Creating Kaltura Server Plugins User Manual

Version: Eagle

Kaltura
open source video

Kaltura Business Headquarters

200 Park Avenue South, New York, NY. 10003, USA

Tel.: +1 800 871 5224

Contents

Preface

This preface contains the following topics:

About this Manual

This document describes how to expand the Kaltura system using server plugins.

NOTE: Please refer to the official and latest product release notes for last-minute updates Technical support may be obtained directly from: Kaltura Support.

Contact Us:

Please send your documentation-related comments and feedback or report mistakes to the Knowledge Management Feedback group.

We are committed to improving our documentation and your feedback is important to us.

Audience

This manual is intended for Kaltura server developers and community members.

To understand this document, you need to be familiar with:

- Kaltura terminology
- Kaltura server architecture
- PHP programming language

Applicability

This document applies to Kaltura API version 3 and later.

Prerequisites

- To access the Kaltura API, you require:
  - A Kaltura partner account
  - Kaltura partner identifiers
- To create a Kaltura server plugin, you require:

- An installed Kaltura server
- A PHP development environment

Document Conventions

Kaltura uses the following admonitions:

- Note
- Workflow

NOTE: Identifies important information that contains helpful suggestions.

Workflow: Provides workflow information.

1. Step 1
2. Step 2

Related Documentation

In addition to this manual, product documentation is available on the Kaltura Knowledge Center.

NOTE: Please remember to review all product release notes for known issues and limitations.

- Kaltura API Documentation Set
- Introduction to the Kaltura API Architecture
- Kaltura API Usage Guidelines
- Introduction to Kaltura Client Libraries

CHAPTER 1

Understanding Kaltura Server Plugins

Platform Overview

The Kaltura Open Source Video Platform figure shows core functionality at the center of the Kaltura platform.

Web services, API clients, and batch processes expose and optimize core functionality.

Widgets and applications access and extend core functionality.

NOTE: The figure relates to client- and server-side components.

For a detailed explanation, refer to The Kaltura Video Platform Architecture Overview.

Server Plugin Definition

A server plugin:

- Extends Kaltura server functionality, such as services and batches
- Can extend:
  - A Kaltura application
  - A Kaltura widget
  - Another server plugin
- Can utilize API clients, web services, and batch processes that expose and optimize Kaltura core functionality

Goals

A server plugin can achieve the following goals:

- Expand the Kaltura server data model.
- Expand Kaltura functionality.

- Expand the Kaltura configuration.

Business Considerations

You create a server plugin to:

- Rapidly and cost-effectively expand upon the core Kaltura platform for your own specific use case.
- Create an application to publish to the community.
- Potentially generate revenue from your contribution to the Kaltura system.

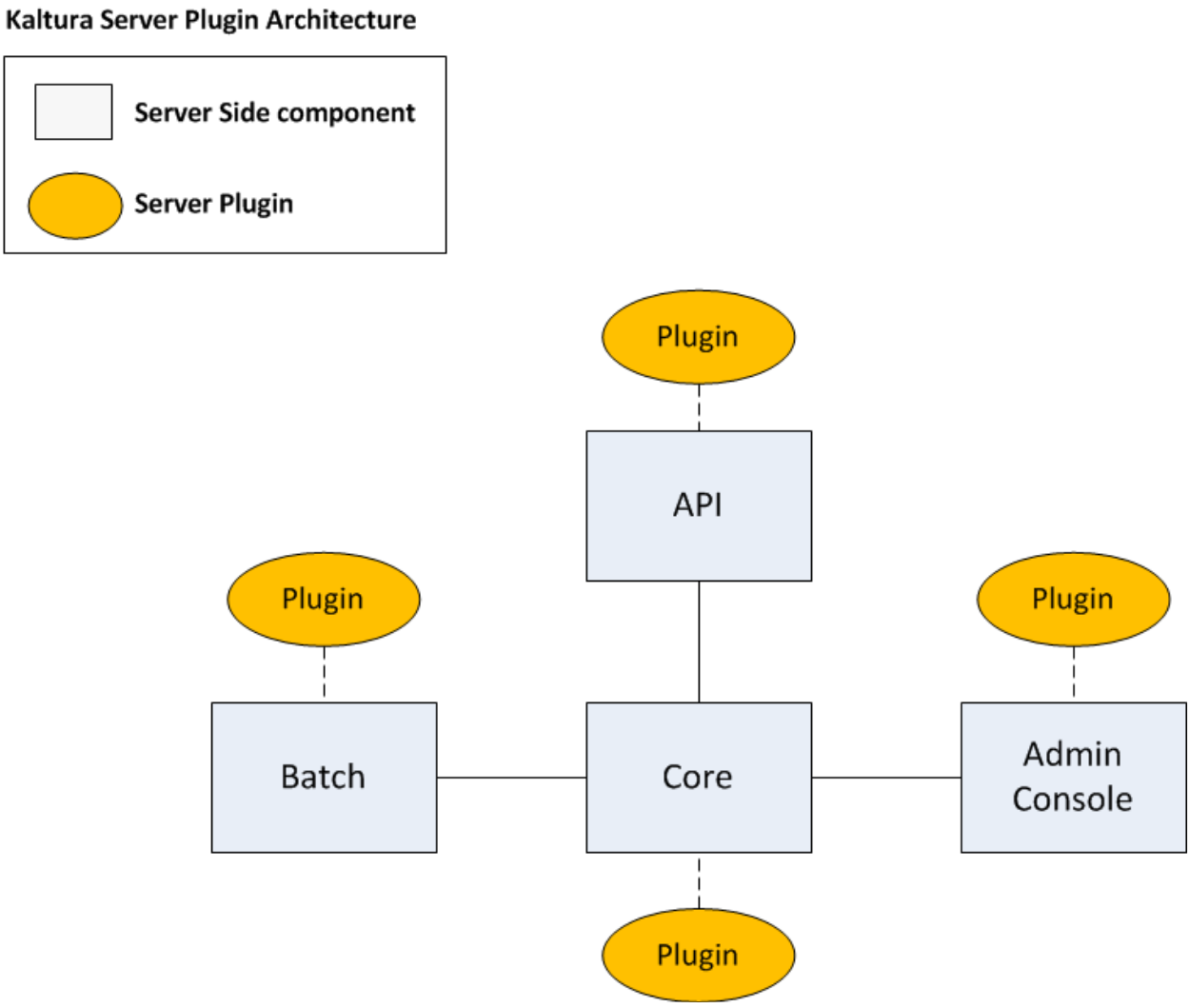
To learn more, refer to the Kaltura Application Exchange.

The Kaltura Application Exchange is a publicly available site where you can offer services and software to the Kaltura community.

CHAPTER 2

Plugin Architecture

Architecture Diagram



Server Plugin Extension Points

You can create a server plugin for the following types of interface extension points:

- API
  - Services
  - API objects
  - API enumerators
  - Partner-level permissions

- Core
  - Bulk upload data
  - Decision layer
  - Configuration
  - DB connection
  - Indexing (Sphinx/Solr)
  - Event management
  - MRSS management
  - Database objects
  - Valid value enumerators
  - Partner-level permissions
- Batch
  - Bulk upload engines
  - Conversion engines
- Admin Console
  - Pages
  - Dialogs
  - Publisher actions
  - Entry investigation

Affected Modules

A server plugin may affect any object in modules such as:

- API
  - API services
  - API client libraries
- Core
- Batch
  - Workers
  - Engine
  - Objects
- Admin Console
  - Pages
  - Forms
  - Menu items
- Configuration
- Database
  - Connections
  - Tables

Sample Implementation

A plugin adds new:

- Tables

- Database connections that access the tables
- Propel objects that load data from the tables
- API objects that reflect the Propel objects
- API object services:
  - Add
  - Update
  - Delete
  - Get
- Batch processes that use the new services
- Admin Console pages that manage plugin components

CHAPTER 3

Plugin Facility Classes

Understanding Plugin Facility Classes

Plugin facility classes:

- Implement the plugin infrastructure
- Are used by the plugin infrastructure
- Serve as interfaces to the plugins from any code in the system

KalturaPluginManager

Enables access to Kaltura plugins at runtime from every code component.

