'Leads', 'action'=>'EditView' , 'record'=>'$NOTNULL$')	EditView (deve esserci record nella request)
array('module'=>'Leads', 'action'=>'EditView')	Creazione nuovo elemento
array('module'=>'Leads', 'action'=>'EditView', 'record'=>'$NOTNULL$', "isDuplicate"=>"true")	Duplicazione

array(‘module’=>’Leads’, ‘action’=>’LeadsAjax’, ‘record’=>’$NOTNULL$’, 'ajxaction'=>'LOADRELATEDLIST', 'header'=>'Products')	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

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

Esempio

```
<?php
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');

// you can set a PopupQquery and a PopupReturnFunction to a field in a table field (VTENEXT 24.08)
SDK::setPopupQuery('field', 'Accounts', 'ml1_f4', 'modules/SDK/examples/Contacts/AccountQuery.php');
// here we have a table field (vcf_3) with an account field (vcf_4) and a contact field (vcf_5), I can view only accounts of rating Market Failed, Project Cancelled and Shutdown and only contacts of these accounts.
$rel_fields = array('processmaker'=>'jQuery("#processmaker").val()','running_process'=>'jQuery("#running_process").val()');
SDK::setPopupQuery('field', 'Processes', 'vcf_3_vcf_4', 'modules/SDK/examples/Contacts/AccountQuery.php', $rel_fields);
$rel_fields['accountid'] = 'jQuery("#vcf_3_vcf_4_"+VTE.EditView.getTableFieldCurrentRow(this)).val()';
SDK::setPopupQuery('field', 'Processes', 'vcf_3_vcf_5', 'modules/SDK/examples/Contacts/ContactsQuery.php', $rel_fields);
?>
```

AccountQuery.php

```
<?php
global $table_prefix;

// check the current position in process
if ($_REQUEST['srcmodule'] == 'Processes') {
    $processmaker = 4;
    $elementid = 'Task_0j1yxi6'; // task condition after the dynaform

    $curr_processmaker = RequestHandler::paramGetInt('processmaker');
    $curr_running_process = RequestHandler::paramGetInt('running_process');

    if ($curr_processmaker != $processmaker) return;
```

```
    require_once('modules/Settings/ProcessMaker/ProcessMakerUtils.php');
    $PMU = ProcessMakerUtils::getInstance();
    $curr_elementid = $PMU->getCurrentElementId($curr_running_process, $curr_processmaker);
    if ($curr_elementid != $elementid) return;
}

$sdk_show_all_button = true;
$query .= " and {$table_prefix}_account.rating not in ('Market Failed','Project Cancelled','Shutdown')";
```

ContactsQuery.php

```
<?php
global $table_prefix;
$accountid = RequestHandler::paramGetInt('accountid');

// check the current position in process
if ($_REQUEST['srcmodule'] == 'Processes') {
    $processmaker = 4;
    $elementid = 'Task_0j1yxi6'; // task condition after the dynaform

    $curr_processmaker = RequestHandler::paramGetInt('processmaker');
    $curr_running_process = RequestHandler::paramGetInt('running_process');

    if ($curr_processmaker != $processmaker) return;

    require_once('modules/Settings/ProcessMaker/ProcessMakerUtils.php');
    $PMU = ProcessMakerUtils::getInstance();
    $curr_elementid = $PMU->getCurrentElementId($curr_running_process, $curr_processmaker);
    if ($curr_elementid != $elementid) return;
}

$sdk_show_all_button = false;
$query .= " and {$table_prefix}_contactdetails.accountid = $accountid";
```

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

Nota: nel caso MassEditSave, è possibile conoscere i record coinvolti richiamando la funzione getListViewCheck($currentModule);

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.php avrà questo contenuto:

```
<?php
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 rispettivamente lettura/scrittura, sola lettura, nascosto)
$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 $sdk_mode	Variabili disponibili	Note
CreateView	""	*$col_fields* *$current_user* *$fieldname* *$readonly* *$success*	1
EditView	"*edit*"		
DetailView	"*detail*"		
MassEdit	"*mass_edit*"		
PopupQuery	"*popup_query*"	*$sdk_columns* *$success*	2
List/RelatedQuery	"*list_related_query*"	*$sdk_columns* *$success*	
Popup	"*popup*"	*$current_user* *$fieldname* *$sdk_columnnames* *$sdk_columnvalues* *$readonly* *$success*	3
Related	"*related*"		4
List	"*list*"		

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
$decription : 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 dattagli 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 $class deve rispettare questa struttura, di seguito esempio:

```
<?php
require_once('modules/Reports/ReportRun.php');


class ReportRunAccounts extends ReportRun {
	
	var $enableExportPdf = true;
	var $enableExportXls = true;
```

```
	var $enablePrint = true;
	var $hideParamsBlock = true;
	
	function __construct($reportid) {
		$this->reports = Reports::getInstance($reportid); // crmv@172034
		$this->reportid = $reportid;
		$this->primarymodule = 'Accounts';
		$this->reporttype = '';
		$this->reportname = 'Account con sito';
		$this->reportlabel = getTranslatedString($this->reportname, 'Reports');
	}
	
	function getSDKBlock() {
		global $mod_strings;
		$sdkblock = '<p><h2>Questo report mostra le aziende con sito</h2></p>';
		// here I can also add custom inputs to filter the report or any html I need
		return $sdkblock;
	}
	
	// overridden, always hide the summary tab
	function hasSummary() {
		return false;
	}
	
	// overridden, always show the total tab
	function hasTotals() {
		return true;
	}
	
	// generate the report
	function GenerateReport($outputformat = "", $filterlist = null, $directOutput=false) {
		global $adb;
		
		// compatibility, please use set them with the proper methods
		if (!empty($outputformat)) {
			$format = "HTML";
			$tab = "MAIN";
			
			if (strpos($outputformat, 'HTML') !== false) $format = "HTML";
			if (strpos($outputformat, 'PRINT') !== false) $format = "PRINT";
```

```
□□□if (strpos($outputformat, 'PDF') !== false) $format = "PDF";
□□□if (strpos($outputformat, 'XLS') !== false) $format = "XLS";
□□□if (strpos($outputformat, 'JSON') !== false) $format = "JSON";
□□□if (strpos($outputformat, 'CV') !== false) $format = "NULL";
□□□
□□□if (strpos($outputformat, 'COUNT') !== false) $tab = "COUNT";
□□□if (strpos($outputformat, 'TOTAL') !== false) $tab = "TOTAL";
□□□if (strpos($outputformat, 'CV') !== false) $tab = "CV";
□□□
□□□$this->setOutputFormat($format, $directOutput);
□□□$this->setReportTab($tab);
□□} else {
□□□$format = $this->outputFormat;
□□□$tab = $this->reportTab;
□□}
□□
□□$format = $this->outputFormat;
□□$direct = $this->directOutput;
□□$tab = $this->reportTab;

