vtlib – vtiger development library

DISCLAIMER:

The vtlib library development is in progress and subject to change. While we make every effort to make sure modules developed using vtlib will be compatible with future versions of vtiger CRM, some incompatible changes may be required for the next release of vtiger. In which case you will have to re-create your modules with an upgraded version of vtlib for the specific version of vtiger CRM.

Table of Contents

API Version History....4
About vtlib....5
vtlib API - Quick Reference....6
Module Types....7
- Entity Module....7
- Extension Module....7
- Language Pack....7
- Bundles....7

Creating a new Entity Module....8
- Backend....8
- FrontEnd....8
- Packaging....8
- About Payslip Module....8

Step 1: Creating Module....9
Step 2: Creating Block (in UI Form)....10
Step 3: Adding Fields....11
- Entity Identifier....12
- Set Picklist Values....12
- Set Related Module....12
- Set Help Information....13
- Set MassEdit property....15

Step 4: Creating Filters....16
- Configure fields....16
- Setup Rules....16

Step 5: Related Lists....17
Step 6: Sharing Access Rules....19
Step 7: Module Tools....20
Optional Step: Module Events....21
Optional Step: Module Webservices....22
Optional Step: Module Templates....23
Optional Step: Custom Links....24
- Special LinkType....25

Final Completed Script (Backend)....26
Executing Module Creation Script....29
Step 8: Creating module files (Frontend)....30
- Language File Preparation....31

New Module Tour....32
- List view....32
- Create view....32

Detail view....33
List view....33
Sharing Access....34
Custom Fields....34
Step 9: Packaging....35
Package Export....35
Package Structure....36
Package Import....38
Package Upgrade....39
Limitations....39
Extension Module....40
Bundles....41
Language Pack....42
Adding License....43
Adding Migration Details....44
Module Manager....46
Disabling Module....47
Enabling Module....47
Exporting Module....48
Importing Module....49
Module Specific Settings....52
Module Upgrade....53
Appendix 1 - API Changes....55
Creating Module....55
Creating Block....55
Creating Field....56
Setting Entity Identifier....56
Set Picklist Values....56
Creating Filter....57
Configure Tools....57
Configure Sharing Access....57
Appendix 2 – Schema Changes....58
Appendix 3 – Using vtiger_imageurl API....59
Appendix 4 – vtlib_handler Method....60
Appendix 5 – vtlib_listview javascript API....61
FAQs – Frequently Asked Questions....62
1. How to write own templates?....62
2. How is module template used?....62
3. Cannot See Module Manager!....63
4. Tips for using field names....63

API Version History

Version	Release Date	Highlights
2.2	**Integrated to 5.1.0**	* Added special module event triggers (vtlib_handler method invocation)
2.1	2009.01.07	* Installing Language Pack, Extension module using Module Manager * ModuleDir was re-organized to be specific to a vtiger version * Module Manager handles License agreement before installation. * Basic support added for module upgrades with Module Manager. * Ability to add custom web links for a module. * Support added to allow help information for the module fields. * Added vtiger_imageurl API
2.0	2008-11-26	* API provided to related modules (related list) * API changes for creating fields, blocks, module * UI type 10 was added for generic popup field
1.4	2008-09-29	* Exporting and Importing was added to Module Manager
1.3	2008-09-01	* Module Manager was added for Administrators to install and enable/disable the new modules developed using vtlib. * API to setup Picklist values
1.2	2008-08-29	* Sharing Access API was added
1.1	2008-08-27	* API's to enable and disable tools like Export, Import were added
1.0	2008-08-19	* Basic API was added to created fields, blocks, module

About vtlib

vtlib is a library to ease new module development for vtiger CRM. vtlib includes APIs to create or modify the backend elements for a module. These APIs help make the necessary changes to the database.

vtlib includes Module Manager which allows new modules to be packaged into zip files that other vtiger CRM installations can easily install and use.

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.

- Vtiger_Module
  - name
  - addBlock()
  - addFilter()
  - initTables()
  - setRelatedList()
  - setDefaultSharing()
  - enableTools()
  - disableTools()
  - save()
  - addLink()

- Vtiger_Menu
  - addModule()

- Vtiger_Block
  - label
  - addField()

- Vtiger_Field
  - table
  - column
  - columntype
  - uitype
  - typeofdata
  - setHelpInfo()
  - setEntityIdentifier()
  - setPicklistValues()
  - setRelatedModules()

- Vtiger_Filter
  - name
  - isdefault
  - addField()
  - addRule()

- Vtiger_Event
  - register()

Module Types

vtiger CRM modules can be classified into following types:

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

Entity Module

Modules in this category will create entity records in vtiger 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 vtiger CRM are also treated as another kind of module by vtlib.

Bundles

Bundles let you install a set of inter dependent vtlib modules in the specified order.

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

Creating a new Entity Module

vtlib simplifies creation of new vtiger CRM modules. Developers can use vtlib to develop vtiger CRM modules that add new functionality to vtiger 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 'Payslip' 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

Packaging

Step 9	Packaging

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)
- ➔ Module Webservices (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 'Payslip' to explain the use of vtlib APIs.

About Payslip Module

It will have the ability to create, edit, delete payslip 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 Vtiger_Module provides an API to work with vtiger CRM modules.

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

$moduleInstance = new Vtiger_Module();
$moduleInstance->name = 'Payslip';
$moduleInstance->save();

$moduleInstance->initTables();

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

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

Table	Naming convention	Description
Basetable	vtiger_<MODULENAME>	Contains the default fields for the new module
Customtable	vtiger_<MODULENAME>cf	Contains custom fields of the module
Grouptable	vtiger_<MODULENAME>grouprel	Used when records are assigned to a group

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

Step 2: Creating Block (in UI Form)

Class Vtiger_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/Vtiger/Module.php');

$blockInstance = new Vtiger_Block();
$blockInstance->label = 'LBL_PAYSLIP_INFORMATION';
$moduleInstance->addBlock($blockInstance);

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

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

Step 3: Adding Fields

Class Vtiger_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/Vtiger/Module.php');

$fieldInstance = new Vtiger_Field();
$fieldInstance->name = 'PayslipName';
$fieldInstance->table = 'vtiger_payslip';
$fieldInstance->column = 'payslipname';
$fieldInstance->columntype = 'VARCHAR(100)';
$fieldInstance->uitype = 2;
$fieldInstance->typeofdata = 'V~M';
$blockInstance->addField($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
$fieldInstance->label	$fieldInstance->name [Mapping entry should be present in module language file as well]

Optional Settings

$fieldInstance->presence	0 – Always Active (Cannot be modified using Layout Editor in 5.1.0) 1 – Mark it In Active (5.1.0 onwards) 2 – Active Property can be modified using Layout Editor (5.1.0 onwards)
$fieldInstance->quickcreate	0 – Enable field in Quick Create Form 1 – Disable field on Quick Create Form
$fieldInstance->masseditable	0 – Permanently Disallow field for mass editing (5.1.0 onwards) 1 – Allow field for mass editing (5.1.0 onwards) 2 – Disallow field for mass editing (but can be made available using Layout Editor 5.1.0 onwards)

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...

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

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:

```
$fieldInstance->setPicklistValues( Array ('Value1', 'Value2') );
```

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:

```
$fieldInstance->setRelatedModules(Array('OtherModule1', 'OtherModule2'));
```

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

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

Set Help Information

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

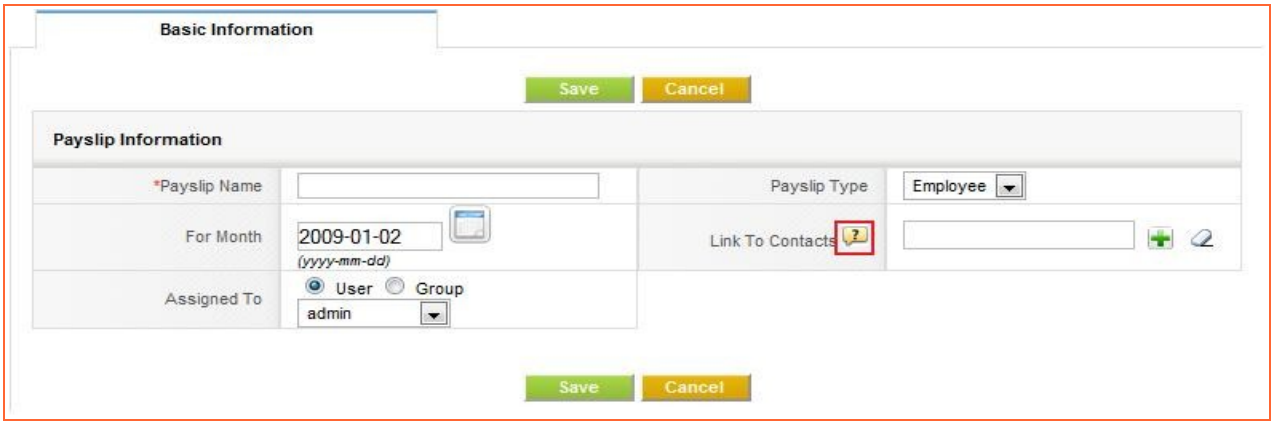
```
include_once('vtlib/Vtiger/Module.php');

$fieldInstance = new Vtiger_Field();
$fieldInstance->name = 'LinkTo';
...
$fieldInstance->helpinfo = 'Relate to an existing contact';
...
$blockInstance->addField($fieldInstance);
```

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

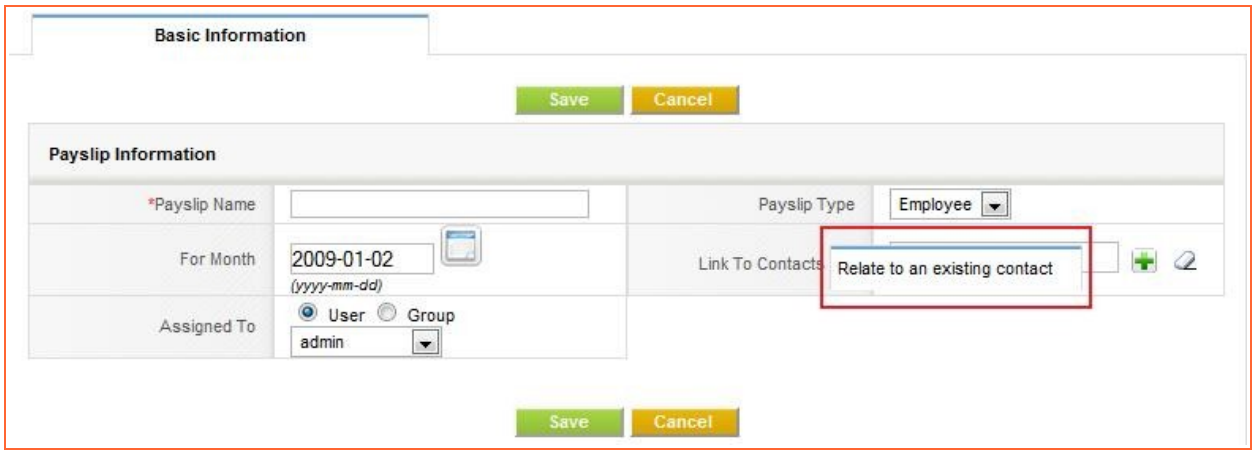
```
$fieldInstance->setHelpInfo('HELP CONTENT');
```

NOTE: HELP CONTENT can be plain or rich text. See the recommended usage below.

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



Clicking on it will show the help content as shown:



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');
```

Recommended:

- Provide translation mapping for the helpinfo being used for a field.

  ```
  Example set the helpinfo as HELP_FILEDNAME_INFO and provide the contents in the language file.

  $fieldInstance->setHelpInfo('HELP_FIELDNAME_INFO');

  In module/<MODULENAME>/language/en_us.lang.php

  $mod_strings = Array(
  ...
  'HELP_FIELDNAME_INFO' => 'Fieldname help contents should be here',
  ...);
  ```

- Avoid newlines in the help content, you can use <br> tag instead
- Preferably escape (', ", <, >) with HTML entities like (", < >)
- It is good to have keep the content less, if you want to provide more details you can link to a external page as shown in the example below:

  Fill your contacts name here. To know more about it
  <a href='http://en.vtiger.com/wiki.html'>see more</a>

Set MassEdit property

NOTE: Mass edit feature is available from vtiger 5.1 onwards

You can make the field available for mass editing use the following ways described below:

When creating the field you can set the property:

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

$fieldInstance = new Vtiger_Field();
$fieldInstance->name = 'TestField';
...
$fieldInstance->masseditable = 1;
...
$blockInstance->addField($fieldInstance);
```

If you have an existing field its property can be updated using the API:

```
$fieldInstance->setMassEditable(value);
```

The value set for masseditable property has the following meaning:

Value	Description
0	Not available for mass edit and this property cannot be controlled by user.
1	Available for mass edit
2	Not available for mass edit but the property can be controlled by user (via Layout Manager etc)

Step 4: 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/Vtiger/Module.php');

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

Configure fields

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

```
$filterInstance->addField($fieldInstance, $columnIndex);
```

Where $columnIndex (optional) is the order/index at which the field should appear in the list view.

