

xf_doc_approval_custom — Developer Guide

Table of Contents

1. [Project Overview](#)
2. [Folder Structure](#)
3. [How Odoo Inheritance Works](#)
4. [Inheriting a Model \(Python\)](#)
5. [Inheriting a View \(XML\)](#)
6. [Inheriting CSS / Styles](#)
7. [Inheriting Fonts \(Google Fonts\)](#)
8. [Key Techniques Used in This Module](#)
9. [Docker Commands](#)
10. [Rules — What NOT to Touch](#)

1. Project Overview

Item	Value
Base module	<code>xf_doc_approval</code> (located at <code>d:\project_v19s\xf_doc_approval</code>)
Custom module	<code>xf_doc_approval_custom</code> (located at <code>d:\project_v19s\odoo19-dev\addons\xf_doc_approval_custom</code>)
Odoo version	19.0
Purpose	Extend the base module with Khmer language labels, new fields, redesigned form, QR code, and role-based comment visibility — without modifying the original module

What we changed vs the base module

Feature	What we did
Navbar color	Changed to <code>#0a5e98</code> via CSS
Font	Added Khmer font (Battambang) for Khmer text only
State labels	Overrode all state labels to Khmer
Section អ្នក (Approvers)	Renamed to Khmer, added <code>sent_date</code> , <code>round</code> , Khmer column headers, document source row
Section ខ្លឹមសារ (Content)	Completely new section with 8 custom fields in a styled table

Feature	What we did
New fields	<code>document_type</code> , <code>decision_requester_id</code> , <code>reference_note</code> , <code>reference_file</code> , <code>doc_number</code> , <code>qr_code</code> , <code>document_source</code> , <code>description</code> (Html override)
QR Code	Auto-generated when document is approved
Auto numbering	<code>ir.sequence</code> → format <code>DOC/YYYY/MM/0001</code>
Comment visibility	Step-based: each approver sees only notes from their step and earlier
Button label	"Send for Approval" → "بمراجعة"

2. Folder Structure

```
xf_doc_approval_custom/
├── __manifest__.py           ← Module declaration (name, depends, data files)
├── __init__.py              ← Python package init (imports models/)
├── models/
│   ├── __init__.py          ← Imports all model files
│   ├── document_package.py   ← Inherits xf.doc.approval.document.package
│   └── document_approver.py  ← Inherits xf.doc.approval.document.approver
├── views/
│   ├── assets.xml            ← Registers CSS + Google Fonts injection
│   └── document_package_form.xml ← Inherits and modifies the form view
├── data/
│   └── sequences.xml         ← Creates ir.sequence for document numbering
└── static/
    ├── src/
    │   └── css/
    │       └── navbar.css    ← All custom CSS (navbar, table, section ⌘)
```

3. How Odoo Inheritance Works

Odoo has **3 layers** you can extend without touching the original code:

Original Module	Your Custom Module
<code>models/</code>	<code>models/</code> ← <code>_inherit = 'original.model.name'</code>

views/	views/	← inherit_id = ref('original.view.id')
static/css/	static/	← registered via ir.asset record

The **golden rule**: never edit the original module. All changes live in your custom module.

4. Inheriting a Model (Python)

Basic pattern

```
# models/my_model.py
from odoo import api, fields, models

class MyCustomExtension(models.Model):
    _inherit = 'original.model.name'  # ← this is the key line

    # Add new fields
    my_new_field = fields.Char(string='My Field')

    # Override an existing field (change labels, default, etc.)
    state = fields.Selection(
        selection=[('draft', 'New Label'), ('done', 'Finished')],
        # Only list what you want to CHANGE. Odoo merges the rest.
    )

    # Override an existing method
    def action_confirm(self):
        # Do something before
        result = super().action_confirm()  # ← always call super()
        # Do something after
        return result
```

How we used it — `document_package.py`

```
class DocApprovalDocumentPackageCustom(models.Model):
    _inherit = 'xf.doc.approval.document.package'

    # 1. New fields added to existing model
    document_type = fields.Selection(...)
    doc_number = fields.Char(default='New')
    qr_code = fields.Image(max_width=0, max_height=0)

    # 2. Override existing field labels only
    state = fields.Selection(
        selection=[('draft', 'បង្កើតលិខិតឯកសារ'), ...]
    )

    # 3. Override existing field type (Text → Html)
    description = fields.Html(translate=True, sanitize=False)
```

```
# 4. Override a method – inject QR generation after approval
def action_finish_approval(self):
    res = super().action_finish_approval() # run original logic first
    approved = self.filtered(lambda r: r.state == 'approved')
    approved._generate_qr_code()
    return res
```

