
VTECRM

VTE SDK 1 - Manual

Summary

vtlib API - Quick Reference....4
Module Types....5
Entity Module....5
Extension Module....5
Language Pack....5
Creating a new Entity Module....6
Backend....6
FrontEnd....6
Additional Options....6
About InstalledBase Module....7
Step 1: Creating Module....8
Step 2: Creating Tabs (optional)....9
Step 3: Creating Blocks (in UI Form)....10
Step 4: Adding Fields....11
Entity Identifier....12
Set Picklist Values....12
Set Related Module....13
Set Help Information....14
Step 5: Creating Filters....15
Configure fields....15
Setup Rules....16
Step 6: Related Lists....17
Limitations....18
Step 7: Sharing Access Rules....19
Step 8: Module Tools....20
Optional Step: Module Events....21
Optional Step: Module Templates....22
Optional Step: Custom Links....23
Special LinkType....23
Step 9: Creating module files (Frontend)....24

Final Completed Script.....25
Appendix – Using vtecrm_imageurl API.....26
Appendix – Uitypes List.....27

vtlib API - Quick Reference

vtlib includes the following APIs that can be used to create new modules. For more details please look at the API docs.

- Vtecrm_Module
  - name
  - addBlock()
  - addFilter()
  - initTables()
  - setRelatedList()
  - setDefaultSharing()
  - enableTools()
  - disableTools()
  - save()
  - addLink()
- Vtecrm_Menu
  - addModule()
- Vtecrm_Block
  - label
  - addField()
- Vtecrm_Field
  - table
  - column
  - columntype
  - uitype
  - typeofdata
  - setHelpInfo()
  - setEntityIdentifier()
  - setPicklistValues()
  - setRelatedModules()
- Vtecrm_Filter
  - name
  - isdefault
  - addField()
  - addRule()
- Vtecrm_Event
  - register()

Module Types

VTE CRM modules can be classified into following types:

1. Entity Module
2. Extension Module
3. Language Pack

Entity Module

Modules in this category will create entity records in vtecrm CRM. The module will provide Create view, Edit view, Detail view and List view. You will be able to create filters etc.

Entity modules are recommended for cases where a new type of data object, e.g. Timesheet, needs to be added into the system as part of the new module. These new data objects can be viewed and managed by administrators and users.

Leads, Contacts, Accounts, Payslip etc... are Entity Modules.

Extension Module

Modules in this category need not follow the general behavior of Entity Module. The records created by Entity module could be used to provide a extended functionality or the records creation/editing can be handled in its own way.

Extension modules can be used when add-on functionality is needed, without the need for new kinds of data objects that users view and manage.

Dashboard, Reports, Portal etc... are Extension Modules.

Language Pack

Language Packs for vtecrm CRM are also treated as another kind of module by vtlib.

NOTE: Module manager will provide the ability to install these different modules.

Creating a new Entity Module

vtlib simplifies creation of new vtecrm CRM modules. Developers can use vtlib to develop vtecrm CRM modules that add new functionality to vtecrm CRM. These modules can then be packaged for easy installation by the Module Manager.

NOTE: In this document we will explain the process of creating a new module by building an example 'InstalledBase' Module. This example code is included as part of vtlib package, and can be used as a starting point to create new modules. Please refer to the 'Using the example code provided with the vtlib API' section in this document for more information.

The following are important steps that should be followed to get a basic working module. The backend section covers database level changes for the module, and the frontend section covers the UI files.

Backend

Step 1	Create module instance, create database tables, and add it to Menu
Step 2	Add UI blocks for the module.
Step 3	Add fields and associate it to blocks. Set at-least one field as entity identifier.
Step 4	Create default list view and additional filters (make sure to create a filter named All which is the default filter)
Step 5	Create Related List (to show in the ''More information'' tab)
Step 6	Setting Sharing Access Rules
Step 7	Setting Module Tools options (i.e., Import/Export)

FrontEnd

Step 8	Creating Module directory and files

Additional Options

➔ Module Templates (to customize Form, List View, and Settings UI)
➔ Module Settings (to allow administrators to configure your module)
➔ Module Events (only available in vtiger CRM version 5.1)

These steps are explained in detail in the course of this section.

