External Content Plugin for ILIAS

Version 1.3 for ILIAS 4.4, Version 1.4 for ILIAS 5

fred.neumann@fim.uni-erlangen.de, jesus.copado@fim.uni-erlangen.de

1. Basic description

This plugin gives to ILIAS the capacity integrate external contents or tools in the platform, it can be used for multiple purposes, like embedding a video from a video server or access an external site with the possibility to pass authentication.

Administrators can define different types of contents and configure their connection with an XML description. A default configuration is provided for the **LTI 1.1** specification, see:

http://developers.imsglobal.org

Please note that the terms *Content* and *Tool* are treated equally in this description. However, *Tool* is now preferred because the plugin allows to connect whole external platforms.

The integration of external tools with LTI 1.1 works like follows:
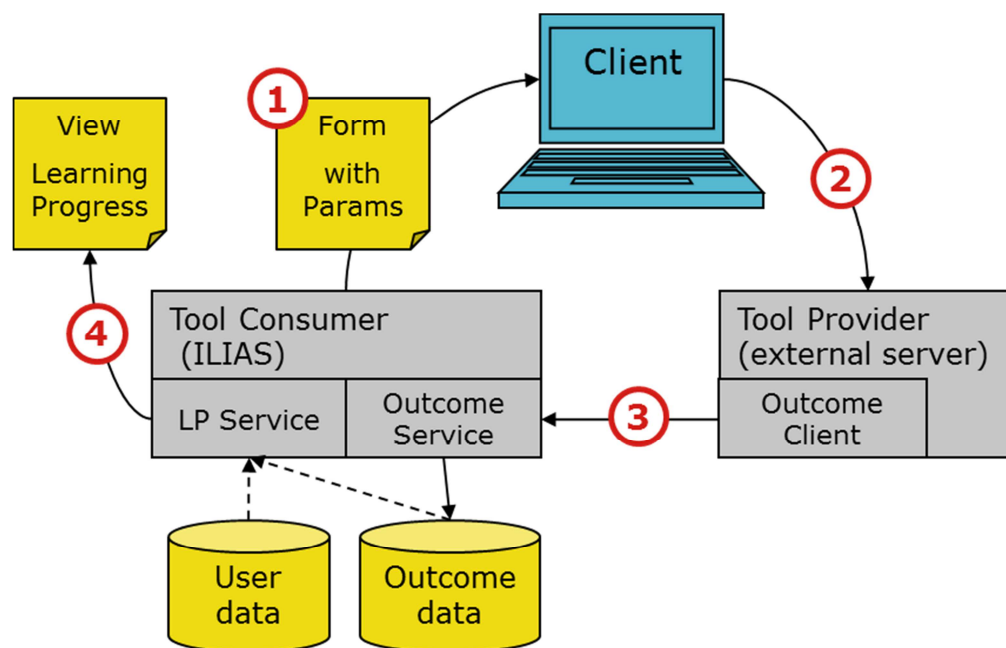
1. ILIAS as a Tool Consumer (TC) launches the external tool by sending a form with hidden fields to the browser. It includes parameters for authentication, resource identification and selected user data. LTI expects these parameters signed with OAuth. The form action is the URL of the tool provider.
2. The browser calls the external Tool Provider (TP) with the parameters given by ILIAS. The TP checks the OAuth signature against a shared secret and provides the requested resource in its own session.
3. The TP can send an outcome value to ILIAS. This is done asynchronously from server to server by calling a URL provided with the launch.

4. ILIAS checks the outcome value against a threshold and sets the learning progress for the related user. This can be viewed by the lecturer or the learner and can be used for an aggregated learning progress of the upper course.

The plugin is not restricted to LTI. By defining your own types you can implement your own protocol of connecting external contents or tools with or without secure authentication.

1.1 Plugin Installation

- Copy the contents of the plugin folder to the following path in your ILIAS installation (create the necessary directories if not existing):
  Customizing/global/plugins/Services/Repository/RepositoryObject/ExternalContent
- Log in as Administrator, go to Administration > Plugins
- For the ExternalContent plugin click choose the actions *Update* . The needed database tables will be created and the plugin activated afterwards.

