EXT: External Import

Extension Key: external_import
Language: en
Keywords: forAdmins, forIntermediates

Table of Contents

Introduction

This extension is designed to fetch data from external sources and store them into tables of the TYPO3 database. The mapping between this external data and the TYPO3 tables is done by extending the syntax of the TCA. A backend module provides a way to synchronize any table manually or to defined a scheduling for all synchronizations. Automatic scheduling can be defined using a Scheduler task.

The main idea of getting external data into the TYPO3 database is to be able to use TYPO3 standard functions on that data (such as enable fields, for example, if available).

Connection to external applications is handled by a class of services called "connectors", the base of which is available as a separate extension (svconnector).

Data from several external sources can be stored into the same table allowing data aggregation.

The extension also provides an API for sending it data from some other source. This data is stored into the TYPO3 database using the same mapping process as when data is fetched directly by the extension.

This extension contains a number of hooks as well as the possibility to call user-defined functions during the import process, which makes it a quite flexible tool. However it was not designed for extensive data manipulation. It is assumed that the data received from the external source is in "palatable" format. If the external data requires a lot of processing, it is probably better to put it through an ETL or ESB tool first, and then import it into TYPO3.

Please also check extension "external_import_tut" which provides a tutorial to this extension.

Questions and support

If you have any questions about this extension, please ask them in the TYPO3 English mailing list (typo3.english), so that others can benefit from the answers. Please use the bug tracker on forge.typo3.org to report problem or suggest features (http://forge.typo3.org/projects/extension-external_import/issues).

Keeping the developer happy

It's unfortunately not possible to rate extensions anymore (maybe that will change again), so feel free to send thanks to the developers or praise the extension in general. If you really want to give something back, you may consider my Amazon wish list: http://www.amazon.co.uk/registry/wishlist/G7DI2AN99Y4F

You may also take a step back and reflect about the beauty of sharing. Think about how much you are benefiting and how much yourself is giving back to the community.

Participating

This tool can be used in a variety of situations and all use cases are certainly not covered by the current version. I will probably not have the time to implement any use case that I don't personally need. However you are welcome to join the development team if you want to bring in new features. If you are interested go to forge.typo3.org and apply to become a project member. I'll get in touch with you.

Installation

Installing this extension does nothing in and of itself. You still need to extend the TCA definition of some tables with the appropriate syntax and create specific connectors for the application you want to connect to.

Automating the imports requires system extension "scheduler".

Version 2.0 requires TYPO3 4.5 or above, and system extensions "extbase" and "fluid".

Extension "gabriel" is no longer supported. Use versions from the 1.x branch if you still need to use Gabriel.

Compatibility issues

Upgrade to 2.0.0

The column configuration "excludedOperations" has been renamed to "disabledOperations", for consistency with the table configuration option. The "excludedOperations" is preserved for now and will log an entry into the deprecation log. You are advised to change the naming of this configuration if you use it, support will be dropped at some point in the future.

Other requirements

As was mentioned in the introduction, this extension makes heavy use of an extended syntax for the TCA. If you are not familiar with the TCA, you are strongly advised to read up on it in the **TCA Reference** documentation.

Configuration

The extension has the following configuration options:

- **Storage PID**: define a general page where all the imported records are stored. This can be overridden specifically for each table (see Administration below).
- **Force time limit**: set a maximum execution time (in seconds) for the manual import processes (i.e. imports launched from the BE module). This time limit affects both PHP (where the default value is defined by max_execution_time) and the AJAX calls triggered by the BE module (where the default limit is 30 seconds). This is necessary if you want to run large imports. Setting this value to -1 preserves the default time limit.
- **Email for reporting**: if an email address is entered here, a detailed report will be sent to this address after every automated synchronization. Mails are not sent after synchronizations started manually from the BE module.
  Note that the mail reporting feature needs a valid e-mail address to be available for sending from. This will either be the mail of the "_cli_scheduler" user or the default mail address of the TYPO3 installation (`$TYPO3_CONF_VARS['MAIL']['defaultMailFromAddress']`). If neither of these mails are available, the report will not be sent and an error will appear in the Admin Tools > Log.
- **Subject of email report**: a label that will be prepended to the subject of the reporting mail. It may be convenient – for example – to use the server's name, in case you have several servers running the same imports.
- **Preview/Debug limit**: this is the maximum number of rows that will be dumped to the devlog when debugging is turned off. It will also be used as the number of rows displayed during a preview, when that feature is implemented.
- **Debug**: check to enable the extension to store some log data (requires an extension such as devlog).
- **Disable logging**: check to disable logging by TCEmain. By default an entry will be written in the Admin Tools > Log for each record touched by the import process. This may create quite a lot of log entries on large imports. Checking this box disables logging for **all** tables. It can be overridden at table-level by the "disableLog" flag (see "General TCA configuration").
  There is one big drawback to this method however. If TCEmain logging is disabled, errors are not tracked at all. This means that the import will run happily all the time and never report errors. You will unfortunately have to choose between errors not being reported and your log being flooded.

