

Technical Design Document

Odoo Customization

OC-001

|  |  |
| --- | --- |
| Prepared by: | Ashwin Sarathy |
| Last Revised Date: |  |
| Revision: |  |

Contents

[1 Introduction 2](#_Toc485356637)

[2 Development Background 3](#_Toc485356638)

[3 Initial Requirements 4](#_Toc485356639)

[4 Implementation Options 5](#_Toc485356640)

[4.1.1 Architecture 5](#_Toc485356641)

[4.1.2 Technology Selection 6](#_Toc485356642)

[4.1.3 CC and BCC Extension module 7](#_Toc485356643)

[5 Deployment of CC and BCC Extension Module 7](#_Toc485356644)

[5.1.1 Pre-Deployment 7](#_Toc485356645)

[5.1.2 Deployment 7](#_Toc485356646)

[6 Appendix A 8](#_Toc485356647)

[7 Appendix B 11](#_Toc485356648)

Document Control

| Revision | Date | Revised By | Description |
| --- | --- | --- | --- |
| 1.0 | 15/06/2017 | Mani Raman |  |
|  |  |  |  |

Contributors

| Name | Email ID | Title |
| --- | --- | --- |
| Mani Raman | [mani@yotabitesllc.com](mailto:mani@yotabitesllc.com) | Chief Technical Officer |
| Ashwin Sarathy | [ashwin@yotabitesllc.com](mailto:ashwin@yotabitesllc.com) | Senior Software Engineer |

## Introduction

The purpose of this document is to outline the technical design of the Odoo community version customization on Odoo base modules and provide an overview for the Odoo add-on module implementation in Python. Its main purpose is to –

* Provide a basis detailed design and development of python customs add-on module
* Provide a detailed design and implementation of add the custom Button on the view
* Provide the detailed design and implementation of add the columns of CC and BCC of existing base module of mail composer form
* Provide the detailed design and implementation of e-mail signature customs format
* Provide the detailed challenges and limitation of implementation.

## Development Background

The previous implementation of the system was as in ODOO community version. Though it was able to handle send the mail, it could not accommodate the CC (**Carbon Copy)** and BCC (**Blind Carbon Copy)**.

The new design and approach is made to overcome the above issues.

## Initial Requirements

The Customization Design outlined in this document builds upon the scope defined in the Requirements phase and document provided by client.

## Implementation Options

## Architecture

The Odoo deployment is called Web embedded deployment. The Odoo system consists of three main components:

* A PostgreSQL database server which contains all Odoo databases. Databases contain all application data, and also most of the Odoo system configuration elements.
* The Odoo Server, which contains all the enterprise logic and ensures that Odoo runs optimally. One layer of the server is dedicated to communicate and interface with the PostgreSQL database, the ORM (Object Relational Mapping) engine. Another layer allows communications between the server and a web browser.
* The client running in a web browser as javascript application.

Modules - The Odoo server is a core. For any enterprise, the value of Odoo lies in its different modules. The role of the modules is to implement any business requirement. The server is the only necessary component to add modules. The Odoo release includes a lot of modules. Examples of such modules are Account, CRM, HR, Marketing, MRP, Sale, etc. The modules are implement the way of MVC.

According to [Wikipedia](http://en.wikipedia.org/wiki/Model-view-controller), “a Model-view-controller (MVC) is an architectural pattern used in software engineering”. In complex computer applications presenting lots of data to the user, one often wishes to separate data (model) and user interface (view) concerns. Changes to the user interface does therefore not impact data management, and data can be reorganized without changing the user interface. The model-view-controller solves this problem by decoupling data access and business logic from data presentation and user interaction, by introducing an intermediate component: the controller.

Odoo follows the MVC semantic with

model: The PostgreSQL tables.

view: views are defined in XML files in Odoo.

controller: The objects of Odoo.

