

SDK 2

- [Include custom php/js/css files](#)
- [Javascript overrides and extensions](#)
- [Standard PHP replacement](#)
- [Inclusion of other files](#)
- [Custom Uitypes](#)
- [Smarty Custom Templates](#)
- [Popup](#)
- [Presave](#)
- [Advanced query](#)
- [Page Header](#)
- [Translations](#)
- [Fields Visibility](#)
- [Home Blocks](#)
- [Custom Buttons](#)
- [Transitions Manager](#)
- [PdfMaker Custom Functions](#)
- [Custom Folders and Reports](#)
- [Turbolift Counter](#)
- [Process](#)
 - [Process log](#)
 - [Import Processes with a script](#)
 - [SDK](#)
- [Portal v2](#)

Include custom php/js/css files

In order to include your custom code (that will be included in every page) you need to register the new file with this call:

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

\$src : the php file path and the file name to be included

To remove the customization:

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

\$src : the php file path and the file name to be removed (the file will NOT be removed from disk)

In case of css/js file you can call:

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

\$id : module id that register the customization (in this case, SDK, 31)

\$type : can be "HEADERCSS" or "HEADERScript"

\$file : the php file path and the file name to be included

Hooks:

include/Webservices/Utils.php

include/squirrelmail/src/redirect.php

install/PopulateSeedData.php

index.php

Javascript overrides and extensions

Some commonly used Javascript functions can be replaced or extended to change their behavior. To do this, simply create a function that has the same name as the function to be modified with the addition of "_override" or "_extension" and the same parameters. The behavior of the two extensions is as follows:

FUNCTION_override()	If present, this function is called instead of the original one. The return value of this function is the return value.
FUNCTION_extension()	If present, this function is called and if it returns false or a value equivalent to false, the original function ends by returning false, while if it returns true or an equivalent value, the execution continues in the original function.

The difference is that, in the first case the original function is completely ignored, while in the second, it is possible to decide whether to continue the standard execution or not. This is very convenient in the case of validation functions, usually very long, in which you simply want to add a control, without copying the entire function for small changes.

The functions that support these extensions are as follows:

File	Functions
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
-------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Standard PHP replacement

You can replace the standard php files of the modules, such as DetailView.php, EditView.php and so on through the method:

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

\$module : the name of the module

\$file : the value of the "action" parameter to be compared

\$newfile: the new php source, without extension and without path

The *\$newfile* must be in the same folder of the module and must be specified without an extension and without a path.

If you want to replace the ListView, you must call setFile twice, once with \$file = "ListView" and once with \$file = "index".

To remove the customization:

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

\$module : the name of the module

\$file : the value of the "action"

Hooks:

include/Ajax/CommonAjax.php
index.php

Inclusion of other files

To associate files or folders to a module, so that they are imported automatically, the following methods are available:

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

\$module : the name of the module

\$src : the path of the file or folder to be associated

To delete the association (but not the files themselves) use:

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

\$module : the name of the module

\$src : the path of the file or folder associated

Custom Uitypes

You can add new types to the existing ones and manage them completely without changing other code. The procedure for creating a new one is:

1. Create a new custom field with the new type (nnn)
2. Create the files:
 - a. nnn.php in modules/SDK/examples
 - b. nnn.js in modules/SDK/examples
 - c. nnn.tpl in Smarty/templates/modules/SDK/examplesThese files manage the behavior of the new field depending on the context (list, detail, and so on)
3. Register the new type with the class method SDK::setUitype.

In modules/SDK/examples/VTE-SDK-2.php there are various examples of field creation and uitype registration.

In modules/SDK/doc/VTE-SDK-2.pdf under **Uitypes List** you can find the list of the main standard uitypes.

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

\$uitype : the number of the new type; it must be nnn (the name of the files)

\$src_php: the path of nnn.php

\$src_tpl: the path of nnn.tpl (without Smarty/templates/ at the beginning)

\$src_js : the path of nnn.js

\$type : the type in webservice format ('text', 'boolean', and so on)

\$params : not used yet

To remove the uitype:

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

\$uitype : it is the number of the uitype to remove (the files associated with it will not be deleted)

We recommend using uitype with a value greater than 2000, to avoid conflicts with the future releases of **vtenext**.

The php script has several variables available, the first one is:

\$sdk_mode : the views that can be customized for the new uitype ("insert", "detail", "edit", "relatedlist", "list", "pdfmaker", "report", and so on)

Depending on the type of `$sdk_mode`, various variables can be read and modified.

detail

to manage the display of the field in DetailView

INPUT

\$module : the current module name

\$fieldlabel: the label of the field

\$fieldname : the name of the field

\$col_fields: (array) the values of the fields

OUTPUT

\$label_fld[] : the label translated

\$label_fld[] : the value to be displayed

edit

to manage the display of the field in EditView

INPUT

\$module_name : the current module name

\$fieldlabel : the label of the field

\$value : the value of the field

OUTPUT

\$editview_label[] : the label translated

\$fieldvalue[] : the value to be displayed

relatedlist, list, pdfmaker

to manage the display of the field in ListView, RelatedList and PDFMaker

INPUT

\$sdk_value : the value of the field

OUTPUT

\$value : the value to be displayed

report

to manage the display of the field in Report

INPUT

\$sdk_value : the value of the field

OUTPUT

\$fieldvalue : the value to be displayed

If the value to be displayed from the interface **is formatted differently** than the value saved in the database (e.g. number 1.000,25 which must be saved as 1000.25) then the following methods must also be managed to save the value in the correct format and search for it.

insert

to convert the value to the format to be saved in the database

INPUT

\$this->column_fields : (array) the values of the fields

\$fieldname : the name of the field

OUTPUT

\$fldvalue : the value to save in the database

formatvalue

to convert the value coming from the \$_REQUEST into the format saved in the database (used in the new management of Conditional Fields)

INPUT

\$value : the value of the field

OUTPUT

\$value : the value converted to database format

querygeneratorsearch

to convert the value searched in the lists and filters by the user

INPUT AND OUTPUT

\$fieldname : the name of the field

\$operator : the comparison operator

\$value : the value searched

customviewsearch

to manage the conversion of the value in the popup filters for the reference fields

INPUT AND OUTPUT

\$tablename : the table of the field

\$fieldname : the name of the field

\$comparator : the comparison operator

\$value : the value searched

popupbasicsearch

to manage the conversion of the value in the popup search for the reference fields

INPUT

\$table_name : the table name

\$column_name : the column name

\$search_string : the value searched

OUTPUT

\$where : the condition of the query

e.g. *\$where = "\$table_name.\$column_name = '".convertToDBFunction(\$search_string)."'";*

popupadvancedsearch

to manage the conversion of the value in the advanced popup search for the reference fields