□□// prepare the output class
□□$output = $this->getOutputClass();
□□$output->clearAll();
□□
□□$return_data = array();
□
□□
□□if ($tab == 'COUNT' && $this->hasSummary()) {

□□□// no summary for this custom report
□□□
□□} elseif ($tab == 'CV') {
□□
□□□// no customview for this report
□□
□□} elseif ($tab == 'MAIN') {

□□□$sSQL = $this->getReportQuery($outputformat, $filterlist);

□□□$result = $adb->query($sSQL);
```

```
		$this->total_count = $adb->num_rows($result);
		
		$error_msg = $adb->database->ErrorMsg();
		if(!$result && $error_msg!=''){
			// Performance Optimization: If direct output is requried
			if($direct) {
				echo getTranslatedString('LBL_REPORT_GENERATION_FAILED', 'Reports') . "<br>" . $error_msg;
				$error_msg = false;
			}
			// END
			return $error_msg;
		}
		
		if($result) {
		
			$this->generateHeader($result, $output);
		
			while ($row = $adb->fetchByAssoc($result)) {
				$colcount = count($row);
				foreach ($row as $column => $value) {
					$cell = array(
						'value' => $value,
						'column' => $column,
						'class' => 'rptData',
					);
					$output->addCell($cell);
				}
				$output->endCurrentRow();
		
			}
		
			$output->countTotal = $this->total_count;
			$output->countFiltered = $this->total_count;
		
			if ($format == 'XLS') {
				$head = $output->getSimpleHeaderArray();
				$data = $output->getSimpleDataArray();
				foreach ($data as $row) {
					$return_data[] = array_combine($head, $row);
				}
```

```
        } else {
            $return_data[] = $output->output(!$direct);
            $return_data[] = $this->total_count;
            $return_data[] = $sSQL;
            $return_data[] = $colcount;
        }
    }

} elseif ($tab == "TOTAL" && $this->hasTotals()) {
    $output->addHeader(array('column' => 'fieldname', 'label' => getTranslatedString('Totals')));
    $output->addHeader(array('column' => 'sum', 'label' => getTranslatedString('SUM')));
    $output->addHeader(array('column' => 'avg', 'label' => getTranslatedString('AVG')));
    $output->addHeader(array('column' => 'min', 'label' => getTranslatedString('MIN')));
    $output->addHeader(array('column' => 'max', 'label' => getTranslatedString('MAX')));

    // fixed totals
    $rows = array(
        array(
            array('column'=> 'fieldname', 'value' => 'Fatturato totale', 'class' => 'rptData'),
            array('column'=> 'sum', 'value' => 2000, 'class' => 'rptTotal'),
            array('column'=> 'avg', 'value' => null, 'class' => 'rptTotal'), // not used
            array('column'=> 'min', 'value' => 850, 'class' => 'rptTotal'),
            array('column'=> 'max', 'value' => null, 'class' => 'rptTotal'), // not used
        ),
    );

    // add them to the output class
    foreach ($rows as $row) {
        foreach ($row as $cell) {
            $output->addCell($cell);
        }
        $output->endCurrentRow();
    }

    // format for xls or html
    if ($format == "XLS") {

        // change the output array to match the expected format for XLS export
```

```
        $return_data = array();
        $data = $output->getSimpleDataArray();
        $fieldName = '';
        foreach ($data as $row) {
            $nrow = array();
            foreach ($row as $key => $value) {
                if ($key == 'fieldname') {
                    $fieldName = $value;
                    continue;
                }
                $klabel = $fieldName.'_'.strtoupper($key);
                $nrow[$klabel] = $value;
            }
            $return_data[] = $nrow;
        }
    
    } else {
        $return_data = $output->output(!$direct);
    }

}

return $return_data;
}

// generate a fixed header for the report
function generateHeader($result, $output, $options = array()) {
    global $adb, $table_prefix;

    $module = 'Accounts';
    $tabid = getTabid($module);
    $count = $adb->num_fields($result);

    for ($x=0; $x<$count; ++$x) {
        $fld = $adb->field_name($result, $x);

        // get the field label from the column (if possible)
        $res = $adb->pquery("SELECT fieldlabel FROM {$table_prefix}_field WHERE columnname = ? and tabid = ?",
array($fld->name, $tabid));
        if ($res && $adb->num_rows($res) > 0) {
```

```
        $fieldlabel = $adb->query_result_no_html($res, 0, 'fieldlabel');
        $headerLabel =  getTranslatedString($fieldlabel, $module);
    } else {
        $headerLabel = $fld->name;
    }
    $hcell = array(
        'column' => $fld->name,
        'label' => $headerLabel,
        'orderable' => false,
        'searchable' => false,
    );
    $output->addHeader($hcell);
  }

 }

// generate the report query
function getReportQuery($outputformat, $filterlist) {
  global $table_prefix;
  $query = 'SELECT accountid, accountname, website, phone FROM '.$table_prefix.'_account WHERE website <>
"" ';
  return $query;
 }
}
```

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:

```
<?php
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 gestista manualmente anche la generazione del filtro temporale, ad esempio usando una funzione come questa:

```
<?php
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

La modalità di visione dei log è cambiata nelle varie release di VTENEXT e viene riassunta come segue.

- VTENEXT 16.09: Log consultabile tramite accesso al filesystem nella cartella logs/ProcessEngine
- VTENEXT 18.X: In Impostazioni → Business Process Manager → ProcessManager sarà disponibile il tasto ‘LOG’ una volta eseguito il seguente script:

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

> Inizialmente viene creato il file 01.log. Quando questo supera i 5MB verrà creato il file 02.log, e così via.

- VTENEXT 19.10: Log consultabili in Impostazioni → Altre Impostazioni → Log di Sistema.

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 medoto 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.

Esempio

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

SDK

È possibile aggiungere funzioni custom ai processi, clicca qui per maggiori dettagli.

Portale v2

Documentazione tema

https://adminlte.io/docs/3.2/

https://github.com/ColorlibHQ/AdminLTE

https://adminlte.io/themes/v3/

Webservice REST

Handler: include/Webservices/CustomerPortal.php

Headers:
• Authorization: Basic base64_encode("username:accesskey")
• Portal-Session-Id: il parametro è opzionale e viene restituito dalla "portal.login". Utilizzare nelle chiamate successive per ottimizzare le performance del portale.
• Content-Type: application/json

Nome	Parametri	Descrizione
portal.info		Permette di ottenere le informazioni sulla licenza e i loghi utilizzati dall'installazione di vtenext collegata.
portal.login	**Body** . username: string (*) . password: string (*) . language: string	Permette di eseguire il login nel portale fornendo l'email (username) e la password del contatto. Il servizio restituisce anche i valori dell'accesskey e dell'id di sessione da utilizzare nelle chiamate successive come header (Authorization e Portal-Session-Id)
portal.logout	**Headers** . Authorization	Permette di eseguire il logout del contatto dal portale.
portal.send_mail_for_password	**Body** . email: string (*) . language: string	Permette il recupero della password di un contatto del portale.
portal.modules_list	**Headers** . Authorization **Body** . language: string (*)	Permette di ottenere una lista di moduli abilitati per il profilo del contatto.

portal.get_list	**Headers** . Authorization **Body** . module: string (*) . language: string . folderid: int . search: array	Permette di ottenere la lista dei record di un modulo filtrati per la visibilità del profilo del contatto. Il parametro "folderid" viene utilizzato dal modulo Documents per ottenere i documenti di una cartella specifica. Il parametro "search" permette di eseguire la paginazione della lista server-side. Es. ['length' => 50, // Numero di record da restituire 'start' => 0, // Offset della paginazione 'search' => '', // Ricerca globale 'search_columns' => [// Ricerca per colonne ['index' => 0, // Indice del campo 'column' => '', // Nome del campo 'search' => '', // Valore del campo]], 'ordering' => [// Ordinamento ['index' => 0, // Indice del campo 'column' => '', // Nome del campo 'dir' => '', // asc/desc]],] Note: 1. Tutte le ricerche vengono eseguite SOLO sulle colonne della lista. 2. La ricerca per colonna viene eseguita in modalità "LIKE". 3. L'ordinamento può essere eseguito solo su una colonna.
portal.get_blocks	**Headers** . Authorization **Body** . module: string (*) . language: string . mode: string (edit, create, detail, list) (*) . app_data: array	Permette di ottenere la lista dei blocchi e dei campi di un modulo filtrati per la visibilità del profilo del contatto. Il parametro "app_data" rappresenta il record che si sta modificando (array con nome campo e valore) ed è utilizzato dalle view SDK per stabilire la visibilità dei campi. Questo parametro verrà utilizzato anche in futuro per la gestione dei campi condizionali.
portal.get_record	**Headers** . Authorization **Body** . module: string (*) . id: int (*)	Permette di ottenere i dati del record indicato se il contatto ha il permesso di visualizzarlo.

portal.save_record	**Headers** . Authorization **Body** . module: string (*) . id: int (*) . values: encoded (*)	Permette il salvataggio del record indicato se il contatto ha il permesso di modificarlo.
portal.delete_record	**Headers** . Authorization **Body** . module: string (*) . id: int (*)	Permette l'eliminazione del record indicato se il contatto ha il permesso di cancellarlo.
portal.write_ticket_comment	**Headers** . Authorization **Body** . id: int (*) . comment: string (*)	Permette la scrittura di un commento all'interno del ticket indicato.
portal.get_attachments	**Headers** . Authorization **Body** . id: int (*)	Permette di ottenere la lista dei documenti relazionati al record indicato filtrati per la visibilità del profilo del contatto.
portal.download_attachment	**Headers** . Authorization **Body** . relid: int (*) . docid: int	Permette il download dell'allegato (relid) se il contatto ha il permesso di visualizzarlo.
portal.upload_attachment	**Headers** . Authorization **Body** . relid: int (*) . title: string (*) **File**	Permette il caricamento di un allegato relazionato al record specificato (relid). Si può indicare il nome (title) del documento che verrà generato.
portal.provide_confidential_info	**Headers** . Authorization **Body** . id: int (*) . comments: string . data: string (*) . request_commentid: int (*)	Permette di rispondere ad una richiesta di informazioni confidenziali. Il parametro "id" indica l'id del ticket, il parametro "comments" indica il commento non cifrato, il parametro "data" indica la risposta confidenziale e il parametro "request_commentid" indica l'id del commento a cui fornire la risposta confidenziale.
portal.get_home_widgets	**Headers** . Authorization **Body** . language: string	Permette di ottenere i widget configurati nel profilo del contatto.
portal.save_authenticate_cookie	**Headers** . Authorization **Body** . contactid: int (*)	Fornisce un hash da memorizzare in un cookie per ricordare l'accesso del contatto.

portal.check_authenticate_cookie	**Headers** . Authorization **Body** . contactid: int (*) . hash: string (*)	Permette di verificare l'hash utilizzato per ricordare l'accesso del contatto.
portal.change_password	**Headers** . Authorization **Body** . username: string (*) . old_password: string (*) . password: string (*) . language: string	Permette di cambiare la password del contatto.

Registrare un nuovo webservice REST

Creare un nuovo file ed eseguirlo (es. plugins/script/script.php).

```
<?php
require('../../config.inc.php');
chdir($root_directory);
require_once('include/utils/utils.php');
require_once('vtlib/Vtecrm/Module.php');
$Vtiger_Utils_Log = true;
global $adb, $table_prefix;
VteSession::start();

SDK::setClass('CustomerPortalRestApi', 'CustomerPortalRestApi2',
'modules/SDK/src/CustomerPortalRestApi2.php');

$parameters = ['param1' => 'string', 'param2' => 'encoded', 'param3' => 'encoded'];
$perm = 'read'; // read, write, readwrite
SDK::setRestOperation('portal.foo', 'modules/SDK/src/CustomerPortalRestApi2.php',
'CustomerPortalRestApi2.foo', $parameters, $perm);
```

Creare un nuovo file che contiene la classe estesa CustomerPortalRestApi2 (es. modules/SDK/src/CustomerPortalRestApi2.php).

```
<?php

require_once('include/Webservices/CustomerPortal.php');

class CustomerPortalRestApi2 extends CustomerPortalRestApi {
```

```
	public function foo($param1, $param2, $param3) {
		$data = [1, 2, 3, 4, 5];
        // ...
		return $data;
	}

}
```

Estendere un webservice REST esistente

Creare un nuovo file ed eseguirlo (es. plugins/script/script.php).

