

## 4. Working with Locations

[ [Adding a new Location to a Content](#) ] [ [Hide/Unhide Location](#) ] [ [Deleting a location](#) ]

### Adding a new Location to a Content

**Full code**

<https://github.com/ezsystems/CookbookBundle/blob/master/Command/AddLocationToContentCommand.php>

We have seen earlier how you can create a Location for a newly created [Content](#). It is of course also possible to add a new [Location](#) to an existing [Content](#).

```
try
{
    $locationCreateStruct = $locationService->newLocationCreateStruct(
    $parentLocationId );
    $contentInfo = $contentService->loadContentInfo( $contentId );
    $newLocation = $locationService->createLocation( $contentInfo,
    $locationCreateStruct );
    print_r( $newLocation );
}
// Content or location not found
catch ( \eZ\Publish\API\Repository\Exceptions\NotFoundException $e )
{
    $output->writeln( $e->getMessage() );
}
// Permission denied
catch ( \eZ\Publish\API\Repository\Exceptions\UnauthorizedException $e )
{
    $output->writeln( $e->getMessage() );
}
```

This is the required code. As you can see, both the [ContentService](#) and the [LocationService](#) are involved. Errors are handled the usual way, by intercepting the Exceptions the used methods may throw.

```
$locationCreateStruct = $locationService->newLocationCreateStruct( $parentLocationId
);
```

Like we do when creating a new [Content](#), we need to get a new [LocationCreateStruct](#). We will use it to set our new [Location](#)'s properties. The new [Location](#)'s parent ID is provided as a parameter to [LocationService::newLocationCreateStruct](#).

In this example, we use the default values for the various [LocationCreateStruct](#) properties. We could of course have set custom values, like setting the [Location](#) as hidden (`$location->hidden = true`), or changed the remoteld (`$location->remoteld = $myRemoteld`).

```
$contentInfo = $contentService->loadContentInfo( $contentId );
```

To add a [Location](#) to a [Content](#), we need to specify the [Content](#), using a [ContentInfo](#) object. We load one using [ContentService::loadContentInfo\(\)](#), using the [Content](#) ID as the argument.

```
$newLocation = $locationService->createLocation( $contentInfo, $locationCreateStruct
);
```

We finally use `LocationService::createLocation()`, providing the `ContentInfo` obtained above, together with our `LocationCreateStruct`. The method returns the newly created `Location Value Object`.

## Hide/Unhide Location



### Full code

<https://github.com/ezsystems/CookbookBundle/blob/master/Command/HideLocationCommand.php>

We mentioned earlier that a `Location`'s visibility could be set while creating the `Location`, using the `hidden` property of the `LocationCreateStruct`. Changing a `Location`'s visibility may have a large impact in the `Repository`: doing so will affect the `Location`'s subtree visibility. For this reason, a `LocationUpdateStruct` doesn't let you toggle this property. You need to use the `LocationService` to do so.

```
$hiddenLocation = $locationService->hideLocation( $location );
$unhiddenLocation = $locationService->unhideLocation( $hiddenLocation );
```

There are two methods for this: `LocationService::hideLocation`, and `LocationService::unhideLocation()`. Both expect the `LocationInfo` as their argument, and return the modified `Location Value Object`.



The explanation above is valid for most `Repository` objects. Modification of properties that affect other parts of the system will require that you use a custom service method.

## Deleting a location

Deleting `Locations` can be done in two ways: `delete`, or `trash`.

```
$locationService->deleteLocation( $locationInfo );
```

`LocationService::deleteLocation()` will permanently delete the `Location`, as well as all its descendants. Content that have only one `Location` will be permanently deleted as well. Those with more than one won't be, as they are still referenced by at least one `Location`.

```
$trashService->trash( $locationInfo );
```

`TrashService::trash()` will send the `Location` as well as all its descendants to the `Trash`, where they can be found and restored until the `Trash` is emptied. Content isn't affected at all, since it is still referenced by the trash items.



The `TrashService` can be used to list, restore and delete `Locations` that were previously sent to trash using `TrashService::trash()`.