Remarks

Since the class constructor is protected, this class cannot be instantiated. Only static methods may be used.

Management Methods

Name	Description
addPlugin	Enables adding a plugin at runtime, even if the plugin is not configured as enabled by default.
getPluginInstance	Returns an instance of all loaded plugins according to name.
getPluginInstances	Returns instances of all loaded plugins. The method may be filtered according to a specific interface.
getPlugins	Returns class names of all enabled plugins.

Facility Methods

NOTE: The facility methods are implementations of interface class methods.

Name	Description
getObjectClass	Returns the class name of extended objects from plugins.
loadObject	Returns an extended object instance from plugins.
mergeConfigs	Returns an extended configuration from plugins.
getExtendedTypes	Returns all enum values that extend the base enum value.

addPlugin

Enables adding a plugin at runtime, even if the plugin is not configured as enabled by default.

```
public static function addPlugin($pluginClass)
```

Context

Used by the API clients generator.

Parameters

Name	Input/Output	Type	Description
pluginClass	Input	String	The class name of the added plugin

getObjectClass

Returns the class name of extended objects from plugins.

```
public static function getObjectClass($baseClass, $enumValue)
```

Remarks

- This method is an implementation of IKalturaObjectLoader::getObjectClass.
- This method may be used for any object inheritance.

Context

Used for Propel and API inherited objects, such as new entry types, asset types, and asset param types.

Parameters

Name	Input/Output	Type	Description
baseClass	Input	String	The extended class
enumValue	Input	String	The value of the pluginable enumerator that is added by a plugin. Indicates the object type.

Return Value

Type	Description
object	The extended class name

getPluginInstance

Returns an instance of all loaded plugins according to name.

```
public static function getPluginInstance($pluginName)
```

Context

- Used by the API type reflector
- Used internally by many plugins to load an instance of themselves

Parameters

Name	Input/Output	Type	Description
pluginName	Input	String	The name of the plugin

Return Value

Type	Description
KalturaPlugin	A plugin instance that implements the interface

getPluginInstances

Returns instances of all loaded plugins. The method may be filtered according to a specific interface.

```
public static function getPluginInstances($interface = null)
```

Context

Used when a specific interface implementation is required.

Parameters

Name	Input/Output	Type	Description
`interface`	Input (optional)	String	The requested interface. Use null to return all plugin instances.

Return Value

Type	Description
Array	`<KalturaPlugin>` plugin instances that implement the interface

getPlugins

Returns class names of all enabled plugins.

```
public static function getPlugins()
```

Return Value

Type	Description
Array	The plugins in a plugin class. Format: `array[pluginName] = pluginClass`

loadObject

Returns an extended object instance from plugins.

```
public static function loadObject($baseClass, $enumValue, array $constructorArgs =
  null)
```

Remarks

This method is an implementation of IKalturaObjectLoader::loadObject.

Parameters

Name	Input/Output	Type	Description
`baseClass`	Input	String	The extended class
`enumValue`	Input	String	The value of the pluginable enumerator that is added by a plugin. Indicates the object type.
`constructorArgs`	Input	Array (optional)	The constructor arguments

Return Value

Type	Description
`object`	The extended class instantiated object

mergeConfigs

Returns an extended configuration from plugins.

```
public static function mergeConfigs(Iterator $config, $configName, $valuesOnly =
  true)
```

Remarks

This method is an implementation of IKalturaConfigurator::getConfig.

Context

Used by the API client generator, testme, and testmeDoc configurations.

Parameters

Name	Input/Output	Type	Description
`config`	Input	`Iterator`	The base configuration object to be merged
`configName`	Input	String	Name of the configuration to be searched in the plugins
`valuesOnly`	Input	Boolean	True —New keys will not be added to the source configuration. False — New keys will be added to the source configuration.

Return Value

Type	Description
`Iterator`	The merged configuration object

getExtendedTypes

Returns all enum values that extend the base enum value.

```
public static function getExtendedTypes($baseClass, $enumValue)
```

Remarks

This method is an implementation of IKalturaTypeExtender::getExtendedTypes.

Context

Used by the API and core to query the database for specific types and all of their extended types.

Parameters

Name	Input/Output	Type	Description
`baseClass`	Input	`Iterator`	The extended class
`enumValue`	Input	String	The value of the base class type. Indicates the object type.

Return Value

Type	Description
Array	The enum values that extend the base enum value

KalturaPlugin

Implements the base instance loader according to interface.

KalturaPlugin is the base abstract class for all Kaltura plugins.

Implements

IKalturaPlugin

Methods

Name	Description
getInstance	Returns itself if it implements the searched interface.

getInstance

Returns itself if it implements the searched interface.

```
public function getInstance($interface)
```

Context

Overridden by plugins to return additional managers that implement different plugin interfaces.

Parameters

Name	Input/Output	Type	Description
interface	Input	String	The searched interface

KalturaDependency

Defines that a plugin is dependent on another Kaltura plugin.

Used by plugins that implement IKalturaPending.

Variables

Name	Type	Description
minVersion	KalturaVersion	The minimum version of the plugin that the current plugin depends on
pluginName	String	The name of the plugin that the current plugin depends on

Methods

Name	Description
__construct	Instantiates a new KalturaDependency object.
getMinimumVersion	Retrieves the lowest Kaltura version that a plugin supports.
getPluginName	Retrieves the name of a plugin that is dependent on a specific Kaltura version.

__construct

Instantiates a new KalturaDependency object.

```
public function __construct($pluginName, KalturaVersion $minVersion = null)
```

Parameters

Name	Input/Output	Type	Description

Name	Input/Output	Type	Description
`pluginName`	Input	String	The name of the plugin
`minVersion`	Input	`KalturaVersion` (optional)	The minimum version identifier

getMinimumVersion

Retrieves the lowest Kaltura version that a plugin supports.

```
public function getMinimumVersion()
```

Return Value

Type	Description
`KalturaVersion`	The minimum version

getPluginName

Retrieves the name of a plugin that is dependent on a specific Kaltura version.

```
public function getPluginName()
```

Return Value

Type	Description
String	The name of the plugin

KalturaVersion

Identifies a Kaltura software build version.

May be used to define:

- A Kaltura plugin version
- A plugin's dependency on a specific version

Methods

Name	Description
__construct	Instantiates a new KalturaVersion object.
getBuild	Retrieves the identifier of a Kaltura software build version.
getMajor	Retrieves the major version identifier of a Kaltura software build.
getMinor	Retrieves the minor version identifier of a Kaltura software build.
isCompatible	Checks whether the current version is equal to or lower than the requested version.
toString	Concatenates the major, minor, and build identifiers of a Kaltura software build.

__construct

Instantiates a new KalturaVersion object.

```
public function __construct($major, $minor, $build, KalturaVersion
   $brokenCompatibilityVersion = null)
```

Parameters

Name	Input/Output	Type	Description
`major`	Input	String	The major version identifier
`minor`	Input	String	The minor version identifier
`build`	Input	String	The build version identifier
`brokenCompatibilityVersion`	Input (optional)	String	The identifier of a version that is incompatible with a plugin

getBuild

Retrieves the identifier of a Kaltura software build version.

```
public function getBuild()
```

Return Value

Type	Description
Integer	The build version identifier

getMajor

Retrieves the major version identifier of a Kaltura software build.

```
public function getMajor()
```

Return Value

Type	Description
Integer	The major version identifier

getMinor

Retrieves the minor version identifier of a Kaltura software build.

```
public function getMinor()
```

Return Value

Type	Description
Integer	The minor version identifier

isCompatible

Checks whether the current version is equal to or lower than the requested version.

```
public function isCompatible(KalturaVersion $version)
```

Parameters

Name	Input/Output	Type	Description
`version`	Input	`KalturaVersion`	The Kaltura version identifier

Return Value

Type	Description
Boolean	True — The Kaltura version is compatible.