```
<?php
require('../../config.inc.php');
chdir($root_directory);
require_once('include/utils/utils.php');
require_once('vtlib/Vtecrm/Module.php');
$Vtiger_Utils_Log = true;
global $adb, $table_prefix;
VteSession::start();

SDK::setClass('CustomerPortalRestApi', 'CustomerPortalRestApi2',
'modules/SDK/src/CustomerPortalRestApi2.php');
```

Creare un nuovo file che contiene la classe estesa CustomerPortalRestApi2 (es. modules/SDK/src/CustomerPortalRestApi2.php).

```
<?php

require_once('include/Webservices/CustomerPortal.php');

class CustomerPortalRestApi2 extends CustomerPortalRestApi {

	public function get_list($module, $language, $folderid = 0, $search = []) {
		$ret = parent::get_list($module, $language, $folderid, $search);
		// your code here ...
		return $ret;
	}

}
```

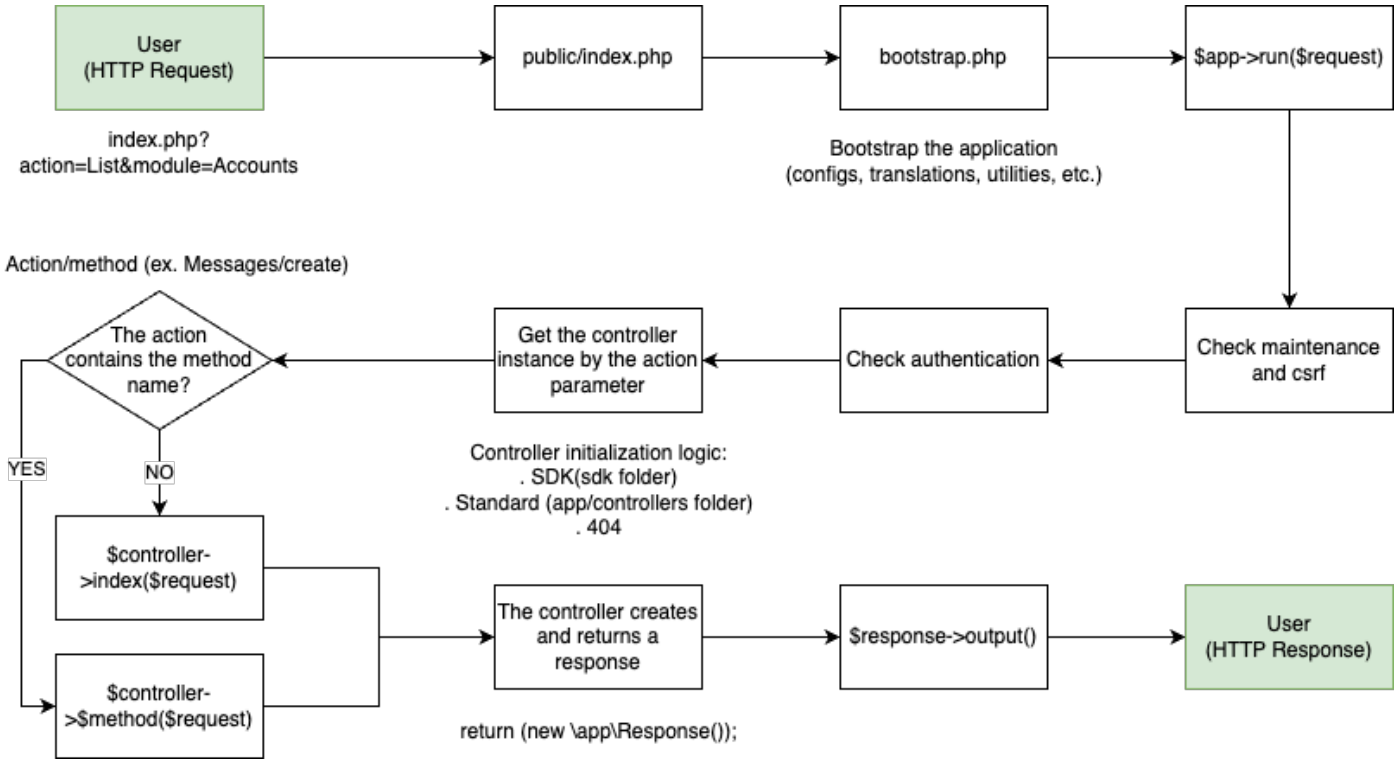
Struttura delle cartelle/file principali

> La cartella del portale si trova in: **VTE_ROOT/portal/v2**

Le cartelle/file principali del nuovo portale sono:

Nome cartella/file	**Descrizione**
app	Contiene i file e le logiche per il funzionamento del portale.
app/controllers	Contiene i file che gestiscono le azioni di default del portale (es. Login, Logout, Edit, Detail, ecc.).
app/fields	Contiene i file che gestiscono i campi dei moduli del portale.
app/modules	Contiene delle logiche custom di alcuni moduli di vtenext (es. Documents, Processes, HelpDesk, ecc.).
app/PortalModule.php	Classe che gestisce le azioni dei moduli del portale (es. Create, Detail, Edit, ecc.). La classe può essere estesa per modificare i comportamenti di default di un modulo.
config	Contiene i file di configurazione del portale.
public	Contiene i file pubblici del portale (es. index.php, css, javascript, immagini, ecc.).
resources	Contiene le risorse utilizzate dal portale come i file per le traduzioni (lang) e i template (templates).
sdk	La cartella è utilizzata per inserire nuove personalizzazioni per il cliente.
storage	Contiene i file temporanei (es. cache, logs, ecc.).
vendor	Contiene le librerie esterne utilizzate dal portale.

Request lifecycle



Configurazione del portale

Il file di configurazione del portale si trova in "config/portal.config.php".

> Per **sovrascrivere i parametri** deve essere utilizzato il file "config/sdk.config.php" altrimenti, in caso di aggiornamento della versione di vtenext, le modifiche potrebbero essere perse.

> **Come spostare il Business Portal in un'altra macchina/cartella**
> Valorizzare i parametri "portal_url", "vte_url" e "csrf_secret" nel file "config/sdk.config.php". Eventualmente cambiare anche il parametro "default_timezone" se la macchina ha un timezone diverso da quello in cui risiede il vte. Cambiare la prop "portal.url" con il link che punta alla cartella v2. es. https://ticket.vtenext.com/v2

Ecco la lista dei parametri supportati dal nuovo portale:

Parametro	Tipo	Default	Descrizione

portal_url	String	$PORTAL_URL (config.inc.php)	Indica l'URL del portale clienti. Se la cartella del portale si trova all'interno della root directory di vtenext, la variabile verrà impostata con il valore della variabile $PORTAL_URL impostata nel file config.inc.php.
vte_url	String	$site_URL (config.inc.php)	Indica l'URL di vtenext e viene utilizzato per ottenere i relativi dati tramite le rest API. Se la cartella del portale si trova all'interno della root directory di vtenext, la variabile verrà impostata con il valore della variabile $site_URL impostata nel file config.inc.php.
default_language	String	it_it	Indica la lingua predefinita utilizzata nel portale clienti. Il valore può essere sostituito con una lingua supportata (vedi parametro "languages").
languages	Array	['en_us' => 'US English', 'it_it' => 'IT Italiano']	Indica le lingue supportate nel portale clienti. Per aggiungere una nuova lingua si deve creare un nuovo file nella cartella resources/lang.
production	Bool	false	Questa configurazione indica se gli errori devono essere visualizzati o meno. Se l'ambiente è di produzione, il valore verrà impostato su true per disabilitare la visualizzazione degli errori. Se l'ambiente è di sviluppo, il valore verrà impostato su false per consentire la visualizzazione degli errori.

default_module	String		Indica il modulo predefinito da caricare dopo l'accesso al portale. Il valore può essere sostituito con il nome del modulo desiderato (deve essere abilitato da profilo). Se il parametro è vuoto, verrà caricata la home del portale.
favicon	String	assets/img/VTENEXT_favicon.ico	Indica il percorso della favicon da caricare nel portale clienti. Il valore è relativo alla cartella public.
login_logo	String	assets/img/VTENEXT_login.png	Indica il percorso del logo da caricare all'interno della pagina di login. Il valore è relativo alla cartella public. [UPDATE] Il logo deve essere caricato nelle impostazioni di vtenext > Loghi.
login_background	String		Indica il percorso dello sfondo da caricare all'interno della pagina di login. Il valore è relativo alla cartella public.
header_logo_sm	String	assets/img/VTENEXT_toggle.png	Indica il percorso dell'icona da caricare nella barra laterale minimizzata. Il valore è relativo alla cartella public. [UPDATE] L'icona deve essere caricata nelle impostazioni di vtenext > Loghi.
header_logo_lg	String	assets/img/VTENEXT_header.png	Indica il percorso dell'icona da caricare nella barra laterale allargata. Il valore è relativo alla cartella public. [UPDATE] L'icona deve essere caricata nelle impostazioni di vtenext > Loghi.