This plugin creates a new repository object type "External Content". **Please do not forget to set the permissions of roles and role templates for this object type. This is not done by default!**

1.2 Quick LTI configuration

- Click *Configure* for the External Content plugin.
- Click *Define a new type* in the table of content types.
- Enter a type name and choose a language version of the provided LTI type models.
- *Save* the new type.
- Check the *Settings* of the type, especially the privacy settings.
- Optionally upload *Icons* that are used for objects of this type.

1.3 Quick LTI usage

- Go to the *Repository.*
- Select *Add New Item* and choose *External Content.*
- Enter an object title and select the LTI content type.
- *Save* the new object.
- Edit the *Settings* of the object. At least you have to enter the *URL*, *Key* and *Secret* of the external tool.
- Optionally edit *Instructions* to be shown on the info page or upload specific *Icons* for this object.
- Move to the *Learning Progress* tab if you want to record a learning progress. Set the modus to "Active" and enter a passing threshold between 0 and 1 for the outcomes sent from the Tool provider.
- Move to the *Content* tab to test how this tool is presented to the users.
- Edit the *Settings* of the object and set it *Online.*

2. Configuration

The plugin allows ILIAS administrators to support different types of external contents or tools. The ILIAS authors (e.g. lecturers) can create repository objects based on these types.

Each type has its own definition of how contents or tools of this type are called, what parameters are fixed for this type (e.g. the address of a video server), what parameters have to be defined by the ILIAS authors (e.g. the id of a video) and what user data are provided to the external server. All data can be encrypted in different ways.

2.1 Create a new type

If you are logged as administrator, go to *Administration* menu, and choose *Plugins*.

Choose the action "Configure" for the External Content plugin. You see a list of available content types:
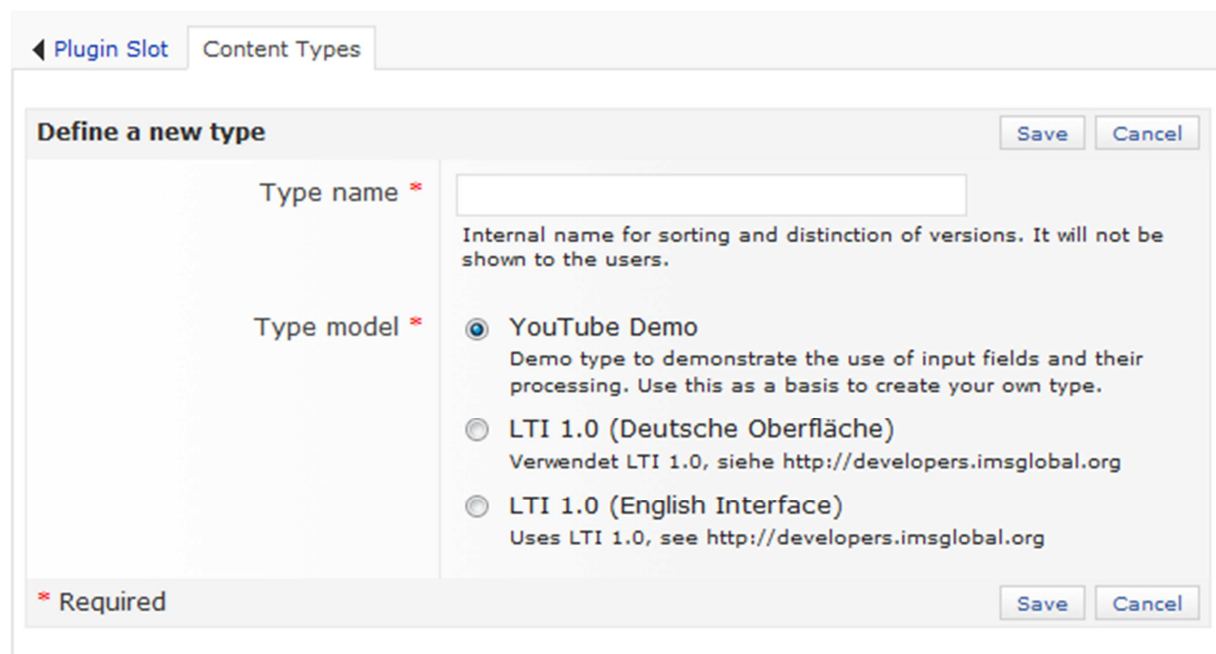
Click *Define a new type* to create a new one. You get a creation form:

Enter an internal type name. This name is only displayed in the administration and can be used to distinguish separate versions of a type. It is not shown to the users.