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 5: Related Lists

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

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

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

$moduleInstance = Vtiger_Module::getInstance('Payslip');

$accountsModule = Vtiger_Module::getInstance('Accounts');

$relationLabel  = 'Accounts';

$moduleInstance->setRelatedList(
      $accountsModule, $relationLabel, Array('ADD','SELECT')
);
```

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

To drop the relation between the modules use the following:

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

About setRelatedList API

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

<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 vtiger_crmentityrel table to keep track of relation between module records. Standard modules available in vtiger CRM 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 6: 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 Payslip module as Private

```
include_once('vtlib/Vtiger/Module.php');
$moduleInstance = Vtiger_Module::getInstance('Payslip');
$moduleInstance->setDefaultSharing('Private');
```

The <PERMISSION_TYPE> can be one of the following:

Public_ReadOnly
Public_ReadWrite
Public_ReadWriteDelete
Private

Step 7: 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 Payslip module

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

$moduleInstance = Vtiger_Module::getInstance('Payslip');

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

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

Optional Step: Module Events

Eventing API is supported from vtiger 5.1 onwards (read more here).

To check if your vtiger CRM supports Eventing use the following:

```
include_once('vtlib/Vtiger/Event.php');
boolean Vtiger_Event::isSupported();
```

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

```
include_once('vtlib/Vtiger/Event.php');
Vtiger_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(Vtiger_Event::hasSupport()) {
	Vtiger_Event::register(
		'Payslip', 'vtiger.entity.aftersave',
		'PayslipHandler', 'modules/Payslip/PayslipHandler.php'
	);

	Vtiger_Event::register(
		'Payslip', 'vtiger.entity.beforesave',
		'PayslipHandler', 'modules/Payslip/PayslipHandler.php'
	);
}
```

modules/Payslip/PayslipHandler.php

```
<?php

class PayslipHandler extends VTEventHandler {

	function handleEvent($eventName, $data) {

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

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

?>
```

Optional Step: Module Webservices

Webservices API is supported from vtiger 5.1 onwards (read more here).

You will need to invoke the setup API to enable the support for the custom modules.

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

$moduleInstance = Vtiger_Module::getInstance('Payslip');
$moduleInstance->initWebservice();
```

NOTE: When the module is imported the Webservice initialize API is automatically invoked.

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 yuu 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/Vtiger/Module.php');

$moduleInstance = Vtiger_Module::getInstance('ModuleName');

$moduleInstance->addLink(<LinkType>, <LinkLabel>, <LinkURL>);
```

LinkType	Type of Link like - DETAILVIEW : Will add a link in the 'More Actions' menu on the Detail View of the record. DETAILVIEWBASIC : Will add a link to the 'Actions' list on the Detail View of the Record. DETAILVIEWWIDGET : Will add a widget on the right hand side of the Detail View of the Record, similar to Tag Cloud widget. LISTVIEW : Will add a link under the 'More Actions' button on the List View of a module. LISTVIEWBASIC : Will add a button on the List View of the module similar to Delete, Mass Edit buttons.
LinkLabel	Label to use for the link when displaying
LinkURL	URL of the link. *You can use variables like $variablename$*

In module's ListView handler page (*modules/Payslip/ListView.php)* you will need this piece of code (before the call to $smarty->display()) :

```
include_once('vtlib/Vtiger/Link.php');

$customlink_params = Array('MODULE' => $currentModule, 'ACTION'=>
vtlib_purify($_REQUEST['action']), 'CATEGORY' => $category);

$smarty->assign('CUSTOM_LINKS', Vtiger_Link::getAllByType(getTabid($currentModule),
Array('LISTVIEW', 'LISTVIEWBASIC'), $customlink_params));
```

In module's DetailView handler page (*modules/Payslip/DetailView.php)* you will need this piece of code (before the call to $smarty->display()) :

```
include_once('vtlib/Vtiger/Link.php');

$customlink_params = Array('MODULE' => $currentModule, 'RECORD' => $focus->id, 'ACTION'=>
vtlib_purify($_REQUEST['action']));

$smarty->assign('CUSTOM_LINKS', Vtiger_Link::getAllByType(getTabid($currentModule),
Array('DETAILVIEW', 'DETAILVIEWBASIC', 'DETAILVIEWWIDGET'), $customlink_params));
```

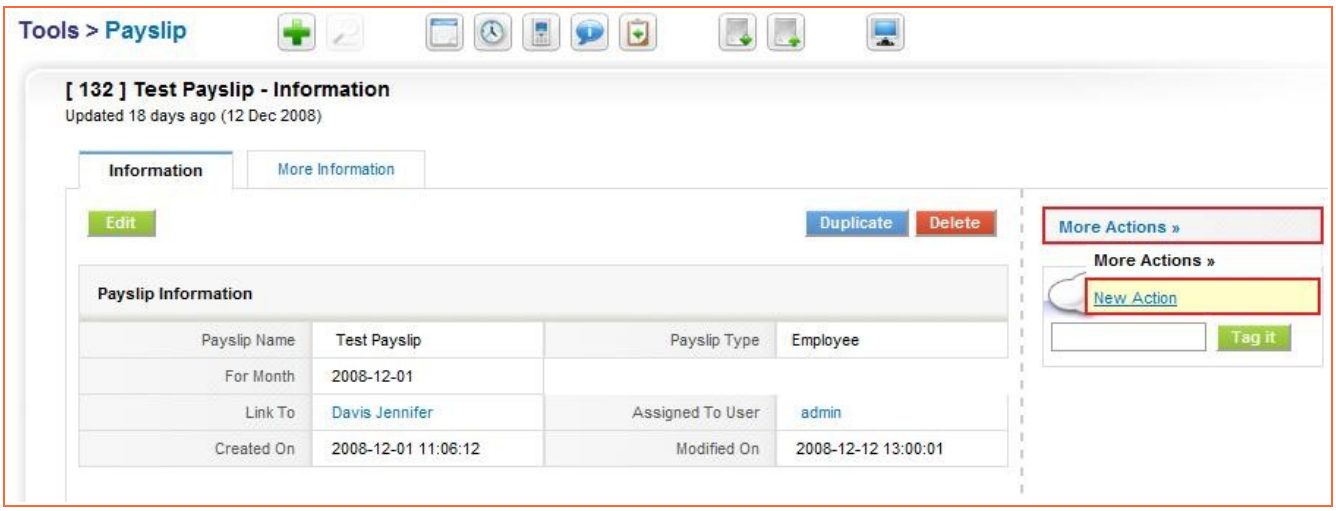
NOTE:
The $MODULE$, $ACTION$ and $RECORD$ variables in the LinkURL, will be replaced with the values set through DetailView.php

The $MODULE$, $ACTION$, $CATEGORY$ variables in the LinkURL, will be replaced with the values set through ListView.php

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

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

$moduleInstance = Vtiger_Module::getInstance('Payslip');

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

On the DetailView page you will find *More Actions* link. When you mouse hovers on this all the related custom links will be shown as a drop down. See the screenshot below:



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.

Final Completed Script (Backend)

Here is the complete script (vtlib.Test.Create.Module1.php) which creates the Payslip module

```
<?php
// Turn on debugging level
$Vtiger_Utils_Log = true;

include_once('vtlib/Vtiger/Menu.php');
include_once('vtlib/Vtiger/Module.php');

// Create module instance and save it first
$module = new Vtiger_Module();
$module->name = 'Payslip';
$module->save();

// Initialize all the tables required
$module->initTables();
/**
 * Creates the following table:
 * vtiger_payslip  (payslipid INTEGER)
 * vtiger_payslipcf(payslipid INTEGER PRIMARY KEY)
 * vtiger_payslipgrouprel((payslipid INTEGER PRIMARY KEY, groupname VARCHAR(100))
 */

// Add the module to the Menu (entry point from UI)
$menu = Vtiger_Menu::getInstance('Tools');
$menu->addModule($module);

// Add the basic module block
$block1 = new Vtiger_Block();
$block1->label = 'LBL_PAYSLIP_INFORMATION';
$module->addBlock($block1);

// Add custom block (required to support Custom Fields)
$block2 = new Vtiger_Block();
$block2->label = 'LBL_CUSTOM_INFORMATION';
$module->addBlock($block2);

/** Create required fields and add to the block */
$field1 = new Vtiger_Field();
$field1->name = 'PayslipName';
$field1->table = $module->basetable;
$field1->column = 'payslipname';
$field1->columntype = 'VARCHAR(255)';
$field1->uitype = 2;
$field1->typeofdata = 'V~M';
$block1->addField($field1); /** Creates the field and adds to block */

// Set at-least one field to identifier of module record
$module->setEntityIdentifier($field1);

$field2 = new Vtiger_Field();
$field2->name = 'PayslipType';
$field2->label = 'Payslip Type';
$field2->columntype = 'VARCHAR(100)';
$field2->uitype = 15;
$field2->typeofdata = 'V~O';// Varchar~Optional
$block1->addField($field2); /** table and column are automatically set */

$field2->setPicklistValues( Array ('Employee', 'Trainee') );

$field3 = new Vtiger_Field();
$field3->name = 'Month';
```

```
$field3->uitype = 23;
$field3->typeofdata = 'D~M'; // Date~Mandatory
$block1->addField($field3);  /** table, column, label, set to default values */

$field4 = new Vtiger_Field();
$field4->name = 'LinkTo';
$field4->label= 'Link To';
$field4->table = 'vtiger_payslip';
$field4->column = 'linkto';
$field4->columntype = 'VARCHAR(100)';
$field4->uitype = 10;
$field4->typeofdata = 'V~O';
$field4->helpinfo = 'Relate to an existing contact';
$block1->addField($field4);
$field4->setRelatedModules(Array('Contacts'));

/** Common fields that should be in every module, linked to vtiger CRM core table
*/
$field5 = new Vtiger_Field();
$field5->name = 'assigned_user_id';
$field5->label = 'Assigned To';
$field5->table = 'vtiger_crmentity';
$field5->column = 'smownerid';
$field5->uitype = 53;
$field5->typeofdata = 'V~M';
$block1->addField($field5);

$field6 = new Vtiger_Field();
$field6->name = 'CreatedTime';
$field6->label= 'Created Time';
$field6->table = 'vtiger_crmentity';
$field6->column = 'createdtime';
$field6->uitype = 70;
$field6->typeofdata = 'T~O';
$field6->displaytype= 2;
$block1->addField($field6);

$field7 = new Vtiger_Field();
$field7->name = 'ModifiedTime';
$field7->label= 'Modified Time';
$field7->table = 'vtiger_crmentity';
$field7->column = 'modifiedtime';
$field7->uitype = 70;
$field7->typeofdata = 'T~O';
$field7->displaytype= 2;
$block1->addField($field7);
/** END */

// Create default custom filter (mandatory)
$filter1 = new Vtiger_Filter();
$filter1->name = 'All';
$filter1->isdefault = true;
$module->addFilter($filter1);

// Add fields to the filter created
$filter1->addField($field1)->addField($field2, 1)->addField($field5, 2);

// Create one more filter
$filter2 = new Vtiger_Filter();
$filter2->name = 'All2';
$module->addFilter($filter2);

// Add fields to the filter
$filter2->addField($field1);
```

```
$filter2->addField($field2, 1);

// Add rule to the filter field
$filter2->addRule($field1, 'CONTAINS', 'Test');

/** Associate other modules to this module */
$module->setRelatedList(Vtiger_Module::getInstance('Accounts'), 'Accounts',
Array('ADD','SELECT'));

/** Set sharing access of this module */
$module->setDefaultSharing('Private');

/** Enable and Disable available tools */
$module->enableTools(Array('Import', 'Export'));
$module->disableTools('Merge');