Type	Description
	False — The Kaltura version is incompatible.

toString

Concatenates the major, minor, and build identifiers of a Kaltura software build.

```
public function toString()
```

Return Value

Type	Description
String	An identifier that concatenates the major, minor, and build identifiers

KalturaAdminConsolePlugin

Enables a plugin to add pages to the Admin Console.

A plugin that adds Admin Console pages must extend and implement the KalturaAdminConsolePlugin abstract class for each page.

Methods

Name	Description
accessCheck	Checks whether a plugin has permission to access the Admin Console.
action	Specifies the name of a plugin action.
doAction	Implements a plugin action.
getNavigationActionLabel	Returns the name of a Admin Console menu label.
getNavigationActionName	Returns the name of a plugin action.
getNavigationRootLabel	Returns the root path of the Admin Console menu label.
getRequiredPermissions	Retrieves the permissions that are required to modify the Admin Console.
getTemplatePath	Returns the absolute file path of a PHTML template.

accessCheck

Checks whether a plugin has permission to access the Admin Console.

```
public function accessCheck($currentPermissions)
```

Parameters

Name	Input/Output	Type	Description
currentPermissions	Input	String	The permissions that currently apply

action

Specifies the name of a plugin action.

```
public function action(Zend_Controller_Action $action)
```

Parameters

Name	Input/Output	Type	Description
action	Input	Zend_Controller_Action	The action executed by the plugin

doAction

Implements a plugin action.

```
abstract public function doAction(Zend_Controller_Action $action);
```

Parameters

Name	Input/Output	Type	Description
action	Input	Zend_Controller_Action	The action executed by the plugin

getNavigationActionLabel

Returns the root path of the Admin Console menu label.

```
public function getNavigationActionLabel()
```

Return Value

Type	Description
String	The Admin Console menu label. Use null to exclude the action from navigation.

getNavigationActionName

Returns the name of a plugin action.

```
public function getNavigationActionName()
```

Return Value

Type	Description
String	The name of the plugin action

getNavigationRootLabel

Returns the root path of the Admin Console menu label.

```
public function getNavigationRootLabel()
```

Return Value

Type	Description
String	The root path of the Admin Console menu label. Use null for the highest level root.

getRequiredPermissions

Retrieves the permissions that are required to modify the Admin Console.

```
abstract public function getRequiredPermissions();
```

getTemplatePath

Returns the absolute file path of a PHTML template.

```
abstract public function getTemplatePath();
```

Return Value

Type	Description
String	The absolute file path of a PHTML template

CHAPTER 4

Plugin Extension Points

Understanding Plugin Extension Points

What is a plugin extension point?

A plugin extension point is an interface that plugins can implement to extend Kaltura server behavior.

Example

`IKalturaMrssContributor` is an extension point that enables plugins to contribute additional XML data to the default core MRSS.

`kMrssManager` uses the `IKalturaMrssContributor` extension point.

After creating the default core MRSS, `kMrssManager`:

- Asks the plugin manager for all of the plugins that implement `IKalturaMrssContributor`
- Individually calls the `contribute` method of the plugins that implement `IKalturaMrssContributor`

Available Plugin Extension Points

You can create server plugins that implement:

- Plugin Extension Points for Admin Console
- Plugin Extension Points for Bulk Upload
- Plugin Extension Points for Server Configuration
- Plugin Extension Points for Content Distribution
- Plugin Extension Points for Search Engines
- Plugin Extension Points for an MRSS
- Plugin Extension Points for Plugin Implementation

Plugin Extension Points for Admin Console

IKalturaAdminConsoleEntryInvestigate

Enables a plugin to add information to the Admin Console's Entry Investigation page (Batch Process Control tab).

Remarks

Called by the Admin Console.

After the core entry investigation page is printed, the Admin Console:

- Asks the plugin manager for all of the plugins that implement `IKalturaAdminConsoleEntryInvestigate`
- Individually calls the `getEntryInvestigatePlugins` method of the plugins that implement `IKalturaAdminConsoleEntryInvestigate`

- Prints the output of each `getEntryInvestigatePlugins` method

Extends

IKalturaBase

Implemented in Existing Plugins

Content Distribution

getEntryInvestigatePlugins

Retrieves additional information to be viewed in the Admin Console's Entry Investigation page.

The `Kaltura_View_Helper_EntryInvestigatePlugin` object represents the additional information, and contains:

- The view PHTML
- The template path
- The template's data array

```
public static function getEntryInvestigatePlugins();
```

Return Value

Type	Description
Array	The Entry Investigation plugins in `Kaltura_View_Helper_EntryInvestigatePlugin`

IKalturaAdminConsolePages

Enables a plugin to add pages and dialogs to the Admin Console.

After the Admin Console builds a menu, the Admin Console requests from the plugin manager all of the plugins that implement IKalturaAdminConsolePages and calls the getAdminConsolePages method for each one. The Admin Console adds the resulting pages to the menu.

Extends

IKalturaBase

Implemented in Existing Plugins

Kaltura Internal Tools

Content Distribution

Virus Scan

getAdminConsolePages

Retrieves pages that a plugin adds to the Admin Console.

A page is an instance of a KalturaAdminConsolePlugin implementation.

```
public static function getAdminConsolePages();
```

Return Value

Type	Description
Array	The pages added to the Admin Console

IKalturaAdminConsolePublisherAction

Enables a plugin to add actions to the Admin Console's Publisher Management page (Publishers tab).

Called by the Admin Console. After the Admin Console prints the partners list page, the Admin Console requests from the plugin manager all of the plugins that implement IKalturaAdminConsolePages and reviews the list of pages generated. For each page that implements IKalturaAdminConsolePublisherAction, the Admin Console adds JavaScript to the partners list page and adds the action options to the partners actions list.

Remarks

The IKalturaAdminConsolePublisherAction extension point:

- Is implemented by Admin Console pages that extend KalturaAdminConsolePlugin
- Is not implemented by Admin Console pages that extend server plugins

Extends

IKalturaBase

getPublisherAdminActionJavascript

Retrieves the JavaScript code to be added to the code of the Admin Console's Publisher Management page.

```
public function getPublisherAdminActionOptions($partner, $permissions);
```

Parameters

Name	Input/Output	Type	Description
`partner`	Input	String	The Kaltura publisher account
`permissions`	Input	String	The group of permission items associated with the added action

Return Value

Type	Description
Array	Strings that consists of `label` and `jsActionFunctionName`

getPublisherAdminActionOptions

Retrieves an action that a plugin adds to the Admin Console's Publisher Management page in the Publisher table's action combo box.

```
public function getPublisherAdminActionJavascript();
```

Return Value

Type	Description
String	The JavaScript code that defines the action that is added to the publisher list view

Plugin Extension Points for Bulk Upload

IKalturaBulkUpload

Enables a plugin to add a bulk upload handler engine.

The batch bulk upload requests from the plugin manager to load the bulk upload engine object according to the bulk upload type (saves as the job sub-type). Each plugin that implements IKalturaBulkUpload must define a new bulk upload type and an engine to handle the new type (for example, csv and xml).

Remarks

- A plugin may add bulk upload types.
- A bulk upload type must enable the following objects to load:
    - kBulkUploadJobData
    - KalturaBulkUploadJobData
    - KBulkUploadEngine.
- A plugin must extend the `BulkUploadType` enum with the new bulk upload type.

Extends

IKalturaBase

IKalturaEnumerator

IKalturaObjectLoader

Implemented in Existing Plugins

CSV Bulk Upload

XML Bulk Upload

Drop Folder Bulk Upload

getFileExtension

Returns the correct file extension for a bulk upload type.

```
public static function getFileExtension($enumValue);
```

Parameters

Name	Input/Output	Type	Description
`enumValue`	Input	Integer	The enum value in the API code

Return Value

Type	Description
String	The file extension for the bulk upload type

IKalturaBulkUploadHandler

Enables a plugin to handle additional data for a bulk upload.