Choose a type model. The models are predefined configurations of types. You may completely change the configuration of your type by editing its XML definitions, but the models are a proven starting point.

Save the new type. This brings you to its *Editing* (see below). You may want to adjust some settings but normally you can start using the type without further configuration.

2.2 Edit a type

2.2.1   Type settings

The editing of a type is directly shown when a new type is created. You can also open it with an action from the list of types. The first editing screen shows its settings:

Some settings are common for all types:

- Type name: internal name for sorting and distinction of versions. It will not be shown to the users.
- Title: this title is shown when a type is selected for a new object.
- Description: this description is shown when a type is selected for a new object.
- Availability: availability of this type in the repository. You may set deprecated types to "Allow existing objects" instead of deleting them.
- Remarks: administrative remarks for this type. They will not be shown to the users.

Some types have specific settings. These are configured below the common ones. The LTI type, for example, allows specifying which personal data sent to the external tool.

Changing the settings of a type affects all existing objects of this type directly. Sometimes it is better to change the availability to "Allow existing objects" and create a new type with other settings. Use the internal type name and remarks to distinguish the versions of the type.

2.2.2 Type Icon

The second sub tab allows you to upload three Icons. These Icons are used for repository objects of this type instead of the common icon for all external content objects.

2.2.3 XML definition

The third sub tab shows the XML definition of a type. This screen is intended for experts only.

All specific input fields for the settings of the type and its objects are defined here. Templates and function fields define how these inputs are processed and how the external tool is launched.

2.3 Delete a type

The list of content types shows the number usages of each type. A type can be deleted if it has no usage. You can delete it by choosing the "delete" action and passing a confirmation screen.

2.4 Learning Progress

Since version 1.3 (ILIAS 4.4) and version 1.4 (ILIAS 5) the plugin is able to record and show the learning progress of users.

As a precondition it is necessary that the learning progress is globally by the ILIAS administrator at "Administration > Statistics and Learning Progress". An important option at this screen is "Anonymized". This determines whether teachers see the learning progress of single users or just a summary.
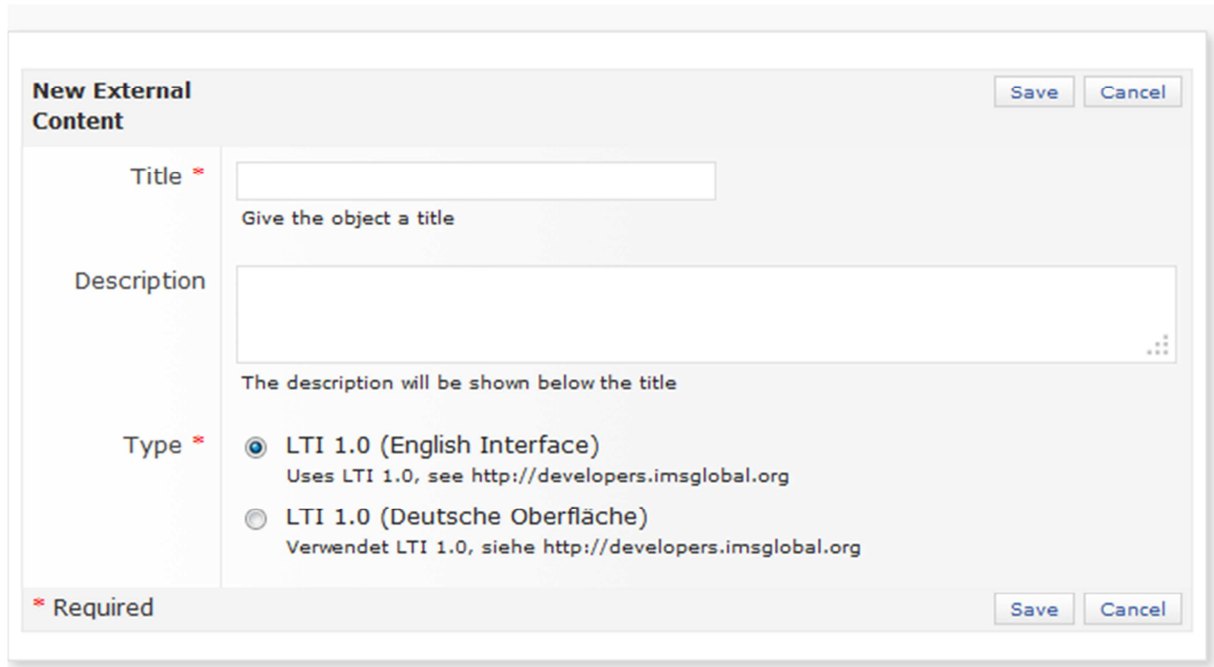
The ability to activate the learning progress for single objects is controlled by the permission "edit learning progress".