helpdesk_logo	String	assets/img/helpdesk.png	Indica il logo utilizzato per visualizzare la risposta data dall'assistenza clienti. Il valore è relativo alla cartella public.

sidebar_theme	String	sidebar-dark-primary	Indica la classe della barra laterale principale. Può avere una luminosità "dark" o "light". Può avere anche una variante di colore, come "primary", "success", "warning", "info", "danger".
enable_sidebar_search	Bool	false	Consente di attivare/disattivare la barra di ricerca nella barra laterale principale.
csrf_secret	String	$csrf_secret (config.inc.php)	Indica la chiave segreta utilizzata per generare un token csrf. Se la cartella del portale si trova all'interno della root directory di vtenext, la variabile verrà impostata con il valore della variabile $csrf_secret impostata nel file config.inc.php.
upload_dir	String		Indica il nome della cartella utilizzata per il caricamento dei file.
browser_title_prefix	String		Indica l'etichetta del prefisso da utilizzare per il titolo del browser. Il valore può essere sostituito con l'etichetta desiderata.
browser_title_suffix	String	customer_portal	Indica l'etichetta del suffisso da utilizzare per il titolo del browser. Il valore può essere sostituito con l'etichetta desiderata.
remember_cookie_name	String	portal_login_hash	Indica il nome del cookie utilizzato per ricordare l'autenticazione dell'utente.
login_expire_time	Int	2592000 (one month)	Indica la scadenza del cookie utilizzato per ricordare l'autenticazione dell'utente. Il valore può essere sostituito con il numero di secondi desiderato.

default_timezone	String	Europe/Rome	Indica il fuso orario predefinito utilizzato nel portale clienti. Questa configurazione deve essere uguale alla variabile $default_timezone impostata nel file config.inc.php di vtenext.
module_icons	Array	[]	Con questa configurazione è possibile sovrascrivere le icone predefinite utilizzate per i moduli abilitati nel portale clienti. Le icone predefinite dei moduli si trovano in app/layouts/PortalLayout.php. I nomi delle icone si possono trovare qui https://fonts.google.com/icons.
sdk_languages	Array	[]	Con questa configurazione si possono aggiungere nuove etichette o modificare quelle esistenti. Si deve creare un nuovo file nella cartella sdk.
sdk_global_php	Array	[]	Con questa configurazione si possono aggiungere file php da caricare in ogni pagina (dovrebbero contenere classi/funzioni). Si deve creare un nuovo file nella cartella sdk.
sdk_global_js	Array	[]	Con questa configurazione si possono aggiungere js da caricare a livello globale. Si deve creare un nuovo file nella cartella public/assets/sdk.
sdk_module_js	Array	[]	Con questa configurazione si possono aggiungere js da caricare per un modulo specifico. Si deve creare un nuovo file nella cartella public/assets/sdk.
sdk_global_css	Array	[]	Con questa configurazione si possono aggiungere css da caricare a livello globale. Si deve creare un nuovo file nella cartella public/assets/sdk.

sdk_controllers	Array	[]	Con questa configurazione si possono aggiungere azioni personalizzate (campo "action" nell'url). L'array rappresenta un'associazione tra il nome dell'azione e il file che contiene la classe controller per gestire la richiesta. Si deve creare un nuovo file nella cartella sdk.
sdk_module	Array	[]	Con questa configurazione si possono aggiungere personalizzazioni su un modulo specifico. L'array rappresenta un'associazione tra il nome del modulo e il file che contiene la classe estesa del modulo. Si deve creare un nuovo file nella cartella sdk.
sdk_menu	Array	[]	Con questa configurazione si possono aggiungere aggiungere voci di menu personalizzate nella barra laterale. Si deve creare un nuovo file nella cartella sdk.

API Reference - Classi, metodi, funzioni e variabili principali

\app\Request

Metodo	**Argomenti**	**Descrizione**

get()	$keys = null, $purify = false	Permette di ottenere uno o più parametri dalla variabile globale $_GET. Se il secondo argomento è impostato a "true", i parametri vengono purificati attraverso la libreria HTML Purifier. Esempio:
		`$request->get('foo');`
		`$request->get(['foo', 'bar']);`
		`$request->get('foo', true);`

post()	$keys = null, $purify = false	Permette di ottenere uno o più parametri dalla variabile globale $_POST. Se il secondo argomento è impostato a "true", i parametri vengono purificati attraverso la libreria HTML Purifier. Esempio:
		`$request->post('foo');`
		`$request->post(['foo', 'bar']);`
		`$request->post('foo', true);`

files()	$key	Permette di ottenere i file caricati attraverso il metodo HTTP POST e organizzati tramite la variabile globale $_FILES. `$request->files('attachments');`
cookie()	$keys = null, $purify = false	Permette di ottenere uno o più parametri dalla variabile globale $_COOKIE. Se il secondo argomento è impostato a "true", i parametri vengono purificati attraverso la libreria HTML Purifier. Esempio: `$request->cookie('foo');`
server()	$keys = null, $purify = false	Permette di ottenere uno o più parametri dalla variabile globale $_SERVER. Se il secondo argomento è impostato a "true", i parametri vengono purificati attraverso la libreria HTML Purifier. Esempio: `$request->server('');`
isGet()		Ritorna true se il metodo della richiesta è GET.