Field types reference

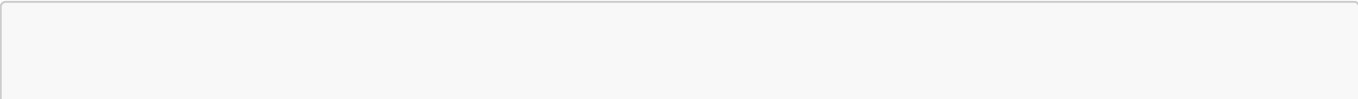
Field	Use case
fields.Char	Short text (one line)
fields.Text	Long plain text
fields.Html	Rich text with editor toolbar
fields.Integer	Whole number
fields.Float	Decimal number
fields.Boolean	True/False checkbox
fields.Date	Date only
fields.Datetime	Date + time
fields.Selection	Dropdown list
fields.Many2one	Link to one other record
fields.One2many	List of related records
fields.Many2many	Multiple related records
fields.Binary	File / raw bytes
fields.Image	Image file (served via /web/image/)

Important: `__init__.py` must import every model file

```
# models/__init__.py
from . import document_package
from . import document_approver
```

5. Inheriting a View (XML)

Basic pattern



```
<record id="my_custom_view" model="ir.ui.view">
  <field name="name">my.custom.view.name</field>
  <field name="model">original.model.name</field>
  <!-- Point to the original view's XML id -->
  <field name="inherit_id" ref="original_module.original_view_id"/>
  <field name="arch" type="xml">

    <!-- Each change is one <xpath> block -->
    <xpath expr="//SELECTOR" position="POSITION">
      <!-- your content here -->
    </xpath>

  </field>
</record>
```

Finding the original view's **ref**

- 1. Go to **Settings** → **Technical** → **Views**
- 2. Search for the model name
- 3. The **External ID** column shows the ref, e.g.
 xf_doc_approval.xf_doc_approval_document_package_form

xpath **expr** — how to select elements

Selector	Example	Selects
By element name	//form	The <form> element
By name attribute	//group[@name='approvers']	<group name="approvers">
By field name	//field[@name='user_id']	<field name="user_id">
By button name	//button[@name='action_confirm']	<button name="action_confirm">
Nested	//field[@name='line_ids']//list//field[@name='qty']	qty field inside list inside line_ids

xpath **position** — where to insert

Position	What it does
before	Insert before the selected element
after	Insert after the selected element

Position	What it does
inside	Insert as child at the end
replace	Replace the selected element entirely
attributes	Only change attributes of the element

Examples used in this module

```

<!-- 1. Change a button label -->
<xpath expr="//button[@name='action_send_for_approval']" position="attributes">
  <attribute name="string">បញ្ជូន</attribute>
</xpath>

<!-- 2. Hide an element -->
<xpath expr="//div[@class='oe_title']" position="attributes">
  <attribute name="invisible">1</attribute>
</xpath>

<!-- 3. Rename a group header -->
<xpath expr="//group[@name='approvers']" position="attributes">
  <attribute name="string">ក. ឋាននុក្រមលំហូរឯកសារ</attribute>
</xpath>

<!-- 4. Add a new element BEFORE another -->
<xpath expr="//group[@name='approvers']/field[@name='approver_ids']"
position="before">
  <div class="my-class">
    <field name="my_new_field" nolabel="1"/>
  </div>
</xpath>

<!-- 5. Add a column to a list -->
<xpath expr="//field[@name='approver_ids']//list//field[@name='notes']"
position="after">
  <field name="sent_date" string="កាលបរិច្ឆេទ"/>
</xpath>

<!-- 6. Replace a field with another -->
<xpath expr="//field[@name='approver_ids']//list//field[@name='notes']"
position="replace">
  <field name="notes_display" string="មតិយោបល់" readonly="1"/>
</xpath>

<!-- 7. Add a whole new group after an existing one -->
<xpath expr="//group[@name='approvers']" position="after">
  <group string="ខ. ខ្លឹមសារ" name="section_b">
    <field name="document_type" nolabel="1"/>
  </group>
</xpath>

```

Conditional visibility

```
<!-- hide when state is not draft -->
<field name="my_field" readonly="state != 'draft'"/>

<!-- hide column in list -->
<field name="method" column_invisible="True"/>

<!-- hide entire group conditionally -->
<group invisible="state != 'approved' or not qr_code">
    ...
</group>
```

6. Inheriting CSS / Styles

Step 1 — Create the CSS file

```
static/src/css/navbar.css
```

Step 2 — Register it in `assets.xml`

```
<record id="my_module_css" model="ir.asset">
    <field name="bundle">web.assets_backend</field>
    <field name="path">my_module/static/src/css/navbar.css</field>
</record>
```

The bundle `web.assets_backend` means it loads in the Odoo backend (not the website/portal).