?>
```

Executing Module Creation Script

1. To execute the vtlib.Test.Create.Module1.php script, open http://localhost/vtigercrm/vtlib.Test.html

2. Click on Create Payslip Module to test creation of Payslip Module

Step 8: 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 'PaySlip' 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
index.php	Module entry point through Menu	index.php
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
CustomView.php	Custom view or Filter handler	CustomView.php
Delete.php	Module record deletion handler	Delete.php
DetailView.php	Detail view handler	DetailView.php
DetailViewAjax.php	Detail view ajax edit handler	DetailViewAjax.php
EditView.php	Edit view handler	EditView.php
ExportRecords.php	Module record export handler	ExportRecords.php
Import.php	Module records import handler	Import.php
ListView.php	List view handler	ListView.php
Popup.php	Popup selection handler for this module record	Popup.php
QuickCreate.php	Quick creation handler	QuickCreate.php
Save.php	Module record save handler	Save.php

Example ModuleDir	Purpose	File under Payslip
TagCloud.php	Tag cloud handler	TagCloud.php
updateRelations.php	Related list record handler (save/delete)	updateRelations.php

Files specific to vtiger 5.1.0 version.

Example ModuleDir	Purpose	File under Payslip
MassEdit.php	Mass Record Edit handler	MassEdit.php
MassEditSave.php	Mass Edit Record Save handler	MassEditSave.php
FindDuplicateRecords.php	Find Duplicate Records handler	FindDuplicateRecords.php
ProcessDuplicates.php	Process Duplicate Records (merging) handler	ProcessDuplicates.php
ListViewPagging.php	List View Paging handler	ListViewPagging.php
UnifiedSearch.php	Global Search handler	UnifiedSearch.php

Files to be updated for a vtlib module build on version 5.1.0, to make it work on 5.2.0 (from vtlib/ModuleDir/5.2.0) :

Example ModuleDir	Purpose of Upgrade	File under Payslip
CallRelatedList.php	Related List View Changes	CallRelatedList.php
DetailViewAjax.php	Related List View Changes	DetailViewAjax.php
ListView.php	Order by issue fixed and Query Generator support	ListView.php
ModuleFile.php	Query Generator support	Payslip.php

Language File Preparation

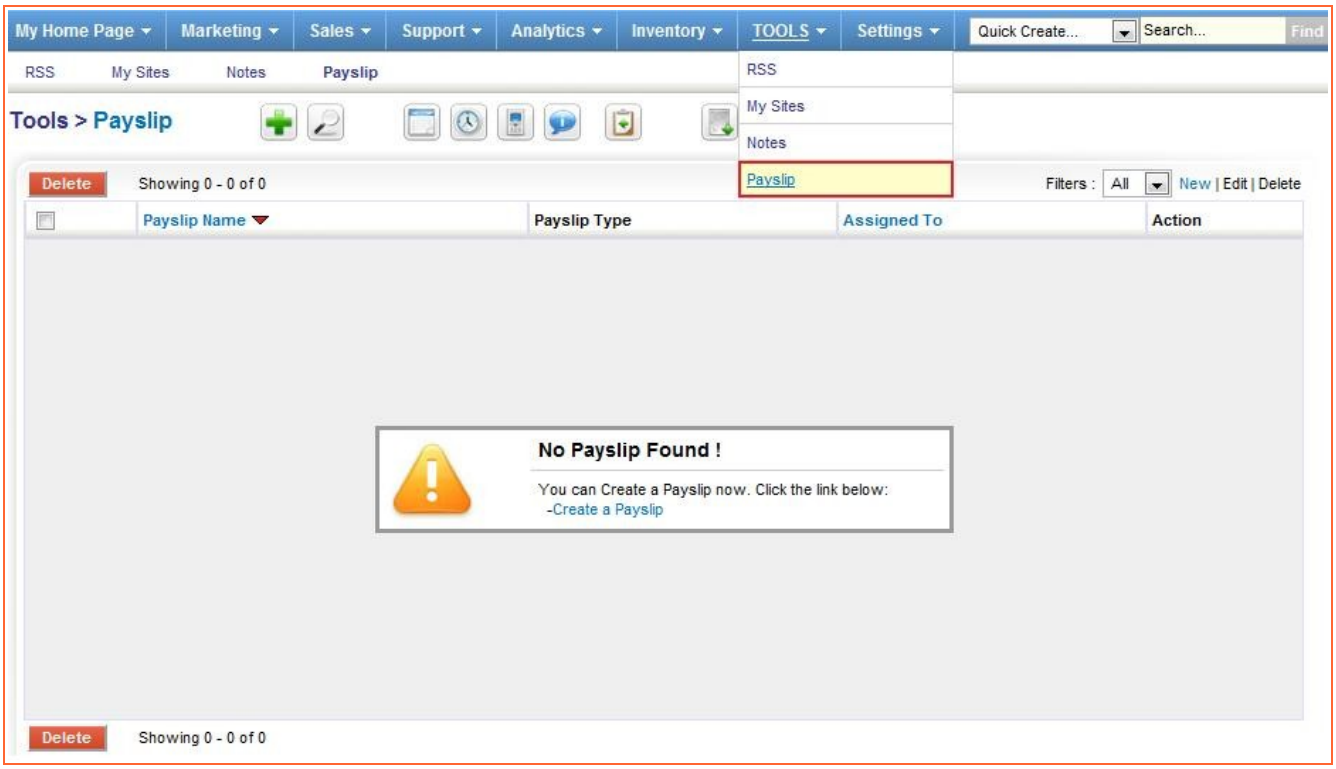
Update the translation mapping in the module language file (modules/<NewModuleName>/language/en_us.lang.php)

Some of the mandatory mapping that should be provided are as follows:

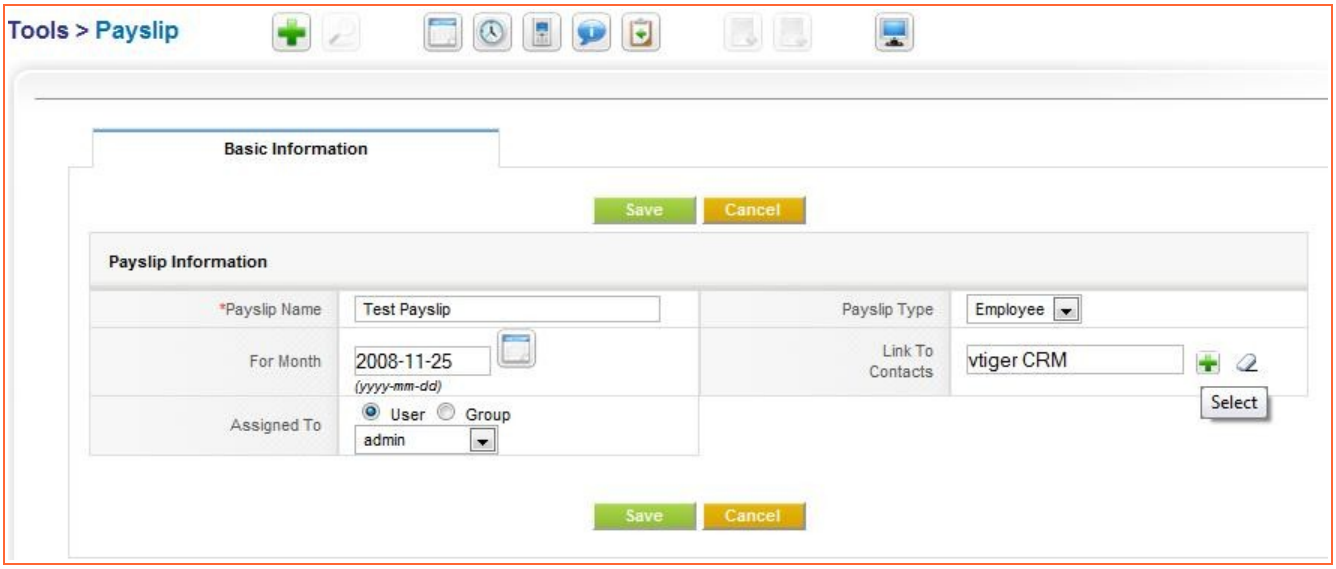
```
$mod_strings = array (

'NewModuleName'          => 'NewModuleName_Translation',

'SINGLE_NewModuleName'   => 'NewModuleNameForSingleRecord',

'LBL_CUSTOM_INFORMATION' => 'Custom Information',

// Label used for your module field
'FieldLabel'             => 'Field Label Translation'

);
```

New Module Tour

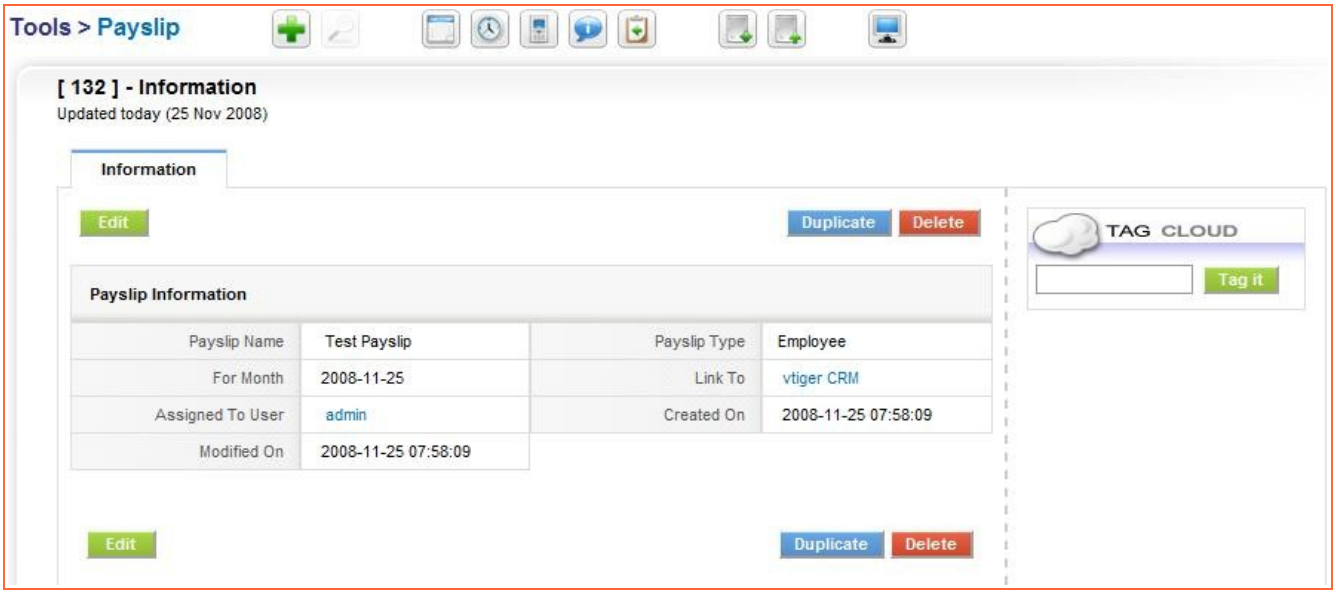
List view



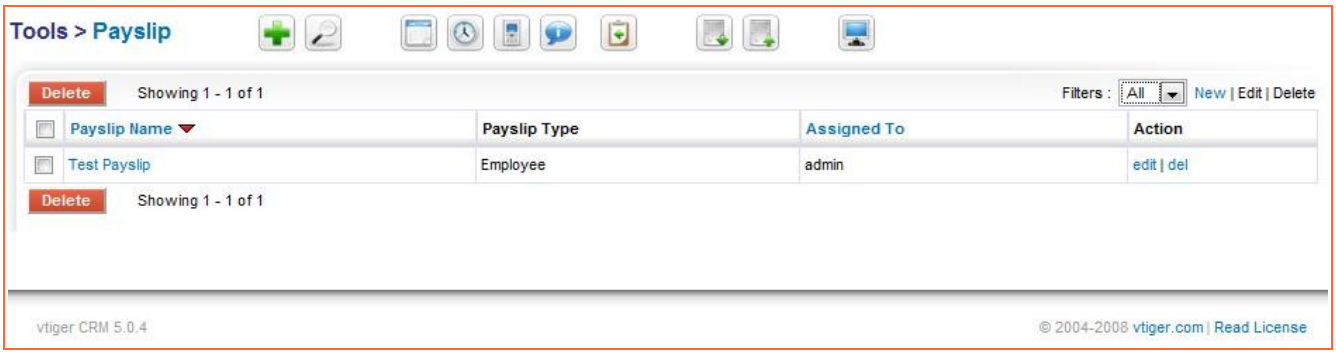
Create view



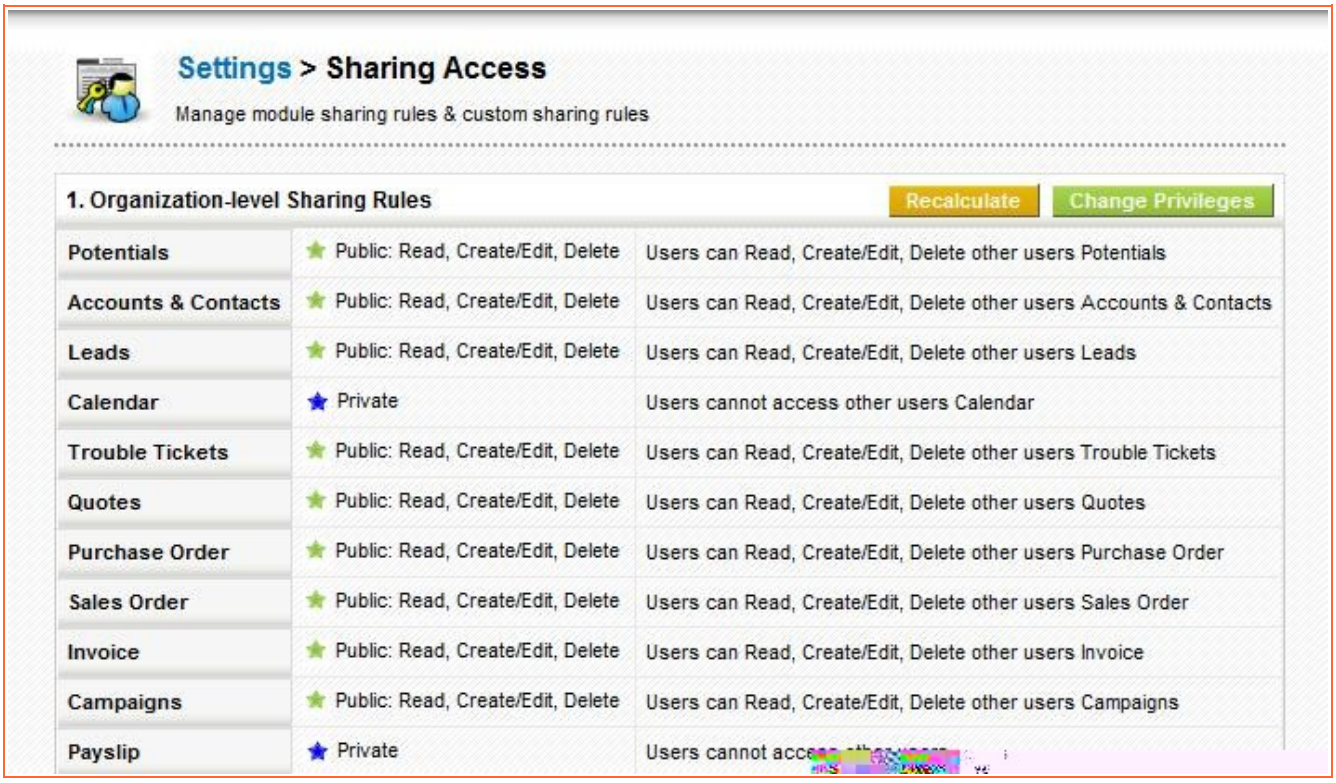
Detail view



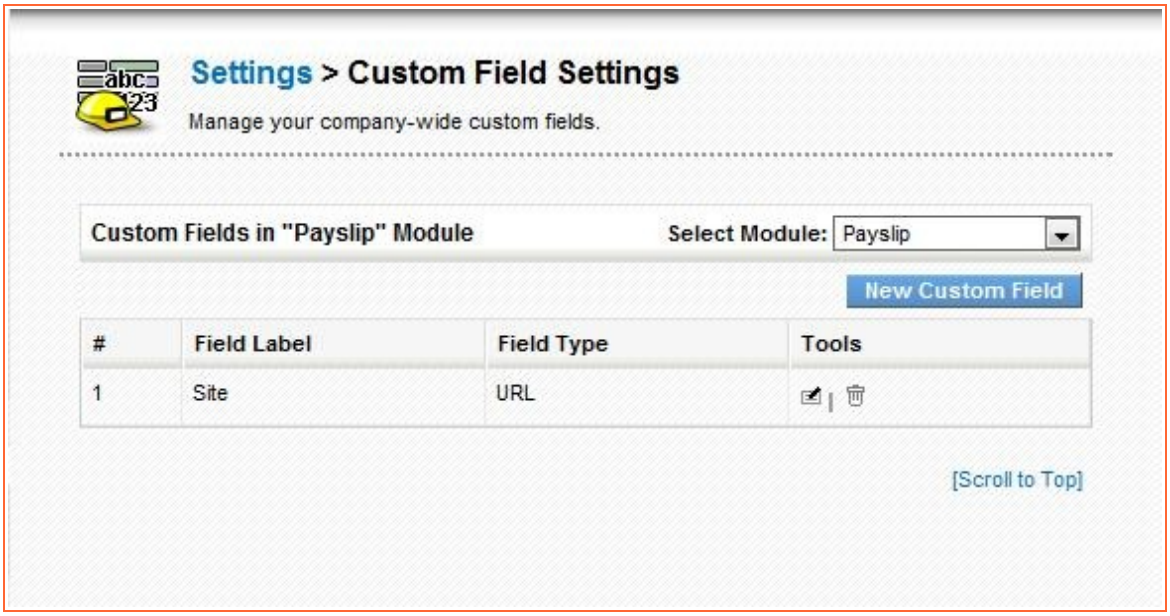
List view



Sharing Access



Custom Fields



Step 9: Packaging

Package Export

vtlib provides API to export module as a zip (package) file which can used for importing through Module Manger.