Currently supported only by CSV Bulk Upload.

Called by the API (on the server side). After bulk upload results are saved, the API calls all the plugins that implement IKalturaBulkUploadHandler for the handleBulkUploadData method. Each plugin can check the additional fields that are unknown to the CSV Bulk Upload engine and check whether these fields are relevant to the called plugin. The plugin can save additional data that relates to the created entry, such as custom metadata.

Extends

IKalturaBase

Implemented in Existing Plugins

Metadata

handleBulkUploadData

Handles additional data from a bulk upload.

```
public static function handleBulkUploadData($entryId, array $data);
```

Parameters

Name	Input/Output	Type	Description
`entryId`	Input	String	The new entry added to a bulk upload
`data`	Input	Array	The new entry's data. Format: `key => value pairs`

Plugin Extension Points for Server Configuration

IKalturaConfigurator

Enables a plugin to append a configuration to an existing server configuration.

Called by testme, testmeDoc and the generator. After loading the configuration file, all plugins are called that implement IKalturaConfigurator for the getConfig method. The plugin manager joins all plugin configurations to the core configuration.

Extends

IKalturaBase

Implemented in Existing Plugins

Admin Console

Kaltura Internal Tools

System Partner

File Sync

Multiple Data Centers

Content Distribution

Document

Metadata

Virus Scan

CSV Bulk Upload

XML Bulk Upload

getConfig

Merges configuration data from the plugin.

```
public static function getConfig($configName);
```

Parameters

Name	Input/Output	Type	Description
`configName`	Input	String	The name of the existing server configuration

Return Value

Type	Description

Type	Description
`Iterator`	The iteration value

Plugin Extension Points for Content Distribution

IKalturaContentDistributionProvider

Enables a plugin to add a content distribution provider (also referred to as a *connector*).

Mostly used to extend the DistributionProviderType by adding the new provider type. Also returns the provider singleton instance.

Extends

IKalturaBase

Implemented in Existing Plugins

Content Distribution

getProvider

Returns the singelton instance of the plugin distribution provider.

```
public static function getProvider();
```

Context

- Exposed through the distribution profile.
- Called by the content distribution flow manager to get the provider and to call the provider methods, supported features, and disabled features. The purpose is to determine what the provider can do, for example, delete or update.

Return Value

Type	Description
`IDistributionProvider`	The singelton instance of the plugin distribution provider

getKalturaProvider

Returns an instance of a Kaltura API distribution provider that represents the singleton instance of the plugin distribution provider.

```
public static function getKalturaProvider();
```

Context

- Called by the `DistributionProvider` API service in the list action to expose the provider as an API object.

Return Value

Type	Description
`KalturaDistributionProvider`	An instance of a Kaltura API distribution provider that represents the plugin distribution provider singleton instance

contributeMRSS

Appends nodes and attributes associated with a specific distribution provider and entry to the Kaltura MRSS XML.

```
public static function contributeMRSS(EntryDistribution $entryDistribution,
   SimpleXMLElement $mrss);
```

Remarks

The action is used to add provider-specific data to the generated MRSS. The data can be used later in an XSL transformation to specify a data structure to send to the provider destination site.

Context

Called by the content distribution MRSS contributor to append provider-specific data to the MRSS XML.

Parameters

Name	Input/Output	Type	Description
`entryDistribution`	Input	`EntryDistribution`	The distribution entry whose data is appended to the MRSS
`mrss`	Input	`SimpleXMLElement`	The MRSS to which the data is appended

Plugin Extension Points for Search Engines

IKalturaCriteriaFactory

Enables a plugin to return an extended `KalturaCriteria` object according to a searched object type.

The `KalturaCriteria` object is used mainly to implement a search in indexing servers before searching the default mysql DB.

When an indexed object is searched, you can call IKalturaCriteriaFactory's getKalturaCriteria method instead of creating the default Propel criteria. The object peer checks its doSelect method to clarify whether the criteria is a KalturaCriteria object and applies its conditions in an external indexing server before applying the criteria on the mysql server.

Extends

IKalturaBase

Implemented in Existing Plugins

Solr Search

Sphinx Search

getKalturaCriteria

Creates a new `KalturaCriteria` for a specified object name.

```
public static function getKalturaCriteria($objectType);
```

Parameters

Name	Input/Output	Type	Description
`objectType`	Input	String	The object type for which to create a `KalturaCriteria`

Return Value

Type	Description
`KalturaCriteria`	The derived object

IKalturaSphinxConfiguration

Enables a plugin to add Sphinx indexes.

Called by plugin installation scripts to collect the configurations for all indexes built by the Sphinx configuration. Plugins may also extend core indexes by adding additional fields.

Extends

IKalturaBase

Implemented in Existing Plugins

Content Distribution

Sphinx Search

getSphinxConfigPath

Retrieves the configuration path for a Sphinx index.

```
public static function getSphinxConfigPath();
```

Return Value

Type	Description
String	The configuration file path

getSphinxSchema

Retrieves a configuration array for a Sphinx index.

```
public static function getSphinxSchema();
```

Return Value

Type	Description
array	The Sphinx index configuration

Plugin Extension Points for an MRSS

IKalturaMrssContributor

Enables a plugin to add XML nodes and attributes to an entry MRSS.

Implemented by the following facilities:

- kContentDistributionMrssManager
- kMetadataMrssManager

Remarks

`IKalturaMrssContributor` is an extension point that enables plugins to contribute additional XML data to the default core MRSS.

`kMrssManager` uses the `IKalturaMrssContributor` extension point.

After creating the default core MRSS, `kMrssManager`:

- Asks the plugin manager for all of the plugins that implement `IKalturaMrssContributor`
- Individually calls the `contribute` method of the plugins that implement `IKalturaMrssContributor`

Extends

IKalturaBase

contribute

Adds data to an MRSS.

```
public function contribute(entry $entry, SimpleXMLElement $mrss);
```

Remarks

The method is used to add provider-specific data to the generated MRSS.

Parameters

Name	Input/Output	Type	Description
`entry`	Input	`entry`	The entry whose data is appended to the MRSS
`mrss`	Input	`SimpleXMLElement`	The MRSS to which the data is appended

Return Value

Type	Description
`SimpleXMLElement`	The generated MRSS

Plugin Extension Points for Plugin Implementation

IKalturaDatabaseConfig

Enables a plugin to add database connections.

Currently not used.

Extends

IKalturaBase

getDatabaseConfig

Retrieves a database configuration.

```
public static function getDatabaseConfig();
```

Return Value

Type	Description
Array	The database configuration

IKalturaEnumerator

Enables a plugin to add enumeration values to those used by the Kaltura core's `BaseEnum` interface.

Called by the generator to generate a complete class with all available values.

Used by the API to check whether a value used is acceptable.

Extends

IKalturaBase

Implemented in Existing Plugins

Content Distribution

Document

Virus Scan

Additional Transcoding Engines

getEnums

Returns a list of enumeration class names that implement the `baseEnumName` interface.

```
public static function getEnums($baseEnumName = null);
```

Remarks

Plugins may add enumeration values to those used by the Kaltura core's `baseEnumName` interface. You implement `baseEnumName` by defining a class for one or more additional enum values. The `getEnums` action returns a list of the class names that you define to implement `baseEnumName`. This enables the plugin API to receive enumeration values that other plugins define, in addition to the values that the core defines.

Context

Called by the API client generator and document generator to expose the enumeration values that are added to the original enum values.

Parameters

Name	Input/Output	Type	Description
`baseEnumName`	Input	String	The core interface that defines enum values. Use null to return all plugin enums.

Return Value

Type	Description
Array	A list of enum class names that extend `baseEnumName`

IKalturaEventConsumers

Enables a plugin to consume server-side events.