The ability to see the learning progress of users is controlled by the permission "read learning progress" (in ILIAS 5 or higher).

3. Usage

3.1 Create an Object

Go to the desired place in the repository and choose *Add New Item*. If the plugin is activated you can select *External Content*. On the create form you have to select a content type:
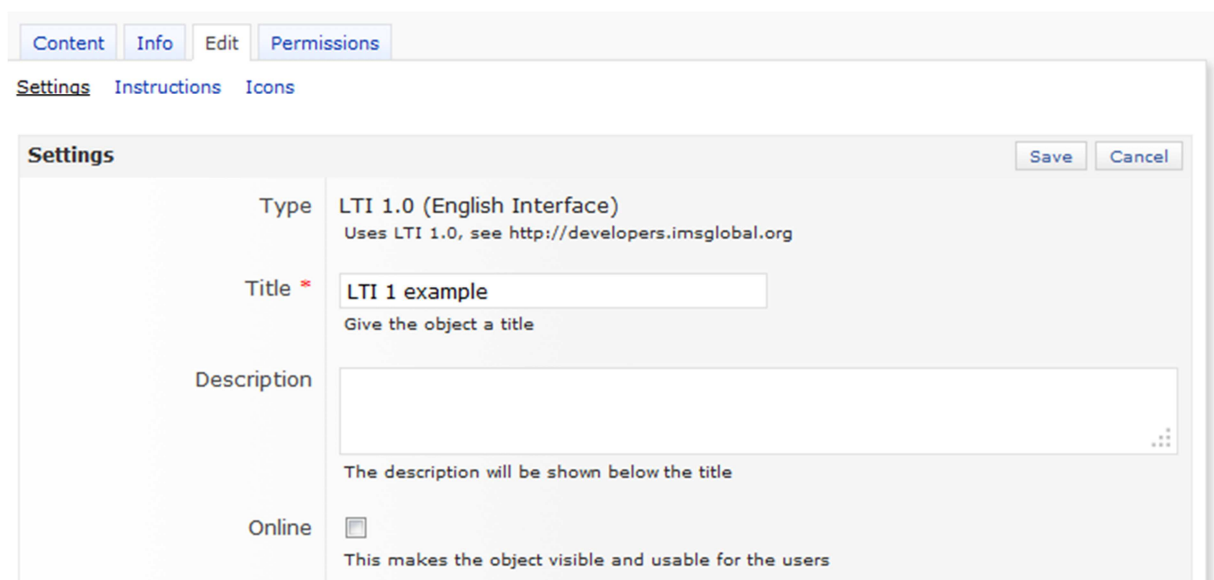
3.2 Edit an Object

The *Edit* tab of an external content object is available to all users with *write* permission. It has three sub tabs: *Settings*, *Instructions* and *Icons*.

3.2.1   Settings

The basic settings of an external content objects are common for all types:

- The *type* just shown and cannot be changed after an object is created.
- *Title* and *Description* work like in all ILIAS objects.

- *Online* makes the object available for all users that have *visible* and *read* permission.

3.2.2 Instruction

Instructions are a free text, probably rich text input. They are displayed on the info tab of the object. Some content types may also include the instructions on the content tab.

3.2.3 Icons

You may upload objects specific icons that are used instead of the type specific or general ones:

3.3 Activate Learning Progress

If the learning progress is enabled in ILIAS and the user has the permission "edit learning progress" on an ExternalContent object, then a *Learning Progress* tab is shown. The *Settings* sub tab allows activating learning progress for the specific object. It is not activated by default!

If activated, a user's status will change from "Not Attempted" to "In Progress" when viewing the content the first time. Other values are only set if the external tool calls the learning outcome service of the plugin according to the LTI 1.1 specification. This call sends a value between 0 and 1. This value is compared to a threshold which can be entered in the learning progress settings. The status is set to "Completed" if it is equal or higher, otherwise to "Failed".