INPUT AND OUTPUT

\$tab_col : the table and the column of the field

\$srch_cond : the comparison operator

\$srch_val : the value searched

reportsearch

to convert the value searched in the reports by the user

INPUT AND OUTPUT

\$table : the table of the field

\$column : the column of the field

\$fieldname : the field name

\$comparator : the comparison operator

\$value : the value searched

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

Smarty Custom Templates

You can create your own templates, which replace the standard ones (such as EditView.tpl and so on). The new template is used if `$_REQUEST` values of the page meet the requirements. The registration of a new template is done through the method:

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

\$params : associative array with requirements (see below)
\$src : path of the new template

In the `$params` variable you can specify a special value ("`$NOTNULL$`") to indicate that this parameter must exist, with any value. The unspecified parameters are ignored. If the rule to be inserted already exists or is not compatible with the existing ones (it could cause ambiguity for some `$_REQUEST`), the insertion fails (and an explanatory message is saved in the log).

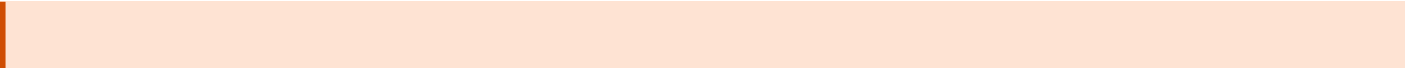
To remove the customization:

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

\$params : associative array with requirements
\$src : path of the template (if NULL, it includes all file)

To completely replace all types of views of a module you need at least 7 rules:

\$params	Notes
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 (with record in <code>\$_REQUEST</code>)
array('module'=>'Leads', 'action'=>'EditView')	New record
array('module'=>'Leads', 'action'=>'EditView', 'record'=>'\$NOTNULL\$', "isDuplicate"=>"true")	Duplicate record
array('module'=>'Leads', 'action'=>'LeadsAjax', 'record'=>'\$NOTNULL\$', 'ajaxaction'=>'LOADRELATEDLIST', 'header'=>'Products')	The related list of leads showing the products



If **multiple rules match**, the most specific will be used. For example, if there are 2 rules, one with "\$NOTNULL\$" and one with the value "Leads" and the request is "Leads", the second rule will be used.

Hooks

Smarty_setup.php

Popup

Two actions are available for managing popup. You can insert a php script before the query is made to load the data. In addition, it is possible to insert another php script before the data is shown, so you can edit this data or the result when the popup is closed. In the first case, there are two methods available:

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

\$type : "field" or "related" to indicate a standard field or popup opened from a related list

\$module: the module in which the popup opens

\$param : the name of the field that opens the popup (must be uitype 10) in the case type = "field", otherwise the name of the connected module

\$src : the path of php file

\$hidden_rel_fields : associative array of fields to pass in the popup request like array(\$urlvalue => \$jscode)

To remove the customization:

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

Same parameters as before

The following variables are available within the php script:

\$query : the query that takes the values to show

\$sdk_show_all_button : if true it shows the button to cancel the SDK restrictions and show all records

In the second case there are the following methods:

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

\$module : the module containing the field that opens the popup

\$fieldname : the name of the field that opens the popup (only uitype 10)

\$src : the php file

To remove the customization:

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

For now the only fields supported are those with uitype 10.

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 PopupQuery and a PopupReturnFunction to a field in a table field
SDK::setPopupQuery('field', 'Accounts', 'ml1_f4', 'modules/SDK/examples/Contacts/AccountQuery.php');
SDK::setPopupQuery('field', 'Processes', 'vcf_2_vcf_35', 'modules/SDK/examples/Contacts/AccountQuery.php');
```

Presave

You can enter your own script when you press the "Save" button in EditView mode. To register a script use the method:

```
SDK::setPreSave($module, $src);
```

\$module : the name of the module

\$src : the path of php file

To remove the customization:

```
SDK::unsetPreSave($module, $src = NULL);
```

\$module : the name of the module

\$src : the path of php file (if NULL, includes all scripts registered for that module)

The following variables are available within the script:

\$type : the form name ("MassEditSave", "DetailView", "EditView", "createTODO", "QcEditView", "ConvertLead", "createQuickTODO", "Kanban")

\$values : (array) new values

For MassEditSave, you can know record that are going to be managed calling
`getListViewCheck($currentModule);`

The following variables can be set:

\$status : (bool) whether or not the submit should be saved

\$message: (string) if not empty a popup with the message is shown

\$confirm: (bool) if true, a Javascript popup is shown asking for confirmation to continue, showing \$message. In this case, \$status must not be set.

\$focus : (string) in case of error, the element that takes the focus (only if \$status = false)

\$changes: (array) values to assign to the fields (only if \$status = false)

The \$focus and \$changes variables are only available when \$status is false and \$type is one of '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

You can modify the query executed to load the data in ListView, RelatedList and Popup mode in order to limit or extend the visibility of the data. This does not affect Administrator users, who have access to all data. In addition, the module must be set as Private.

Editing the query is done through a custom php function (see below). Only one function of this type can be used in each module. To register the function use:

```
SDK::setAdvancedQuery($module, $func, $src);
```

\$module : the module in which to apply the function (if a function is already registered for the module, the new one is not inserted)

\$func : the name of the php function

\$src : the php file that contains the function

To remove the customization:

```
SDK::unsetAdvancedQuery($module);
```

\$module : the name of the module

The \$func function must be defined as follows:

```
<?php

function myFunction($module) {
    // Your code ...
}
```

\$module : the module that calls the function

It returns a string:

"" : (empty string) the query is unchanged

? : (not empty string) this string is added to the query

Hooks

data/CRMEntity.php

Page Header

You can customize the user icon, the settings icon or the blue bars at the top of the pages of VTE to incorporate new features. To do this, simply extend the method `setCustomVars` of class `VTEPageHeader` as follows:

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

The file `NewPageHeader.php` will have this content:

```
<?php
require_once('include/utls/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);
    []}
}
```

Translations

Translations can be customized for each language and module installed. To modify or insert a new translation use the method:

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

\$module : the module name that contains the string

\$langid : the code of the language (e.g. "en_us", "it_it")

\$label : the label (e.g. LBL_TASK_TITLE)

\$newlabel : the translation of the label

If the label already exists for the chosen module and language, it will be replaced. As a module you can specify "APP_STRINGS" to insert a global translation or "ALERT_ARR" to make the translation available in JavaScript files. To load a string simultaneously in multiple languages, the method is:

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

\$module : the name of the module

\$label : the label

\$strings : associative array with the translations (e.g. array("it_it"=>str1, ...))

To remove a translation:

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

\$module : the name of the module

\$langid : the code of the language

\$label : the label (if NULL, all the strings that match)

Fields Visibility

You can change the visibility of the various fields (value of `$readonly`) and other variables in the different modes (ListView, EditView, and so on) via SDK. To register a new "view" use the method:

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

`$module` : the name of the module to apply the view

`$src` : the path of php file

`$mode` : how the rule is applied (see below)

`$success`: what to do after applying the rule (see below)

The views defined for each module are applied in the order in which they are registered. When they register, they are added to the queue of views for that module. The `$mode` variable only makes sense if you change the `$readonly` variable and it admits the following values:

"constrain" : forces the value of `$readonly` to take the new value given in the script

"restrict" : changes the value of `$readonly` only for a more restrictive value (from 1 to 99 or 100, from 99 to 100, not vice versa)

The `$success` variable instead can be:

"continue" : after applying the view, continue with the next

"stop" : if the view returns `$success = true`, no other rules are executed

The following variables are available within the scripts:

`$sdk_mode` : one of "" (create), "edit", "detail", "popup_query", "list_related_query", "popup", "related", "list", "mass_edit"

`$readonly` : the readonly value for the current field (1, 99, 100)

`$col_fields`: values of fields, only for `$sdk_mode` = "edit", "detail" e ""

`$fieldname` or `$fieldName`: the name of the current field

`$current_user`: the current user

And you can set the `$success` variable with true or false.

Depending on the mode (ListView, EditView, and so on) there are different ways to edit the queries and different variables available.

Mode	Value of <code>\$sdk_mode</code>	Available variables	Notes
CreateView	""	<code>\$col_fields</code> <code>\$current_user</code>	1

EditView	"edit"		
DetailView	"detail"		
MassEdit	"mass_edit"		
PopupQuery	"popup_query"	<code>\$sdk_columns</code> <code>\$success</code>	2
List/RelatedQuery	"list_related_query"	<code>\$sdk_columns</code> <code>\$success</code>	
Popup	"popup"	<code>\$current_user</code> <code>\$fieldname</code>	3
Related	"related"	<code>\$sdk_columnnames</code> <code>\$sdk_columnvalues</code> <code>\$readonly</code>	4
List	"list"	<code>\$success</code>	

Notes:

1. In these modes the values of the fields are in `$col_fields[nameOfTheField]`
2. These modes are used to modify the query so that additional fields can be picked up. In `$sdk_columns` variable there are the database columns to add to the query. Include then the php file "modules/SDK/AddColumnsToQueryView.php"
3. To get values from other fields, specified in the PopupQuery and ListRelatedQuery modes, write them in the `$sdk_columnnames` variable, include the php file "modules/SDK/GetFieldsFromQueryView.php" and then take them from `$sdk_columnvalues`
4. In the case of the related "Activity history", only the variables `$recordId` and `$readonly` are available and apply to the entire row, not to the single field.

To remove the view:

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

\$module : the name of the module

\$src : the path of php file

Hooks

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


Home Blocks

New blocks can be added to the home of VTE via SDK. The blocks cannot be deleted from the interface. The method for creating a new block is:

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

\$size : the horizontal size of the block (from 1 to 4)

\$url : the address to be shown within the block. It can also have a protocol at the beginning (e.g. <http://www.mysite.com/file>)

\$title : the label of the block (it can be translated via API)

\$userid : array containing the ids of the users who can see the block. If you leave null, the block is visible to all users

\$useframe: if true the content will be inside an <iframe> otherwise the file is included directly

Users created later will see all previously registered blocks.

Block cancellation is possible via 2 methods:

```
SDK::unsetHomelframe($stuffid);
```

\$stuffid : the id of the block

```
SDK::unsetHomelframeByUrl($url);
```

\$url : the url of the block

Blocks are removed for all users.

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

Custom Buttons

Buttons can be added under the main menu. To insert a new button use the following method:

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

\$type : the type of button, it can be 'fixed' or 'contextual'; in the first case the button appears on the left and is always visible, in the second case the button is inserted on the right and will be visible only in the chosen module and for the chosen action.

\$title : the label of the button

\$onclick : the javascript code to execute. It is NOT possible to use double quotes! (")

\$image : the button image. It must be specified without path and reside in themes/softed/images folder even in the smallest version (e.g. img.png e img_min.png)

\$module : if type = 'contextual', the module in which the button is visible

\$action : if type = 'contextual', the action (request action) in which the button is visible

\$condition : string like FunctionName:PathPhp representing a function (in the PathPhp file) to be called before showing the button. If it returns false, the button is not shown. The function has only one parameter of type reference to an array with the information of the button.

To remove the button use:

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

\$type : the type of the button

\$id : the id of the button

Hooks

Smarty/templates/Buttons_List.tpl

Transitions Manager

You can change the selection options for the picklists managed by the transitions manager, as well as add messages to the "State manager" block to the right of the record detail. To register this functionality use the method:

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

\$module : the name of the module to handle

\$fieldname : the name of the field managed by transition

\$file : the path of the php file that contains the function to call

\$function : the function to call

To remove the customization:

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

The function called by the transitions manager has the following format:

```
<?php

function myFunction($module, $fieldname, $record, $status, $values) {
    // Your code ...
}
```

\$module : the current module

\$fieldname : the name of the field managed by transition

\$record : the current record

\$status : the value of the field managed by transition of the current record

\$values : array of admissible values for the status

It must return null if you do not want to change the transitions manager behavior or an array with the following format:

```
array(
    'values' => array(..) // array of admissible values for the status
    'message' => " // html code to be included under the transitions manager block
);
```

Hooks

modules/Transitions/Transitions.php

modules/Transitions/Statusblock.php

Smarty/templates/modules/Transitions/StatusBlock.tpl

PdfMaker Custom Functions

Custom functions can be added in the PDFMaker module. To insert one use:

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

\$label: the label of the function (it is translated into the PDFMaker module)

\$name: the name of the function

\$params: array with the names of the function parameters

Registered functions must be saved in php files in modules/PDFMaker/functions/ to be used by the PDFMaker module

To remove the function:

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

\$name: the name of the function

Custom Folders and Reports

Custom folders can be created using the following API.

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

\$name : the name of the folder

\$description : the description of the folder

The created folder will be on the reports page. It will be visible to all users and cannot be changed. The name and description can be translated with the translation API.

Folders created via API can be deleted with:

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

\$name : the name of the folder to be deleted

\$delreports : if true it also deletes all reports (created via API) in that folder (files are not deleted)

Inside the folders you can insert customized reports with the following statement:

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

\$name : the name of the report (it can be translated via API)

\$description : the description of the report (it can be translated via API)

\$foldername : the name of the folder created via API where to insert the report

\$reportrun : the path of the php file that contains the class that generates the report

\$class : the name of the class that handles the report

\$jsfunction : the name of the javascript function to run when the "Generate Report" button is pressed

The report created via API can be deleted with:

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

\$name : the name of the report to delete (files are not deleted)

The class specified in \$class must follow the following structure:

```
<?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 && $error_msg!=""){
            // Performance Optimization: If direct output is required
            if($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);
    }

}

// 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;
}
}

```

The javascript function specified in \$jsfunction must already be declared and return a string to be added to the 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;
}

```

When updating a VTE to version 16.09 (or later), some features of the customized reports must be verified.

