SQLIImport eZ Publish extension

Version 1.0 - 2010 Jerome Vieilledent for SQLi

Date: 2010/08/26

Table of contents

PRESENTATION

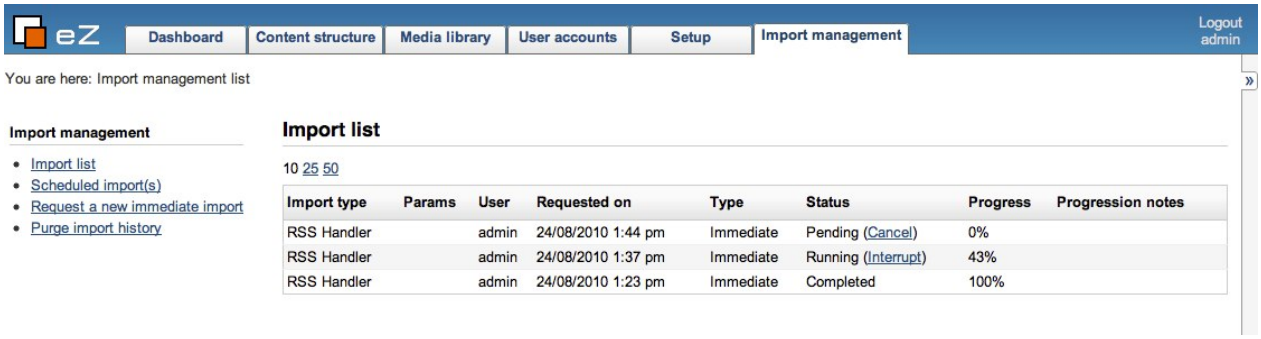
SQLIImport is an extension allowing to import external content in eZ Publish. It provides a framework for content manipulation and a simple interface for developing import handlers, so as a GUI to administrate your imports in the admin interface Import handlers are classes you need to develop in order 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 *SQLIImport*, a new **Import Management** tab appears (you'll need to have access to *sqliimport* 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 :



Immediate & Scheduled

There are two types of imports :

- Immediate
- Scheduled

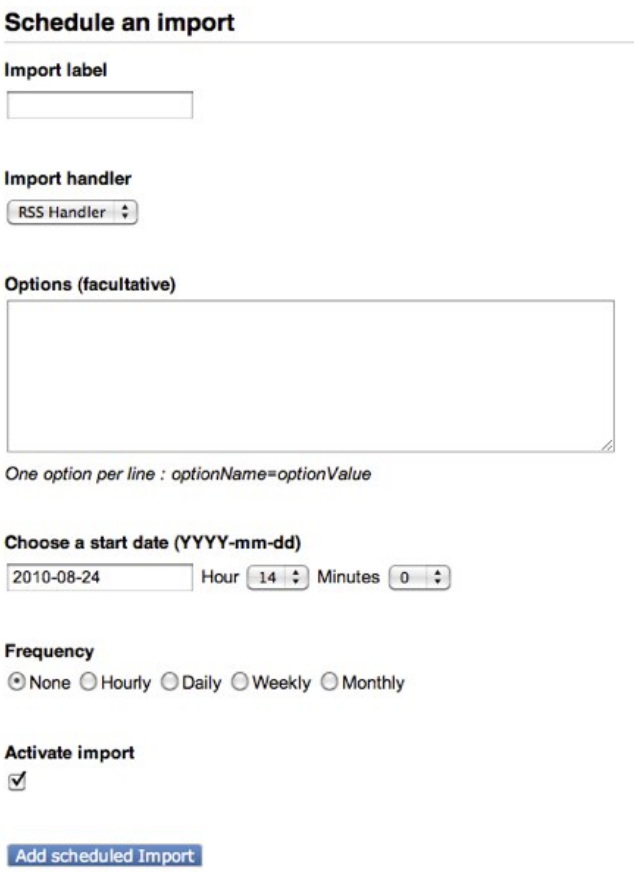
Each import is stored in the database as pending and awaits for the cronjob to process it. Running imports are safely interruptable from the admin interface. Pending imports can be cancelled while the cronjob has not processed it.