4. Type Definition

The type models that can be chosen when a new type is created are just predefined types. You may change the behavior of a type or create a complete different type by editing the type definition in XML.

The following code shows the basic structure of an XML definition for external content types:

```xml
<interface type="link|page|embed">
        <title>Title of the type (is chosen at object creation)</title>
        <description>Longer description</description>
        <template><![CDATA[ Code with {FIELD_NAME} placeholders ]]></template>
        <metasource><![CDATA[ URL of a meta data source ]]></metasource>
        <fields>
                <field name = "FIELD_NAME"
                        type = "text"
                        encoding = "plain" >
                        <title>title to be shown for an input field</title>
                        <description>Description shown</description>
                </field>
        </fields>
</interface>
```

4.1 Main elements

4.1.1 Interface

A type definition must exactly have one `<interface>` as its root element.

The only attribute `type` defines how an external content is embedded:

- "`link`": By a hyperlink with href generated from the type template.
- "`page`": As an own HTML page, generated from the type template.
- "`embed`": HTML code embedded to an ILIAS page and generated from the type template.

4.1.2 Title

The `<title>` element corresponds to the title on the settings screen and is automatically updated when the title is changed there. This title is shown for the type selection when new objects are created.

4.1.3 Description

The `<description>` element corresponds to the description on the settings screen and is automatically updated when the description is changed there. This description is shown for the type selection when new objects are created.

4.1.4 Template

Depending on the interface type, the element `<template>` contains a CDATA section defining a URL or HTML code. Field names are included as placeholders in curly brackets {}. Example:

```
<template><![CDATA[
<iframe src="{INPUT_SRC}" with="{INPUT_WIDTH}" height="{INPUT_HEIGHT} />
]]></template>
```

The field names should follow the conventions for naming variables (no special characters except "_"). These names refer either pre-defined fields (starting with "ILIAS") or name attributes of `<field>` elements (see below).

Placeholders of existing fields are replaced by the actual field values. For input fields, the values are taken from the type or object settings. For ILIAS fields the values may depend on the current user and context of an object. Template fields have their own template as a sub element which is processed recursively.

4.1.5 Metasource

`<metasource>` is optional, works like `<template>` and produces a URL to request meta data from the external content. These have to be provides as an xml structure (see "Interface for meta data") and are shown on the info page of the external object.

4.2 Fields

The element `<fields>` of an interface contains a couple of `<field>` elements that can be used in a template.

Field elements have the following common attributes:

- `name` is the field name and has to be unique. This name, enclosed in {} is used as a placeholder in templates.
- `type` is the field type. Each type has its own processing rules and additional attributes.
- `encoding` is the name or a list of encoding functions that should be applied to the field value before a placeholder for this field is replaced in an upper template. See section "Encodings" below.

The type attribute of a field specifies the purpose of a field. Some fields are shown on the settings screen of content types and objects. Others are just used for intermediate values in the template processing.

4.2.1 Input fields

Input fields are shown on the settings screens. The following attributes are common to all input fields:

- `level` defines where an input field is shown. `"type"` will show the field on the settings screen of a content type in the administration. `"object"` (default) will show the field on the settings screen of an object.
- `parentfield` defines the name of a field that should be the a parent of a nested input field.
- `parentvalue` must be used in conjunction with `parentfield` and defines the value of a radio field option that should be the parent of a nested input field.
- `default` defines a value that should be used if the field has no input.
- `required` is set to `"1"` if an input is required.

Input fields can have the following sub elements:

- `<title>` defines the field title shown in the input form.
- `<description>` defines the info text shown below the input field.

Input fields can have the following values in their `type` attribute:

- `"header"` will show a section header . Only the title is used. All other properties are ignored.
- `"description"` will show non-editable text. Only the description is used. All other properties are ignored.
- `"text"` will show a single line text input field. The attribute `size` defines the length of the field.
- `"textarea"` will show a multiline text input field. The attributes `rows` and `cols` define the size of the field. The attribute `richtext` is set to `"1"` if a rich text editor should be used.
- `"password"` will show a password input field.
- `"checkbox"` will show a checkbox. If it is checked, the value of the field will be "1".
- `"radio"` defines a group of radio options.

Input fields with `type="radio"` define their options with `<option>` sub elements. The attribute `value` of an option defines the value that is taken when an option is chosen. The sub element

`<title>` of an option defines the option title and the sub element `<description>` defines the description shown with the title.