One of the changed parts is the management of **time filters**, which previously could be modified by extending the `getPrimaryStdFilterHTML` and `getSecondaryStdFilterHTML` methods. To obtain the same result, you need to extend the `getStdFilterFields` function which returns an array of available fields, for example:

```

<?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;
}

```

When the report is generated the variable `$this->stdfilters` contains the time filter to be used. If the report query is completely customized, the generation of the time filter must also be managed manually, for example using a method like this:

```

<?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

Turbolift Counter

When changing the standard extraction criteria of a related list, for example using another method or redefining it by extending the class that contains it, the counter of the number of the linked records visible in the Turbolift (the right column with the list of modules in record detail) may no longer be valid. In this case, therefore, a method that returns the correct count must be defined through the following API.

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

\$relation_id : relation id (table vte_relatedlists)

\$method : the method name that returns the correct count

The method must be implemented in the class that contains the relation (e.g. in the Contact relationship connected to a Company we mean the Accounts class or any extensions to it)

To remove the customization:

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

The defined method can be the related list method (column **name** in vte_relatedlists) or a custom method that returns an integer.

Process

Process log

The way of viewing the logs has changed in the various **vtenext** releases and is summarized as follows.

- **vtenext 16.09**: Logs can be consulted by accessing the filesystem in the folder logs/ProcessEngine
- **vtenext 18.X**: In Settings → Business Process Manager → ProcessManager the button 'LOG' will be available once the following script has been executed:

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

The file 01.log is initially created. When the file exceeds the 5MB, the file 02.log will be created and so on.

- **vtenext 19.10**: Logs are available in Settings → Other Settings → System Logs

Import Processes with a script

From version 18.05 (rev. 1696) it is possible to import processes previously exported in the format vtebpmn (diagram + configuration) or bpmn (diagram only) with php scripts using the importFile method of the ProcessMakerUtils class.

The importFile method takes in the first parameter the path of the file to be installed (.vtebpmn / .bpmn) and in the second one a Boolean value (true/false) depending on whether you want to automatically activate the process or not.

This method is useful for installing processes at the end of the installation of a new module: just include the process file in the installation zip and execute the code in the case 'postinstall' of the method vtlib_handler of the module class.

Example

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

Process

SDK

You can add custom functions to processes, click [here](#) for more details.

Portal v2

Theme documentation

<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: the parameter is optional and is returned by the "portal.login". Use in subsequent calls to optimize portal performance.
- Content-Type: application/json

Name	Parameters	Description
portal.info		Allows you to obtain information on the license and logos used by the connected vtenext installation.
portal.login	Body <ul style="list-style-type: none">. username: string (*). password: string (*). language: string	Allows you to log in to the customer portal by providing the email (username) and password of the contact. The service also returns the accesskey and session id values to be used in subsequent calls as headers (Authorization and Portal-Session-Id).
portal.logout	Headers <ul style="list-style-type: none">. Authorization	Allows you to logout the contact from the portal.
portal.send_mail_for_password	Body <ul style="list-style-type: none">. email: string (*). language: string	Allows you to recover the password of a contact.
portal.modules_list	Headers <ul style="list-style-type: none">. Authorization Body <ul style="list-style-type: none">. language: string (*)	Allows you to obtain a list of modules enabled for the contact's profile.

portal.get_list	<p>Headers</p> <ul style="list-style-type: none"> . Authorization <p>Body</p> <ul style="list-style-type: none"> . module: string (*) . language: string . folderid: int . search: array 	<p>Allows you to obtain the list of records of a module filtered by the contact profile.</p> <p>The "folderid" parameter is used by the Documents module to get the documents of a specific folder.</p> <p>The "search" parameter allows the paging of the list on the server-side.</p> <p>Ex.</p> <pre>['length' => 50, // Number of records to return 'start' => 0, // Paging offset 'search' => "", // Global search 'search_columns' => [// Search by columns ['index' => 0, // Index of the field 'column' => "", // Name of the field 'search' => "", // Value of the field]], 'ordering' => [// Ordering ['index' => 0, // Index of the field 'column' => "", // Name of the field 'dir' => "", // asc/desc]],]</pre> <p>Note:</p> <ol style="list-style-type: none"> 1. All searches are performed ONLY on the columns of the list. 2. Column search is performed in "LIKE" mode. 3. Sorting can only be done on one column.
portal.get_blocks	<p>Headers</p> <ul style="list-style-type: none"> . Authorization <p>Body</p> <ul style="list-style-type: none"> . module: string (*) . language: string . mode: string (edit, create, detail, list) (*) . app_data: array 	<p>Allows you to obtain the list of the blocks and the related fields of a module filtered by the contact profile.</p> <p>The "app_data" parameter represents the record being edited (array with field name and value) and is used by the SDK views to establish the visibility of the fields.</p> <p>This parameter will also be used in the future for managing conditional fields.</p>
portal.get_record	<p>Headers</p> <ul style="list-style-type: none"> . Authorization <p>Body</p> <ul style="list-style-type: none"> . module: string (*) . id: int (*) 	<p>If the contact has visibility permissions for the indicated record, it allows you to obtain its data.</p>

portal.save_record	Headers . Authorization Body . module: string (*) . id: int (*) . values: encoded (*)	If the contact has write permissions for the indicated record, it allows its saving.
portal.delete_record	Headers . Authorization Body . module: string (*) . id: int (*)	If the contact has delete permissions for the indicated record, it allows its deletion.
portal.write_ticket_comment	Headers . Authorization Body . id: int (*) . comment: string (*)	Allows you to write a comment within the indicated ticket.
portal.get_attachments	Headers . Authorization Body . id: int (*)	Allows you to obtain the list of documents of the specified record filtered by the visibility of the contact profile.
portal.download_attachment	Headers . Authorization Body . relid: int (*) . docid: int	If the contact has permission to view the attachment (relid), it allows its download.
portal.upload_attachment	Headers . Authorization Body . relid: int (*) . title: string (*) File	Allows the uploading of an attachment related to the specified record (relid). You can indicate the name (title) of the document that will be generated.
portal.provide_confidential_info	Headers . Authorization Body . id: int (*) . comments: string . data: string (*) . request_commentid: int (*)	Allows you to respond to a request for confidential information. The "id" parameter indicates the ticket id, the "comments" parameter indicates the unencrypted comment, the "data" parameter indicates the confidential response, and the "request_commentid" parameter indicates the id of the comment to which the confidential response should be provided .
portal.get_home_widgets	Headers . Authorization Body . language: string	Allows you to obtain the widgets configured in the profile associated with the contact.
portal.save_authenticate_cookie	Headers . Authorization Body . contactid: int (*)	Provides an hash to be stored in a cookie to remember the contact's login.

portal.check_authenticate_cookie	Headers . Authorization Body . contactid: int (*) . hash: string (*)	Allows you to verify the hash used to remember the contact's login.
portal.change_password	Headers . Authorization Body . username: string (*) . old_password: string (*) . password: string (*) . language: string	Allows you to change the contact's password.

Register a new REST webservice

Create a new file and execute it (e.g. 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_Utills_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);
```