All event consumers are called synchronously.

Used by the kEventsManager. After calling the core event consumers, lists all of the event consumers from all of the plugins that implement IKalturaEventConsumers and calls their consume methods.

Extends

IKalturaBase

Implemented in Existing Plugins

Multiple Data Centers

Content Distribution

Annotation

Audit Trail

Document

Metadata

Short Link

Virus Scan

Solr Search

Sphinx Search

getEventConsumers

Retrieves the event consumers used by the plugin.

```
public static function getEventConsumers();
```

Remarks

An event consumer implements the event consumer interfaces according to the events it desires to consume. The consumer interface always requires implementing the method that is called whenever the event is raised. Implementing the method enables the plugin to react to the event raised in that method.

Context

Called by the event manager to call all of the event consumers that are relevant to the raised event.

Return Value

Type	Description
Array	The list of event consumers

IKalturaMemoryCleaner

Enables a plugin to clean unused memory, instances, and pools.

Called by the API to clean all of the pools between different requests in the same multi request.

Called by the feed renderer once in every chunk of entries to ensure that the server does not run out of memory.

Extends

IKalturaBase

Implemented in Existing Plugins

Content Distribution

Annotation

Audit Trail

Metadata

Short Link

Virus Scan

Solr Search

Sphinx Search

cleanMemory

Cleans unused memory, instances, and pools.

```
public static function cleanMemory();
```

IKalturaObjectLoader

Enables a plugin to load and search extended objects and types.

Very generic and commonly used.

Used wherever an instantiated object may be extended or implemented by a plugin. For example, when an asset is instantiated from the database, the Propel checks whether it should instantiate a flavor asset, a thumbnail asset, or another type of asset that is defined by any plugin, such as caption asset or attachment asset.

Extends

IKalturaBase

Implemented in Existing Plugins

Multiple Data Centers

Content Distribution

Document

Metadata

Virus Scan

Additional Transcoding Engines

loadObject

Returns an object that is known only to the plugin, and extends the `baseClass`.

```
public static function loadObject($baseClass, $enumValue, array $constructorArgs =
   null);
```

Context

- Called by the Core `DistributionProfilePeer` to load the relevant `DistributionProfile` object.
- Called by each of the batch `KAsyncDistribute` extensions to load the `DistributionEngine` that is relevant to the job type and the provider.
- Called by the Admin Console `DistributionProfileConfigureAction` class to load the form configuration that extends `Form_ProviderProfileConfiguration` and is relevant to a specific configured provider.
- Called by the API `KalturaDistributionJobData` object when translated from a core object to load the `KalturaDistributionJobProviderData` object that is relevant to the provider.
- Called by the API `KalturaDistributionJobData` object when translated to a core object to load the `kDistributionJobProviderData` object that is relevant to the provider.
- Called by the API `KalturaDistributionProfileFactory` to load the correct `KalturaDistributionProfile` that is relevant to the provider.

Parameters

Name	Input/Output	Type	Description
`baseClass`	Input	String	The base class of the loaded object
`enumValue`	Input	String	The enumeration value of the loaded object
`constructorArgs`	Input (optional)	Array	The constructor arguments of the loaded object

Return Value

Type	Description

Object	The loaded object instance

getObjectClass

Retrieves a class name that is defined by the plugin and is known only to the plugin, and extends the `baseClass`.

```
public static function getObjectClass($baseClass, $enumValue);
```

Context

- Called by the Core `DistributionProfilePeer` to load the relevant `DistributionProfile` object.
- Called by each of the batch `KAsyncDistribute` extensions to load the `DistributionEngine` that is relevant to the job type and the provider.
- Called by the Admin Console `DistributionProfileConfigureAction` class to load the form configuration that extends `Form_ProviderProfileConfiguration` and is relevant to a specific configured provider.
- Called by the API `KalturaDistributionJobData` object when translated from a core object to load the `KalturaDistributionJobProviderData` object that is relevant to the provider.
- Called by the API `KalturaDistributionJobData` object when translated to a core object to load the `kDistributionJobProviderData` object that is relevant to the provider.
- Called by the API `KalturaDistributionProfileFactory` to load the correct `KalturaDistributionProfile` that is relevant to the provider.

Parameters

Name	Input/Output	Type	Description
`baseClass`	Input	String	The base class of the searched class
`enumValue`	Input	String	The enumeration value of the searched class

Return Value

Type	Description
String	The name of the searched object's class

IKalturaPending

Enables a plugin to define a dependency on another plugin.

Used by the plugin manager to decide whether a plugin can be used, according to the plugin's dependencies.

Extends

IKalturaBase

Implemented in Existing Plugins

Content Distribution

Drop Folder

dependsOn

Returns a Kaltura dependency object that defines the relationship between two plugins.

```
public static function dependsOn();
```

Context

Called by the plugin manager to check whether all required plugins are enabled.

Return Value

Type	Description
Array	The Kaltura dependency object

IKalturaPermissions

Enables a plugin to define the partners allowed to use the plugin.

Used by a plugin in its event consumers and API services. Before calling the event consumers or the API actions, checks that the partner is allowed to use the plugin.

Extends

IKalturaBase

Implemented in Existing Plugins

Admin Console

System Partner

File Sync

Content Distribution

Annotation

Audit Trail

Metadata

Virus Scan

isAllowedPartner

Grants or denies a partner permission to use a plugin.

```
public static function isAllowedPartner($partnerId);
```

Context

- Called by the API to decide whether a specific partner is allowed to use the API.
- Called by the flow manager to decide whether a specific event is relevant to the partner.

Parameters

Name	Input/Output	Type	Description
`partnerId`	Input	Integer	The partner ID of the partner being checked for permission

Return Value

Type	Description
Boolean	True — The partner is allowed to use the plugin. False — The partner is not allowed to use the plugin.

IKalturaPlugin

Associates a plugin with the Kaltura system.

Implemented by all plugins. Returns the plugin name.

Will be used in the future to add functionalities common to all plugins, such as version and description.

Remarks

Must be implemented by all plugins.

Extends

IKalturaBase

Implemented in Existing Plugins

Document

getPluginName

Retrieves the name of a plugin.

```
public static function getPluginName();
```

Return Value

Type	Description
String	The name of the plugin

IKalturaSearchDataContributor

Enables a plugin to return additional data to be saved in an indexed object.

Used by indexing managers such as kSphinxManager to collect additional values to be indexed on indexable objects. The indexing managers save the collected data on the indexing server.

Extends

IKalturaBase

Implemented in Existing Plugins

Content Distribution

Metadata

getSearchData

Returns an array of search data to be associated with the object.

```
public static function getSearchData(BaseObject $object);
```

Parameters

Name	Input/Output	Type	Description
`object`	Input	`BaseObject`	The object in which the data is saved

Return Value

Type	Description
Array	Key value pair of field to search => data

IKalturaServices

Enables a plugin to add new Kaltura API services.

Called by the generator to list all of the API services from all plugins in order to generate the client libraries.

Called by the API to check which class should be called for a requested service.

Extends

IKalturaBase

Implemented in Existing Plugins

Admin Console

Kaltura Internal Tools

System Partner

File Sync

Multiple Data Centers

Partner Aggregation

Content Distribution

Annotation

Audit Trail

Document

Metadata

Short Link

Virus Scan

getServicesMap

Retrieves a map of Kaltura API services.

```
public static function getServicesMap();
```

Return Value

Type	Description
Array	The map of API services. Format: `array[serviceName] = serviceClass`

IKalturaVersion

Enables passing a Kaltura version identifier to a plugin.

Used by the plugin manager to decide whether a plugin's version satisfies the plugin dependency.

Remarks

The Kaltura version may be important for dependencies between plugins.

Extends

IKalturaBase

Implemented in Existing Plugins

Content Distribution

XML Bulk Upload

getVersion

Retrieves the identifier of a Kaltura version.

```
public static function getVersion();
```

Return Value

Type	Description
`KalturaVersion`	The Kaltura version identifier

CHAPTER 5

How to Write a Plugin

This section describes how to write the components that are required for plugins.