We are using the example module 'InstalledBase' to explain the use of vtlib APIs.

About InstalledBase Module

It will have the ability to create, edit, delete installedbase records. You can create Custom Filters for the Listview, which displays the list of payslip instances.

We shall associate this module with the Tools menu.

Step 1: Creating Module

Class Vtecrm_Module provides an API to work with vtecrm CRM modules.

```
include_once('vtlib/Vtecrm/Menu.php');
include_once('vtlib/Vtecrm/Module.php');

$moduleInstance = new Vtecrm_Module();
$moduleInstance->name = 'InstalledBase';
$moduleInstance->save();

$moduleInstance->initTables();

$moduleInstance=Vtecrm_Module::getInstance('InstalledBase');

$menuInstance = Vtecrm_Menu::getInstance('Tools');
$menuInstance->addModule($moduleInstance);
```

Vtecrm_Module->initTables() API will initialize (create) the 3 necessary tables a module should have as explained below:

Table	Naming convention	Description
Basetable	Vtecrm_<MODULENAME>	Contains the default field for the new module
Customtable	Vtecrm_<MODULENAME>cf	Contains custom fields of the module

Vtecrm_Menu->addModule(<ModuleInstance>) API will create menu item which serves as UI entry point for the module.

Step 2: Creating Tabs (optional)

Class Vtecrm_Panel provides API to work with a Module tabs, which are visible in detailview and editview (feature available only in VTE 16.09 and newer).

The example given below describes the way of creating new tabs for the module created earlier:

```
include_once('vtlib/Vtecrm/Module.php');
include_once('vtlib/Vtecrm/Block.php');

$moduleInstance = Vtecrm_Module::getInstance('InstalledBase');

$panelInstance = new Vtecrm_Panel();
$panelInstance->label = 'LBL_TAB_MAIN';
$moduleInstance->addPanel($panelInstance);

$panelInstance2 = new Vtecrm_Panel();
$panelInstance2->label = 'Second panel';
$moduleInstance->addPanel($panelInstance2);
```

NOTE: LBL_TAB_MAIN is the label for the default tab. When creating a new module, and no tabs are added manually, a default tab is created with this label and all blocks are inside this only tab.

Step 3: Creating Blocks (in UI Form)

Class Vtecrm_Block provides API to work with a Module block, the container which holds the fields together.

The example given below describes the way of creating new blocks for the module created earlier:

```
include_once('vtlib/Vtecrm/Module.php');
include_once('vtlib/Vtecrm/Block.php');

$moduleInstance=Vtecrm_Module::getInstance('InstalledBase');

$blockInstance = new Vtecrm_Block();
$blockInstance->label = 'LBL_InstalledBase_INFORMATION';
$moduleInstance->addBlock($blockInstance);

$blockInstance2 = new Vtecrm_Block();
$blockInstance2->label = 'LBL_CUSTOM_INFORMATION';
$moduleInstance->addBlock($blockInstance2);

$blockInstance3 = new vtecrm_Block();
$blockInstance3->label = 'LBL_DESCRIPTION_INFORMATION';
$moduleInstance->addBlock($blockInstance3);
```

NOTE: LBL_CUSTOM_INFORMATION block should always be created to support Custom Fields for a module.

If you want to put blocks in specific tabs (see previous chapter), use the following syntax:

```
include_once('vtlib/Vtecrm/Module.php');
include_once('vtlib/Vtecrm/Block.php');

$moduleInstance=Vtecrm_Module::getInstance('InstalledBase');

// $panelInstance2 is an instance to the previously created panel
$blockInstance = new Vtecrm_Block();
$blockInstance->label = 'Block in second panel';
$panelInstance2->addBlock($blockInstance);
```

NOTE: If you add blocks to the module instance, they will be put in the first tab (and if no tabs exists in the module, a default one will be created).

Step 4: Adding Fields

Class Vtecrm_Field provides API to work with a Module field, which are the basic elements that store and display the module record data.