Create a new file that contains the CustomerPortalRestApi2 class (e.g. 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;
}

}

```

Extend an existing REST webservice

Create a new file and execute it (e.g. 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_Utills_Log = true;
global $adb, $table_prefix;
VteSession::start();

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

```

Create a new file that contains the CustomerPortalRestApi2 class (e.g. 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;
    }

}

```

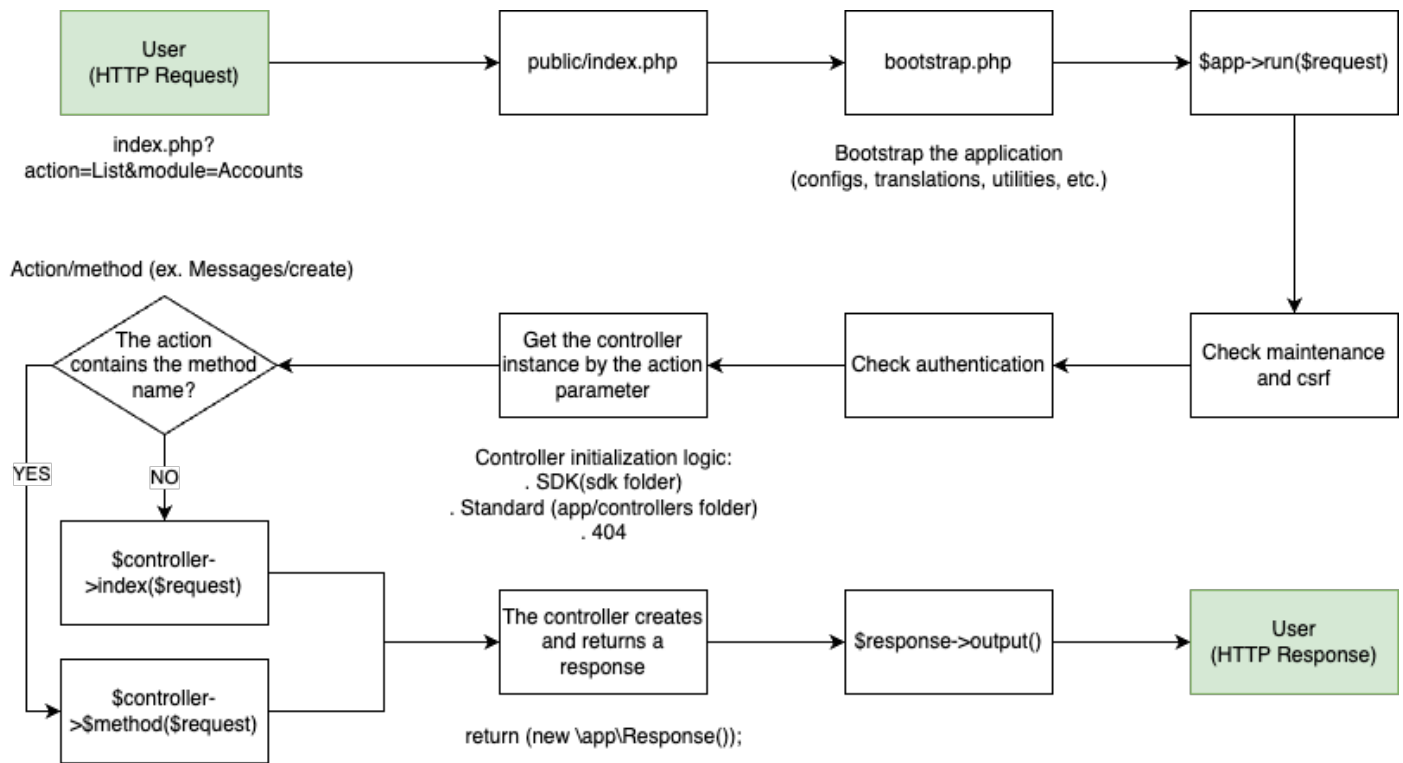
Structure of main folders/files

The portal folder is located at: **VTE_ROOT/portal/v2**

The main folders/files of the new portal are:

Folder/file name	Description
app	Contains the files and the logic of the portal.
app/controllers	Contains the files that manage the portal's default actions (e.g. Login, Logout, Edit, Detail, etc.).
app/fields	Contains the files that manage the portal module fields.
app/modules	Contains custom logic of some vtenext modules (e.g. Documents, Processes, HelpDesk, etc.).
app/PortalModule.php	Class that manages the actions of the portal modules (e.g. Create, Detail, Edit, etc.). The class can be extended to change the default behaviors of a module.
config	Contains the portal configuration files.
public	Contains the portal's public files (e.g. index.php, css, javascript, images, etc.).
resources	Contains the resources used by the portal such as translation files (lang) and templates (templates).
sdk	The folder is used to insert new customizations for the customer.
storage	Contains temporary files (e.g. cache, logs, etc.).
vendor	Contains the external libraries used by the portal.

Request lifecycle



Portal configuration

The portal configuration file is located in "config/portal.config.php".

To **overwrite the parameters** the "config/sdk.config.php" file must be used otherwise, in case of updating the version of vtenext, the changes could be lost.

How to move the Business Portal to another host/folder

Update the "portal_url", "vte_url" and "csrf_secret" parameters in the "config/sdk.config.php" file.

Also change the "default_timezone" parameter if the host has a different timezone from the one in which the vte is located. Change the "portal.url" prop to the link pointing to the v2 folder. e.g. <https://ticket.vtenext.com/v2>

Here is the list of parameters supported by the new portal:

Parameter	Type	Default	Description
-----------	------	---------	-------------

portal_url	String	\$PORTAL_URL (config.inc.php)	Indicates the URL of the customer portal. If the portal folder is inside the vtenext root directory, the variable will be set to the value of the \$PORTAL_URL variable set in the config.inc.php file.
vte_url	String	\$site_URL (config.inc.php)	Indicates the URL of vtenext and is used to obtain its data via the rest API. If the portal folder is inside the vtenext root directory, the variable will be set to the value of the \$site_URL variable set in the config.inc.php file.
default_language	String	it_it	Indicates the default language used in the customer portal. The value can be replaced with a supported language (see "languages" parameter).
languages	Array	['en_us' => 'US English', 'it_it' => 'IT Italiano']	Indicates the languages supported in the customer portal. To add a new language you need to create a new file in the resources/lang folder.
production	Bool	false	This configuration indicates whether errors should be displayed or not. If the environment is production, the value will be set to true to disable error display. If the environment is development, the value will be set to false to allow errors to appear.
default_module	String		Indicates the default module to load after logging into the portal. The value can be replaced with the name of the desired module (must be enabled from profile). If the parameter is empty, the portal home will be loaded.
favicon	String	assets/img/VTENEXT_favicon.ico	Indicates the path of the favicon. The value is relative to the public folder.

login_logo	String	assets/img/VTENEXT_login.png	Indicates the path of the logo to load on the login page. The value is relative to the public folder. [UPDATE] The logo must be loaded in vtenext settings > Logos.
login_background	String		Indicates the path of the background to load on the login page. The value is relative to the public folder.
header_logo_sm	String	assets/img/VTENEXT_toggle.png	Indicates the path of the icon to load in the minimized sidebar. The value is relative to the public folder. [UPDATE] The icon must be loaded in vtenext settings > Logos.
header_logo_lg	String	assets/img/VTENEXT_header.png	Indicates the path of the icon to load in the expanded sidebar. The value is relative to the public folder. [UPDATE] The icon must be loaded in vtenext settings > Logos.
helpdesk_logo	String	assets/img/helpdesk.png	Indicates the logo used to display the response given by customer support. The value is relative to the public folder.
sidebar_theme	String	sidebar-dark-primary	Indicates the class of the main sidebar. It can have a "dark" or "light" brightness. It can also have a color variant, such as "primary", "success", "warning", "info", "danger".
enable_sidebar_search	Bool	false	Enable/disable the search bar in the main sidebar.
csrf_secret	String	\$csrf_secret (config.inc.php)	Indicates the secret key used to generate a csrf token. If the portal folder is inside the vtenext root directory, the variable will be set to the value of the \$csrf_secret variable set in the config.inc.php file.

upload_dir	String		Indicates the name of the folder used for uploading files.
browser_title_prefix	String		Indicates the prefix label to use for the browser title. The value can be replaced with the desired label.
browser_title_suffix	String	customer_portal	Indicates the suffix label to use for the browser title. The value can be replaced with the desired label.
remember_cookie_name	String	portal_login_hash	Indicates the name of the cookie used to remember user authentication.
login_expire_time	Int	2592000 (one month)	Indicates the expiration of the cookie used to remember user authentication. The value can be replaced with the desired number of seconds.
default_timezone	String	Europe/Rome	Indicates the default time zone used in the customer portal. This configuration must be the same as the \$default_timezone variable set in the vtenext config.inc.php file.
