magmodules



Ontwikkelaarsgids

Cross-Linking for Magento 2

Using the Cross-Linking extension from your own code. The extension links up product descriptions, category descriptions and CMS content on its own, but any other content — a blog post, a landing page, a block from a custom extension — can run through the same rules with one call. The rules, the limits and the store scoping a merchant configured in the admin all apply, so content from your own code behaves exactly like content the extension handles itself.

The service contract

`Magmodules\CrossLinking\Api\CrossLink\ReplacerInterface` is the entry point. Inject it like any other service:

```
use Magmodules\CrossLinking\Api\CrossLink\ReplacerInterface;

class MyBlogPostRenderer
{
    public function __construct(
        private readonly ReplacerInterface $crossLinks
    ) {
    }

    public function render(string $html): string
    {
        return $this->crossLinks->apply($html);
    }
}
```

That is the whole integration. The call returns the HTML with the active rules applied.

apply()

```
public function apply(
    string $html,
    ?int $storeId = null,
    ?int $maxLinks = null,
    ?string $currentUrl = null,
    ?string $contentType = null
): string;
```

Parameter	What it does when you leave it out
`$html`	Required. Empty content comes back untouched.

Parameter	What it does when you leave it out
`$storeId`	The store the visitor is on. Pass it when you render content for a store other than the current one, such as from a console command or a cron job.
`$maxLinks`	The **Default Max Links Per Page** setting under General. Pass a number to use your own limit for this piece of content.
`$currentUrl`	No self-link check. Pass the URL of the page you are rendering and a rule pointing at that same page is skipped, so the page does not link to itself.
`$contentType`	The rule applies whatever kind of content this is. Pass one of `ContentType::PRODUCT`, `CATEGORY`, `CMS` or `OTHER` and a rule limited to certain content types knows whether it is wanted here.

The method respects the extension's own switches. With Cross-Linking disabled, or when no store can be resolved, the HTML comes back exactly as it went in — your code does not need to check anything first.

From a template

For a block that renders its own content, there is a ViewModel, so the template needs no PHP class of its own. Wire it up in layout XML:

```
<block name="my.content" template="My_Module::content.phtml">
    <arguments>
        <argument name="crossLinks" xsi:type="object">
            Magmodules\CrossLinking\ViewModel\CrossLinks
        </argument>
    </arguments>
</block>
```

And in the template:

```
<?php $crossLinks = $block->getData('crossLinks'); ?>
<?= /* @noEscape */ $crossLinks->apply($html) ?>
```

The output is HTML the extension built itself: attribute values on the links it inserts are escaped, and your own content is returned as you passed it in.

What the rules do to your content

The same guarantees apply as for the content the extension handles itself:

- **Text inside existing links is left alone.** A keyword that already sits in an `<a>` tag is not linked a second time.
- **HTML attributes are left alone.** A keyword in an `alt` or `title` attribute is not turned into a link.
- **Rules run in priority order,** and at equal priority the longest keyword goes first, so "duffle bag" wins over "bag".
- **Limits are respected per keyword and per page.** A rule with its own maximum stops there; the page as a whole stops at the limit you passed or the configured default.
- **Only rules for the store apply,** including rules set to all store views.

Reading and writing rules

Rules themselves are available through `Magmodules\CrossLinking\Api\CrossLink\RepositoryInterface`, with the usual `get()`, `save()`, `delete()`, `deleteById()` and `getList()`. `getActiveForStore()` returns the active rules for one store in the order they are applied. The rule itself is a `Magmodules\CrossLinking\Api\CrossLink\DataInterface` with a getter and setter per field.

Use this when you want to create or maintain rules from your own code — an import, a sync with another system — rather than through the admin.

Upgrade safety

Everything under `Magmodules\CrossLinking\Api` is marked `@api` and follows semantic versioning: it will not change in a way that breaks your code within a major version. Classes outside `Api` are internal. `Magmodules\CrossLinking\Service\CrossLink\Replacer` in particular is the implementation behind the contract, not the contract itself — inject the interface, not the class.

Need More Help?

Documentation:

- All Help Articles - Complete documentation overview

Support:

- Contact Support - Get help from our team

For a complete overview of features and configuration options, see the Cross-Linking extension on magmodules.eu

All articles for Cross-Linking

Installation

1	Installatie met Composer (aanbevolen)
2	Installatie via de Adobe Marketplace
3	Installatie via FTP en SSH

Configuration

1	Configuratie Gids
2	Snelstart Gids

Troubleshooting

1	Troubleshooting

Other

1	**Ontwikkelaarsgids (current)**

Grids

1	Cross-Links Grid
2	Link Placements

Background

1	Best Practices
2	CLI Commando's