User manual

General considerations

The purpose of this extension is to take data from somewhere else (called the external source) than the local TYPO3 database and store it into that local database. Data from the external source is matched to local tables and fields using information stored in the TCA, using the extended syntax provided by this extension.

The extension can either fetch the data from some external source or receive data from any kind of script using the provided API. Fetching data from an external source goes through a standardized process.

Connecting to an external source is achieved using connector services (see extension "svconnector"), that will return the fetched data to the external import. Once such a connector exists, it can be related to one or more TYPO3 tables (with additional parameters if needed) using the extended TCA syntax. From then on the table can be synchronized with the external source. Every time a synchronization is started (either manually or according to a schedule), the connector service is called upon to fetch the data. Such tables are referred to as "**synchronizable tables**". This type of action is called "pulling data".

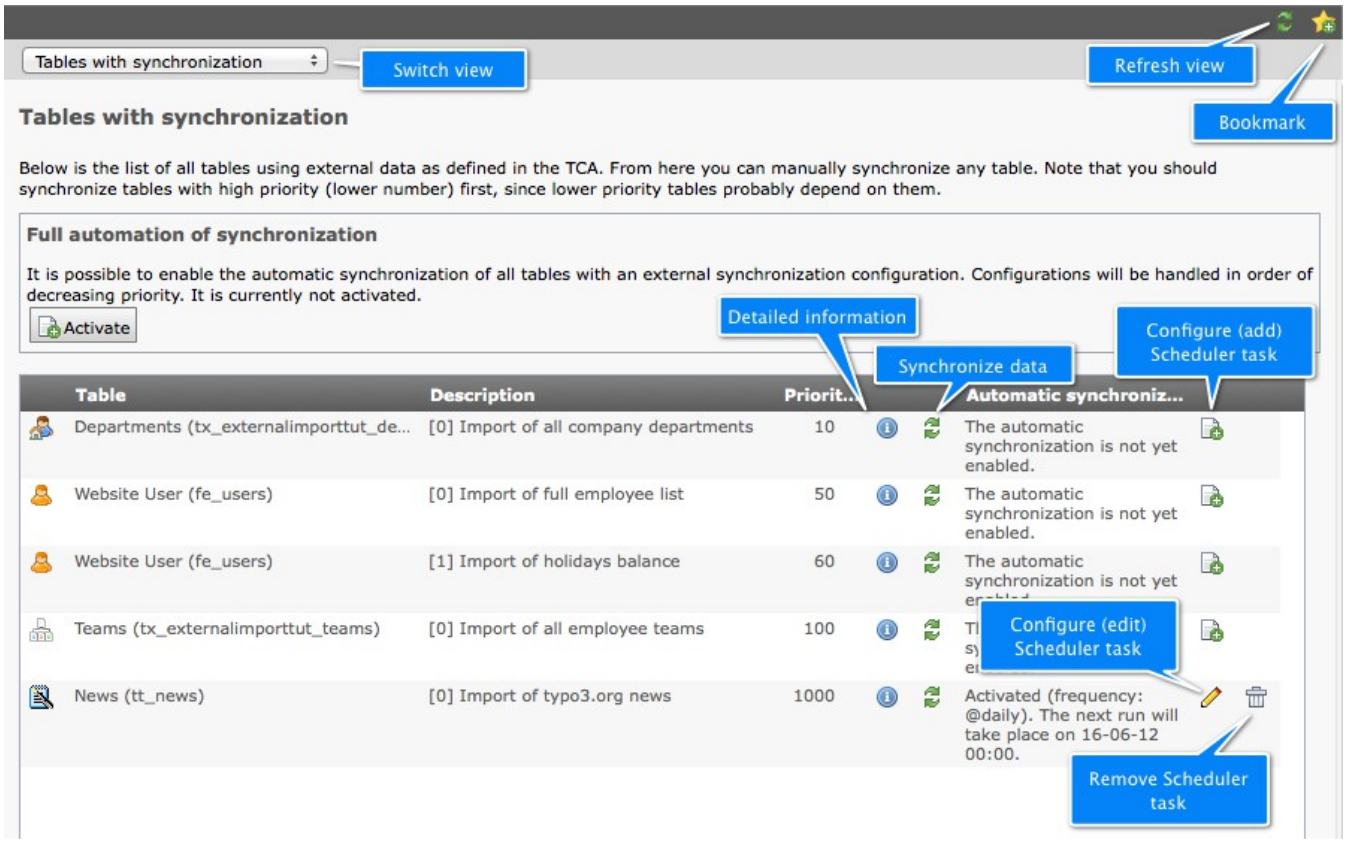
On the other hand this extension also provides an API that can be called up to pass data directly to the external import process. No connector services are used in this case. The extension is called on a need-to basis by any script that uses it. As such it is not possible to synchronize those tables from the BE module, nor to schedule their synchronization. Such tables are referred to as "**non-synchronizable tables**". This type of action is called "pushing data".