isPost()		Ritorna true se il metodo della richiesta è POST.
isAjax()		Ritorna true se la richiesta è AJAX.
purify()	$input	Purifica la variabile $input attraverso la libreria HTML Purifier.

\app\Response

Metodo	**Argomenti**	**Descrizione**

__construct()	$content = '', $statusCode = 200, $headers = []	Inizializza un nuovo oggetto \app\Response() con il contenuto della risposta ($content), il codice di ritorno ($statusCode) e gli header di default ($headers).
setContent()	$content	Imposta il contenuto della risposta.
setStatusCode()	$statusCode	Imposta il codice di ritorna della risposta.
setHeader()	$header, $replace = true	Imposta un nuovo header nella risposta. Con il secondo argomento è possibile indicare se l'header deve sostituire oppure no un header precedente già impostato.
setMimeType()	$mimeType = 'text/html'	Imposta il mime del contenuto della risposta.
json()	$data	Imposta il contenuto della risposta con $data convertito in formato JSON e mime 'application/json'.
redirect()	$page	Esegue un redirect verso $page.
downloadFile()	$fullpath	Permette lo scaricamento di un file posizionato in $fullpath.
output()		Esegue l'output del contenuto, del codice di risposta e degli header impostati.

\app\Session

Metodo	**Argomenti**	**Descrizione**
set()	$key, $value = ''	Imposta il valore $value con chiave $key in $_SESSION.
get()	$key	Permette di ottenere il valore con chiave $key da $_SESSION.
flash()	$key	Permette di ottenere il valore con chiave $key da $_SESSION. Successivamente, la chiave $key verrà eliminata da $_SESSION.
remove()	$key	Elimina la chiave $key da $_SESSION.
hasKey()	$key	Ritorna true se esiste la chiave $key in $_SESSION.
setMulti()	$keys	Permette di scrivere valori multipli in $_SESSION.

removeMulti()	$keys	Elimina valori multipli in $_SESSION.
append()	$key, $value = ''	Imposta la chiave $key come un array in $_SESSION e il valore $value viene aggiunto a quest'ultimo.

\app\Config

Metodo	**Argomenti**	**Descrizione**
has()	$key	Ritorna true se esiste la chiave $key nella configurazione globale.
get()	$key	Permette di ottenere il valore con chiave $key dalla configurazione globale.
set()	$key, $value	Imposta il valore $value con chiave $key nella configurazione globale.
getAll()		Permette di ottenere una lista chiave-valore con tutta la configurazione globale.
setMulti()	$values	Permette di scrivere valori multipli nella configurazione globale.
clear()	$key	Elimina la chiave $key dalla configurazione globale.
clearAll()		Elimina tutti i valori dalla configurazione globale.

\app\PortalModule

Variabile	**Default**	**Descrizione**
$hasComments	false	Indica se il modulo supporta il blocco commenti.
$hasAttachments	false	Indica se il modulo supporta gli allegati.
$enableEdit	true	Indica se il modulo può essere modificato (modalità edit).
$formColumns	3	Indica il numero di colonne da utilizzare per la visualizzazione dei campi in Create, Edit e Detail.
$listTemplate	List.tpl	Indica il template utilizzato per la visualizzazione di una lista (action List).

$referenceListTemplate	sections/ReferenceList.tpl	Indica il template utilizzato per la visualizzazione di una lista relazionata (uitype 10).
$detailTemplate	Detail.tpl	Indica il template utilizzato per la visualizzazione del dettaglio di un record (action Detail).
$editTemplate	Edit.tpl	Indica il template utilizzato per la visualizzazione della modifica di un record (action Edit).
$notAuthorizedTemplate	PageNotAuthorized.tpl	Indica il template utilizzato per visualizzare un errore di permesso (es. record non trovato o errori di permessi).

Metodo	**Argomenti**	**Descrizione**
__construct()	$module	$module indica il nome del modulo per ottenere un'instanza della classe. Se il modulo è stato esteso tramite sdk allora verrà ritornata un'istanza della classe estesa.
prepareList()	$viewer, $request	Metodo utilizzato per la visualizzazione di una lista (action List).
postProcessList()	$viewer, $request	Questo metodo può essere utilizzato dalle classi estese per inserire/modificare i dati impostati nella prepareList().
prepareEdit()	$viewer, $request	Metodo utilizzato per la visualizzazione della modifica di un record (action Edit).
postProcessEdit()	$viewer, $request	Questo metodo può essere utilizzato dalle classi estese per inserire/modificare i dati impostati nella prepareEdit().
prepareDetail()	$viewer, $request	Metodo utilizzato per la visualizzazione del dettaglio di un record (action Detail).
postProcessDetail()	$viewer, $request	Questo metodo può essere utilizzato dalle classi estese per inserire/modificare i dati impostati nella prepareDetail().
saveRecord()	$request	Metodo utilizzato per il salvataggio di un record (action Save).

postProcessSaveValues()	$request, &$values	Questo metodo può essere utilizzato dalle classi estese per inserire/modificare i dati impostati nella saveRecord().
isPermitted()	$module, $action = self::ACTION_LIST, $recordValues = []	Ritorna se il contatto ha il permesso di eseguire una determinata azione. Lista azioni supportate: . ACTION_LIST . ACTION_CREATE . ACTION_EDIT . ACTION_DETAIL . ACTION_DELETE . ACTION_SAVE . ACTION_ADD_COMMENTS . ACTION_UPLOAD_ATTACHMENTS . ACTION_CHANGE_PWD . ACTION_SOLVE_TICKET

\app\clients\PortalRestClient

Metodo	**Argomenti**	**Descrizione**
get()	$restName, $queryParameters = [], $headers = []	Permette di eseguire una richiesta GET verso vtenext.