An example from the LTI definition shows the use of radio options and nested inputs:

```
<field name="LAUNCH_TARGET" type="radio" required="1" default="top">
  <title>Tool Presentation</title>
  <option value="iframe">
    <title>Embedded</title>
    <description>Show the tool embedded on the content page</description>
  </option>
  <option value="top">
    <title>Same Window</title>
    <description>Open the tool in the same window</description>
  </option>
  <option value="blank">
    <title>New Window</title>
    <description>Open the tool in a new browser window</description>
  </option>
</field>
<field name="LAUNCH_WIDTH" parentfield="LAUNCH_TARGET" parentvalue="iframe"
type="text">
  <title>Width</title>
</field>
<field name="LAUNCH_HEIGHT" parentfield="LAUNCH_TARGET"
parentvalue="iframe" type="text">
  <title>Height</title>
</field>
```

4.2.2 Template fields

Fields with `type="template"` are used to concatenate the string values of other fields and apply an encoding to the concatenation. This is for example useful to calculate an md5 hash and append it as a signature to a url.

Template fields have just the basic attributes `name`, `type` and `encoding`. Additionally they have a sub element `<template>` that is processed like the main template of the interface: first all placeholders are replaced by the values of their corresponding fields. Then the encoding is applied.

4.2.3 Calculated fields

Fields with `type="calculated"` provide a more sophisticated processing of values than template fields. They have an attribute `function` that defines which calculation should be applied. The parameters for this function are defined as `<param>` sub elements. Each parameter has a `name` attribute which should be unique for a calculated field.

The text value of a parameter can either be a fixed string or the placeholder of a single field. It is *not* a template which means that a concatenation of text and fields will not work within params.

A typical example is used in the LTI definition to set what kind of user id is sent based on the value of a radio input field:

```
<field name="LTI_USER_ID" type="calculated" function="selectByName">

  <param name="value">{SEND_USER_ID}</param>

  <param name="none"></param>

  <param name="coded">{ILIAS_USER_CODE}</param>

  <param name="numeric">{ILIAS_USER_ID}</param>

  <param name="login">{ILIAS_USER_LOGIN}</param>

</field>
```

The radio input field is SEND_USER_ID which may have the values "none", "coded", "numeric" or "login". The function "selectByName" selects the value to be sent by comparing the value of the first parameter with the names of the other parameters. In case of "none", an empty string is returned. In the other cases the values of pre-defines ILIAS fields are returned.

See the section Functions below in this manual for a list of available functions.

4.2.4   ILIAS fields

ILIAS fields are pre-defined values that can be used with {} placeholders but do not have a <field> definition in the type XML. The following ILIAS fields are available:

- ILIAS_REF_ID   Ref-Id of the object in the ILIAS repository
- ILIAS_TITLE   title of the ILIAS object
- ILIAS_DESCRIPTION   description of the ILIAS object
- ILIAS_INSTRUCTIONS   instructions for the ILIAS object

- ILIAS_CONTEXT_ID   Ref-Id of the surrounding course or group
- ILIAS_CONTEXT_TYPE   Object Type of the surrounding course or group
- ILIAS_CONTEXT_TITLE   Title of the surrounding course or group

- ILIAS_REMOTE_ADDR   IP address of the browser
- ILIAS_TIME   current server time (yyyy-mm-dd hh:mm:ss)
- ILIAS_TIMESTAMP   current server time (integer)
- ILIAS_SESSION_ID   ID of the php session
- ILIAS_TOKEN   Token used for callback authentication
- ILIAS_RESULT_ID   result id to be used calling the outcome service (for LTI 1.1)

- ILIAS_CALLBACK_URL   URL to be called for callback authentication (not active)
- ILIAS_EVENT_LOG_URL   URL to be called for logging events (nut active)
- ILIAS_RETURN_URL   URL to return to the content object in ILIAS
- ILIAS_RESULT_URL   URL of the outcome service (for LTI 1.1)

- ILIAS_USER_ID   numerical ID of the user
- ILIAS_USER_CODE   unique anonymous code for the user
- ILIAS_USER_LOGIN   login name of the user
- ILIAS_USER_FIRSTNAME   first name of the user
- ILIAS_USER_LASTNAME   family name of the user

