



SQLIImport eZ Publish extension

Version 1.0 - 2010 Jerome Vieilledent for SQLi

Date: 2010/08/26

Table of contents

SQLIImport eZ Publish extension	1
PRESENTATION	1
LICENCE	1
USAGE	2
Immediate & Scheduled	2
Immediate	2
Scheduled	2
Runtime Options	3
CLI SCRIPT	3
PERFORMANCE SETTINGS	3
ViewCaching	4
ObjectIndexing	4
Content objects update	4
Datatypes for which a specific diff handler is provided :	4
HANDLER DEVELOPMENT	4
Simplified content API	5
Source Handlers	6
DIFF HANDLER DEVELOPMENT	7

PRESENTATION

SQLIImport is an extension allowing to import external content in eZ Publish. It provides a framework for content manipulation and a simple interface for developing import handlers, so as a GUI to administrate your imports in the admin interface. Import handlers are classes you need to develop in order to process external data (XML, CSV, ...) and import it into eZ Publish (see [Handler Development](#) section).

LICENCE

This eZ Publish extension is provided as *is*, in GPL v2 (see LICENCE).

USAGE

You can manage your imports via the admin interface. After installing *SQLImport*, a new **Import Management** tab appears (you'll need to have access to *sqlimport* module or to simply be administrator to be able to see it from eZ Publish 4.3). Click on it to start managing your imports :

You are here: Import management list

Import management

- [Import list](#)
- [Scheduled import\(s\)](#)
- [Request a new immediate import](#)
- [Purge import history](#)

Import list

10 25 50

Import type	Params	User	Requested on	Type	Status	Progress	Progression notes
RSS Handler		admin	24/08/2010 1:44 pm	Immediate	Pending (Cancel)	0%	
RSS Handler		admin	24/08/2010 1:37 pm	Immediate	Running (Interrupt)	43%	
RSS Handler		admin	24/08/2010 1:23 pm	Immediate	Completed	100%	

Immediate & Scheduled

There are two types of imports :

- Immediate
- Scheduled

Each import is stored in the database as pending and awaits for the cronjob to process it. Running imports are safely interruptable from the admin interface. Pending imports can be cancelled while the cronjob has not processed it.

Immediate

Immediate imports are *one-shot*, meaning that they will not repeat in time, contrary to scheduled imports. To add an immediate import, click on *Request a new immediate import* in the left menu. Choose your import handler and eventually add options (see [Runtime Options](#) section below).

Scheduled

Scheduled imports will be launched at chosen *start date*. They can be one-shot (Frequency = none) or recurring. You can add a label to the scheduled import and deactivate it :

Schedule an import

Import label

Import handler

Options (facultative)

One option per line : `optionName=optionValue`

Choose a start date (YYYY-mm-dd)

2010-08-24

 Hour

14

 Minutes

0

Frequency

☒ None

☐ Hourly

☐ Daily

☐ Weekly

☐ Monthly

Activate import

☒

Add scheduled Import

To add a scheduled import, go to *Scheduled import(s)* by clicking the link in the left menu, and click *Add a scheduled import*. Choose your import handler and eventually add options (see [Runtime Options](#) section below).

Runtime Options

If your import handler supports **Runtime options** (see [Handler Development](#) section), you can add them from the admin interface. You can only add one option per line with format **optionName=optionValue**. Options will be passed to the import handler at runtime (in the handler constructor).

CLI SCRIPT

SQLIimport provides both a cronjob and a *regular* CLI script. The cronjob is used to process imports added from the admin interface (immediate and scheduled). The regular CLI script can be used to trigger a quick one shot import, without having to go into the admin interface.

Usage : `php extension/sqliimport/bin/php/sqlidoimport.php [OPTION]...`

Options :

`--source-handlers=VALUE`

Comma separated source handlers identifiers. If not provided, all source handlers will be processed.

`--list-source-handlers`

Lists all available handlers

`--options=VALUE`

Options for import handlers. Should be something like `--options="handler1::foo=bar,foo2=baz|handler2::someoption=biz"`

PERFORMANCE SETTINGS

Several *performance settings* are set in **sqliimport.ini** configuration file. For more details, read the inline comments in the INI file.

ViewCaching

View caching is disabled by default for performance reasons. It's disabled only for the import script. ViewCache is cleared once import has been done, via `sqlimport_cleanup` cronjob (launched after `sqlimport_run`)

ObjectIndexing

Same as for ViewCaching above. Import will be much faster with ObjectIndexing set to disabled. Will just activate `site.ini SearchSettings.DelayedIndexing` for current import script. Content objects will be indexed once import has been done, via `sqlimport_cleanup` cronjob.