post()	$restName, $queryParameters = [], $bodyParameters = [], $headers = []	Permette di eseguire una richiesta POST verso vtenext.
postMultipart()	$restName, $queryParameters = [], $bodyParameters = [], $fileParameters = [], $headers = []	Permette di eseguire una richiesta POST multipart verso vtenext.
patch()	$restName, $queryParameters = [], $bodyParameters = [], $headers = []	Permette di eseguire una richiesta PATCH verso vtenext.
delete()	$restName, $queryParameters = [], $headers = []	Permette di eseguire una richiesta DELETE verso vtenext.
postDownload()	$restName, $queryParameters = [], $bodyParameters = [], $headers = []	Permette di eseguire una richiesta POST per scaricare un file di vtenext in modalità stream.

Helpers

Funzione	**Argomenti**	**Descrizione**
preprint()	$var	print_r formattato con tag <pre>
predump()	$var	var_dump formattato con tag <pre>

encodeForHtml()	$value, $charset = 'UTF-8'	Codifica il valore $value per essere inserito in una pagina HTML.
encodeForHtmlAttr()	$value, $enclosing = '"'	Codifica il valore $value per essere inserito all'interno di un attributo di un tag HTML.
encodeForJs()	$value, $enclosing = '"'	Codifica il valore $value per essere inserito all'interno di uno <script> javascript.
htmlAttr()	$attributes	Codifica una lista di attributi per essere inseriti all'interno di un tag HTML.
config()	$key	Permette di ottenere il valore con chiave $key dalla configurazione globale.
trans()	$key, $args = []	Permette la traduzione di un'etichetta $key.
listUrl()	$module, $extraParams = []	Genera un link per aprire una lista.
createUrl()	$module, $extraParams = []	Genera un link per aprire la creazione di un record.
createDocUrl()	$module, $folderId, $extraParams = []	Genera un link per aprire la creazione di un documento.
detailUrl()	$module, $record, $extraParams = []	Genera un link per aprire il dettaglio di un record.
editUrl()	$module, $record, $extraParams = []	Genera un link per aprire la modifica di un record.
downloadUrl()	$record, $documentId, $extraParams = []	Genera un link per eseguire il download di un documento.
docFolderUrl()	$module, $folderId, $extraParams = []	Genera un link per aprire una cartella specifica nel modulo documenti.
returnUrl()	$request	Genera un link di ritorno indietro (es. azione "Annulla").
portalLanguage()		Ritorna la lingua utilizzata nel portale.
setPortalLanguage()	$language	Imposta la lingua $language nel portale.
getBrowserTitle()	$title	Ritorna il titolo da impostare in una pagina HTML del portale.
getModuleLabel()	$module	Ritorna la traduzione del modulo $module.
getSingleModuleLabel()	$module	Ritorna la traduzione singolare del modulo $module.

getMaxUploadSize()		Ritorna la dimensione massima di upload nel portale.
setPortalCookie()	$name, $value = "", $expires_or_options, $httponly = false	Imposta un cookie nel portale.
unsetPortalCookie()	$name, $httponly = false	Rimuove un cookie dal portale.
csrfToken()		Ritorna il token csrf.
csrfInputName()		Ritorna il nome dell'input per l'invio del token csrf.
flashPortalError()	$error	Permette di visualizzare un messaggio di errore temporizzato.
flashPortalMessage()	$message	Permette di visualizzare un messaggio temporizzato.
customerId()		Ritorna l'id del contatto autenticato.
customerEmail()		Ritorna l'email del contatto autenticato.
customerUsername()		Ritorna il nome e cognome del contatto autenticato.
resourcever()	$filename	Permette il versionamento di file css e javascript.
basePath()	$path = ''	Ritorna il percorso della cartella base del portale. Se viene indicato $path viene creato e restituito un percorso a partire dalla cartella base.
appPath()	$path = ''	Ritorna il percorso della cartella "app" del portale. Se viene indicato $path viene creato e restituito un percorso a partire dalla cartella "app".
configPath()	$path = ''	Ritorna il percorso della cartella "config" del portale. Se viene indicato $path viene creato e restituito un percorso a partire dalla cartella "config".
resourcesPath()	$path = ''	Ritorna il percorso della cartella "resources" del portale. Se viene indicato $path viene creato e restituito un percorso a partire dalla cartella "resources".
langPath()	$lang	Ritorna il percorso del file della lingua $lang indicata.

storagePath()	$path = ''	Ritorna il percorso della cartella "storage" del portale. Se viene indicato $path viene creato e restituito un percorso a partire dalla cartella "storage".
publicPath()	$path = ''	Ritorna il percorso della cartella "public" del portale. Se viene indicato $path viene creato e restituito un percorso a partire dalla cartella "public".
vtePath()	$path = ''	Ritorna il percorso della cartella di vtenext. Se viene indicato $path viene creato e restituito un percorso a partire dalla cartella di vtenext.
sdkPath()	$path = ''	Ritorna il percorso della cartella "sdk" del portale. Se viene indicato $path viene creato e restituito un percorso a partire dalla cartella "sdk".
sdkAssetsPath()	$path = ''	Ritorna il percorso della cartella "public/assets/sdk" del portale. Se viene indicato $path viene creato e restituito un percorso a partire dalla cartella "public/assets/sdk".
publicRelPath()	$assetPath	Ritorna un percorso relativo a partire dalla cartella "public".

Esempi

View SDK

Ecco un esempio di come modificare la visibilità dei campi del portale tramite view SDK.