CSS selectors used in this module

```
/* Target by field name (most reliable) */
[name="approver_ids"] .o_list_table thead th { ... }

/* Target Odoo's built-in classes */
.o_main_navbar { background-color: #0a5e98 !important; }

/* Target a custom class added via view */
.o_xf_section_b { ... }
.o_xf_row { ... }
.o_xf_cell_label { ... }

/* Target buttons */
button.oe_highlight { background-color: #0a5e98 !important; }
```

Important CSS rules for Odoo

```
/* Always use !important to override Odoo's built-in Bootstrap styles */
.my-class { color: red !important; }

/* Use specific font list – Khmer font as FALLBACK, not first */
body, p, span, td {
    font-family: 'Segoe UI', Roboto, Arial, 'Battambang', sans-serif;
}
/* If you put Battambang first, ALL text (including icons) uses Khmer font */

/* Hide checkbox column in list without breaking layout */
[name="approver_ids"] .o_list_record_selector {
    width: 0 !important;
    padding: 0 !important;
    overflow: hidden !important;
}
```

7. Inheriting Fonts (Google Fonts)

CSS `@import url(...)` does **not** work inside Odoo's asset bundle (Docker blocks external HTTP at compile time). Instead, inject a `<link>` tag directly into the HTML `<head>` by inheriting `web.layout`:

```
<!-- In assets.xml -->
<template id="my_font_template" inherit_id="web.layout" name="My Font">
    <xpath expr="//head" position="inside">
        <link rel="preconnect" href="https://fonts.googleapis.com"/>
        <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin=""/>
        <link rel="stylesheet"
            href="https://fonts.googleapis.com/css2?
family=Battambang:wght@400;700&display=swap"/>
    </xpath>
</template>
```

Note: `&` must be used instead of `&` inside XML attribute values.

8. Key Techniques Used in This Module

8.1 Auto document numbering (ir.sequence)

```
<!-- data/sequences.xml -->
<record id="seq_xf_doc_approval_document_package" model="ir.sequence">
    <field name="code">xf.doc.approval.document.package</field>
    <field name="prefix">DOC/%(year)s/%(month)s</field>
```



```
<field name="padding">4</field>
</record>
```

```
# In model method:
record.doc_number = self.env['ir.sequence'].next_by_code(
    'xf.doc.approval.document.package'
) or 'New'
# Result: DOC/2026/04/0001
```

8.2 Override a method and call super()

Always call `super()` unless you deliberately want to block the original behavior.

```
def action_send_for_approval(self):
    # Run YOUR code first
    for record in self:
        if not record.doc_number or record.doc_number == 'New':
            record.doc_number = self.env['ir.sequence'].next_by_code(...)
    # Then run the ORIGINAL method
    return super().action_send_for_approval()
```

8.3 Computed field (non-stored, user-dependent)

Used for `notes_display` — each user sees different notes based on their step:

```
notes_display = fields.Text(
    compute='_compute_notes_display',
    # No store=True → recomputes every time, never saved to DB
)

@api.depends('notes', 'step', 'document_package_id.approver_ids.step')
def _compute_notes_display(self):
    current_uid = self.env.uid
    for record in self:
        my_step = max(
            record.document_package_id.approver_ids
                .filtered(lambda a: a.user_id.id == current_uid)
                .mapped('step') or ['00']
        )
        record.notes_display = record.notes if record.step <= my_step else False
```

8.4 QR Code generation

```
# fields.Image serves via /web/image/ – do NOT use fields.Binary with
attachment=True
qr_code = fields.Image(max_width=0, max_height=0)

def _generate_qr_code(self):
    import qrcode, base64, io
    for record in self:
        qr = qrcode.QRCode(version=None,
                           error_correction=qrcode.constants.ERROR_CORRECT_L,
                           box_size=15, border=4)
        qr.add_data(record.doc_number) # Keep content SHORT and ASCII-only
        qr.make(fit=True)
        img = qr.make_image(fill_color='black', back_color='white')
        buf = io.BytesIO()
        img.save(buf, format='PNG')
        record.write({'qr_code': base64.b64encode(buf.getvalue())})

# Trigger after approval
def action_finish_approval(self):
    res = super().action_finish_approval()
    self.filtered(lambda r: r.state == 'approved')._generate_qr_code()
    return res
```

Key lesson: Keep QR content to ASCII-only short text.

