Extend the Frontend Form

Sublogin Documentation

Guide for Magento administrators, technical project owners, and implementation teams.

LAST UPDATED
2026-10-13

SOURCE
Open this article online

EXTENSION
Magento 2 Extension Sublogin

Extend the Frontend Form

Use a Magento plugin on `MageB2B\Sublogin\Block\Sublogin\Edit::getFormFields()` when a custom module needs to change the fields rendered on the customer-facing Sublogin create or edit form.

This hook controls presentation only. A new field still needs its own validation, persistence, loading, authorization and API or import behavior where applicable.

Inspect the current field array first

The returned fields depend on store configuration and installed add-ons. Base fields can include name, unique email, password handling, active and expiry state, email preference, delegated sublogin management, parent addresses and payment or shipping restrictions.

Role, Budget, Order Approval and Custom Catalog can add their own definitions. Inspect the result from your installed version before choosing a position or renderer.

Register a frontend plugin

In your custom module's `etc/frontend/di.xml`:

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManag
er/etc/config.xsd">
    <type name="MageB2B\Sublogin\Block\Sublogin\Edit">
        <plugin name="company_sublogin_add_form_field"
type="Company\SubloginFields\Plugin\AddFormField" />
    </type>
</config>
```

Then append a field without replacing definitions contributed by other modules:

```
<?php

declare(strict_types=1);

namespace Company\SubloginFields\Plugin;

use MageB2B\Sublogin\Block\Sublogin\Edit;

final class AddFormField
{
```

```
    public function afterGetFormFields(Edit $subject, array
$fields): array
    {
        $fields[] = [
            'name' => 'purchasing_reference',
            'label' => __('Purchasing Reference'),
            'type' => 'text',
            'default' => '',
            'visible' => true,
            'position' => 1400,
        ];

        return $fields;
    }
}
```

Common definition keys include `name`, `label`, `type`, `default`, `options`, `visible` and `position`. Copy renderer conventions from the current array instead of assuming every Magento form type is supported.

Register the field for saving

Rendering the field is not enough. Since Sublogin 2.3.9, storefront create and edit requests pass through a POST field allow-list in `MageB2B\Sublogin\Controller\Sublogin\PostDataProcessor`: any posted key that is not registered is silently dropped before the sublogin is saved. A custom field therefore renders but never persists unless you register it.

Add the field name to the `frontendAllowedFields` argument in your module's `etc/frontend/di.xml` (or global `di.xml`):

```
<type
name="MageB2B\Sublogin\Controller\Sublogin\PostDataProcessor">
    <arguments>
        <argument name="frontendAllowedFields" xsi:type="array">
            <item name="purchasing_reference"
xsi:type="string">purchasing_reference</item>
        </argument>
    </arguments>
</type>
```

This is the mechanism the first-party add-ons use: `mageb2b/sublogin-role` registers `role_id` and `group_id`, `mageb2b/sublogin-orderapproval` registers `order_needs_approval` and `order_approval_amount_check`. Registering a field only allows it through the filter — validation and persistence are still your code's job.

Complete the data contract

Before releasing the field, decide:

1. where the value is stored;
2. which main customer, sublogin or administrator may read and change it;
3. how create and edit requests validate it;
4. whether Admin, REST, SOAP, import and export need the value;
5. how it behaves when a customer is transformed into a sublogin or back;
6. whether it contains personal or sensitive data.

Do not write directly into vendor module tables without a reviewed schema and service boundary.

Verify

Test create and edit as the main customer and as a delegated sublogin. Submit invalid input, reload the saved record, and check every installed storefront theme. If Hyvä is used, confirm the relevant compatibility package and custom template render the new field.