```
<?php
require('../../config.inc.php');
chdir($root_directory);
require_once('include/utils/utils.php');
require_once('vtlib/Vtecrm/Module.php');
$Vtiger_Utils_Log = true;
global $adb, $table_prefix;
VteSession::start();

$module = '';
$src = '';
$mode = 'constrain';
```

```
$success = 'continue';
SDK::addView($module, $src, $mode, $success);
```

```
<?php

global $sdk_mode, $table_prefix;

switch ($sdk_mode) {
	case 'portal.create':
		$readonly = 100;
		$success = true;
		break;
	case 'portal.edit':
	case 'portal.detail':
		if ($col_fields['field3'] !== 'Open') {
			$readonly = 99;
			$success = true;
		}
		if (in_array($fieldname, ['field1', 'field2'])) {
			$readonly = 100;
			$success = true;
		}
		break;
}
```

Creazione di un nuovo controller

Ecco un esempio di come creare un nuovo controller per gestire un'azione custom.

- Modificare "config/sdk.config.php" inserendo il nuovo controller

```
return [
	'sdk_controllers' => [
		'SampleVte' => 'controllers/SampleVteController.php',
	]
];
```

- Implementare la classe SampleVteController in "sdk/controllers/SampleVteController.php"

```
<?php
```

```
class SampleVteController extends \app\controllers\BaseController {
	
	public function index($request) {
		return $this->displaySomething($request);
	}

	protected function displaySomething($request) {
		$parameter1 = $request->get('parameter1', true);
		$parameter2 = $request->get('parameter2', true);

		$this->viewer->assign('PARAMETER1', $parameter1);
		$this->viewer->assign('PARAMETER2', $parameter2);

		$layout = \app\LayoutFactory::getPortalLayout($this->viewer, $this->client, $request);
		$output = $this->fetchWithLayout('sdk/SampleVte.tpl', $layout);

		return new \app\Response($output);
	}
	
}
```

- Creare un nuovo template in "resources/templates/sdk/SampleVte.tpl"

```
{extends file='layouts/PortalLayout.tpl'}

{block name=content}
	<h1>Sample Vte</h1>
{/block}
```

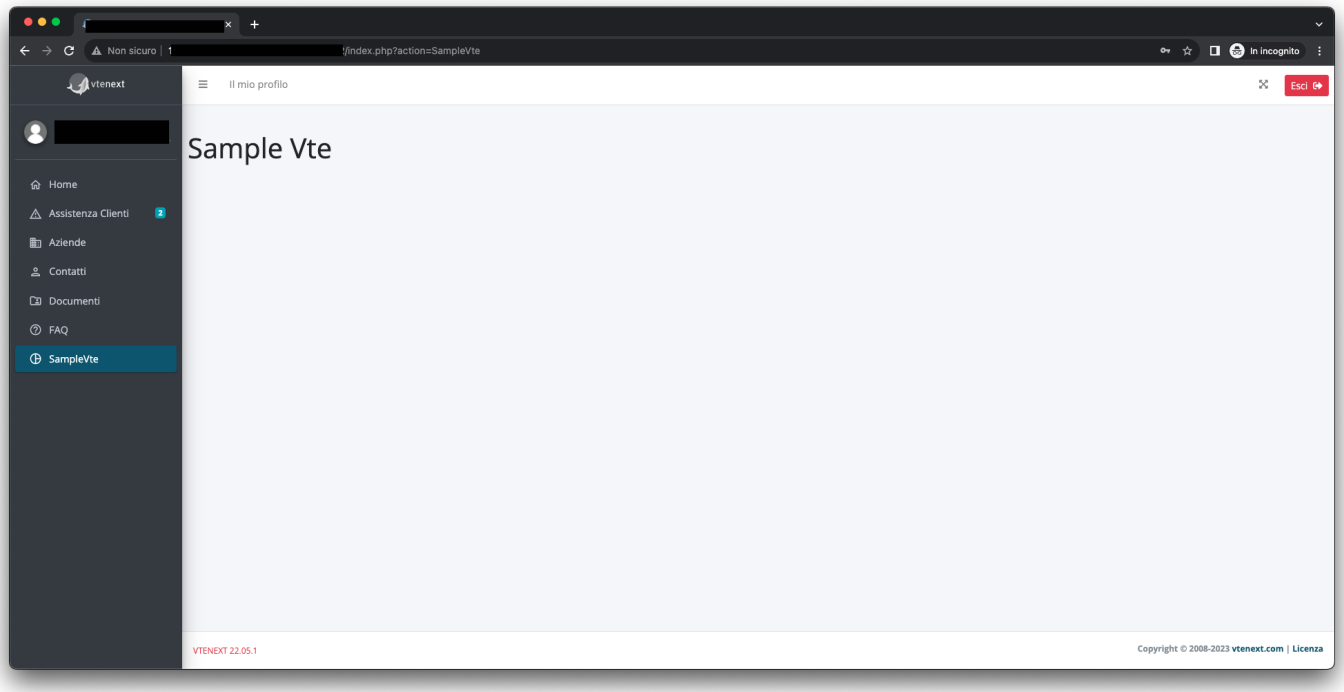
- Modificare "config/sdk.config.php" indicando il file sdk per l'inserimento delle nuovi voci nel menù laterale

```
return [
	'sdk_controllers' => [
		'SampleVte' => 'controllers/SampleVteController.php',
	],
	 'sdk_menu' => [
		'samplevte-menu.php',
	]
];
```

- Modificare il file "sdk/samplevte-menu.php"

```
<?php

return [
    [
        'text' => trans('SampleVte'),
        'active' => false,
        'prefix' => [
            'type' => 'icon',
            'icon_style' => 'material',
            'icon_name' => 'pie_chart',
        ],
        'action' => [
            'type' => 'link',
            'link_href' => "index.php?action=SampleVte",
        ],
    ],
];
```



Estensione di un modulo

Ecco un esempio di come estendere le funzionalità di un modulo.

- Modificare "config/sdk.config.php"

```
return [
    'sdk_module' => [
        'Contacts' => 'modules/ContactsModule.php',
    ]
];
```

- Implementare la classe ContactsModule in "sdk/modules/ContactsModule.php". In questo esempio viene attivato il blocco per inserire i commenti e viene cambiato il layout passando ad una visualizzazione a due colonne.

```
<?php

class ContactsModule extends \app\PortalModule {

    public $hasComments = true;

    public $formColumns = 2;

    public function canAddComments() {
        return true;
    }

}
```