Kaltura MediaSpace Module Development Guidelines and Best Practices

Developer Guide

Version: Kaltura MediaSpace 5

Kaltura
open source video

Kaltura Business Headquarters

5 Union Square West, Suite 602, New York, NY, 10003, USA

Tel.: +1 800 871 5224

Contents

SECTION 1

Understanding Kaltura MediaSpace Modules

Kaltura MediaSpace (also referred to as KMS) enables community, collaboration and social activities by leveraging the power of online video by using an out-of-the-box video portal.

KMS is a Kaltura API based application. KMS uses its associated Kaltura Account to store content, metadata and user information and does not have a local database of its own.

KMS is a highly customizable application. Administrators can configure and manage the application through an administration interface and developers can extend its functionality and add new features through its modular architecture of Model View Controller (MVC) based modules.

KMS Architecture – Core and Modules

This section describes the following:

- Core Architecture
- Modules

Core Architecture

KMS is implemented as a Zend-Framework application and as such, the main architectural feature to consider is the Model-View-Controller (MVC) based modules.

Kaltura MediaSpace's core is divided into two main parts:

- Application MVC – the core set of models, views and controllers. The main features of KMS, which are considered core, are implemented in this part of the system.
- KMS library – a set of classes that the application uses to manage different flows and resources.

The KMS library also implements the generic infrastructure that allows extending MediaSpace via modules and themes.

NOTE: Custom themes are not covered in this document.

Kaltura MediaSpace Modules

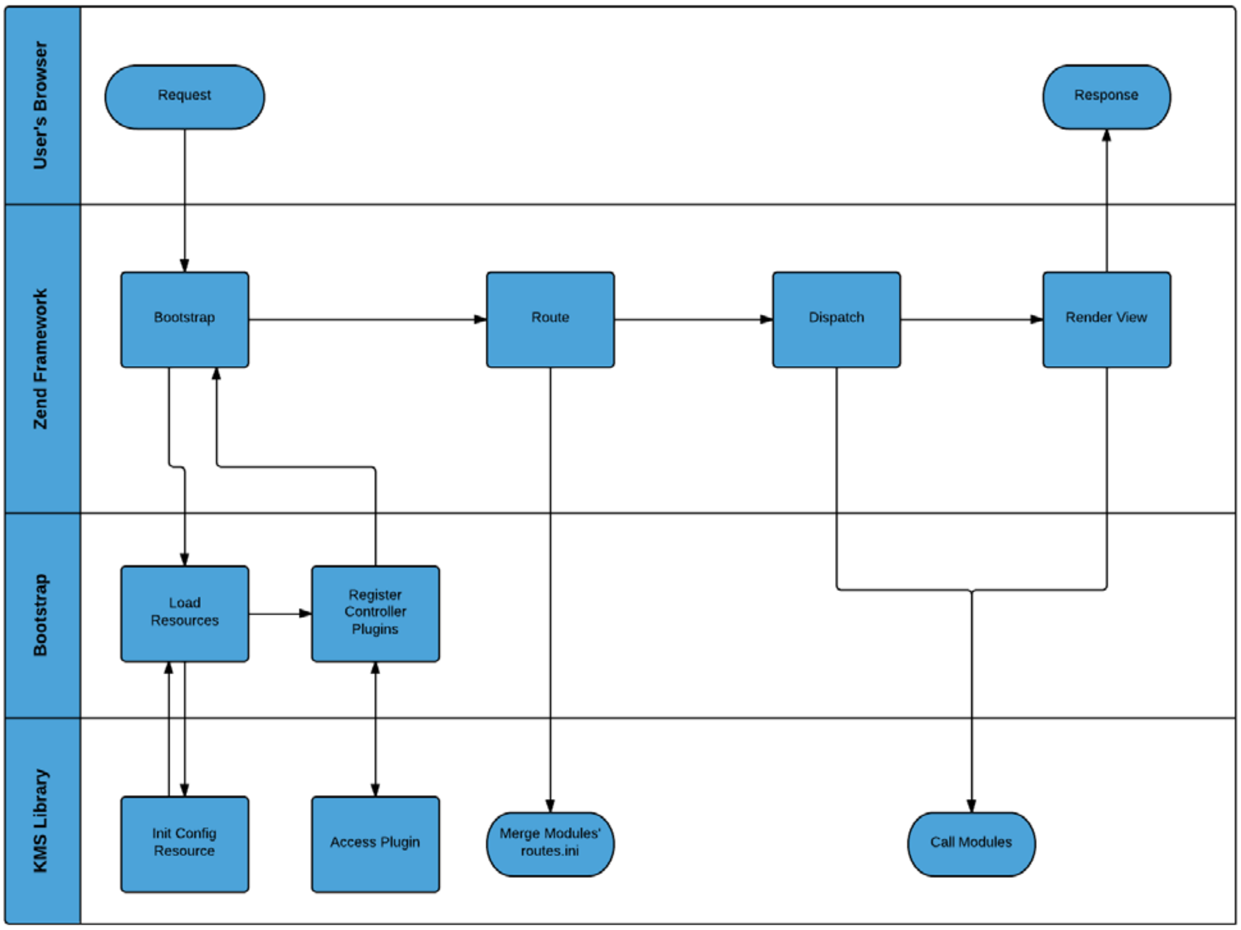
Kaltura MediaSpace modules are code bundles that are deployed in a KMS installation.