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 :



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 *sqliimport_cleanup* cronjob (launched after *sqliimport_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 *sqliimport_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 :

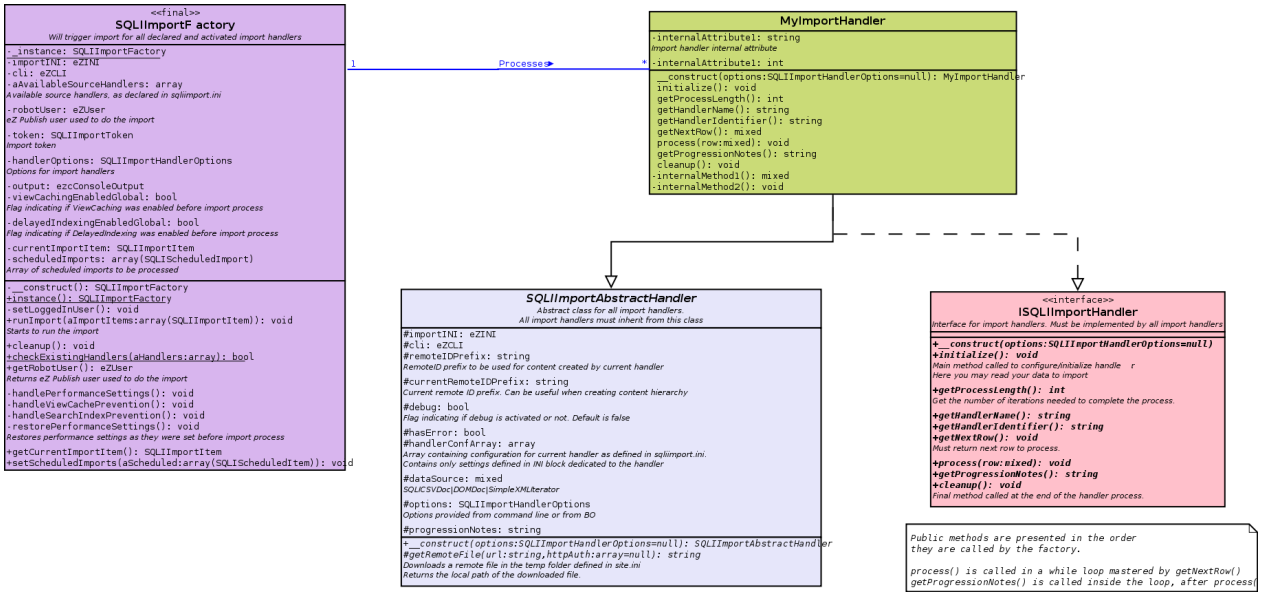
- ezimage
- ezbinaryfile

HANDLER DEVELOPMENT

To import external content into eZ Publish with SQLIImport, 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 **SQLIImportAbstractHandler** and implements **ISQLIImportHandler**. You will also need to declare it in an override of **sqliimport.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 *sqliimport.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 datatypes *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.

Note about HTML content :