Khmer Unicode text forces QR version 17+ (810×810 px) → unreadable when scaled down.

8.5 Override field type (Text → Html)

The base model had `description = fields.Text(...)`.

We override it to get a rich-text editor:

```
# Custom module – overrides the type for this model
description = fields.Html(translate=True, sanitize=False)
```

`sanitize=False` is needed to allow file attachments embedded by the HTML editor.

8.6 One2many field in view — show only file column

```
<field name="document_ids" nolabel="1" readonly="state != 'draft'">
  <list editable="bottom">
    <field name="name" column_invisible="True"/> <!-- hidden but still
present -->
    <field name="file" widget="binary" filename="file_name" string="ឯកសារ"/>
    <field name="file_name" column_invisible="True"/>
  </list>
</field>
```

`column_invisible` hides the column in list view but keeps the field in the record.

8.7 Radio widget inline (horizontal)

```
<field name="document_source" widget="radio" nolabel="1" readonly="state !=  
'draft'"/>
```

```
/* Force horizontal layout */  
[name="document_source"] .o_field_radio {  
    display: flex !important;  
    flex-direction: row !important;  
    gap: 16px !important;  
}
```

9. Docker Commands

Start the containers

```
cd d:/project_v19s/odoo19-dev  
docker compose up -d
```

Stop the containers

```
cd d:/project_v19s/odoo19-dev  
docker compose down
```

Update the custom module (after code/view changes)

```
docker exec odoo19_app odoo -u xf_doc_approval_custom -d demo_db --stop-after-init  
docker restart odoo19_app
```

First-time install of a new module

```
docker exec odoo19_app odoo -i xf_doc_approval_custom -d demo_db --stop-after-init  
docker restart odoo19_app
```

Open Python shell (for debugging / manual fixes)

```
docker exec -i odoo19_app odoo shell -d demo_db --no-http
```

Inside the shell:

```
# Browse a record
r = env['xf.doc.approval.document.package'].browse(4)
print(r.state, r.doc_number)

# Run a method manually
r._generate_qr_code()
env.cr.commit() # save changes
```

View live logs

```
docker logs odoo19_app -f
```

10. Rules — What NOT to Touch

Rule	Reason
Never edit <code>d:\project_v19s\xf_doc_approval\</code>	It is the original base module. Any change there will be overwritten by updates and breaks the separation of concerns.
Never use <code>@import url()</code> in CSS files	Odoo's Docker environment cannot fetch external URLs at CSS compile time. Use <code>web.layout</code> template injection instead.
Never put <code>font-family: Battambang</code> as the FIRST font	It will replace Font Awesome icons with broken characters. Always put it as a fallback.
Never use <code>fields.Binary(attachment=True)</code> for images shown with <code>widget="image"</code>	The <code>/web/image/</code> controller does not serve <code>attachment=True</code> Binary fields for custom models. Use <code>fields.Image</code> instead.
Never put <code><field></code> inside a plain <code><div></code> when replacing the entire <code><sheet></code>	OWL (Odoo 19's frontend framework) requires fields to be inside proper Odoo form elements. Use targeted <code><xpath></code> instead of full sheet replacement.
Always call <code>super()</code> when overriding methods	Without <code>super()</code> , the original module's logic is skipped entirely (approvals won't work, emails won't send, etc.).

Quick Inheritance Checklist

When you want to add something new:

- [] 1. Find the model name in the original module (grep for `_name = '...'`)
- [] 2. Find the view XML ID in Settings → Technical → Views
- [] 3. Add `_inherit = 'model.name'` in a new Python file under `models/`
- [] 4. Import the new file in `models/__init__.py`
- [] 5. Add `inherit_id = ref('module.view_id')` in your XML view file
- [] 6. Write xpath to target the exact element you want to change
- [] 7. Add CSS in `static/src/css/` and register it in `assets.xml`
- [] 8. Run: `docker exec odoo19_app odoo -u your_module -d demo_db --stop-after-init`
- [] 9. Run: `docker restart odoo19_app`
- [] 10. Hard refresh browser (Ctrl+Shift+R)