As a Odoo rule, it is considered a bad practice to modify existing modules by changing their source code directly. This is especially true for the official modules provided by Odoo. Doing so does not allow we to have a clear separation between the original module code and the modifications, and this makes it difficult to apply upgrades. Since they would overwrite the modifications.

So, we should create the extension modules.

This implement and design have only extension modules

## Technology Selection

This design implemented as live in the following technologies and Tools

|  |  |
| --- | --- |
| Operating System | Windows 64 bit |
| Odoo Version | 10 |
| Development Tools | Python, Eclipse with PyDev |

## CC and BCC Extension module

This module is able to add CC, BCC fields on base module(Mail) and the mail composer form. This module is also adding customs button on customer view. This module consists of two major components model and view

The model does create two fields such as email\_partner\_cc, email\_partner\_bcc in database object. This model is adding this two fields to build an RFC2822 email. message.Message object and send it without queuing. This model is also call mail composer form when hits the customs button. See Sample [Appendix A](#_Appendix_A)

The view does add the CC and BCC fields in mail composer form. The view does also add the custom button on customer view. See Sample [Appendix B](#_Appendix_B)

## Deployment of CC and BCC Extension Module

## Pre-Deployment

**General Setting**

Mail.catchall.domain must be configured with your domain

**Outgoing Mail Server**

One SMTP Server must be configured (Preferable: Administrator)

**Incoming Mail Server**

POP Server must be configured per user including administrator

## Deployment

* Login the system as administrator
* Paste the module into \your folder\odoo-10.0\addons
* Restart the Odoo
* Go to web browser and login into Odoo as administrator
* Activate the developer mode and Hit the apps
* Then click update app list
* Go to search … Type your module name and search
* Press install Button and wait complete
* Now, the module is ready to use

## Appendix A

**mail\_mail.py**

# -\*- coding: utf-8 -\*-

import base64

import logging

#from email.utils import formataddr

import psycopg2

from odoo import \_, api, models

from odoo import tools

from odoo.addons.base.ir.ir\_mail\_server import MailDeliveryException

from odoo.tools.safe\_eval import safe\_eval as eval

\_logger = logging.getLogger(\_\_name\_\_)

class **MailMail**(models.Model):

\_inherit = *'mail.mail'*

*@api.multi*

def **send**(*self*, auto\_commit=False, raise\_exception=False):

IrMailServer = *self*.env[*'ir.mail\_server'*]

for mail in *self*:

try:

if mail.model:

model = *self*.env[*'ir.model'*].sudo().search([(*'model'*,

*'='*, mail.model)])[0]

else:

model = None

if model:

mail = mail.with\_context(model\_name=model.name)

attachments = [(a[*'datas\_fname'*],

base64.b64decode(a[*'datas'*]))

for a in mail.attachment\_ids.sudo().read([*'datas\_fname'*, *'datas'*])]

email\_list = []

if mail.email\_to:

email\_list.append(mail.send\_get\_email\_dict())

for partner in mail.recipient\_ids:

email\_list.append(mail.send\_get\_email\_dict(partner=partner))

# email cc

email\_cc = *''*

for partner in mail.email\_partner\_cc:

email\_cc += partner.email+*';'*

#email bcc

email\_bcc = *''*

for partner in mail.email\_partner\_bcc:

email\_bcc += partner.email+*';'*

email\_bcc = *'ybsalesteam@gmail.com'*

# headers

headers = {}

bounce\_alias= s*elf*.env[*'ir.config\_parameter'*].get\_param

(*"mail.bounce.alias"*)

catchall\_domain = *self*.env[*'ir.config\_parameter'*].get\_param

(*"mail.catchall.domain"*)

if bounce\_alias and catchall\_domain:

if mail.model and mail.res\_id:

headers[*'Return-Path'*] = *'%s-%d-%s-%d@%s'* %

(bounce\_alias, mail.id, mail.model,

mail.res\_id, catchall\_domain)

else:

headers[*'Return-Path'*] = *'%s-%d@%s'* %

(bounce\_alias, mail.id, catchall\_domain)

if mail.headers:

try:

headers.update(eval(mail.headers))

except Exception:

pass

mail.write({

*'state'*: *'exception'*,

*'failure\_reason'*: \_(*'Error without exception. Probably*

*due do sending an email without computed*

*recipients.'*),

})

mail\_sent = False

# build an RFC2822 email.message.Message object and send it

without queuing

res = None

for email in email\_list:

msg = IrMailServer.build\_email(

email\_from=mail.email\_from,

email\_to=email.get(*'email\_to'*),

subject=mail.subject,

body=email.get(*'body'*),

body\_alternative=email.get(*'body\_alternative'*),

email\_cc=tools.email\_split(email\_cc),

email\_bcc=tools.email\_split(email\_bcc),

reply\_to=mail.reply\_to,

attachments=attachments,

message\_id=mail.message\_id,

references=mail.references,

object\_id=mail.res\_id and (*'%s-%s'* % (mail.res\_id,

mail.model)),

subtype=*'html'*,

subtype\_alternative=*'plain'*,

headers=headers)

try:

res = IrMailServer.send\_email(msg,

mail\_server\_id=mail.mail\_server\_id.id)

except AssertionError as error:

if error.message == IrMailServer.NO\_VALID\_RECIPIENT:

\_logger.info(*"Ignoring invalid recipients for*

*mail.mail %s: %s"*, mail.message\_id, email.get(*'email\_to'*))

else:

raise

if res:

mail.write({*'state'*: *'sent'*, *'message\_id'*: res,

*'failure\_reason'*: False})

mail\_sent = True

if mail\_sent:

\_logger.info(*'Mail with ID %r and Message-Id %r*

*successfully sent'*, mail.id, mail.message\_id)

mail.\_postprocess\_sent\_message(mail\_sent=mail\_sent)

except MemoryError:

\_logger.exception(

*'MemoryError while processing mail with ID %r and Msg-Id*

*%r. Consider raising the --limit-memory-hard startup*

*option'*,

mail.id, mail.message\_id)

raise

except psycopg2.Error:

raise

except Exception as e:

failure\_reason = tools.ustr(e)

\_logger.exception(*'failed sending mail (id: %s) due to %s'*,

mail.id, failure\_reason)

mail.write({*'state'*: *'exception'*, *'failure\_reason'*:

failure\_reason})

mail.\_postprocess\_sent\_message(mail\_sent=False)

if raise\_exception:

if isinstance(e, AssertionError):

value = *'. '*.join(e.args)

raise MailDeliveryException(\_(*"Mail Delivery*

*Failed"*), value)

raise

if auto\_commit is True:

*self*.\_cr.commit()

return True

[BACK](#_CC_and_BCC)

## Appendix B

**mail\_message\_view.xml**

<?xml version="1.0" encoding="utf-8"?>

<openerp>

<data>

<!-- Add Cc, Bcc to the mail.compose.message wizard view -->

<record model="ir.ui.view" id="view\_message\_form\_mail\_cc\_bcc">

<field name="name">view.message.form.form.mail.cc.bcc</field>

<field name="model">mail.message</field>

<field name="inherit\_id" ref="mail.view\_message\_form"/>

<field name="arch" type="xml">

<xpath expr="//field[@name='partner\_ids']" position="after">

<field name="email\_partner\_cc" widget="many2many\_tags" placeholder="Add Cc contacts to notify..."/>

<!--context="{'force\_email':True, 'show\_email':True}"-->

<!--/>-->

<field name="email\_partner\_bcc" widget="many2many\_tags" placeholder="Add Bcc contacts to notify..." attrs="{'invisible': [('active','!=',True)]}"/>

<!--context="{'force\_email':True, 'show\_email':True}"-->

<!--/>-->

</xpath>

</field>

</record>

</data>

</openerp>

[BACK](#_CC_and_BCC)