```
require_once('vtlib/Vtiger/Package.php');
require_once('vtlib/Vtiger/Module.php');
$package = new Vtiger_Package();
$package->export('<MODULE Instance>', '<DESTINATION DIR>', '<ZIPFILE NAME>', <DIRECT DOWNLOAD>);
```

<MODULE Instance>	Vtiger_Module instance to be exported (packaged)
<DESTINATION DIR>	(Optional: Default=test/vtlib) Directory where the zipfile output should be created.
<ZIPFILE NAME>	(Optional: Default=modulename-timestamp.zip) Zipfile name to use for the output file.
<DIRECT DOWNLOAD>	(Optional: Default=false) If true, the zipfile created will be streamed for download and zipfile will be deleted after that.

Example:

```
require_once('vtlib/Vtiger/Package.php');
require_once('vtlib/Vtiger/Module.php');
$package = new Vtiger_Package();
$package->export(
      Vtiger_Module::getInstance('Payslip'),
      'test/vtlib',
      'Payslip-Export.zip',
      true
);
```

NOTE: Please make sure test/vtlib directory exists under vtigercrm root directory and is writeable.

Package Structure

The exported zipfile (package) has the following structure:

```
manifest.xml
modules/
        ModuleName/
                <Module Related Files>
                language/
                        en_us.lang.php
                        <Other language Files>

templates/
        <Smarty templates of the Module>
```

manifest.xml has the meta information that will be useful during the import process as shown:

```
<?xml version="1.0" encoding="utf-8"?>
<module>
  <exporttime>YYYY-MM-DD hh:mm:ss</exporttime>
  <name>MODULE NAME</name>
  <version>1.0</version>
  <label>MODULE LABEL</label>
  <parent>MENU</parent>

  <dependencies>
      <vtiger_version>VTIGER_VERSION_NUMBER</vtiger_version>
     <vtiger_max_version>VTIGER_MAX_VERSION_SUPPORTED</vtiger_max_version>
  </dependencies>

  <tables>
    <table>
      <name>TABLENAME</name>
      <sql>TABLE SQL</sql>
    </table>
  </tables>

  <blocks>
    <block>
      <label>BLOCK LABEL</label>
      <fields>
        <field>
          <fieldname>payslipname</fieldname>
          <columnname>payslipname</columnname>
          <uitype>UI TYPE</uitype>
          <tablename>TABLE NAME</tablename>
          <generatedtype>GEN TYPE</generatedtype>
          <fieldlabel>FIELD LABEL</fieldlabel>
          <readonly>READONLY</readonly>
          <presence>PRESENCE</presence>
          <selected>SELECTED</selected>
          <maximumlength>MAXLEN</maximumlength>
          <typeofdata>TYPEOFDATA</typeofdata>
          <quickcreate>QUICKCREATE</quickcreate>
          <displaytype>DISPTYPE</displaytype>
          <info_type>INFOTYPE</info_type>
          <helpinfo><![CDATA[HELP INFORMATION]]></helpinfo>
          <masseditable>MASSEDIT VALUE</masseditable>
        </field>
      </fields>
    </block>
  </blocks>

  <customviews>
```

```
    <customview>
      <viewname>VIEWNAME</viewname>
      <setdefault>0</setdefault>
      <setmetrics>1</setmetrics>
      <fields>
        <field>
          <fieldname>FIELDNAME</fieldname>
          <columnindex>0</columnindex>
        </field>
      </fields>
    </customview>
  </customviews>
  <sharingaccess>
    <default>private</default>
  </sharingaccess>

  <actions>
    <action>
      <name>Export</name>
      <status>enabled</status>
    </action>
    <action>
      <name>Import</name>
      <status>enabled</status>
    </action>
  </actions>

  <customlinks>
    <customlink>
      <linktype>DETAILVIEW</linktype>
      <linklabel>Visit Site</linklabel>
      <linkurl><![CDATA[http://www.vtiger.com]]></linkurl>
      <linkicon><![CDATA[themes/images/vtiger-paw.jpg]]></linkicon>
      <sequence>0</sequence>
     </customlink>
  <customlinks>

  <events>
    <event>
      <eventname>EVENT_NAME</eventname>
      <classname>EVENT_HANDLER_CLASS</classname>
      <filename>EVENT_HANDLER_CLASS_FILE</filename>
      <condition><![CDATA[modulename in ['MODULENAME'] ]]></condition>
    </event>
  </events>

</module>
```

Package Import

You can import a module from package (zip) file using the following API

```
require_once('vtlib/Vtiger/Package.php');

$package = new Vtiger_Package();
$package->import(<MODULE ZIPFILE>, <overwrite>);
```

<MODULE ZIPFILE>	Module zipfile (package).
<overwrite>	(Optional: Default=false) Overwrite the existing module directory if present *NOTE: overwrite flag is ignored currently. It will be implemented in future.* *Please make sure to check for directory non-existence before importing.*

The package file should be validated before Importing which can be done using the following API

```
require_once('vtlib/Vtiger/Package.php');
$package = new Vtiger_Package();
$package->checkzip(<MODULE ZIPFILE>);
```

<MODULE ZIPFILE>	Module zipfile (package).

checkzip returns **true** if package structure in zipfile is as specified in the section **Package Export** above.

Detecting Module Name being Imported

```
require_once('vtlib/Vtiger/Package.php');
$package = new Vtiger_Package();
$package->getModuleNameFromZip(<MODULE ZIPFILE>);
```

<MODULE ZIPFILE>	Module zipfile (package).

getModuleNameFromZip returns **ModuleName** if checkZip succeeds.

Example:

```
require_once('vtlib/Vtiger/Package.php');
require_once('vtlib/Vtiger/Module.php');

$package = new Vtiger_Package();
$module = $package->getModuleNameFromZip('test/vtlib/Payslip.zip');

$module_exists = false;
$module_dir_exists = false;
if($module == null) {
	echo “Module zipfile is not valid!”;
} else if(Vtiger_Module::getInstance($module)) {
	echo "$module already exists!";
	$module_exists = true;
} else if(is_dir("modules/$module")) {
	echo "$module folder exists! Overwrite?";
	$module_dir_exists = true;
}
if($module_exists == false && $module_dir_exists == false) {
	$package->import('test/vtlib/Payslip.zip');
}
```

Package Upgrade

NOTE: Currently this module upgrade feature does not support deletion and modification of exiting module fields. Before you use this feature, please ensure your modified module does not change or delete existing fields.

You can upgrade a module that was imported earlier using the following API:

```
require_once('vtlib/Vtiger/Package.php');
require_once('vtlib/Vtiger/Module.php');

$package = new Vtiger_Package();
$package->update(<ModuleInstance>, <MODULE ZIPFILE>, <overwrite>);
```

<ModuleInstance>	Vtiger_Module instance which needs to be upgraded.
<MODULE ZIPFILE>	Module zipfile (package).
<overwrite>	(Optional: Default=true) Overwrite the existing module directory if present

Example:

```
require_once('vtlib/Vtiger/Package.php');
require_once('vtlib/Vtiger/Module.php');

$package = new Vtiger_Package();
$moduleInstance = Vtiger_Module::getInstance('Payslip');
$package->update($moduleInstance, 'test/vtlib/Payslip.zip');
```

Limitations

1. Any property change to existing block or field will not applied during module upgrade.

NOTE: Look at Adding_Migration_Details section to know more about adding migration information to the package file through manifest.xml

Extension Module

Module Manager lets you install an extension module provided the manifest.xml (in package) has the following information. *This feature is available from vtiger CRM 5.1.0 onwards only.*

manifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<module>
      <type>extension</type>
      <name>MODULENAME</name>
      <label>MODULE LABEL</label>
      <parent>Tools</parent>
       <version>1.0</version>
      <dependencies>
            <vtiger_version>5.1.0</vtiger_version>
      </dependencies>
      <tables>
            <table>
                  <name>TABLE-NAME</name>
                  <sql><![CDATA[CREATE_TABLE_SQL]]></sql>
            </table>
      </tables>
      <events>
            <event>
                  <eventname>EVENT_NAME</eventname>
                  <classname>EVENT_HANDLER_CLASS</classname>
                  <filename>EVENT_HANDLER_CLASS_FILE</filename>
                  <condition><![CDATA[modulename in ['MODULENAME']]]></condition>
            </event>
      </events>
</module>
```

type	Mandatory	Should have the value **extension**
name	Mandatory	Module name (should not contain spaces or special characters)
label	Mandatory	Label used to display on the UI
parent	Optional	Menu to which this Module needs to be attached
dependencies (vtiger_version)	Mandatory	Version for which the package is intended for
tables	Optional	Tables that needs to be created during installation of module
events	Optional	Events that needs to be registered during installation of module

Package File

The following file structure is recommended for extension module package (zip file).

```
manifest.xml

modules/
    MODULENAME/
        language/
            en_us.lang.php
    index.php
    <other module files>

templates/
    <smarty templates>
```

Bundles

Module Manager lets you install bundles provided the manifest.xml (in package) has the following information. *This feature is available from vtiger CRM 5.2.0 onwards only.*

manifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<module>
	<name>MODULENAME</name>
	 <version>1.0</version>
	<dependencies>
		<vtiger_version>5.1.0</vtiger_version>
		<vtiger_max_version>5.*</vtiger_max_version>
	</dependencies>
	<modulebundle>true</modulebundle>
	<modulelist>
		<dependent_module>
			<name>BUNDLE_PART_1</name>
			<install_sequence>1</install_sequence>
			<filepath>MODULE_FILE_1.zip</filepath>
		</dependent_module>
		<dependent_module>
			<name>BUNDLE_PART_2</name>
			<install_sequence>2</install_sequence>
			<filepath>MODULE_FILE_2.zip</filepath>
		</dependent_module>
	</modulelist>
</module>
```

modulebundle	Mandatory	Should have the value **true**
name	Mandatory	Module name (should not contain spaces or special characters)
dependencies (vtiger_version)	Mandatory	Version for which the package is intended for
modulelist	Optional	List of modules which are part of the bundle with their name, installation sequence and the file name

Package File

The following file structure is recommended for bundles package (zip file).