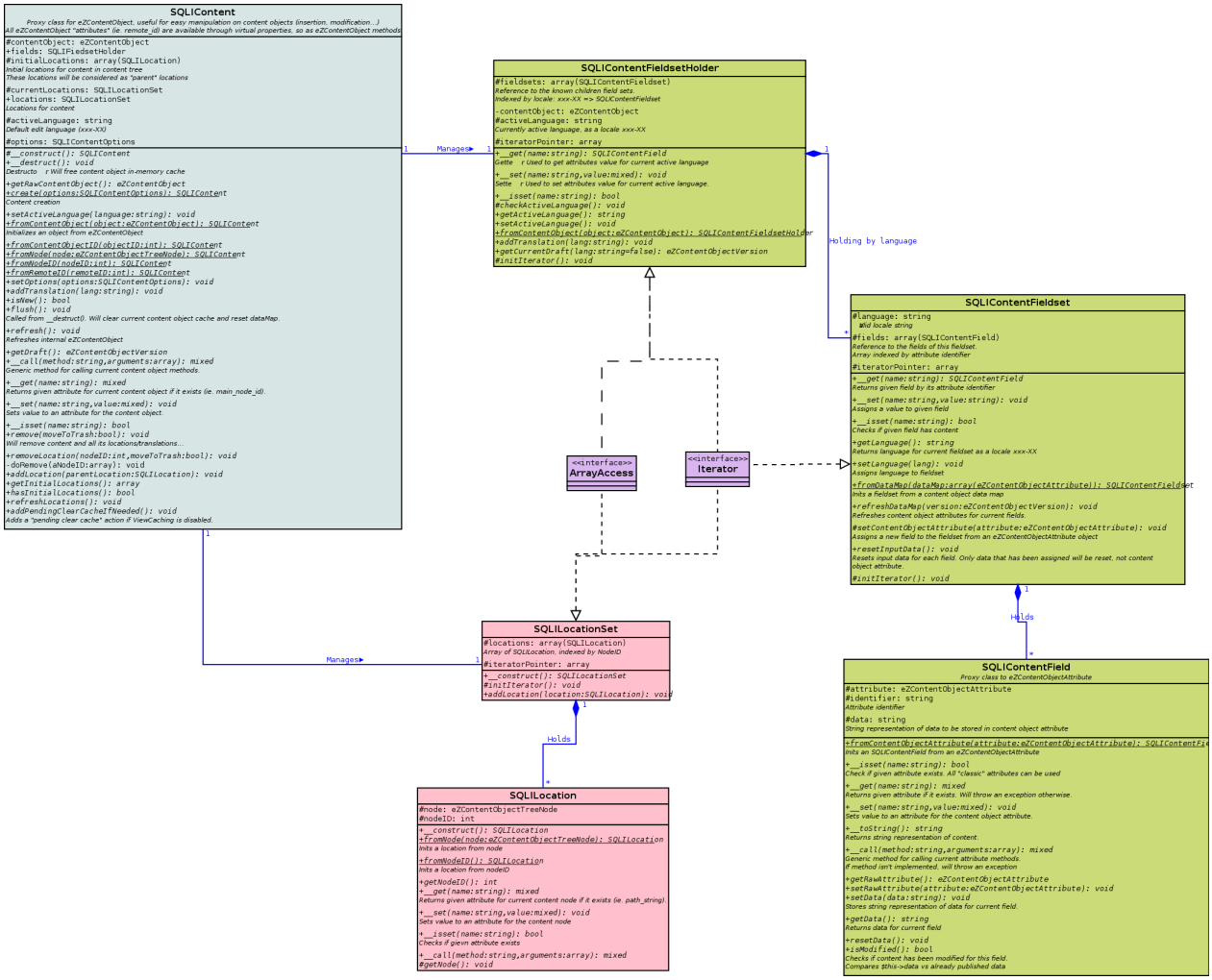
A shorthand method is available to handle HTML content import. It is available in *Import Handlers* and outside import handlers :

```
// Code below is available in an import handler
$content->fields->intro = $this->getRichContent( $myHTMLContent );

// Code below is available everywhere
$eZXMLContent = SQLIContentUtils::getRicheContent( $myHTMLContent );
```