- ILIAS_USER_FULLNAME   complete name of the user
- ILIAS_USER_EMAIL   e-mail address of the user
- ILIAS_USER_IMAGE   path to a user image (if public)
- ILIAS_USER_LANG   2-character language key of the user
- ILIAS_USER_WRITE_ACCESS   1 of the user has write access to the object, 0 otherwise

- ILIAS_VERSION   version number of the ILIAS installation
- ILIAS_CONTACT_EMAIL   feedback recipient of the ILIAS client
- ILIAS_CLIENT_ID   ID of the ILIAS client
- ILIAS_HTTP_PATH   HTTP path of the ILIAS directory
- ILIAS_LMS_URL   base URL of the ILIAS installation (with client)
- ILIAS_LMS_GUID   Dotted concatenation of ILIAS client, path and domain
- ILIAS_LMS_NAME   Short installation name or client name, if empty
- ILIAS_LMS_DESCRIPTION   ILIAS header title or client description, if empty

4.3 Encodings

Encodings are named functions that take a string value as input and return a string value as output. They can be set as an attribute for all input, template and function fields. When the field is used in a template or function, then the encoding is applied to the field value before it is used. An encoding for a function field will only work if the return type of the function is a string value.

The plugin defines a set of basic, commonly used encodings. They can be listed with comma separation for a combination. In this case they are applied from the left to the right, e.g.:

```
encoding="strip_tags,singlequotes,no_break,trim"
```

The following encodings are defined by the plugin:

- plain   does not modify the input
- url   applies URL encoding
- base64   applies BASE64 encoding
- dechex   hexadecimal representation of numeric strings
- md5   calculates an MD5 message digest
- sha1   calculates an sha1 hash
- strip_tags   removes html tags
- no_break   converts line breaks to spaces
- nl2br   converts line breaks to <br /> elements
- singlequotes   changes double quotes(") to single quotes (')
- doublequotes   changes single quotes(') to double quotes (")
- trim   removes white spaces from the beginning and the end
- htmlentities   converts utf-8 special characters to their HTML entities
- html_entity_decode   converts HTML entities to their utf-8 characters
- htmlspecialchars   concerts < > & " to their HTML entities
- htmlspecialchars_decode   converts the HTML entities of < > & " back

Any other encoding refers to a user defined encoding function. If the encoding attribute is "ilMyEncoding" then the plugin will search for the following class file:

Customizing/global/encodings/class.ilMyEncoding.php

This class name must be ilMyEncoding and the class has to provide a static method **encode()** that takes a string as input and provides the encoded string as return value.

4.4 Functions

Functions are used by calculated fields with their function attribute.   They get a list of named input parameters and return a calculated result. The data types of parameters and results are not fixed and depend on the semantics of a function.

The following functions are defined by the plugin:

4.4.1   createArray

This returns an associative array with the parameter names as keys and the parameter values as values. This function is needed for example in the LTI type to create the parameter array for signOAuth.

4.4.2   getBasename

This returns the basename of a URL.

4.4.3   signOAuth

This signs parameters according to the OAuth specification. The input parameters are:

- url   url to be signed
- key   key used for the signature
- secret   shared secret used for the signature
- callback   callback URL, is "about:blank" in the LTI case
- sign_method   "HMAC_SHA1", "PLAINTEXT" or "RSA_SHA1"
- http_method   "POST" or "GET"
- token   is empty in the LTI case
- data   associative array of parameters to be signed

4.4.4   createHtmlInputFields

This creates a list of HTML input fields. The parameters are:

- type   "visible" or "hidden"
- data   associative array of data

The keys of the data array are taken as field names and the values as field values. Note that no encoding is applied to the values before they are written into the html value attributes.

4.4.5   showValues

This generates a HTML output for debugging and shows the data as "key = value" lines. The parameters are:

- data   associative array of data

4.4.6   selectByName

Compares the value of one parameter with the names of the other parameters and returns the value of the found parameter. If no parameter matches, the value of a parameter with no name is returned. The compare parameter must have the name "value".

See the section of calculated fields for an example.

5. Interface for Meta Data

The address called by <metasource> has to deliver the meta data of an external object as an XML structure with the following format:

```
<meta>
        <group name="...">
                <title>...</title>
                <fields>
                        <field name="...">
                                <title>...</title>
                                <content>[CDATA[...]]</content>
                        </field>
                        ...
                </fields>
                ...
        </group>
        ...
</meta>
```

The fields will be matched by the field names with the corresponding LOM attributes.