```
manifest.xml

MODULENAME_1.zip

MODULENAME_2.zip

...
```

Language Pack

Module Manager lets you install language packs to your vtiger CRM installation. The Language package should follow the package structure as explained below:

manifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<module>
      <type>language</type>
      <name>English</name>
      <label>US English</label>
      <prefix>en_us</prefix>
       <version>1.0</version>
      <dependencies>
            <vtiger_version>5.1.0</vtiger_version>
      </dependencies>
</module>
```

type	Mandatory	Should have the value **language**
name	Optional	Language pack complete name
label	Mandatory	Label to identify the language on the UI (on login page)
prefix	Mandatory	The filename prefix used for language file. Only these files will be extracted from the package file during installation.
dependencies (vtiger_version)	Mandatory	Version for which the package is intended for.

Package File

The following file structure is recommended for extension language package (zip file).

```
manifest.xml

modules/
    Accounts/
        language/
            <prefix>.lang.php

    Contacts/
        language/
            <prefix>.lang.php

    Leads/
        language/
            <prefix>.lang.php

    ...
```

Adding License

The manifest.xml of the package can contain license information which will be displayed to user during Module Manager installation process. You will need to add **<license>** node in the manifest.xml as described below:

Inline License:

```
<module>
	<name>MODULENAME</name>
	<label>MODULE_LABEL</label>
	 <version>1.0</version>

	<dependencies>
		<vtiger_version>5.0.4</vtiger_version>
	</dependencies>
	<license>
		<inline><![CDATA[This is under vtiger Public License ]]></inline>
	</license>
	...
</module>
```

License from File:

You can specific the LICENSEFILE in the package that contains the License information.

```
<module>
	<name>MODULENAME</name>
	<label>MODULE_LABEL</label>
	 <version>1.0</version>
	<dependencies>
		<vtiger_version>5.0.4</vtiger_version>
	</dependencies>
	<license>
		<file>LICENSEFILE</file>
	</license>
	...
</module>
```

Adding Migration Details

Module Manager supports upgrade of modules built with vtlib. In some cases, custom schema changes and data migration will be required for these module upgrades.

When a new version of a module is released it might have schema changes w.r.t older version.

The upgrade process might not be complete unless required schema changes and data migration are applied. In such cases, you can add the migration information in your manifest.xml as described below:

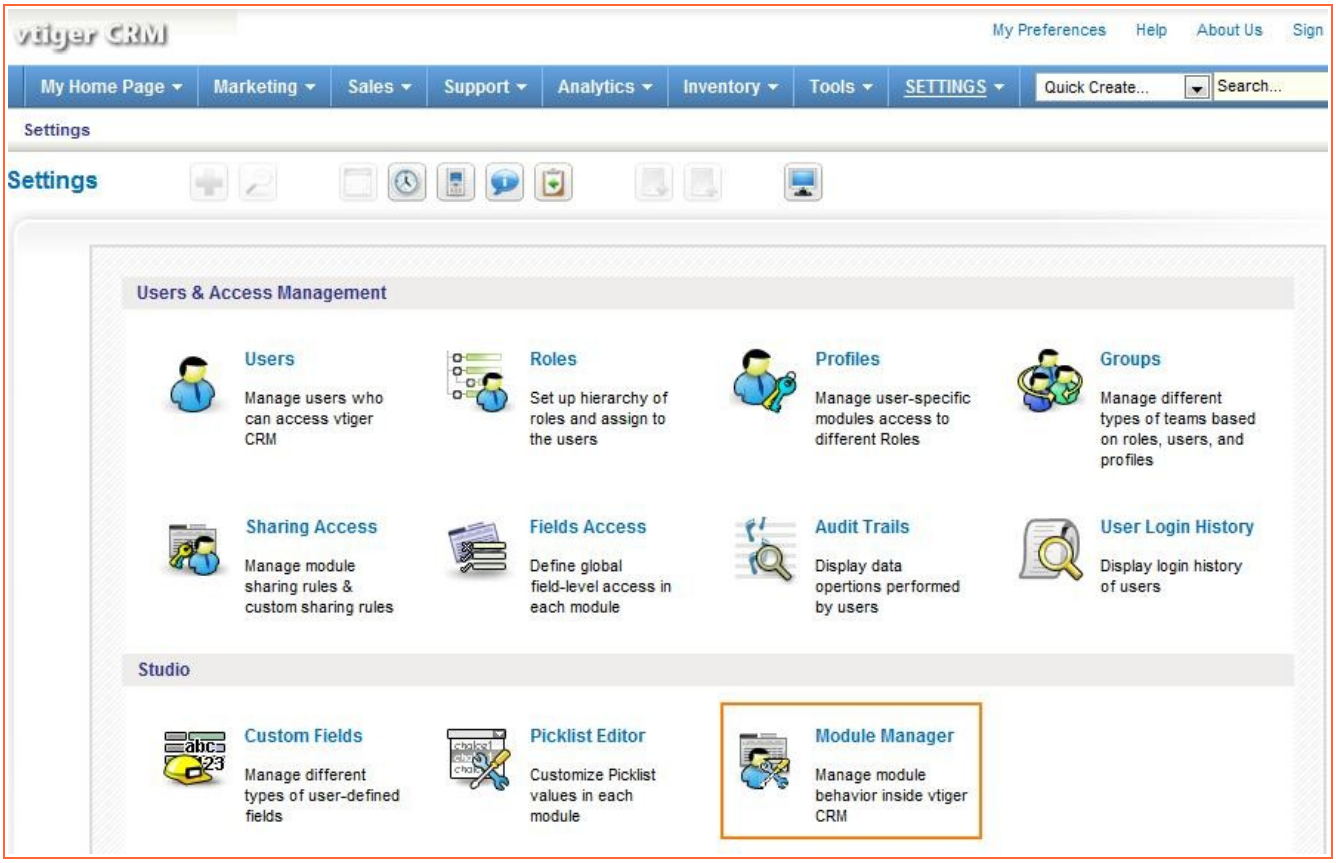
```
<?xml version="1.0" encoding="utf-8"?>
<module>
	<name>MODULENAME</name>
	<label>MODULE LABEL</label>
	<parent>Tools</parent>
	 <version>1.2</version>

	<dependencies>
		<vtiger_version>5.0.4</vtiger_version>
	</dependencies>

	 <migrations>
		 <migration version='1.0'>
			<tables>
				<table>
					<name>TABLE-NAME</name>
					<sql><![CDATA[ALTER TABLE MyTable ADD COLUMN
NewColumn INT]]></sql>
				</table>
			</tables>
		 </migration>
		 <migration version='1.1'>
			<tables>
				<table>
					<name>TABLE-NAME</name>
					<sql><![CDATA[UPDATE MyTable SET NewColumn=1 WHERE
NewColumn is NULL]]></sql>
				</table>
			</tables>
		 </migration>
	 </migrations>

	<tables>
		<table>
			<name>TABLE-NAME</name>
			<sql><![CDATA[CREATE_TABLE_SQL]]></sql>
		</table>
	</tables>
	...
</module>
```

NOTE: The above snippet of manifest.xml is for version 1.2 of a module. It contains migration information for version 1.0 and 1.1

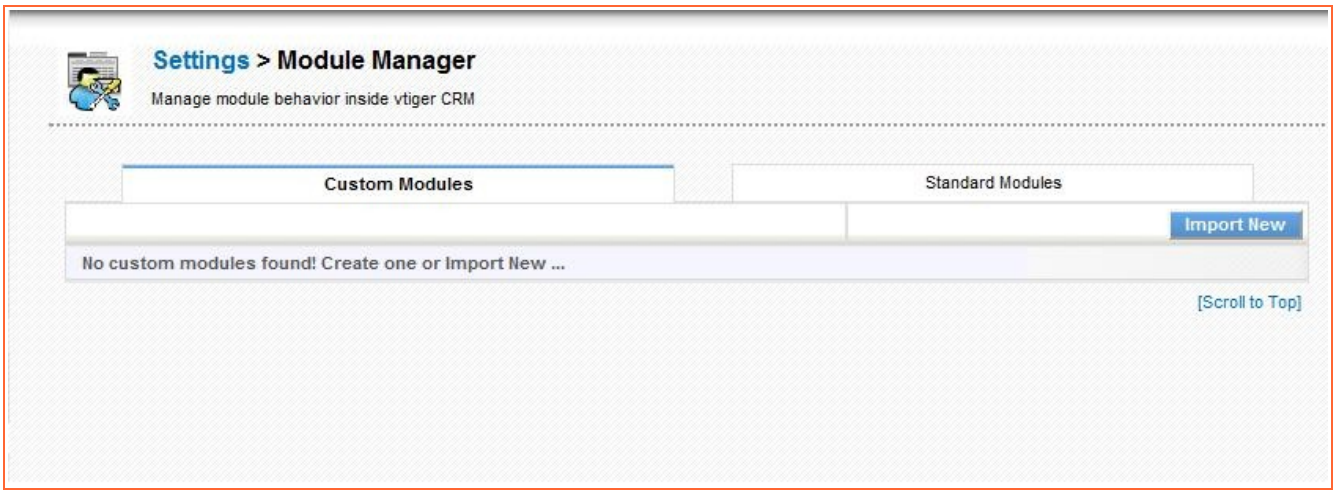
Migration node details	
<migration version='1.0'...	This version indicates the source version (earlier version) for which the migration should be applied.
<tables> <table> <name>... <sql>...	Table name to migrate. SQL to use for migration.

Module Manager

Once vtlib is installed, it provides the Module Manager configuration tool under Settings. With this you can enable, disable or control settings of vtiger CRM modules. On disabling a module, it won't be shown on the Menu and access is restricted (including for the administrator).

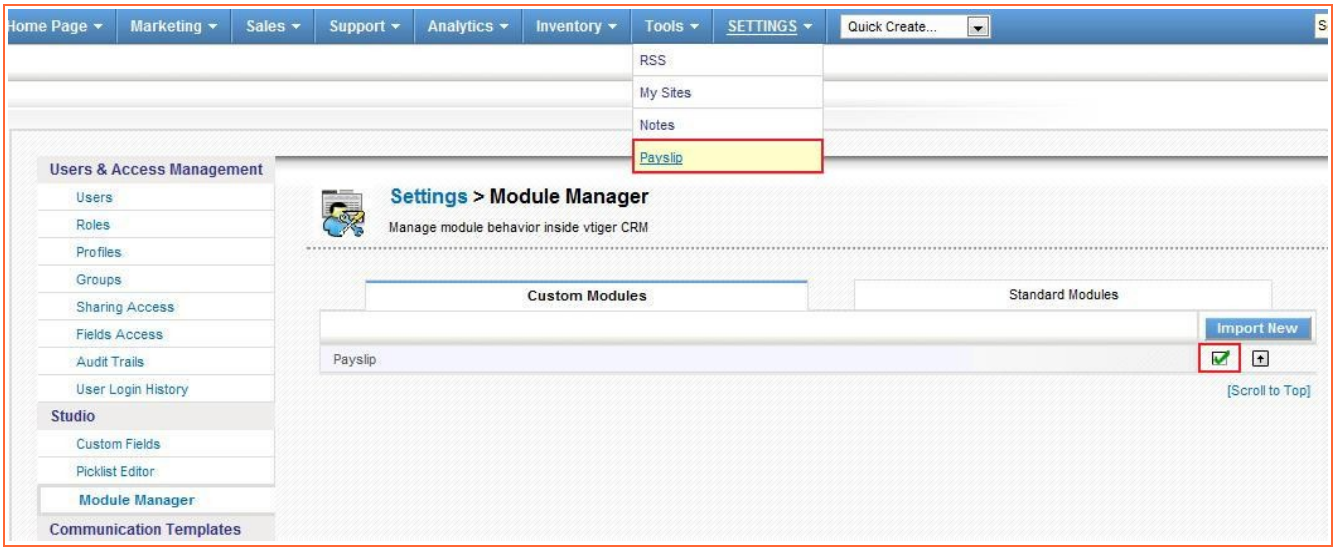


Modules are categorized as Standard (which are provided as a core part of vtiger CRM), and Custom (which you have imported or created)



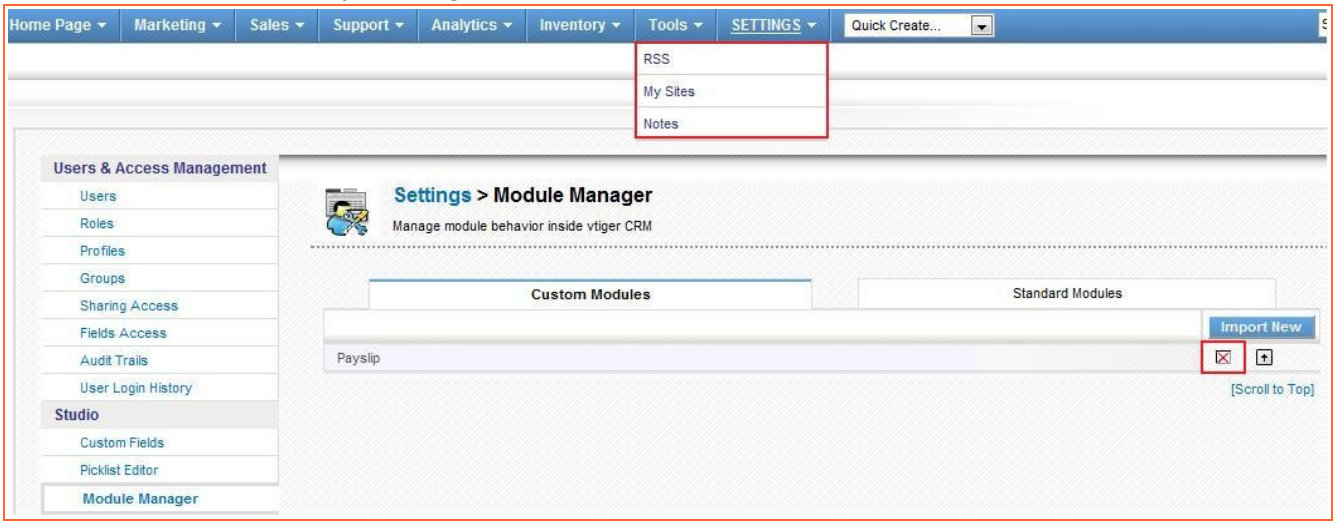
Disabling Module

You can disable module by clicking on the green tick mark icon.