Content objects update

If bundled content manipulation framework is used, the system will do comparisons in order to check if it is really necessary to create a new content object version. By default it compares the string representation of each attribute content, but the diff system is extendable. It is thus possible to define new diff handlers for each datatype.

For more information, please read the [DIFF HANDLER DEVELOPMENT](#) section.

Datatypes for which a specific diff handler is provided :

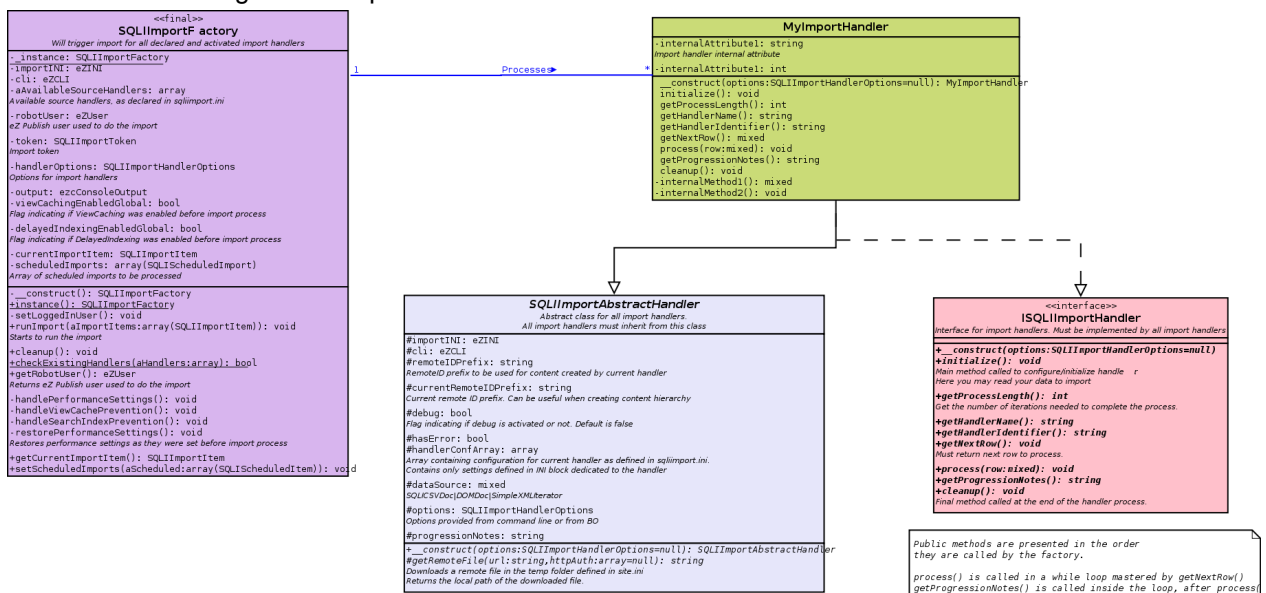
- ezimage
- ezbinaryfile

HANDLER DEVELOPMENT

To import external content into eZ Publish with SQLImport, you need to develop a handler that *understands* the external source (whatever it is) and maps it with your eZ Publish content structure.

Developing an import handler is fairly easy. You just need to create a PHP class that extends **SQLImportAbstractHandler** and implements **ISQLImportHandler**. You will also need to declare it in an override of `sqlimport.ini` by creating a dedicated section (please read inline INI comments for further details).

Here's the UML diagram for import handlers :



Handler method call order :

- `__construct()` - You'll need to call the parent constructor in it
- `initialize()` - Main method called to init your handler. Here you may read your external data source
- `getProcessLength()` - Get the number of iterations needed to complete the process
- `getHandlerName()`
- `getHandlerIdentifier()`
- `getNextRow()` - Must return next row to process or *false* when import process is finished for the handler
- `process()` - Called in a loop mastered by `getNextRow()`. Result of `getNextRow` is passed to this method
- `getProgressionNotes()` - Called inside the loop, after `process()`
- `cleanup()` Final method called at the end of the handler process

A full working example is provided (**SQLIRSSImportHandler**). Check it to understand the mechanism.

Note that all configuration set in your INI handler block in `sqlimport.ini` will be available in your handler in **`$this->handlerConfArray`**.