Define Name

You must name the plugin.

Define Version

If other plugins may be dependent on the plugin, a plugin version is required.

Implement IKalturaVersion to return the current plugin version.

Recommendations:

- Hard code the version that is returned.
- Update the version whenever the code is changed.

Define Dependency

Define the plugins that are used by this plugin.

Implement IKalturaPending and return the list of required plugins.

Define Permitted Partners

The plugin may permit (hard-coded) access to specific partners, according to their configuration or ID. This is useful for built-in partners such as admin console (-2), batch (-1), shared content (0), and template (99).

In addition, the code can check whether a partner has the permission required to use the plugin.

Implement IKalturaPermissions to clarify whether the partner is permitted to use the plugin.

DB Table

Create a Propel schema XML file and generate your objects using the propel-gen pear command.

Object

Recommendations:

- Place your server-side objects under the lib/model folder
- Place your objects that are exposed to the API under the lib/api folder.

Core DB (Propel) Object

To add an object table using the Propel generator:

1. In a plugin's `config` folder, create a schema.xml file with your new objects (new tables).
2. In the `config` folder, execute the Propel generator to create all objects and peers.

Sphinx Index

To expand an existing Sphinx index or to add a new Sphinx index:

Implement IKalturaSphinxConfiguration.

To save data in a Sphinx index:

Extend IKalturaSearchDataContributor.
getSearchData returns the data to save in the index to Sphinx.

To search for data in a Sphinx index:

In a plugin, add a core class for a filter to:

- Extend `AdvancedSearchFilterItem`
- Implement the `apply` function to add clauses to select from the Sphinx index.

Search Data

Search data is additional textual content that is indexed in a search engine, such as Sphinx, in addition to the original object data.

For example, the Metadata plugin enables saving additional information related to an entry object, such as Author. This field should be searchable, so if someone sets the Author field value to *Judy Garland*, the entry should be returned in the search results when either *Judy* or *Garland* is searched.

For this entry to be found, you need to add the searched values to the entry search engine, even though the searched data is not part of the entry data, but is part of the related metadata that belongs to the Metadata plugin.

To add search data, you must implement the IKalturaSearchDataContributor interface and the getSearchData method that returns an array. The array keys are the fields in the search engine that should be populated, such as *metadata_data*, and the values are the content that should be available for search.

To support your own plugin fields, use the Sphinx configurator (see Sphinx Index).

To add search functionality on the indexed data, you may implement KalturaSearchItem or one of its inherited classes. This class may be added as an advanced search object on every KalturaFilter object.

API Object

Create a folder that contains the name *api* (lower-case), such as *plugins/myPlugin/lib/api*.

Create the Kaltura API objects that are required to expose your data model through the API.

API Service

(Recommendation) Create your service class under the *my_plugin/services* folder.

Implement IKalturaServices to return the list of services implemented in your plugin.

Configure a generator.ini (under *my_plugin/config*) file to include or exclude your services and actions from generated client libraries and implement IKalturaConfigurator to return the configuration to the client generator.

Batch Worker

Implement your worker classes (recommendation: place worker classes under *my_plugin/batch*).

To use these workers, configure your batch_config.ini file.

Conversion Engine

1. Implement KOperationEngine.
2. Implement KDLOperatorBase.
3. Implement IKalturaObjectLoader to return your implementation of KOperationEngine and KDLOperatorBase.
4. Implement IKalturaEnumerator to return additional conversion engine types for the conversionEngineType enum.

Admin Console Page

1. Implement KalturaAdminConsolePlugin to load your Admin Console action data. Recommendation: Place the action data under the *my_plugin/admin* folder. For example, *my_plugin/admin/MyPluginCustomAction.php*
2. Create a template phtml file and ensure that your implementation of KalturaAdminConsolePlugin points to its path in the getTemplatePath method. For example, *my_plugin/admin/scripts/plugin/my-plugin-custom-action.phtml*. Recommendation: In the template file name, include contain lower-case letters; use a capital letter in the action file for a new word and separate the words with a dash (-).
3. Implement IKalturaAdminConsolePages to return the new Admin Console pages.

Entry Investigation Information

1. Implement `Kaltura_View_Helper_EntryInvestigatePlugin` to return the data array to be applied to the template and the template to be used to generate the HTML.
2. Implement the PHTML template.
3. Implement IKalturaAdminConsoleEntryInvestigate to return your implementation of `Kaltura_View_Helper_EntryInvestigatePlugin`.

Admin Console Publisher Action

1. Implement Admin Console Page.
2. Ensure that your implementation of KalturaAdminConsolePlugin also implements IKalturaAdminConsolePublisherAction to return the list of options to be added and the JavaScript code to be added to the page.

Extending an Enum

This section describes pluginable enums.

Implement the core interface enum that you wish to extend. For example, if you want to add a new entry status value, create a class that implements the entryStatus interface and add a new constant for the new value. For example:

```
class MyEntryStatus implements IKalturaPluginEnum, entryStatus
{
   const MY_CONSTANT_OF_MY_VALUE = 'MyConstantOfMyValue';

   // Must be implemented:
   public static function getAdditionalValues()
   {
       return array(
              'MY_CONSTANT_OF_MY_VALUE' => self::MY_CONSTANT_OF_MY_VALUE,
       );
   }

   /**
    * @return array
    * Enable you to add documentation for the auto-generated testmeDoc.
    */
   public static function getAdditionalDescriptions()
   {
       return array(

       MyPlugin::getApiValue(self::MY_CONSTANT_OF_MY_VALUE) => 'My special value',
);
   }
}
```

Implement IKalturaEnumerator and its getEnums method, where the $baseEnumName can be null or a core interface name. When $baseEnumName is null, always return all the enum class names that you implemented. Otherwise, return only the enum class names that implement the requested interface name.

Handling Events

This section describes how to handle events, including how to cancel and continue events.

Implement IKalturaEventConsumers and its getEventConsumers method.

This method returns an array of class names that implement different event consumers, such as kObjectCreatedEventConsumer, kObjectChangedEventConsumer or any consumer that extends KalturaEventConsumer.

Each consumer must implement all the base consumer abstract methods. Usually the implemented method will be a *should* method, such as shouldConsumeChangedEvent or shouldConsumeCreatedEvent. A *should* method returns a Boolean value that defines whether the main consumer method should be called and one consumer main method, such as objectChanged or objectCreated, which reacts to the raised event.

Available event consumers that may be implemented:

- kBatchJobStatusEventConsumer
  Handles the kBatchJobStatusEvent that is raised when the status of job changes.
- kObjectCreatedEventConsumer
  Handles the kObjectCreatedEvent that is raised by the Propel object after the object is saved to the database for the first time.
- kObjectAddedEventConsumer
  Handles the kObjectAddedEvent that is raised by the developer whenever the developer decides that the object is complete and is considered as added. For example, the flavor

object is considered as added only after its asset file is synchronized on one of the data centers.

- kObjectChangedEventConsumer
  Handles the kObjectChangedEvent that is raised by the Propel object after the object is saved to the database.
- kObjectUpdatedEventConsumer
  Handles the kObjectUpdatedEvent that is raised by the developer whenever the developer decides that the object update is complete and is considered as updated. For example, the flavor object is considered as updated only after its new asset file is synchronized on one of the data centers.
- kObjectCopiedEventConsumer
  Handles the kObjectCopiedEvent that is raised by the Propel object after the object is saved to the database for the first time and is a copy of an existing object.
- kObjectDataChangedEventConsumer
  Handles the kObjectDataChangedEvent that is raised by the developer whenever the developer decides that the object data change is complete and is considered as changed. For example, after changing metadata content, the data is considered as changed only after the new metadata version is saved in the database and the new metadata file is synchronized on at least one of the data centers.
- kObjectDeletedEventConsumer
  Handles the kObjectDeletedEvent that is raised by the developer whenever the developer decides that the object deletion is complete and is considered as deleted. Usually called from the Propel following the update method, after comparing the object status.