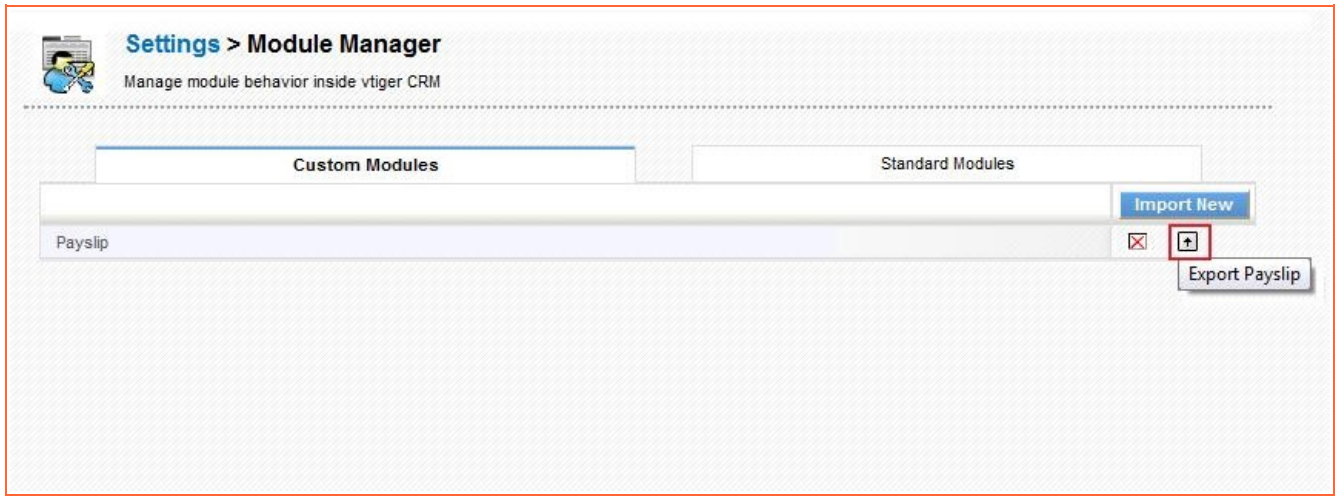
Enabling Module

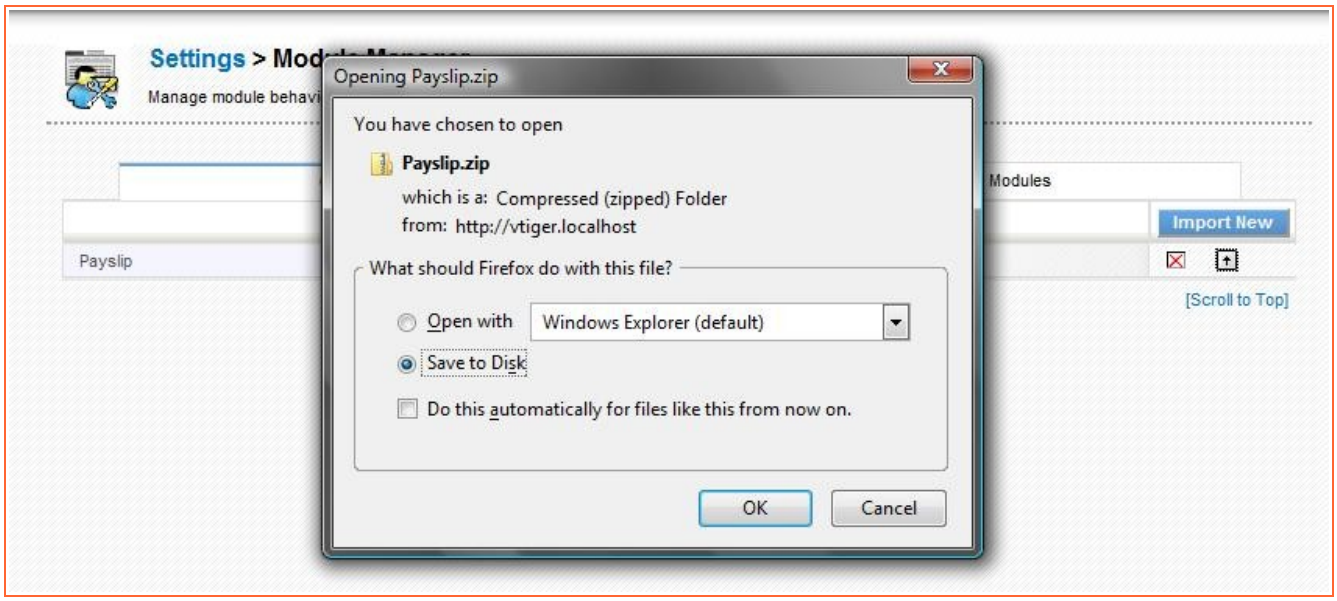
You can enable module by clicking on the red tick mark icon.



Exporting Module

Click on the UP arrow icon in the module manager, which will export the module as a zip file.

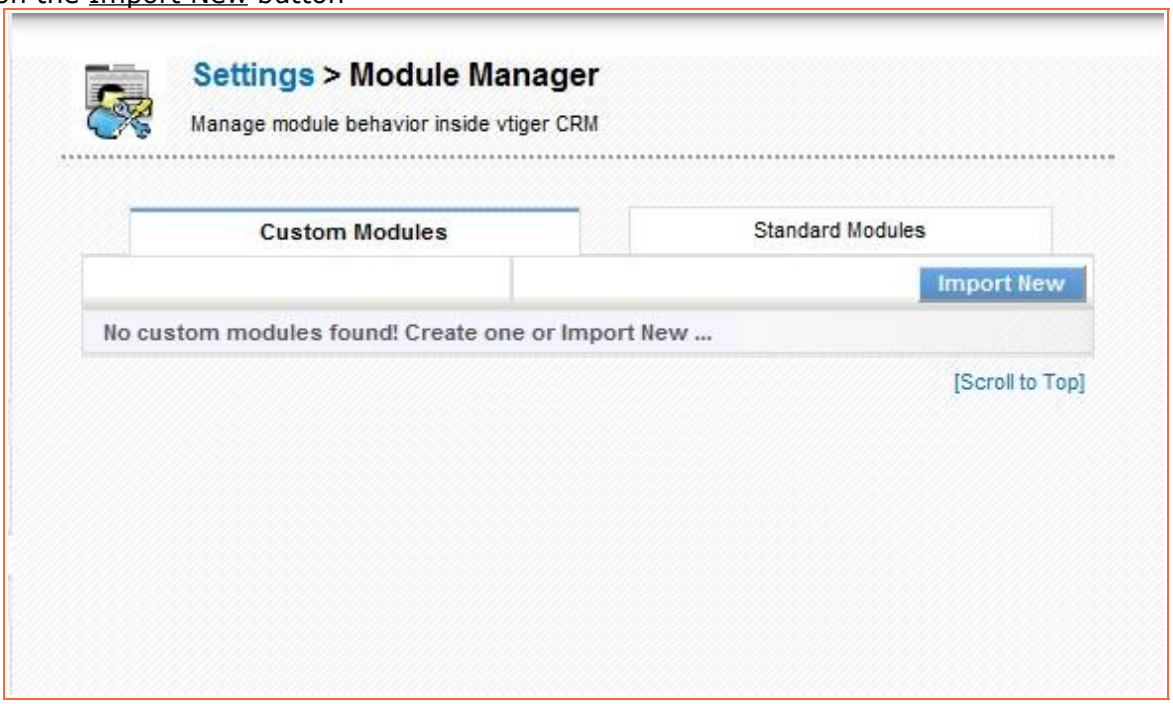




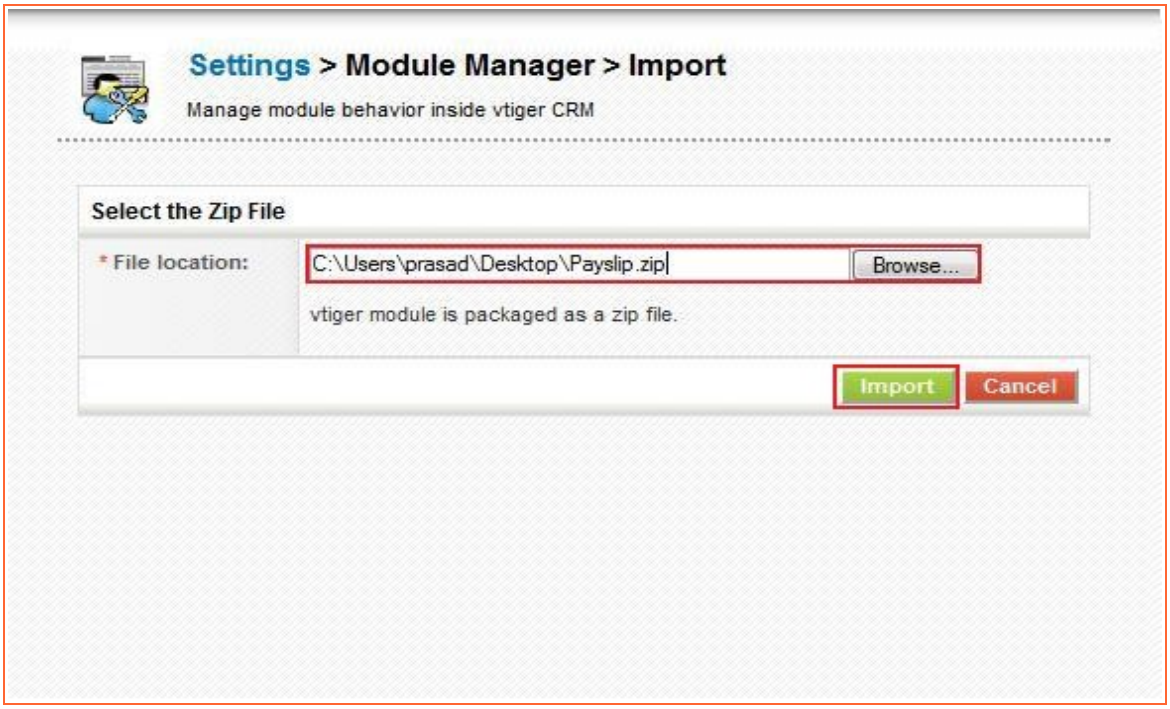
Importing Module

Module manager will let you import new modules. Follow the steps given below:

