Creating a Custom Module for MediaSpace

Last Modified on 08/11/2020 5:00 pm IDT

Kaltura MediaSpace Module Structure

A module of MediaSpace is an implementation of MVC web application, based on Zend-Framework folder structure and naming convention.

This section describes the folder structure of a MediaSpace module including typical files in each folder.

In the following example, replace "{module}" with the name of your module. Notice that the replacement should be case-sensitive, so if you see {Module} and your module's name is test you should replace with Test.

```
{module}/
         controllers/
                     IndexController.php
         models/
                {Module}.php --- the model file of the module, without it the module will not be fu
nctional at all
         views/
               scripts/
                       index/ --- as the name of the controller we defined
                              index.phtml --- view script for indexAction within IndexController
                              other.phtml --- view script for otherAction within IndexController
         assets/ --- css, js and images to be used by the module.
         default.ini --- default settings file of the module.
         admin.ini --- settings file of the module to expose configuration options in configuration
 management UI
         module.info --- information file of the module to present data in configuration management
 UI
```

Model Class

The model class in a MediaSpace module is responsible for "declaring" each and every feature the module extends.

Extending a MediaSpace feature through a module is done by implementing one of the interfaces (your_KMS_site.mediaspace.kaltura.com/kb/tab/interfaces} that are available in MediaSpace.

Interfaces

Your model class should implement any of the interfaces, according to the features you would like to provide through your module.

A basic model class of the 'mymodule' module would look like the following:

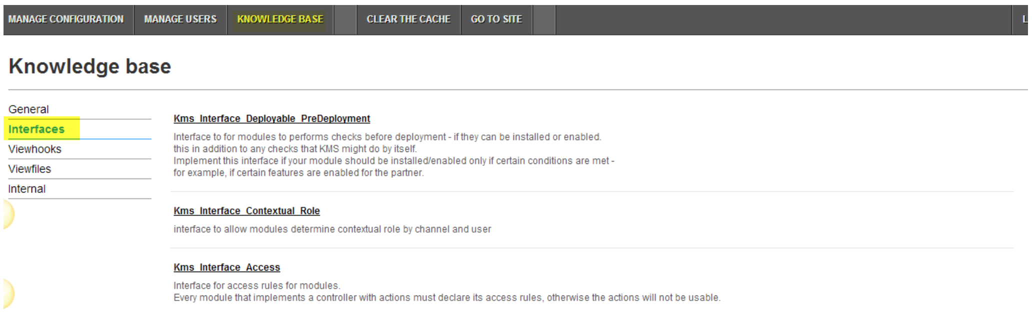
```
class Mymodule_Model_Mymodule extends Kms_Module_BaseModel
{
}
```

The abstract class Kms_Module_BaseModel implements 2 interfaces:

- Kms_Interface_Model_ViewHook - allows modules to provide their HTML output to be included in core views of

MediaSpace.
- Kms_Interface_Access - every module that has a controller must declare the access rules for its actions to integrate with MediaSpace's roles.

To learn and review up to date interfaces, select the Knowledge Base tab in the MediaSpace Admin and then select Interfaces.



Knowledge base

Kms_Interface_Deployable_PreDeployment

Interface to for modules to performs checks before deployment - if they can be installed or enabled.
this in addition to any checks that KMS might do by itself.
Implement this interface if your module should be installed/enabled only if certain conditions are met -
for example, if certain features are enabled for the partner.

Kms_Interface_Contextual_Role

interface to allow modules determine contextual role by channel and user

Kms_Interface_Access

Interface for access rules for modules.
Every module that implements a controller with actions must declare its access rules, otherwise the actions will not be usable.

Additional NameSpaces

If you need to add additional independent classes to your module, you can store them in one of the following namespaces that MediaSpace includes as part of the auto-loading process.

Note that you have to keep ZF naming convention so files will be found by the autoloader.

- plugins
- services
- views/helpers

For example, if you want to add a class which communicates with a 3rd party API (i.e. service) you should add your file under 'services' folder (in your module).

Your folder/file structure would look like:

```
{module}/
         controllers/
                     ...
         models/
                {module}.php
         views/
               scripts/
                       ...
         assets/
                ...
         services/
                  Thirdparty.php
         admin.ini
         default.ini
         module.info
```

Your class name would look like:

```
class {Module}_Service_Thirdparty
{
}
```

Note that {Module} should be replaced with the name of your module with the first character in upper case.

Module Assets

Modules can contain js, css, flv, and image files to be used in their views. The files are located in the module's 'assets' folder.

To access the files, use a URL of the form:

```
http://[kms url]/[build number]/[module name]/asset/[file name]
```

View Hooks

Modules are allowed to add HTML content to different locations in KMS pages.

This capability, in KMS, is called "View Hook" - the ability to hook into an existing view and adding output to that view.

This is built in a way that KMS invokes (internally) page requests and uses the response as the HTML that is added in the "core" view script.

A module that implements viewhook must have, at least, the following:

- Model class
  - Declares which view hooks are implemented by the module. For each viewhook - specify which action and controller of the module should be invoked, and the importance order between other modules implementing the same viewhook.
  - Set access rules for any of the controllers and actions provided by the module.
- At least one controller - to expose actions that are invoked as viewhooks.
- Relevant view scripts to serve as the output for each of the actions.