The example given below describes the way of creating new field for the module created earlier:

```
include_once('vtlib/Vtecrm/Block.php');
include_once('vtlib/Vtecrm/Field.php');
include_once('vtlib/Vtecrm/Module.php');

$moduleInstance=Vtecrm_Module::getInstance('InstalledBase');
$blockInstance=Vtecrm_Block::getInstance('LBL_InstalledBase_INFORMATION',
$moduleInstance);

$fieldInstance = new Vtecrm_Field();
$fieldInstance->name = 'installedbase_name';
$fieldInstance->table = 'vte_installedbase';
$fieldInstance->label = 'InstalledBase Name';
$fieldInstance->columntype = 'VARCHAR(255)';
$fieldInstance->uitype = 1;
$fieldInstance->typeofdata = 'V~M';
$fieldInstance->quickcreate = 0;
$blockInstance->addField($fieldInstance);
```

NOTE: The fieldInstance name is a mandatory value to be set before saving / adding to block. Other values (if not set) are defaulted as explained below:

$fieldInstance->table	Module's basetable
$fieldInstance->column	$fieldInstance->name in lowercase [The table will be altered by adding the column if not present]
$fieldInstance->columntype	VARCHAR(255)
$fieldInstance->uitype	1
$fieldInstance->typeofdata	V~O [V stand for varchar / D stand for date / … O stand for optional / M stand for mandatory]
$fieldInstance->label	$fieldInstance->name [Mapping entry should be present in module language file as well]

NOTE: Uitype is the graphical representation of the field. At the end of the manual you can find the list of the existent uitypes (Appendix – Uitypes List).

Entity Identifier

One of the mandatory field should be set as entity identifier of module once it is created. This field will be used for showing the details in 'Last Viewed Entries' etc...

```
include_once('vtlib/Vtecrm/Field.php');
include_once('vtlib/Vtecrm/Module.php');

$moduleInstance=Vtecrm_Module::getInstance('InstalledBase');
$fieldInstance=Vtecrm_Field::getInstance('installedbasename',$moduleInstance);

$moduleInstance->setEntityIdentifier($fieldInstance);
```

NOTE: The Entity Identifier must run before the step 3

Set Picklist Values

If the field is of Picklist type (uitype **15**, 16, 33, 55, 111) then you can configure the initial values using the following API:

```
include_once('vtlib/Vtecrm/Block.php');
include_once('vtlib/Vtecrm/Field.php');
include_once('vtlib/Vtecrm/Module.php');

$moduleInstance=Vtecrm_Module::getInstance('InstalledBase');
$blockInstance=Vtecrm_Block::getInstance('LBL_InstalledBase_INFORMATION',
$moduleInstance);

$fieldInstance = new Vtecrm_Field();
$fieldInstance->table = 'vte_installedbase';
$fieldInstance->column = 'installedbasetype';
$fieldInstance->name = 'installedbasetype';
$fieldInstance->label = 'Type';
$fieldInstance->columntype = 'VARCHAR(100)';
$fieldInstance->uitype = 15;
$fieldInstance->typeofdata = 'V~M';
$blockInstance->addField($fieldInstance);
$fieldInstance->setPicklistValues( Array ('-- Nessuno -- ', 'In corso di
evasione', 'Evasa', 'Fatturata') );
```

Set Related Module

If the field is of Popup select type (uitype=10), you can configure the related modules which could be selected via Popup using the following API:

```
include_once('vtlib/Vtecrm/Module.php');
include_once('vtlib/Vtecrm/Block.php');
include_once('vtlib/Vtecrm/Field.php');

$moduleInstance=Vtecrm_Module::getInstance('InstalledBase');
$blockInstance=Vtecrm_Block::getInstance('LBL_InstalledBase_INFORMATION',
$moduleInstance);

$fieldInstance = new Vtecrm_Field();
$fieldInstance->name = 'account_id';
$fieldInstance->table = 'vte_installedbase';
$fieldInstance->label= 'Account Id';
$fieldInstance->column = 'accountid';
$fieldInstance->uitype = 10;
$fieldInstance->columntype = 'I(19)';
$fieldInstance->typeofdata = 'I~O';
$fieldInstance->displaytype= 1;
$fieldInstance->quickcreate = 0;
$blockInstance->addField($fieldInstance);
$fieldInstance->setRelatedModules(Array('Accounts'));
```