module_icons	Array	[]	With this configuration you can override the default icons used for modules enabled in the customer portal. The default form icons are found in app/layouts/PortalLayout.php. The names of the icons can be found here https://fonts.google.com/icons .
sdk_languages	Array	[]	With this configuration you can add new labels or modify existing ones. You need to create a new file in the sdk folder.
sdk_global_php	Array	[]	With this configuration you can add php files to load on each page (they should contain classes/functions). You need to create a new file in the sdk folder.

sdk_global_js	Array	[]	With this configuration you can add js to load globally. You need to create a new file in the public/assets/sdk folder.
sdk_module_js	Array	[]	With this configuration you can add js to load for a specific module. You need to create a new file in the public/assets/sdk folder.
sdk_global_css	Array	[]	With this configuration you can add css to load globally. You need to create a new file in the public/assets/sdk folder.
sdk_controllers	Array	[]	With this configuration you can add custom actions ("action" field in the URL). The array represents an association between the action name and the file that contains the controller class to handle the request. You need to create a new file in the sdk folder.
sdk_module	Array	[]	With this configuration you can add customizations on a specific module. The array represents an association between the module name and the file that contains the module's extended class. You need to create a new file in the sdk folder.
sdk_menu	Array	[]	With this configuration you can add custom menu items in the sidebar. You need to create a new file in the sdk folder.

API Reference - Main classes, methods, functions and variables

\app\Request

Method	Arguments	Description
get()	\$keys = null, \$purify = false	<p>Allows you to get one or more parameters from the global variable \$_GET. If the second argument is set to "true", the parameters are purified through the HTML Purifier library. Example:</p> <pre>\$request->get('foo'); \$request->get(['foo', 'bar']); \$request->get('foo', true);</pre>
post()	\$keys = null, \$purify = false	<p>Allows you to get one or more parameters from the global variable \$_POST. If the second argument is set to "true", the parameters are purified through the HTML Purifier library. Example:</p> <pre>\$request->post('foo'); \$request->post(['foo', 'bar']); \$request->post('foo', true);</pre>
files()	\$key	<p>Allows you to get files uploaded via the HTTP POST method and organized via the global variable \$_FILES. Example:</p> <pre>\$request->files('attachments');</pre>
cookie()	\$keys = null, \$purify = false	<p>Allows you to obtain one or more parameters from the global variable \$_COOKIE. If the second argument is set to "true", the parameters are purified through the HTML Purifier library. Example:</p> <pre>\$request->cookie('foo');</pre>
server()	\$keys = null, \$purify = false	<p>Allows you to get one or more parameters from the global variable \$_SERVER. If the second argument is set to "true", the parameters are purified through the HTML Purifier library. Example:</p> <pre>\$request->server("");</pre>
isGet()		Returns true if the request method is GET.

isPost()		Returns true if the request method is POST.
isAjax()		Returns true if the request is AJAX.
purify()	\$input	Purify the \$input variable through the HTML Purifier library.

\app\Response

Method	Arguments	Description
__construct()	\$content = "", \$statusCode = 200, \$headers = []	Initializes a new \app\Response() object with the response content (\$content), return code (\$statusCode), and default headers (\$headers).
setContent()	\$content	Set the content of the response.
setStatusCode()	\$statusCode	Set the response return code.
setHeader()	\$header, \$replace = true	Set a new header in the response. With the second argument it is possible to indicate whether or not the header must replace a previous header already set.
setMimeType()	\$mimeType = 'text/html'	Sets the response content mime.
json()	\$data	Set response content with \$data converted to JSON format and 'application/json' mime.
redirect()	\$page	Performs a redirect to \$page.
downloadFile()	\$fullpath	Allows downloading of a file located in \$fullpath.
output()		Outputs the content, response code, and headers you set.

\app\Session

Method	Arguments	Description
set()	\$key, \$value = ""	Set the value \$value with key \$key in \$_SESSION.
get()	\$key	Allows you to get the value with key \$key from \$_SESSION.

flash()	\$key	Allows you to get the value with key \$key from \$_SESSION. Next, the \$key will be deleted from \$_SESSION.
remove()	\$key	Delete the \$key from \$_SESSION.
hasKey()	\$key	Returns true if \$key exists in \$_SESSION.
setMulti()	\$keys	Allows you to write multiple values to \$_SESSION.
removeMulti()	\$keys	Delete multiple values in \$_SESSION.
append()	\$key, \$value = ''	Set the \$key as an array in \$_SESSION and the value \$value is added to it.

\app\Config

Method	Arguments	Description
has()	\$key	Returns true if \$key exists in the global configuration.
get()	\$key	Allows you to get the value with key \$key from the global configuration.
set()	\$key, \$value	Set the value \$value with key \$key in the global configuration.
getAll()		It allows you to obtain a key-value list with all the global configuration.
setMulti()	\$values	Allows you to write multiple values into the global configuration.
clear()	\$key	Delete the \$key from the global configuration.
clearAll()		Delete all values from the global configuration.

\app\PortalModule

Variable	Default	Description
\$hasComments	false	Indicates whether the module supports comments.
\$hasAttachments	false	Indicates whether the module supports attachments.
\$enableEdit	true	Indicates whether the module can be modified (edit mode).

\$formColumns	3	Indicates the number of columns to use for displaying fields in Create, Edit and Detail.
\$listTemplate	List.tpl	Indicates the template used for displaying a list (action List).
\$referenceListTemplate	sections/ReferenceList.tpl	Indicates the template used for displaying a related list (uitype 10).
\$detailTemplate	Detail.tpl	Indicates the template used to display the detail of a record (action Detail).
\$editTemplate	Edit.tpl	Indicates the template used to display the edit of a record (action Edit).
\$notAuthorizedTemplate	PageNotAuthorized.tpl	Indicates the template used to display a permission error (e.g. record not found or permission errors).