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

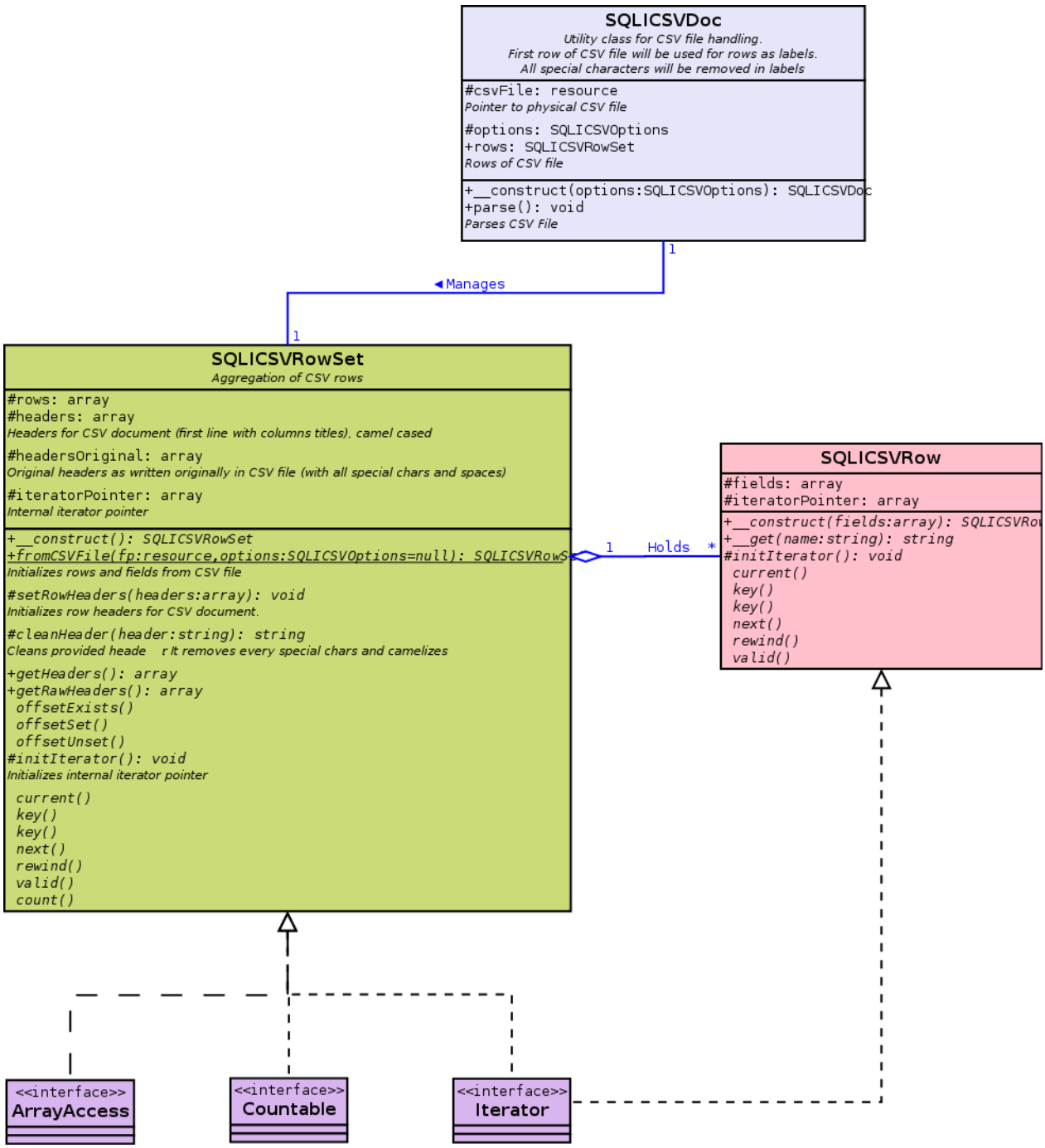
You can also have a look at the UML diagram below.



Source Handlers

2 source handlers are provided :

- **SQLIXMLParser** - 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*).



DIFF HANDLER DEVELOPMENT

When updating a content, **SQLIContentPublisher** only publishes really modified content by default. It makes a diff between already stored content and new content. This is done via diff handlers.

3 diff handlers are provided by default :

- *SQLIDefaultDiffHandler* - will basically compare attributes string representation
- *SQLIImageDiffHandler*
- *SQLIBinaryFileDiffHandler*

You can develop your own diff handler for your datatypes by creating a class implementing **ISQLIDiffHandler** interface. Only one static method is needed : **contentIsModified()**. Please read

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