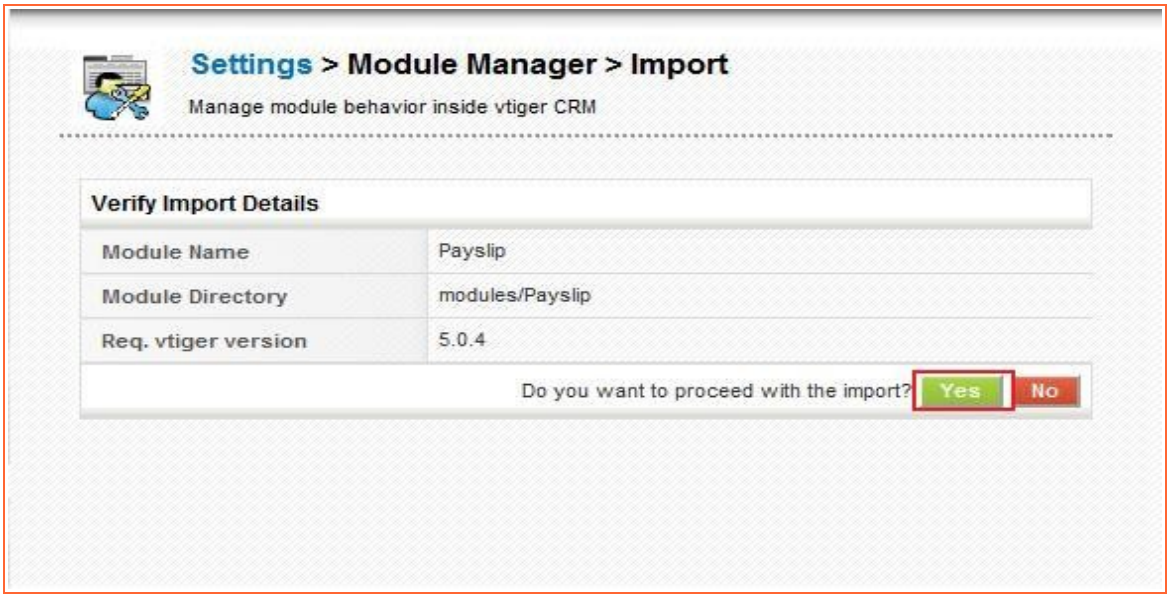
Click on the Import New button



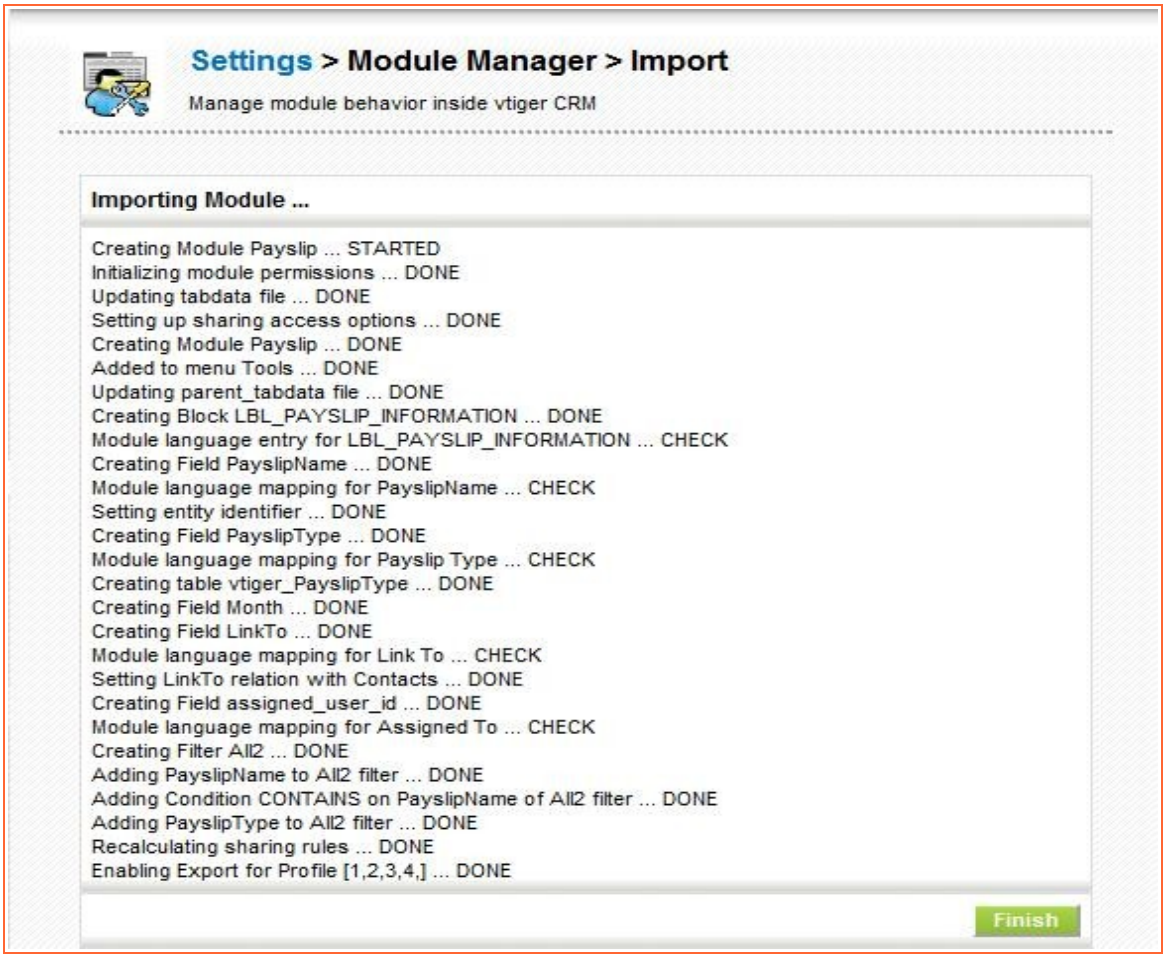
Select the module zip (package) file that was previously exported or created.



Verify the import details parsed from zipfile. Click Yes to proceed or No to cancel.



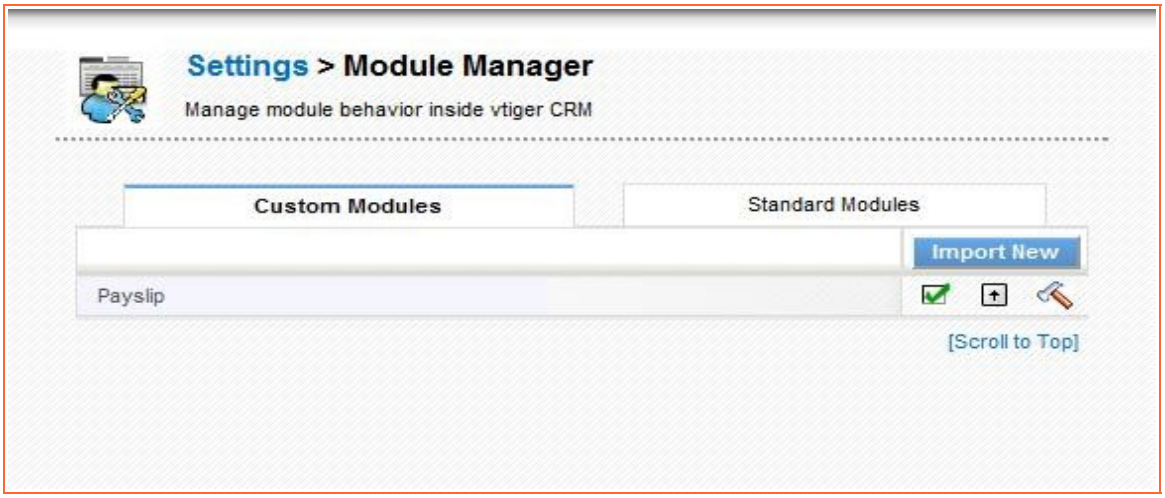
Click on Finish to complete the module import.



NOTE: If you are trying to import a module which already exists or a directory which is present in the modules folder you will see the following message.

Module Specific Settings

A module can have its own specific settings. In such cases, Settings.php should be created under the module folder. This file will be invoked (if found) when Settings icon is clicked.



Example: Sample Settings.php for Payslip module

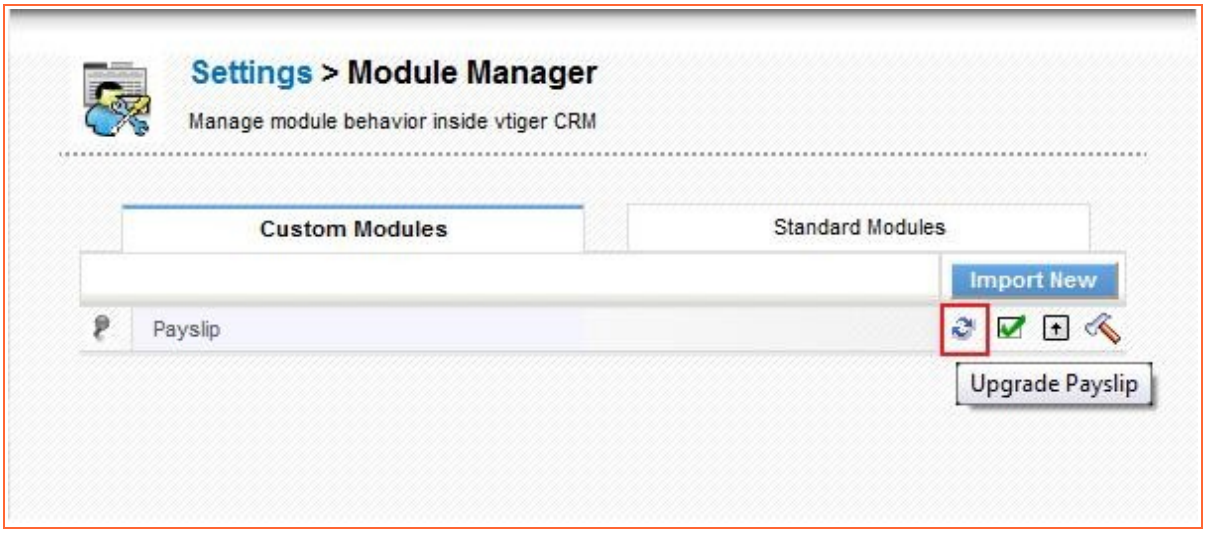
```
<?php

$thisModule = $_REQUEST['formodule'];

require_once('Smarty_setup.php');

$smarty = new vtigerCRM_Smarty();

// Use the module specific template file
// modules/Payslip/MySettings.tpl

$smarty->display(vtlib_getModuleTemplate('Payslip', 'MySettings.tpl'));
?>
```

Module Upgrade

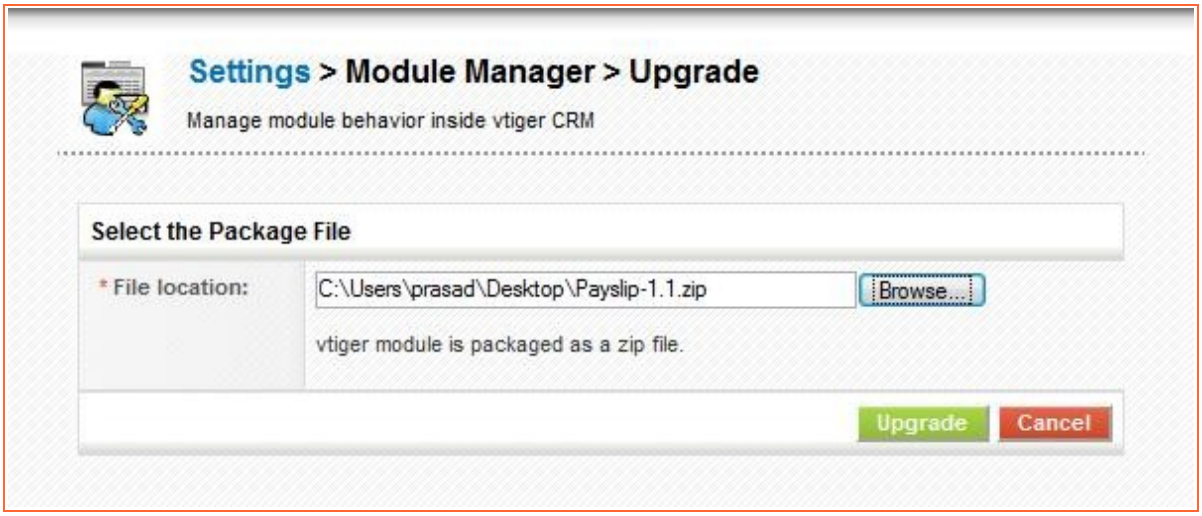
Upgrading the module to next version is now possible through Module Manager.

NOTE: Currently this module upgrade feature does not support deletion and modification of exiting module fields. Before you use this feature, please ensure your modified module does not change or delete existing fields.