Simplified content API

A framework is provided to manage eZ Publish content without hassle (please read examples in the API doc) :

```
$cli->notice( 'Creation of a new "comment" object' );
$options = new SQLIContentOptions( array(
    'class_identifier' => 'comment',
    'remote_id'       => 'my_ubber_cool_remote_id',
    'language'        => 'fre-FR'
) );
$comment = SQLIContent::create( $options );
$cli->notice( 'Current version : '.$comment->current_version );
$comment->fields->subject = 'Mon super sujet';
$comment->fields->author = 'Moi !';
$comment->fields->message = 'Le commentaire de la mort';

$comment->addTranslation( 'eng-MS' );
$comment->fields['eng-US']->subject = 'My great subject';
$comment->fields['eng-US']->author = 'Batman';
$comment->fields['eng-US']->message = 'Death comment';

$comment->addLocation( SQLILocation::fromNodeID( 2 ) );
$comment->addLocation( SQLILocation::fromNodeID( 43 ) );

$publisher = SQLIContentPublisher::getInstance();
$publisher->publish( $comment );

$cli->notice( 'Current version : '.$comment->current_version );

// Loop against locations
foreach( $comment->locations as $nodeID => $location )
{
    // Regular node attributes are available as virtual properties
    $cli->notice( $nodeID.' => '.$location->path_string.' ( '.$comment->locations[$nodeID]->path_identification_string.' ) );
}

// Now free memory.
// unset() on SQLIContent triggers eZContentObject::clearCache()
// and eZContentObject::resetDataMap()
unset( $comment );
```

SQLIContent framework relies on string representation of content attributes. It makes use of `fromString()` / `toString()` methods, implemented in every kernel datatypes since eZ Publish 3.9. So if you use custom datatypes, make sure they implement these methods for better result. If they are not present, the framework will use *data_text* instead.

For more information about string representation of kernel datatypes, please read **fromString.txt** appendix.

A shorthand method is available to handle HTML content import. It is available in *Import Handlers* and outside import handlers :

For more examples, please check scripts located in the *stubs/* directory.

UML Class Diagram illustrating the structure of the **SQLContent** project, showing classes, interfaces, and their relationships.

```

classDiagram
    class SQLContent {
        #contentObject: eZContentObject
        #fields: SQLFieldsetHolder
        #initialLocations: array(SQLLocation)
        #currentLocations: SQLLocationSet
        #locations: SQLLocationSet
        #activeLanguage: string
        #defaultLanguage: string
        #options: SQLContentOptions
        +_construct()
        +_destruct()
        +getActiveLanguage()
        +setActiveLanguage()
        +fromContentObject()
        +fromNode()
        +fromNodeAndPath()
        +fromNodeAndPathAndLanguage()
        +refresh()
        +getDraft()
        +_get()
        +_set()
        +_isset()
        +removeLocation()
        +deleteLocation()
        +getInitialLocations()
        +refreshLocations()
        +hasPending()
    }
    class SQLFieldsetHolder {
        #fields: array(SQLContentFieldset)
        #referenceToKnownChildrenFieldSet
        #contentObject: eZContentObject
        #activeLanguage: string
        #iteratorPointer: array
        +_get()
        +_set()
        +_isset()
        +_checkActiveLanguage()
        +_getActiveLanguage()
        +_fromContentObject()
        +_addTranslation()
        +_getCurrentDraft()
        +_newInstance()
    }
    class SQLContentFieldset {
        #language: string
        #local: string
        #fields: array(SQLContentField)
        #referenceToFieldsByAttributeIdentifier
        #iteratorPointer: array
        +_get()
        +_set()
        +_isset()
        +_getLanguage()
        +_setLanguage()
        +_fromFieldMap()
        +_refresh()
        +_setContentObjectAttribute()
        +_resetInputData()
        +_newInstance()
    }
    class SQLContentField {
        #attribute: eZContentObjectAttribute
        #identifier: string
        #data: string
        +_fromContentObjectAttribute()
        +_isset()
        +_get()
        +_set()
        +_toString()
        +_call()
        +_getRawAttribute()
        +_setAttribute()
        +_setData()
        +_getData()
        +_reset()
        +_isModified()
        +_check()
    }
    class SQLLocation {
        #node: eZContentObjectNode
        #mode: int
        +_construct()
        +_fromNode()
        +_get()
        +_set()
        +_isset()
        +_call()
    }
    class SQLLocationSet {
        #locations: array(SQLLocation)
        #mode: int
        #iteratorPointer: array
        +_construct()
        +_newInstance()
        +_addLocation()
    }
    class SQLInterfaceArrayAccess {
    }
    class SQLInterfaceIterator {
    }

    SQLContent --> SQLFieldsetHolder : Holds
    SQLContent --> SQLLocationSet : Holds
    SQLContent --> SQLContentFieldset : Holds
    SQLContentFieldset --> SQLContentField : Holds
    SQLFieldsetHolder --> SQLContentFieldset : Holds
    SQLFieldsetHolder --|> SQLInterfaceArrayAccess
    SQLFieldsetHolder --|> SQLInterfaceIterator
    SQLLocationSet --|> SQLInterfaceArrayAccess
    SQLLocationSet --|> SQLInterfaceIterator
    SQLLocationSet --> SQLLocation : Holds
    
```

SQLContent (Proxy class for eZContentObject, useful for easy manipulation on content objects (insertion, modification...))