If you want is possible create more than one related. In this situation, you can see a picklist where you find all the related modules.

```
include_once('vtlib/Vtecrm/Module.php');
include_once('vtlib/Vtecrm/Block.php');
include_once('vtlib/Vtecrm/Field.php');

$moduleInstance=Vtecrm_Module::getInstance('InstalledBase');
$blockInstance=Vtecrm_Block::getInstance('LBL_InstalledBase_INFORMATION',
$moduleInstance);

$fieldInstance = new Vtecrm_Field();
$fieldInstance->name = 'accoun     t_id';
$fieldInstance->table = 'vte_installedbase';
$fieldInstance->label= 'Account Id';
$fieldInstance->column = 'accountid';
$fieldInstance->uitype = 10;
$fieldInstance->columntype = 'I(19)';
$fieldInstance->typeofdata = 'I~O';
$fieldInstance->displaytype= 1;
$fieldInstance->quickcreate = 0;
$blockInstance->addField($fieldInstance);
$fieldInstance->setRelatedModules(Array('Accounts','Contacts'));
```

To unset the related module you can use the following API.

```
$fieldInstance->setRelatedModules(Array('OtherModule2'));
```

Set Help Information

Providing help information for module field will be useful to educate users.

```
include_once('vtlib/Vtecrm/Module.php');
include_once('vtlib/Vtecrm/Block.php');
include_once('vtlib/Vtecrm/Field.php');

$moduleInstance=Vtecrm_Module::getInstance('InstalledBase');
$blockInstance=Vtecrm_Block::getInstance('LBL_InstalledBase_INFORMATION',
$moduleInstance);
…
$fieldInstance->helpinfo = 'Relate to an existing account';
…
$fieldInstance->setRelatedModules(Array('Accounts'));
```

You can provide set the help text for an existing field using the following API:

```
$fieldInstance->setHelpInfo('HELP CONTENT');
```

NOTE: HELP CONTENT can be plain or rich text.

When a field has help information, helpicon will be shown beside the field label.

NOTE: Given below is the snippet of code that should be added to EditView.php of existing module to enable Help Icon support.

```
// ...
// Gather the help information associated with fields
$smarty->assign('FIELDHELPINFO', vtlib_getFieldHelpInfo($currentModule));
// END
// ...
if($focus->mode == 'edit') $smarty->display('salesEditView.tpl');
else $smarty->display('CreateView.tpl');
```

Step 5: Creating Filters

Class Vtiger_Filter provides API to work with a Module's custom view or filter. The list view display is controlled via these filters.

The example given below describes the way of creating new filter for the module:

```
include_once('vtlib/Vtecrm/Filter.php');
include_once('vtlib/Vtecrm/Module.php');

$moduleInstance=Vtecrm_Module::getInstance('InstalledBase');

$filterInstance = new Vtecrm_Filter();
$filterInstance->name = 'All';
$filterInstance->isdefault = true;
$moduleInstance->addFilter($filterInstance);
```

Configure fields

To add fields to the filter you can use the following API:

```
include_once('vtlib/Vtecrm/Module.php');
include_once('vtlib/Vtecrm/Block.php');
include_once('vtlib/Vtecrm/Field.php');
include_once('vtlib/Vtecrm/Filter.php');

$moduleInstance=Vtecrm_Module::getInstance('InstalledBase');
$blockInstance=Vtecrm_Block::getInstance('LBL_InstalledBase_INFORMATION',
$moduleInstance);
$filterInstance=Vtecrm_Filter::getInstance('All',$moduleInstance);

$fieldInstance1=Vtecrm_Field::getInstance('installedbasename',$moduleInstance);
$fieldInstance2=Vtecrm_Field::getInstance('installedbasetype',$moduleInstance);
$fieldInstance3=Vtecrm_Field::getInstance('account_id',$moduleInstance);

$filterInstance->addField($fieldInstance1, 1);
$filterInstance->addField($fieldInstance2, 2);
$filterInstance->addField($fieldInstance3, 3);
```