Content Distribution Connector

Refer to Creating a Custom Distribution Destination Plugin: Developer Guide.

MRSS XML Data

To add MRSS XML data to a plugin:

1. In the plugin, implement `IKalturaSchemaContributor`.
2. In `isContributingToSchema`, return `true` if the given type is `SchemaType::SYNDICATION`.

Clean Memory

Implement IKalturaMemoryCleaner to clean all instances, pools, and static objects.

Search Engine

1. Implement KalturaCriteria to search your indexing server and translate the criteria into single simple criteria on the IDs.
2. Implement IKalturaCriteriaFactory to return your implementation of KalturaCriteria for the indexed object types.

Additional Configuration

1. Create configuration files to be appended to existing known system configurations, such as generator, testme, and testmeDoc.

2. Implement IKalturaConfigurator to return implementations of `Iterator`, such as Zend_Config_Ini or Zend_Config_Xml, which contain your configuration additions.

Bulk Upload Engine

1. Implement KBulkUploadEngine to handle your file type.
2. Implement IKalturaBulkUpload.
    - Return your file type extension.
    - Implement writeBulkUploadLogFile to print the log result of your file actions. Recommendation: Use the same format as the bulk upload file.
    - Return the new file type for the BulkUploadType enum.
    - Return the implementations for the following objects:
        - kBulkUploadJobData
        - KalturaBulkUploadJobData
        - KBulkUploadEngine

CHAPTER 6

Existing Plugins

This section lists the extension points that are implemented in a selection of existing plugins, organized by category.

The first section of each category also describes the function of each extension point in the plugin.

Admin Console

This section describes plugins that are used by the Admin Console.

Admin Console

Adds API services to be used only by the Admin Console built-in partner.

Implemented Extension Point Interfaces

Name	Function
IKalturaPermissions	Blocks all partners—except the Admin Console built-in partner—from using ad-hoc API services.
IKalturaServices	Exposes API services used by the Admin Console: • entryAdmin • flavorParamsOutput • mediaInfo • thumbParamsOutput • uiConfAdmin
IKalturaConfigurator	Excludes added services from the Test Me console.

Kaltura Internal Tools

Exposes a few Kaltura developer facilities in the Admin Console.

Implemented Extension Point Interfaces

Name
IKalturaServices
IKalturaAdminConsolePages
IKalturaConfigurator

System Partner

Enables the Admin Console to manage and control a publisher's configuration.

Implemented Extension Point Interfaces

Name
IKalturaPermissions
IKalturaServices
IKalturaConfigurator

Storage

This section describes plugins that are relate to data management.

File Sync

Exposes the file sync to the API for internal synchronization and listing.

Used by the Admin Console and batch built-in partners only.

Implemented Extension Point Interfaces

Name	Function
IKalturaPermissions	Blocks all partners—except the Admin Console and Batch built-in partners—from using ad-hoc API services.
IKalturaServices	Exposes new API services: fileSync
IKalturaConfigurator	Generator – excludes fileSync service from all generated clients except admin console and batch libraries. Testme - excludes fileSync service from testme console and testmeDoc documentation page.

Multiple Data Centers

Adds system facilities to maintain multiple data centers.

Implemented Extension Point Interfaces

Name
IKalturaServices
IKalturaEventConsumers
IKalturaObjectLoader
IKalturaConfigurator

Partner Aggregation

Adds an API that exposes a publisher's data warehouse statistics.

Implemented Extension Point Interfaces

Name
IKalturaServices

Functionality

This section describes plugins that extend Kaltura functionality.

Content Distribution

Enables a publisher to distribute entry content to multiple external media providers.

Implemented Extension Point Interfaces

Name	Function
IKalturaPermissions	Blocks event consumer and API access to unauthorized partners.
IKalturaServices	Exposes new API services: • contentDistributionBatch • distributionProfile • distributionProvider • entryDistribution • genericDistributionProviderAction • genericDistributionProvider
IKalturaEventConsumers	Adds event consumers.
IKalturaEnumerator	Expands: • BatchJobType • FileSyncObjectType
IKalturaVersion	Returns the plugin version.
IKalturaSearchDataContributor	Adds search data to an entry's indexed documents.
IKalturaObjectLoader	Loads: • New implementations of ISyncableFile • New extensions of kJobData and KalturaJobData
IKalturaAdminConsolePages	Adds pages to: • Admin Console • Content distribution profile management • Generic distribution provider management
IKalturaAdminConsoleEntryInvestigate	Adds a distribution information table to the Admin Console's Entry Investigation page (Batch Process Control tab).
IKalturaPending	Defines dependency (not version-specific) on a metadata plugin.
IKalturaMemoryCleaner	Cleans new Propel peer instance pools.
IKalturaConfigurator	Adds a client generator configuration.
IKalturaContentDistributionProvider	Adds remote distribution integration interface (also referred to as *connector*).
IKalturaSphinxConfiguration	Adds Sphinx distribution index and adds fields and attributes to the entry index.

Annotation

Enables a partner to annotate a video entry.

Implemented Extension Point Interfaces

Name
IKalturaServices
IKalturaPermissions
IKalturaEventConsumers
IKalturaMemoryCleaner

Audit Trail

Enables a partner to audit object creation, modification, and deletion.

Implemented Extension Point Interfaces

Name
IKalturaPermissions
IKalturaServices
IKalturaEventConsumers
IKalturaMemoryCleaner

Document

Enables a partner to upload a document file to an entry.

Implemented Extension Point Interfaces

Name
IKalturaPlugin
IKalturaServices
IKalturaObjectLoader
IKalturaEventConsumers
IKalturaEnumerator
IKalturaConfigurator

Metadata

Enables a partner to add custom metadata fields to an object.

Implemented Extension Point Interfaces

Name
IKalturaPermissions
IKalturaServices
IKalturaEventConsumers
IKalturaObjectLoader

Name
IKalturaBulkUploadHandler
IKalturaSearchDataContributor
IKalturaMemoryCleaner
IKalturaConfigurator

Short Link

Enables a partner to use a short URL instead of a long URL.

Implemented Extension Point Interfaces

Name	Function
IKalturaServices	Exposes new API: servicesshortLink
IKalturaEventConsumers	Deletes a short link object associated with a deleted user.
IKalturaMemoryCleaner	Cleans the short link peer instances pool.

Virus Scan

Enables a partner to scan files for viruses.

Implemented Extension Point Interfaces

Name
IKalturaPermissions
IKalturaServices
IKalturaEventConsumers
IKalturaEnumerator
IKalturaObjectLoader
IKalturaMemoryCleaner
IKalturaAdminConsolePages
IKalturaConfigurator

Search Engines

This section describes plugins that are enable search engines to act as indexing servers .

Solr Search

Uses Solr as an indexing server for the Kaltura system.

Remarks

Development of this plugin is not complete.

Implemented Extension Point Interfaces

Name	Function
IKalturaEventConsumers	Updates the indexes when object data changes.

Name	Function
IKalturaCriteriaFactory	Adds implementation criteria for the Solr index.
IKalturaMemoryCleaner	Cleans the Solr log peer instances pool.

Sphinx Search

Uses Sphinx as an indexing server for the Kaltura system.

Implemented Extension Point Interfaces

Name
IKalturaEventConsumers
IKalturaCriteriaFactory
IKalturaMemoryCleaner
IKalturaSphinxConfiguration

Transcoding Engines

This section describes plugins that are enable use of transcoding engines.

Additional Transcoding Engines

Enables a publisher to use additional transcoding engines.

Implemented Extension Point Interfaces

Name	Function
IKalturaObjectLoader	Loads new extensions of KOperationEngine and KDLOperatorBase.
IKalturaEnumerator	Expands conversionEngineType.