Method	Arguments	Description
__construct()	\$module	\$module indicates the name of the module to obtain an instance of the class. If the module has been extended via SDK then an instance of the extended class will be returned.
prepareList()	\$viewer, \$request	Method used for displaying a list (action List).
postProcessList()	\$viewer, \$request	This method can be used by extended classes to insert/modify data set in prepareList().
prepareEdit()	\$viewer, \$request	Method used to display the edit of a record (action Edit).
postProcessEdit()	\$viewer, \$request	This method can be used by extended classes to insert/modify the data set in prepareEdit().
prepareDetail()	\$viewer, \$request	Method used to display the detail of a record (action Detail).
postProcessDetail()	\$viewer, \$request	This method can be used by extended classes to insert/modify data set in prepareDetail().
saveRecord()	\$request	Method used to save a record (action Save).
postProcessSaveValues()	\$request, &\$values	This method can be used by extended classes to insert/modify data set in saveRecord().

isPermitted()	\$module, \$action = self::ACTION_LIST, \$recordValues = []	Returns if the contact has permission to perform a certain action. List of supported actions: . 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

Method	Arguments	Description
get()	\$restName, \$queryParameters = [], \$headers = []	Allows you to perform a GET request to vtenext.
post()	\$restName, \$queryParameters = [], \$bodyParameters = [], \$headers = []	Allows you to perform a POST request to vtenext.
postMultipart()	\$restName, \$queryParameters = [], \$bodyParameters = [], \$fileParameters = [], \$headers = []	Allows you to perform a multipart POST request to vtenext.
patch()	\$restName, \$queryParameters = [], \$bodyParameters = [], \$headers = []	Allows you to execute a PATCH request to vtenext.
delete()	\$restName, \$queryParameters = [], \$headers = []	Allows you to execute a DELETE request to vtenext.
postDownload()	\$restName, \$queryParameters = [], \$bodyParameters = [], \$headers = []	Allows you to perform a POST request to download a vtenext file in stream mode.

Helpers

Function	Arguments	Description
preprint()	\$var	print_r formatted with <pre> tag
predump()	\$var	var_dump formatted with <pre> tags
encodeForHtml()	\$value, \$charset = 'UTF-8'	Encode the \$value value to be inserted into an HTML page.
encodeForHtmlAttr()	\$value, \$enclosing = ''	Encode the \$value value to be placed inside an HTML tag attribute.

encodeForJs()	\$value, \$enclosing = ''	Encode the \$value value to be placed inside a javascript <script>.
htmlAttr()	\$attributes	Encode a list of attributes to be placed inside an HTML tag.
config()	\$key	Allows you to get the value with key \$key from the global configuration.
trans()	\$key, \$args = []	Translate the label \$key.
listUrl()	\$module, \$extraParams = []	Generate a link to open a list.
createUrl()	\$module, \$extraParams = []	Generates a link to open the creation of a record.
createDocUrl()	\$module, \$folderId, \$extraParams = []	Generates a link to open the creation of a document.
detailUrl()	\$module, \$record, \$extraParams = []	Generates a link to open the detail of a record.
editUrl()	\$module, \$record, \$extraParams = []	Generates a link to open the editing of a record.
downloadUrl()	\$record, \$documentId, \$extraParams = []	Generates a link to download a document.
docFolderUrl()	\$module, \$folderId, \$extraParams = []	Generates a link to open a specific folder in the documents module.
returnUrl()	\$request	Generate a return link (e.g. "Cancel" action).
portalLanguage()		Returns the language used in the portal.
setPortalLanguage()	\$language	Set the \$language in the portal.
getBrowserTitle()	\$title	Returns the title to be set in an HTML page of the portal.
getModuleLabel()	\$module	Returns the translation of the module \$module.
getSingleModuleLabel()	\$module	Returns the singular translation of the module \$module.
getMaxUploadSize()		Returns the maximum upload size in the portal.
setPortalCookie()	\$name, \$value = '', \$expires_or_options, \$httponly = false	Set a cookie in the portal.
unsetPortalCookie()	\$name, \$httponly = false	Removes a cookie from the portal.

csrfToken()		Returns the csrf token.
csrfInputName()		Returns the name of the input for sending the csrf token.
flashPortalError()	\$error	Allows you to display a timed error message.
flashPortalMessage()	\$message	Allows you to display a timed message.
customerId()		Returns the ID of the authenticated contact.
customerEmail()		Returns the email of the authenticated contact.
customerUsername()		Returns the name and surname of the authenticated contact.
resourcever()	\$filename	Allows the versioning of css and javascript files.
basePath()	\$path = ""	Returns the path to the portal base folder. If \$path is specified, a path is created and returned starting from the base folder.
appPath()	\$path = ""	Returns the path to the portal's "app" folder. If \$path is specified, a path is created and returned starting from the "app" folder.
configPath()	\$path = ""	Returns the path to the portal's "config" folder. If \$path is specified, a path is created and returned starting from the "config" folder.
resourcesPath()	\$path = ""	Returns the path to the portal's "resources" folder. If \$path is specified, a path is created and returned starting from the "resources" folder.
langPath()	only \$	Returns the path to the indicated language file \$lang.
storagePath()	\$path = ""	Returns the path to the portal's "storage" folder. If \$path is specified, a path is created and returned starting from the "storage" folder.
publicPath()	\$path = ""	Returns the path to the portal's "public" folder. If \$path is specified, a path is created and returned starting from the "public" folder.

vtePath()	\$path = ""	Returns the path to the vtenext folder. If \$path is indicated, a path is created and returned starting from the vtenext folder.
sdkPath()	\$path = ""	Returns the path to the portal's "sdk" folder. If \$path is indicated, a path is created and returned starting from the "sdk" folder.
sdkAssetsPath()	\$path = ""	Returns the path to the portal's "public/assets/sdk" folder. If \$path is specified, a path is created and returned starting from the "public/assets/sdk" folder.
publicRelPath()	\$assetPath	Returns a relative path starting from the "public" folder.

Examples

SDK view

Here is an example of how to change the visibility of portal fields via view SDK.