Setup Rules

Once the field is added to filter you can setup rule (condition) for filtering as well using the following API:

```
$filterInstance->addRule($fieldInstance, $comparator, $compareValue,
$columnIndex);
```

Where comparator could be one of the following:

EQUALS
NOT_EQUALS
STARTS_WITH
ENDS_WITH
CONTAINS
DOES_NOT_CONTAINS
LESS_THAN
GREATER_THAN
LESS_OR_EQUAL
GREATER_OR_EQUAL

$compareValue is the value against with the field needs to be compared.

$columnIndex (optional) is the order at which this rule condition should be applied.

Step 6: Related Lists

One module could be associated with multiple records of other module that is displayed under "More Information" tab on Detail View.

```
Vtecrm_Module->setRelatedList(<TARGET MODULE>[, <HEADER LABEL>, <ALLOWED
ACTIONS>,
<CALLBACK FUNCTION NAME>]);
```

The example given below describes the way of creating a relation between a Payslip and Accounts module:

NOTE: If you do not define the model of related. The related default is related_list the type N -> N

```
include_once('vtlib/Vtecrm/Module.php');

$moduleInstance = Vtecrm_Module::getInstance('InstalledBase');
$accountsModule = Vtecrm_Module::getInstance('Accounts');
$relationLabel = 'Accounts';
$moduleInstance->setRelatedList(

      $accountsModule, $relationLabel, Array('ADD','SELECT','get_related_list')
);
```

With this you can Add one or more Accounts to InstalledBase records.

To drop the relation between the modules use the following:

```
$moduleInstance->unsetRelatedList($targetModuleInstance);
```

About setRelatedList API

<TARGET MODULE>	Module name to which relation is being setup.
<HEADER LABEL>	Optional (default = <TARGET MODULE>) Label to use on the More Information related list view.
<ALLOWED ACTIONS>	Optional ADD or SELECT (default = false) What buttons should be shown in the related list view while adding records.
<CALLBACK FUNCTION NAME>	Optional (default = get_related_list) The function should be defined in the <SOURCE MODULE> class. This should generate the listview entries for displaying.

NOTE:
This API will create an entry in the vtecrm_crmentityrel table to keep track of relation between module
records. Standard modules available in vtecrm handles the relation in separate tables and performs the JOIN to fetch data specific to each module.

This is an attempt to achieve generic behavior. You can write custom call back functions to handle related
list queries that will meet your requirements.

Limitations

Following limitations apply for the related list APIs

1. Standard module class variables are not set as required by the get_related_list vtlib module API.

Case handling should be handled @function vtlib_setup_modulevars in include/utils/VtlibUtils.php

2. get_related_list API added to module class does not handle JOIN

on tables where some modules like (Accounts) store information hence complete details are not fetched in the Related List View.
(Example Sorting on the city field on related list view will fail if dieOnError is true)

Step 7: Sharing Access Rules

Sharing access configuration for the module can be done as shown below:

The example given below describes the way to configure the InstalledBase module as Private

```
include_once('vtlib/Vtecrm/Module.php');

$moduleInstance = Vtecrm_Module::getInstance('InstalledBase');

$moduleInstance->setDefaultSharing('Private');
```

The <PERMISSION_TYPE> can be one of the following:

Public_ReadOnly
Public_ReadWrite
Public_ReadWriteDelete
Private

Step 8: Module Tools

Features like Import, Export are termed as module tools. Such tools can enabled or disabled as shown below:

The example given below describes the way to enable and disable the tools for InstalledBase module

```
include_once('vtlib/Vtecrm/Module.php');

$moduleInstance = Vtecrm_Module::getInstance('InstalledBase');

$moduleInstance->enableTools(Array('Import', 'Export'));

$moduleInstance->disableTools('Export');
```

Optional Step: Module Events

To register an event for a module, use the following:

```
include_once('vtlib/Vtecrm/Event.php');
Vtecrm_Event::register('<MODULENAME>', '<EVENTNAME>',
                       '<HANDLERCLASS>', '<HANDLERFILE>');
```