KMS modules are used to add new capabilities and features to the core functionality of KMS, and even affect or change the way KMS was designed to behave.

To keep the system's behavior cohesive, unified and supported, modules are also expected to use the MVC pattern and be implemented according to the guidelines described in this document.

Kaltura MediaSpace Application Workflow

The following diagram describes the application flow of KMS.



Most of the flow actually represents processes that are managed by the Zend Framework (such as bootstrap and dispatch).

The interesting and important part of the diagram is the bottom swim lane – controlled by the KMS Library. These are the main entry points where KMS interacts with modules.

How Are Modules Initiated?

During different phases KMS "reaches out" to all enabled modules, through different predefined class interfaces, to allow modules to participate in the application flows.

For example, during the initialization of the Access Plugin, KMS loads all the modules that implement an interface called "*Kms_Interface_Access*" and calls the *getAccessRules()* method that these modules are obliged to implement.

Any KMS module would implement the list of interfaces, which define the entry points that the module would like to be called upon.

Why Use Class Interfaces?

The primary reason for choosing to use class interfaces to implement the KMS modularity is to be able to maintain backwards compatibility.

Interfaces should not change between KMS versions, unless absolutely necessary.

During development we cautiously consider any changes before applying a change to an existing interface.

The secondary reason for choosing to use class interfaces to implement the KMS modularity is to be able to easily determine which module should be called for what entry point.

Module Development Guidelines

Except for modules implemented for private use, there are three distribution channels for custom KMS modules:

- Bundled within the Kaltura MediaSpace download package (as a default module, for self-hosted KMS installations)
- Deployed on Kaltura's MediaSpace SaaS Environment
- Downloadable via Kaltura Exchange to be used in self-hosted KMS installations

You are encouraged to consider the recommendations in this section for all distribution channels, including modules implemented for private use only.

The goal of these recommendations is to ensure high quality software to be developed and distributed as a KMS module.

KMS Module Coding Standards

KMS follows Zend Framework coding standards – Kaltura MediaSpace modules are required to follow these standards as well.

KMS Module Naming Convention

It is recommended to prefix the module name with the company or service name to avoid conflicts with other modules.

Refrain from using the customer name as a module name.

Currently, KMS expects the module folder name to be all lowercase, and based on Zend Framework naming conventions, class names of the module should start with a capital letter.

For example, if your module's name is *example*, your model class name would be *Example_Model_Example*.

General Do's and Don'ts

- Stick to what KMS allows you to do.
  - Implement the available interfaces (under library/Kms/Interfaces/).
  - Use the asset-delivery action for providing your modules JS/CSS/image (or other static) files. Remember to build your URLs via the *cdnUrl* ViewHelper.
  - Avoid implementing your module through hacks based on Zend Framework behavior and capabilities.