- #contentObject: eZContentObject
- #fields: SQLFieldsetHolder
- #initialLocations: array(SQLLocation)
- #currentLocations: SQLLocationSet
- #locations: SQLLocationSet
- #activeLanguage: string
- #defaultLanguage: string
- #options: SQLContentOptions
- +_construct(): SQLContent
- +_destruct(): void
- +getActiveLanguage(): string
- +setActiveLanguage(language: string): void
- +fromContentObject(object: eZContentObject): SQLContent
- +fromNode(mode: eZContentObjectNode): SQLContent
- +fromNodeAndPath(mode: eZContentObjectNode, path: string): SQLContent
- +fromNodeAndPathAndLanguage(mode: eZContentObjectNode, path: string, language: string): SQLContent
- +refresh(): void
- +getDraft(): eZContentObjectVersion
- +_get(name: string): mixed
- +_set(name: string, value: mixed): void
- +_isset(name: string): bool
- +removeLocation(mode: int, moveToTrash: bool): void
- +deleteLocation(mode: int, location: SQLLocation): void
- +getInitialLocations(): array
- +refreshLocations(): void
- +hasPending(): bool

SQLFieldsetHolder

- #fields: array(SQLContentFieldset)
- #referenceToKnownChildrenFieldSet
- #contentObject: eZContentObject
- #activeLanguage: string
- #iteratorPointer: array
- +_get(name: string): SQLContentFieldset
- +_set(name: string, value: mixed): void
- +_isset(name: string): bool
- +_checkActiveLanguage(): void
- +_getActiveLanguage(): string
- +_fromContentObject(object: eZContentObject): SQLFieldsetHolder
- +_addTranslation(language: string): void
- +_getCurrentDraft(language: string): eZContentObjectVersion
- +_newInstance(): void

SQLContentFieldset

- #language: string
- #local: string
- #fields: array(SQLContentField)
- #referenceToFieldsByAttributeIdentifier
- #iteratorPointer: array
- +_get(name: string): SQLContentField
- +_set(name: string, value: string): void
- +_isset(name: string): bool
- +_getLanguage(): string
- +_setLanguage(language: string): void
- +_fromFieldMap(fieldMap: array(eZContentObjectAttribute): SQLContentFieldset)
- +_refresh(mode: eZContentObjectVersion): void
- +_setContentObjectAttribute(attribute: eZContentObjectAttribute): void
- +_resetInputData(): void
- +_newInstance(): void

SQLContentField (Proxy class for eZContentObjectAttribute)

- #attribute: eZContentObjectAttribute
- #identifier: string
- #data: string
- +_fromContentObjectAttribute(attribute: eZContentObjectAttribute): SQLContentField
- +_isset(name: string): bool
- +_get(name: string): mixed
- +_set(name: string, value: mixed): void
- +_toString(): string
- +_call(method: string, arguments: array): mixed
- +_getRawAttribute(): eZContentObjectAttribute
- +_setAttribute(attribute: eZContentObjectAttribute): void
- +_setData(data: string): void
- +_getData(): string
- +_reset(): void
- +_isModified(): bool
- +_check(): void

SQLLocation

- #node: eZContentObjectNode
- #mode: int
- +_construct(): SQLLocation
- +_fromNode(mode: eZContentObjectNode): SQLLocation
- +_get(name: string): mixed
- +_set(name: string, value: mixed): void
- +_isset(name: string): bool
- +_call(method: string, arguments: array): mixed

SQLLocationSet

- #locations: array(SQLLocation)
- #mode: int
- #iteratorPointer: array
- +_construct(): SQLLocationSet
- +_newInstance(): void
- +_addLocation(location: SQLLocation): void

SQLInterfaceArrayAccess

SQLInterfaceIterator

2 source handlers are provided :

- **SQLXMLParser** - Catches parse errors and fetches XML string. Transforms PHP errors into exceptions. Works with DOM or SimpleXML (example in *stubs/xml.php*).
- **SQLICSVDoc** - Set of classes to manage CSV structures as easily as with SimpleXML (example in *stubs/csv.php*).

interface PHPDoc for further information. You can also check the code of provided handlers for examples.