<MODULNAME>	Module for which events should be registered
<EVENTNAME>	vtiger.entity.aftersave vtiger.entity.beforesave
<HANDLERCLASS>	Event handler class, look at the example below
<HANDLERFILE>	File where HANDLERCLASS is defined (should be within vtiger CRM directory)

Example: Registering event callback before and after save.

```
if(Vtecrm_Event::hasSupport()) {
      Vtecrm_Event::register(
            'InstalledBase', 'vtecrm.entity.aftersave',
            'InstalledBaseHandler',
'modules/InstalledBase/InstalledBaseHandler.php'
            );

            Vtecrm_Event::register(
                  'InstalledBase', 'vtiger.entity.beforesave',
                  'InstalledBaseHandler',
'modules/InstalledBase/InstalledBaseHandler.php'
            );
}
```

modules/InstalledBase/ InstalledBaseHandler.php

```
<?php

class InstalledBaseHandler extends VTEventHandler {

      function handleEvent($eventName, $data) {

            if($eventName == 'vtecrm.entity.beforesave') {
                  // Entity is about to be saved, take required action
            }

            if($eventName == 'vtecrm.entity.aftersave') {
                  // Entity has been saved, take next action
            }
      }
}
?>
```

Optional Step: Module Templates

If you would like to customize the list view or have a custom Settings page for the module, then you will need to create a Smarty template accordingly. You will need to have some knowledge of Smarty templates usage before you proceed.

Your module specific Smarty template files should be created under Smarty/templates/modules/<NewModuleName>.

Use vtlib_getModuleTemplate($module, $templateName) API (include/utils/VtlibUtils.php) as:

```
$smarty->display(vtlib_getModuleTemplate($currentModule, 'MyListview.tpl'));
```

Optional Step: Custom Links

You can add custom web link to the module using the following API:

```
include_once('vtlib/Vtecrm/Module.php');
$moduleInstance = Vtecrm_Module::getInstance('ModuleName');
$moduleInstance->addLink(<LinkType>, <LinkLabel>, <LinkURL>);
```

LinkType	Type of Link like DETAILVIEW etc..
LinkLabel	Label to use for the link when displaying
LinkURL	URL of the link. *You can use variables like $variablename$*

Given below is an example which adds a link to the DetailView of the Module.

```
include_once('vtlib/Vtecrm/Module.php');

$moduleInstance = Vtecrm_Module::getInstance('InstalledBase');
$moduleInstance->addLink(
'DETAILVIEW',
'New Action',
'index.php?
module=OtherModule&action=SomeAction&src_module=$MODULE$&src_record=$RECORD$'
);
```

NOTE: The $MODULE$ and $RECORD$ variables for the 'New Action' link will be replaced with the values set through DetailView.php

Special LinkType

Following LinkTypes are treated specially while processing for display:

Linktype	Description
HEADERSCRIPT	The link will be treated as a javascript type and will be imported in the head section of the HTML output page as *<script type='text/javascript' src='linkurl'></script>*
HEADERCSS	The link will be treated as a CSS type and will be imported in the head section of the HTML output page as *<link rel='stylesheet' type='text/css' href='linkurl>*
HEADERLINK	You can see these link grouped under More on the top header panel. Useful if you want to provide utitlity tools like Bookmarklet etc.

Step 9: Creating module files (Frontend)

Each new module should have a directory under modules/ folder. To help speed up the module code creation, vtlib comes bundled with skeleton module structure based on the 'InstalledBase' module. This code is include in vtlib/ModuleDir folder which can be used as a template for new module that is created. It contains source files that needs to be changed as explained below.

NOTE: ModuleDir has sub-directories specific to vtiger version, please make sure to use the right one.