NOTE: Keeping these recommendations will ensure maximum forward compatibility of your module, thus lowering the chances of significant update work when upgrading your module to future versions of KMS.

  - Your module should NEVER modify KMS core or rely on hacks or patches applied to core files of KMS.

NOTE: Failing to comply with these recommendations or modifying core code makes the KMS upgrade process more difficult and complex.

- o Write safe code so errors or misconfigurations in your module will not crash the entire application.

NOTE: Keeping this recommendation ensures that even if your module is malfunctioning it does not stop the application from working, thus allowing more flexibility in support.

- o Modules should not modify or override KMC core/default CSS rules or JavaScript code.

NOTE:

- Keeping this recommendation should assist keeping highest forward compatibility.
- It is OK for your module to provide CSS for its own element, however you are recommended to use available CSS definitions from Twitter Bootstrap (utilized by KMS).
- It is OK for your module to provide JS for its own functionality.

- o Keep your code organized correctly, according to an MVC pattern.
  - o Implement logic and any API (for example, Kaltura API) calls in the ***model.***
  - o Call model methods from controller-action and prepare view data.
  - o Keep only simple loop and if logic in view while rendering HTML.
  - o Implement ViewHelper if more complicated (or repeatable) logic is required within a view.
- o Use KMS cache infrastructure when possible and relevant.
- o Do not leave calls to *Kms_Log::trace()* in your final code.
- o Choose smartly when to use available public methods from existing models (for implementing API calls to Kaltura) or implement your own code for making API calls.
- o When using existing KMS models you should usually get a model's instance through *Kms_Resource_Models::get{ModelName}()*. For example:

```
$entryModel = Kms_Resource_Models::getEntry();
```

NOTE: Refrain from instantiating model objects like:
$entryModel = new Application_Model_Entry();

If you do choose to instantiate model objects, be extra careful as doing so can have global implications on KMS behavior.

- If your module depends on another module, remember to implement the *Kms_Interface_Model_Dependency* interface.
- Always set "enabled=0" in your module's default.ini file. 3rd party modules should always be disabled by default.

Security Considerations

Avoid performing any actions that you would not do if you were to host the code on your own server.

General Security Guidelines

- Make sure your code is safe against OWASP Top 10 vulnerabilities.

- Avoid implementing file uploads directly to KMS servers. Use Kaltura's API and widgets to upload content to Kaltura.
- Do not use PHP's eval() function to run user-input-based code.
- Never output sensitive information to the browser (for example, in case of errors).
- Avoid using ini_set in your module's code.
- Do not use execute shell commands (for example, functions such as: exec, system, or passthru).

Specific Security Guidelines

- If you implement a form of your own to update data, protect against CSRF by having your form class extend the Application_Form_Base that handles CSRF protection for you.
- If you perform any redirects in your actions, make sure you only allow internal redirects or redirects to URLs that are valid and accepted. In other words – sanitize user-input URLs before redirecting to such.
- If your module writes cookie with sensitive data, set the httpOnly flag on your cookie so it cannot be read by malicious client-side code.
- Sanitize any user input to protect against XSS.

Expected Documentation

The following documentation is expected for KMS Modules.

- Module Description
- Code Comments
- Release Notes
- Setup Guide or Manual

Module Description

KMS modules are expected to include a file called "module.info" at the root of the module folder.

The "module.info" file is expected to be in INI format.

You are expected to provide this file with your module so it will be clear to the KMS Administrators and other developers what your module does.

Please use the "description" property in this file and provide a short description of the module added functionality.

Code Comments

Provide meaningful code comments in your module, specifically block-comments for classes and methods, so whoever does the code review for certification purposes will be able to easily understand what each and every piece of code is responsible for.

Release Notes

You are expected to publish release notes with every version release of your module, listing any new or modified features, known and resolved issues and important notes or configurations if there are such.

The release notes should also include contact information for your sales and support, including phone and email. Remember that although your module may be bundled with KMS you are still the owner of the code for support purposes.