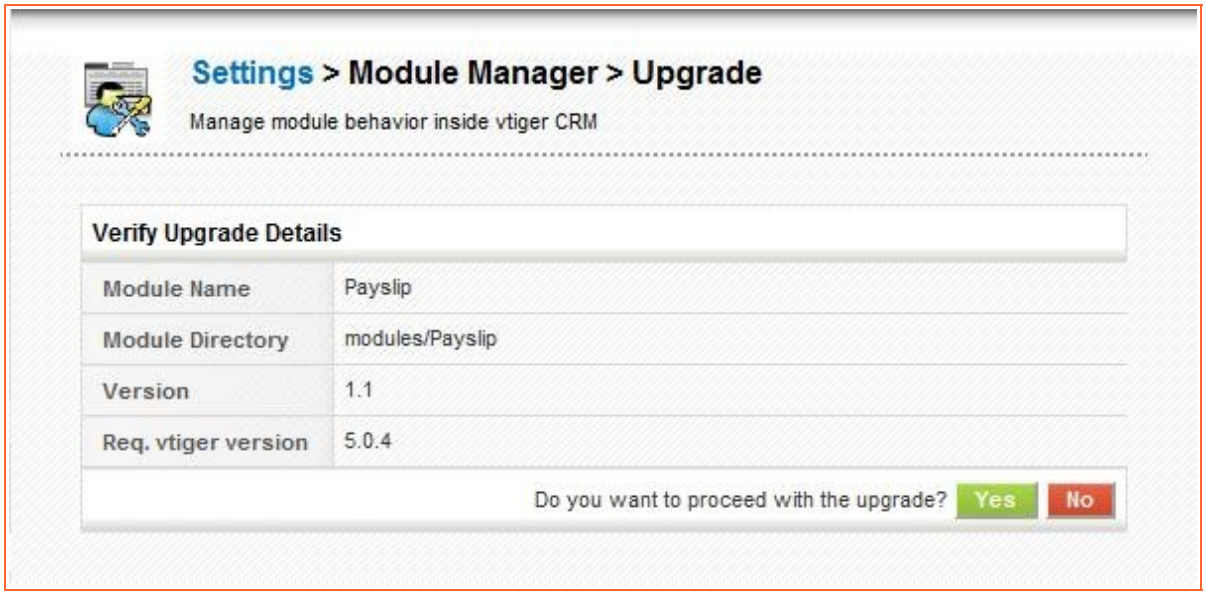
Click on the upgrade icon:



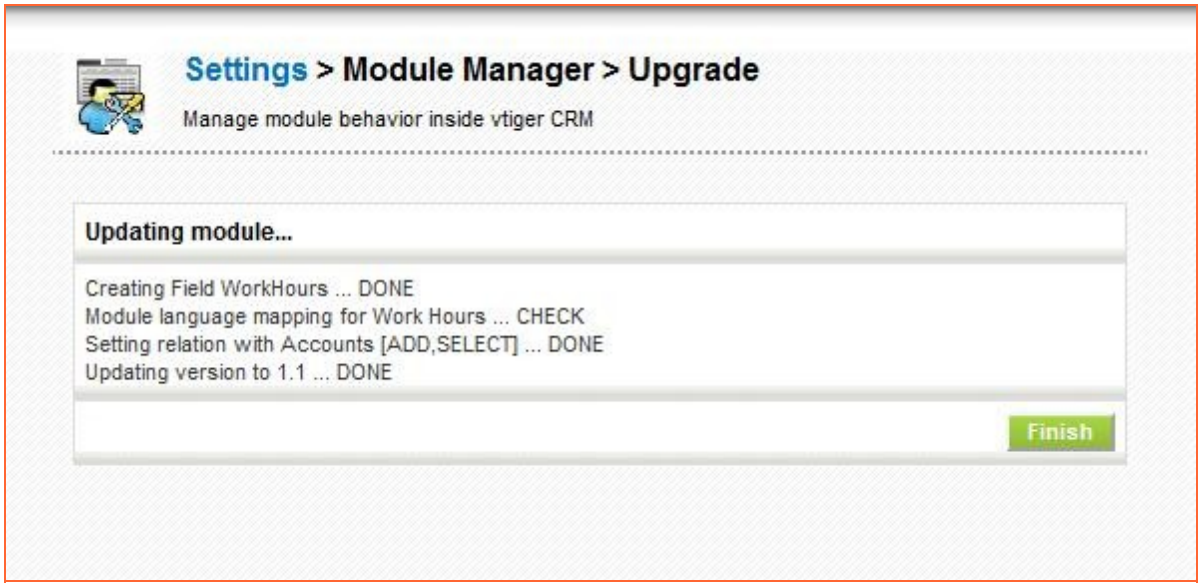
Select the new package file for the module:



Verify the package details before you upgrade:



Finally your module will be upgraded:



Appendix 1 - API Changes

vtlib 2.0 contains changes to the APIs previous provided in 1.x version. The new APIs are more modular and adhere to the OOD model.

We explain the changes below.

Creating Module

Using vtlib 1.x

```
Vtiger_Tab::create('Payslip', 'Payslip', 'Tools');
Vtiger_Utils::CreateTable('vtiger_payslip', '(payslipid integer)');
Vtiger_Utils::CreateTable('vtiger_payslipcf', '(payslipid integer, primary key
(payslipid))');
Vtiger_Utils::CreateTable('vtiger_payslipgrouprel',
      '(payslipid integer, groupname varchar(100), primary key(payslipid))');
```

Using vtlib 2.x

```
$moduleInstance = new Vtiger_Module();
$moduleInstance->name = 'Payslip';
$moduleInstance->save();
$moduleInstance->initTables();

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

Creating Block

Using vtlib 1.x

```
Vtiger_Block::create('Payslip', 'LBL_PAYSLIP_INFORMATION');
```

Using vtlib 2.x

```
$blockInstance = new Vtiger_Block();
$blockInstance->label = 'LBL_PAYSLIP_INFORMATION';
$moduleInstance->addBlock($blockInstance);
```

Creating Field

Using vtlib 1.x

```
$fieldInstance = new Vtiger_Field();
$fieldInstance-> set('module',        'Payslip')
      -> set('columnname',    'payslipname')
      -> set('tablename',     'vtiger_payslip')
      -> set('columntype',    'varchar(255)')
      -> set('generatedtype', '1')
      -> set('uitype',        2)
      -> set('fieldname',     'payslipname')
      -> set('fieldlabel',    'PayslipName')
      -> set('readonly',       '1')
      -> set('presence',       '0')
      -> set('selected',       '0')
      -> set('maximumlength',  '100')
      -> set('sequence',       null)
      -> set('typeofdata',     'V~M')
      -> set('quickcreate',    '1')
      -> set('block',          null)
      -> set('blocklabel',     'LBL_PAYSLIP_INFORMATION')
      -> set('displaytype',    '1')
      -> set('quickcreatesequence', null)
      -> set('info_type',      'BAS');
$fieldInstance->create();
```

Using vtlib 2.x

```
$fieldInstance = new Vtiger_Field();
$fieldInstance->name = 'PayslipName';
$fieldInstance->table = 'vtiger_payslip';
$fieldInstance->column = 'payslipname';
$fieldInstance->columntype = 'VARCHAR(100)';
$fieldInstance->uitype = 2;
$fieldInstance->typeofdata = 'V~M';
$blockInstance->addField($fieldInstance);
```

Setting Entity Identifier

Using vtlib 1.x

```
$fieldInstance->set('entityidfield', 'payslipid')
              ->set('entityidcolumn', 'payslipid');
$fieldInstance->setEntityIdentifier();
```

Using vtlib 2.x

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

Set Picklist Values

Using vtlib 1.x

```
$fieldInstance->setupPicklistValues( Array ('Employee', 'Trainee') );
```

Using vtlib 2.x

```
$fieldInstance->setPicklistValues( Array ('Employee', 'Trainee') );
```

Creating Filter

Using vtlib 1.x

```
Vtiger_CustomView::create('Payslip', 'All',true);
$cv = new Vtiger_CustomView('Payslip', 'All');
$cv->addColumn($fieldInstance1)
      ->addColumn($fieldInstance2, 1);
```

Using vtlib 2.x

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

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

Configure Tools

Using vtlib 1.x

```
Vtiger_Module::disableAction('Payslip','Import');
Vtiger_Module::enableAction('Payslip', 'Export');
```

Using vtlib 2.x

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

Configure Sharing Access

Using vtlib 1.x

```
Vtiger_Module::setDefaultSharingAccess('Payslip', 'Private');
```

Using vtlib 2.x

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

Appendix 2 – Schema Changes

Some of the vtlib API make the schema changes (either adding a new table or new column to existing table) the details are captured in this section

Table	Column	Action	Description
vtiger_field	helpinfo TEXT	Column Addition	This column will store the helptext associated with vtiger_field
vtiger_language		Table Addition	Captures languages installed for vtiger CRM
vtiger_links		Table Addition	Captures details of custom module links
vtiger_tab	version VARCHAR(10)	Column Addition	Track version of module in use. Useful during migration or upgrade of module.
vtiger_tab_info		Table Addition	Captures some of the module preferences like minimum version supported, maximum version supported etc.
vtiger_crmentityrel		Table Addition	Captures the relation between module records. For Generic related list handling.
vtiger_fieldmodulerel		Table Addition	Captures related module information for the field of uitype 10
vtiger_mailer_queue		Table Additions	These tables will be added when Vtiger_Mailer class will be use for sending mails asynchronously.
vtiger_mailer_queueinfo			

Appendix 3 – Using vtiger_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'|@vtiger_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:

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

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

Appendix 4 – vtlib_handler Method

Module class should define vtlib_handler method to handle special events triggered as described below:

Event Type	Description
module.postinstall	Once the module import is completed.
module.preupdate	Before updating module (package).
module.postupdate	After module (package) is updated.
module.disabled	When module is disabled.
module.enabled	When module is enabled.
module.preuninstall	Before module instance is deleted.

Example:

vtlib_handler function should be defined as Module Class method.

```
/**
 * Invoked when special actions are performed on the module.
 * @param String Module name
 * @param String Event Type
 */
function vtlib_handler($modulename, $event_type) {

	if($event_type == 'module.postinstall') {
		// TODO Handle post installation actions

	} else if($event_type == 'module.disabled') {
		// TODO Handle actions when this module is disabled.

	} else if($event_type == 'module.enabled') {
		// TODO Handle actions when this module is enabled.

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

	} else if($event_type == 'module.preupdate') {
		// TODO Handle actions before this module is updated.

	} else if($event_type == 'module.postupdate') {
		// TODO Handle actions after this module is updated.

	}
}
```

Usecase

When a module is disabled, through vtlib_handler now it can de-register any Event notification unless it is enabled again.

Appendix 5 – vtlib_listview javascript API

vtlib_listview javascript API provides the ability to register listener function for some predefined event types.

```
vtlib_listview.register( <EVENT_TYPE>, <HANDLER_FUNCTION>,
[<HANDLER_FUNCTION_OPTIONAL_PARAMETERS>] );
```

cell.onmouseover	When the mouse moves over the listview cell.
cell.onmouseout	When the mouse moves out of the listview cell.

The handler function will be invoked with two set of arguments:

Event Parameters	{
	event : <EVENT_TYPE>
	domnode : HTML DOM node used to detect the event
	module : Module Name to which field belongs
	fieldname: Field Name
	recordid : Record ID displayed on the listview
	}

Optional Parameters	This is the one that was passed to the vtlib_listview.register API

Example: You can have the following in your javascript file

```
vtlib_listview.register( 'cell.onmouseover', function(evtparams) {
    evtparams.domnode.style.backgroundColor = '#FFFD7C';
});
vtlib_listview.register( 'cell.onmouseout', function(evtparams) {
    evtparams.domnode.style.backgroundColor = 'white';
});
```

NOTE:

- To make the trigger on all the modules, you will need to configure the javascript as HEADERSCRIPT (refer Custom Links#Special LinkType)

FAQs – Frequently Asked Questions

1. How to write own templates?

Reference: http://forums.vtiger.com/viewtopic.php?p=75410#75410

```
Now I want to set my own templates. The VTLib documentation states that I do
this:
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'));

My question is where do I place this piece of code? In the module creator?
```

Solution:

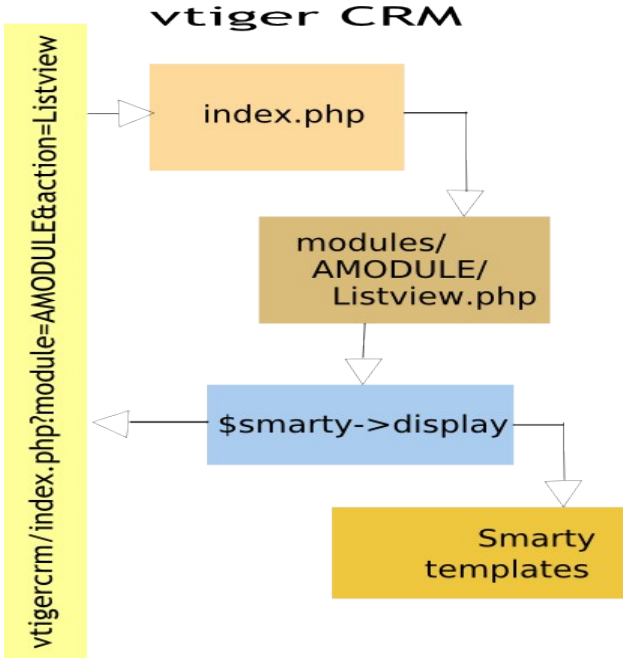
Let us assume you want to create your own Listview for your module TestModule, what you need to do is the following:

1. Create the MyListview.tpl under Smarty/templates/modules/TestModule/MyListView.tpl
2. In your modules/TestModule/Listview.php you will need to call the smarty display as:
   `$smarty->display(vtlib_getModuleTemplate($currentModule, 'MyListview.tpl'));`

Please refer: http://www.smarty.net/manual/en/ to learn more about Smarty usage.

2. How is module template used?

Example below explains how the module listview action gets processed using the smarty.



3. Cannot See Module Manager!

```
I've installed vtlib on vitger 5.0.4 I can not seem to see the module manager
under settings on either, is there a file or dir I need to move some where in
order for the module manager to populate?
```

Solution:

- Enable write access to `modules/, Smarty/, cron/, test/` directory before unzipping vtlib-x.y.zip
- Delete the files under folder Smarty/templates_c (having extension *.tpl.php) and refresh the Settings page, you should see Module Manager.

4. Tips for using field names

1. Preferably use small case characters for field (name and columnname).
2. Avoid any special characters like (_,:,-) in names. You can use it for labels
3. Having same value for field (name and columnname) would makes it easier to avoid confusion to start with.