1. Copy ModuleDir/*<target_vtiger_version>* contents to newly created modules/<NewModuleName> folder.
2. Rename <NewModuleName>/ModuleFile.php as <NewModuleName>/<NewModuleName>.php (as noted in the table below)
3. Rename <NewModuleName>/ModuleFileAjax.php as <NewModuleName>/<NewModuleName>Ajax.php
4. Rename <NewModuleName>/ModuleFile.js to <NewModuleName>/<NewModuleName>.js
5. Edit <NewModuleName>/<NewModuleName>.php
    a. Rename Class ModuleClass to <NewModuleName>
    b. Update $table_name and $table_index (Module table name and table index column)
    c. Update $groupTable
    d. Update $tab_name, $tab_name_index
    e. Update $list_fields, $list_fields_name, $sortby_fields, $list_link_field
    f. Update $detailview_links
    g. Update $default_order_by, $default_sort_order
    h. Update $required_fields
    i. Update $customFieldTable
    j. Rename function ModuleClass to function <NewModuleName> [This is the Constructor Class]

NOTE: Other files under modules/<NewModuleName> need not be changed.

Example ModuleDir	Purpose	File under Payslip
ModuleFile.php	Module class definition file.	Payslip.php
ModuleFileAjax.php	Base file for ajax actions used under Listview etc...	PayslipAjax.php
ModuleFile.js	Module specific javascript function can be written here	Payslip.js
CallRelatedList.php	More Information Detail view handler	CallRelatedList.php
DetailViewAjax.php	Detail view ajax edit handler	DetailViewAjax.php
EditView.php	Edit view handler	EditView.php
Save.php	Module record save handler	Save.php

Final Completed Script

The file modules/SDK/examples/VTE-SDK-1.php contains a complete script for the creation of a module.

Appendix – Using vtecrm_imageurl API

There are reusable images under themes/images folder and theme specific images will be under themes/<THEMENAME>/images folder.

You can let the image easily configurable for each theme, please make sure to follow the steps below:

In YourSmartyFile.tpl (*Smarty template file*)

```
<img src="{'myimage.gif'|@vtecrm_imageurl:$THEME}">
```

$THEME variable will be sent by the calling script as follows:

```
global $theme;
$smarty->assign('THEME', $theme);
$smarty->display('YourSmartyFile.tpl');
```

This gets translated to:

<img src="themes/<THEMENAME>/images/myimage.gif">	If myimage.gif exists under <THEMENAME> folder
<img src="themes/images/myimage.gif">	Default path if theme specific image is not found

If you directly building the UI from PHP script, make sure to use the API as follows:

```
vtecrm_imageurl ( 'imagename', 'themename' );
```

NOTE: vtecrm_imageurl API is defined in include/utils/VtlibUtils.php

Appendix – Uitypes List

This table describe the representation of each uitype and suggests the value of the other properties for the creation of the field when you create a field using the API described on **Step 3: Adding Fields.**

uitype	Description	typeofdata	columntype	readonly	displaytype
1	Text box	V~O	C(255)	1	1
4	Automatic numeration	V~O	C(100)	1	1
5	Date	D~O	D	1	1
7	Number (only integer)	I~O	I(19)	1	1
7	Number (with 2 decimals)	N~O	N(19.2)	1	1
9	Percentual	N~O	N(7.3)	1	1
10	Relation with other record	V~O	I(19)	1	1
11	Phone	V~O	C(30)	1	1
13	Email	E~O	C(100)	1	1
15	Picklist	V~O	C(255)	1	1
16	Picklist non-role based	V~O	C(255)	1	1
17	URL	V~O	C(255)	1	1
19	Textarea with colspan=2	V~O	XL	1	1
21	Textarea	V~O	X	1	1
33	Multi-select combo box	V~O	X	1	1
52	User picklist	V~O	I(19)	1	1
53	User and group picklist	V~M	I(19)	1	1
56	Checkbox	C~O	I(1)	1	1
69	Attachment	V~O	C(255)	1	1
71	Currency	N~O	N(19.2)	1	1
77	User picklist	I~O	I(19)	1	1
85	Skype	V~O	C(255)	1	1
210	Textarea with ckeditor	V~O	X or XL	1	1
1013	Fax	V~O	C(50)	1	1
1014	Mobile	V~O	C(50)	1	1
1015	Picklist multi-language	V~O	C(255)	1	1