The release notes file will be named CHANGELOG.txt and placed on the root in your module directory, the formatting of the file should adhere to the Markdown syntax guidelines.

If you host a public version of the release notes on your server, customers will appreciate having a link to a public release notes in the description field within the module.info file (which will show in the module admin panel).

Setup Guide or Manual

If enabling your module also requires special configurations or registration to 3rd party services, please provide a module setup guide (or module user manual) for users to be able to setup, manage and use your module.

A link to the guide should be placed in the description within module.info file.

Certification

For all distribution channels, a module must go through certification process and be certified before made available and recommended to Kaltura MediaSpace customers.

Submitting a Module for Certification

Modules and all following versions of the module (no matter how small of a change) are required to go through the Kaltura code review certification before being deployed on Kaltura's SaaS, and/or recommended for Kaltura customers' use.

When your module is ready for certification, send your module to your Kaltura point of contact and ask for the module to go through the certification process.

To properly track versions, bug fixes and new additions, and significantly shorten certification times we require that each module (and new versions of it) submitted for review, will be on a GitHub.com repository that can be accessed by the Kaltura engineers who will perform the certification code review.

FAQ

Who will be performing the module certification process?

An official developer of the Kaltura MediaSpace Engineering team will perform the certification process.

What does the certification process include?

The certification process includes the following:

1. Code review for compatibility with developer guidelines mentioned in this document, security review of the code submitted, and that the module performs what was stated in its release notes.
2. Sanity admin and user interfaces and workflows QA.
3. Validation of all documentation required including setup manual, and release notes file.

NOTE: Code review may also include the use of automatic code-analysis tools if Kaltura finds the need for it.

What do I do if my module failed the certification process?

Fix the issues reported by Kaltura and resubmit the module for certification.

How do I release new versions or apply bug patches to my KMS module?

Regardless of the significance of changes applied to the new version, it is required that every module version submitted for deployment will undergo proper certification by the Kaltura MediaSpace Engineering team. To make sure that your certification is completed quickly, please make proper use of a GitHub.com repository when developing your module, which allows the code reviewer to track the changes at the code level.

What may cause my module certification to fail?

The following may cause the certification of a module to fail (not ordered by significance):

- Critical or major bugs found during certification.
- Exposing of sensitive information to user or sending such information to external systems. Examples for sensitive information: Kaltura account information, Kaltura admin secret, and information about the webserver.
- Inability to perform QA due to missing documentation.
- Violation of any item from the Specific Security Guidelines section.
- Violation of any item from the General Security Guidelines section, under the discretion of the reviewer.
- Modification of KMS core code for the module to be operable.
- Exploiting Zend Framework behavior/capabilities instead of (or in addition to) using dedicated KMS interfaces – under the discretion of the reviewer.
- Trace logs left in the code.
- Not implementing the dependency interface while using another module’s code or decisions.
- Any attempt to obscure what your module or parts of its code are doing.
  For example, if you want to use a minified JS it is OK but you must include a non-minified version of the JS code in the module as well.
- Loading static assets not via *cdnUrl* ViewHelper.
- Missing Release Notes on your CHANGELOG.txt file.
- Missing critical setup or configuration notes required for successfully enabling your module.
- Breaking standard user interfaces in MediaSpace.

A module’s certification may fail for additional reasons not listed above.

For all cases of certification failure, the module developers will be notified about the reasons for failure and guidelines to resolve the issues found. You will be given the opportunity to resubmit the module for certification after you fix/resolve the issues in the module.

SECTION 4

UI Considerations

The purpose of your custom module is to provide an integrated experience that is part of Kaltura MediaSpace and enables additional functionality while following existing user experience and user flows.

If your custom module introduces a user interface that is vastly different from Kaltura MediaSpace, you may be forcing the user to learn a completely new interface which may be difficult for them to embrace. Be certain that the user interface elements are consistent with the Kaltura MediaSpace interface and HTML tags to support the CSS file, so your user interface does not "stand out".