```
<?php
require('.././config.inc.php');
chdir($root_directory);
require_once('include/utils/utils.php');
require_once('vtlib/Vtecrm/Module.php');
$Vtiger_Utills_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;
    }

```

Creating a new controller

Here is an example of how to create a new controller to manage a custom action.

- Edit "config/sdk.config.php" to insert a new controller

```

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

```

- Implement the SampleVteController class 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);
}
}
}

```

- Create a new template in "resources/templates/sdk/SampleVte.tpl"

```

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

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

```

- Modify "config/sdk.config.php" indicating the sdk file for inserting the new entries in the side menu

```

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

```

- Edit the "sdk/samplevte-menu.php" file

```

<?php

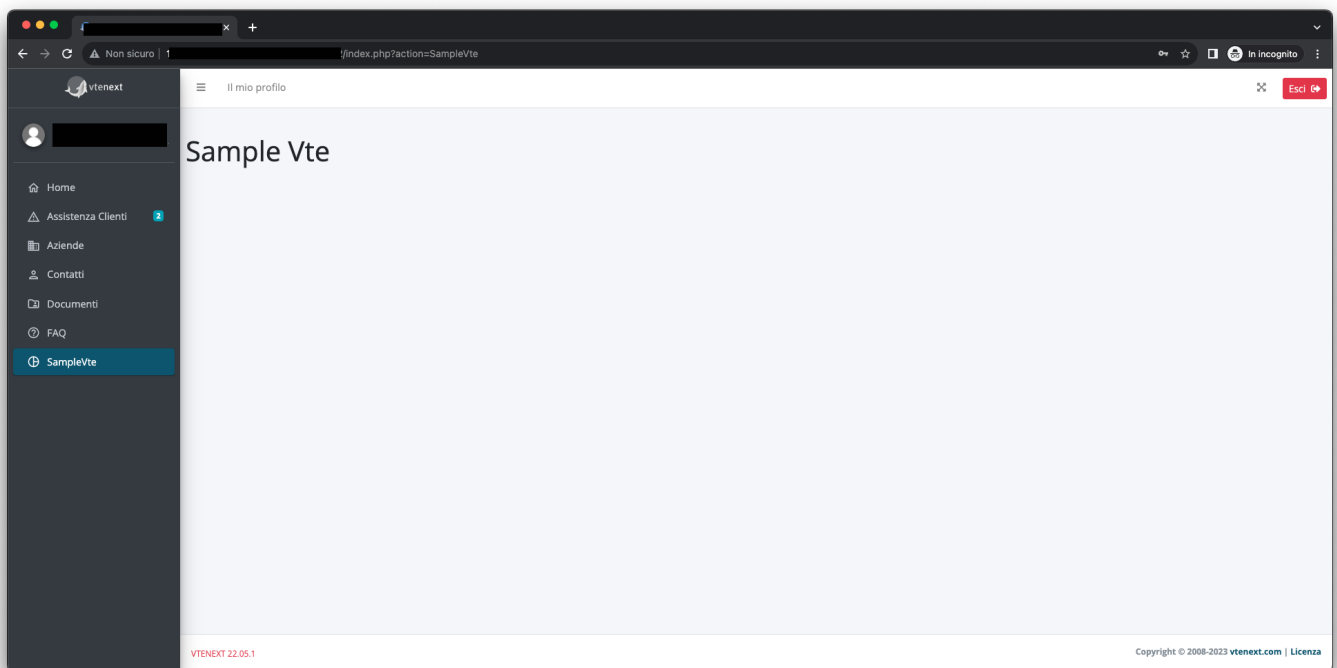
return [
    [
        'text' => trans('SampleVte'),
    ],
];

```

```

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

```



Extending a module

Here is an example of how to extend the functionality of a module.

- Edit "config/sdk.config.php"

```

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

```

- Implement the ContactsModule class in "sdk/modules/ContactsModule.php"

```
<?php

class ContactsModule extends \app\PortalModule {

    public $hasComments = true;

    public $formColumns = 2;

    public function canAddComments() {
        return true;
    }
}
```