Transcoding engine examples:

- Avi demux
- Fast start
- Inlet aramada
- Mp4 box
- Quick time
- Segmenter
- VLC

Ingestion

This section describes plugins that enable partners to ingest content.

CSV Bulk Upload

Enables a partner to use a CSV (comma-separated values) file for bulk upload of media files.

Implemented Extension Point Interfaces

Name	Function
IKalturaBulkUpload	Defines an engine to handle CSV files.
IKalturaConfigurator	Adds required API actions to the batch-generated client.

XML Bulk Upload

Enables a partner to use an XML file for bulk upload of media files.

Implemented Extension Point Interfaces

Name
IKalturaBulkUpload
IKalturaVersion
IKalturaConfigurator

Drop Folder

Enables a partner to upload files to an entry using a drop folder.

Implemented Extension Point Interfaces

Name
IKalturaBulkUpload
IKalturaPending

CHAPTER 7

How to Create a New Extension Point

For the Community

An extension point for the community is a shared plugin that may be used by other Kaltura installations.

A new community extension point must be included within a plugin. Any new plugin that wants to implement the new created extension point must be dependent on the plugin that includes the extension point.

Implemented in the Kaltura Server Core

Kaltura developers can create a server core extension point that may be used by any plugin.

A Kaltura server core extension point must be located under the *infra/plugins/interfaces* folder.

Using a New Extension Point

A new extension point is a new interface that may be implemented by plugins.

To use a new extension point, you can use the following API call to get all installed plugins that implement the extension point interface and to call the plugin methods.

```
KalturaPluginManager::getPluginInstances('myNewInterface')
```

Interface Example

```
interface myNewInterface
{
   public function doSomething();
}
```

Usage Example

```
$pluginInstances = KalturaPluginManager::getPluginInstances('myNewInterface');
foreach($pluginInstances as $pluginInstance)
   $pluginInstance->doSomething();
```

CHAPTER 8

Folder Naming Conventions and Structure

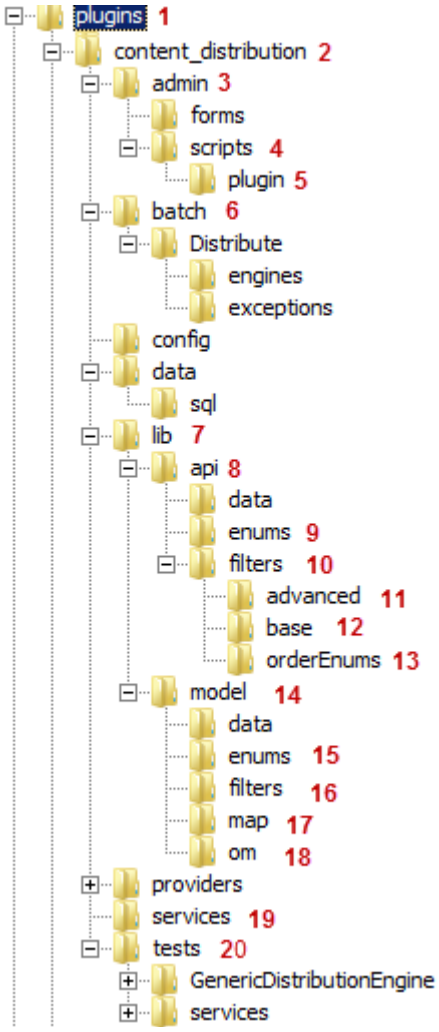
Your server plugin must comply with Kaltura conventions, including folder names and structure.

Unless otherwise noted:

- Folder name format: Lower-case with underscores. Example: content_distribution.
- File name format: Upper- and lower-case with no spaces or underscores. Example: ContentDistributionPlugin.php
- All files are PHP files.

The figure shows an example of a server plugin folder structure.

Numbers refer to the table that describes required and optional folders.



#	Folder	Description	Required	Files		Notes
				Type	Auto-Generated	
1.	plugins	Highest-level folder	✓	—	—	The folder is defined in the server code. Do not modify the existing folder name. All server plugins must have folders under *plugins*. (Optional) You can add a plugin group folder under *plugins*.
2.	content_distribution	Specific server plugin folder	✓	Plugin class file Bulk Upload XML file (optional)	—	Use a folder name that describes the plugin Plugin class file format: *PluginName*Plugin.php. Example: ContentDistributionPlugin.php Bulk Upload XML file format: *PluginName*BulkUploadXmlPlugin.php Example: ContentDistributionBulkUploadXmlPlugin.php
3.	admin	Admin Console folder	—	Files that extend the Admin Console	—	
4.	scripts	Admin Console extension script folder	—	Page templates (.phtml)	—	Page template file format: lower-case with dashes *page-template-name*.phtml Example: entry-investigate-distribution.phtml
5.	plugin	Admin Console customized page template folder	—	Customized page templates (.phtml)	—	Customized page templates file format: lower-case with dashes *customized-page-template-class-name*-action.phtml Example: distribution-profile-list-action.phtml
6.	batch	Batch worker folder	✓	—	—	For multiple workers, add a folder under *batch* for each worker. Capitalize the folder name for a worker.
7.	lib	Library folder	✓	Event consumers Facilities	—	*lib* includes the plugin interface class and all files required for plugin operation.

#	Folder	Description	Required	Files		Notes
				Type	Auto-Generated	
				Managers		
8.	api	API folder	✓	Object files	—	
9.	enums	Enumerator folder	—	Enumerator files	—	
10.	filters	Filter folder	—	Filter files	✓	Filter file format: *FilterName*Filter.php Example: KalturaEntryDistributionFilter.php
11.	advanced	Advanced filter folder	—	Advanced filter files	—	
12.	base	Base filter folder	—	Base filter files	✓	Filter file format: *FilterName*BaseFilter.php Example: KalturaEntryDistributionBaseFilter.php
13.	orderEnums	orderEnums filter folder	—	Filter files that order by enumerator	✓	Filter file format: *FilterName*OrderBy.php Example: KalturaEntryDistributionOrderBy.php
14.	model	Database model folder	✓	Propel object and peer files	—	File formats: *ObjectName*.php Example: EntryDistribution.php *ObjectName*Peer.php Example: EntryDistributionPeer.php
15.	enums	Database enumerator folder	✓	Database enumerator files	—	
16.	filters	Database filter folder	✓	Database filter files	—	Filter file formats: *FilterName*Filter.php Example: EntryDistributionFilter.php *FilterName*Filter.class.php Example: GenericDistributionProviderFilter.class.php

#	Folder	Description	Required	Files		Notes
				Type	Auto-Generated	
17.	map	Database table map folder	✓	Database table map files	✓	Table map file format: *MapName*TableMap.php Example: EntryDistributionTableMap.php
18.	om	Base object and peer folder	✓	Base object and peer files	✓	File formats: Base*ObjectName*.php Example: BaseEntryDistribution.php Base*ObjectName*Peer.php Example: BaseEntryDistributionPeer.php
19.	services	Folder for service classes that the plugin provides	✓	Service classes that the plugin provides	—	
20.	tests	Unit test folder	✓	—	—	Add a folder under *tests* for each unit test. A unit test may include files that are not PHP.

GLOSSARY

Term	Definition
Destination	A media provider or video sharing site
Distribution	Publishing media entries in a destination site
Distribution Provider	A module that publishes Kaltura entries in the destination site
Entry	Kaltura's database and API representation of a content entity and its metadata. Entry types include media, video, audio, image, data, mix, document, and playlist. Entry metadata includes type, storage location, title, tag, and rating.
Extension Point	
Kaltura Administration Console	An application for administering the Kaltura system, including administration of multiple Kaltura accounts. The Admin Console typically is accessed by Kaltura system administrators and the IT team.
MRSS	Media RSS file format
Partner	An individual or organization with a Kaltura system account
Partner ID	A numeric identifier that uniquely identifies a partner in the Kaltura database
Plugin	An application that provides new features to other applications
Publisher	See Partner.