Note that it is perfectly possible to also push data towards synchronizable tables. The reverse is not true (non-synchronizable tables cannot pull data).

Using the backend module

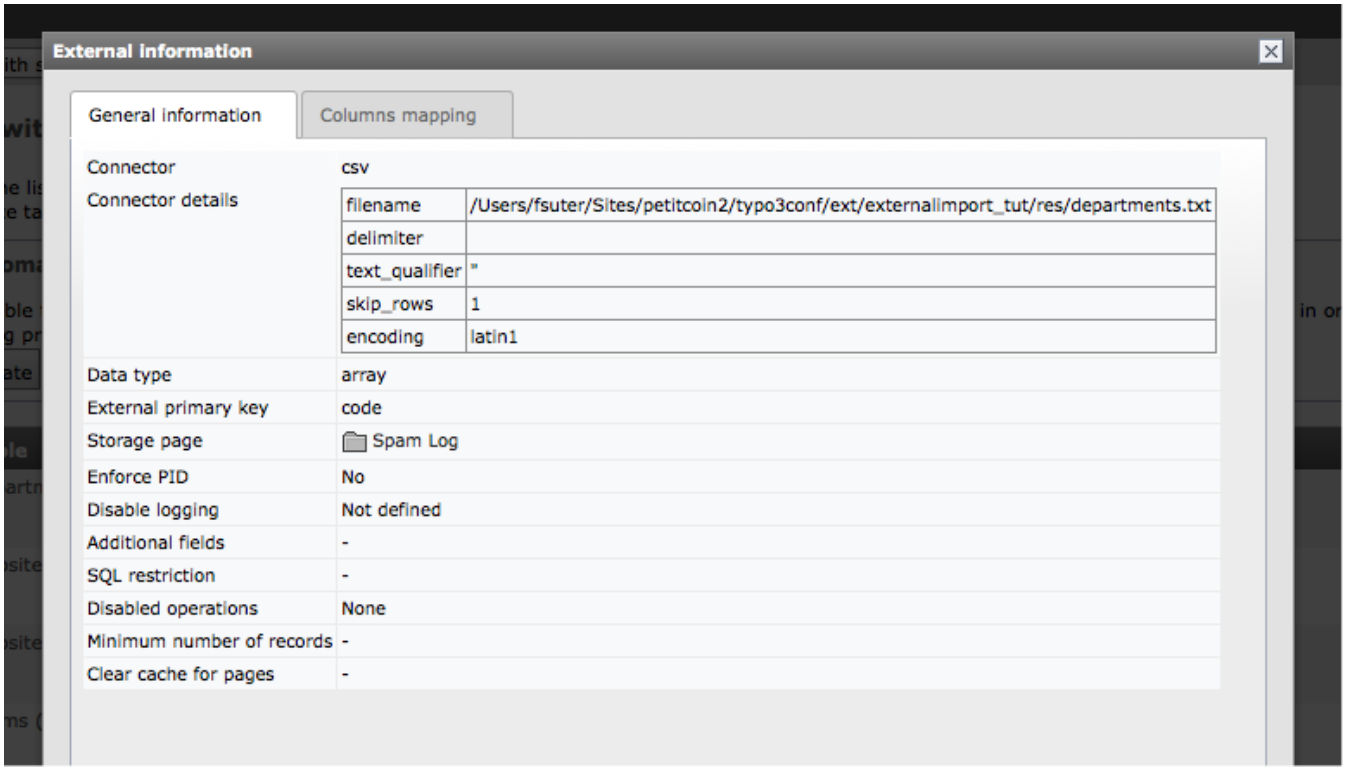
Synchronizable tables

The first function of the BE module – called "Tables with synchronization" – displays a list of all synchronizable tables. The various features are summarized in the picture below.

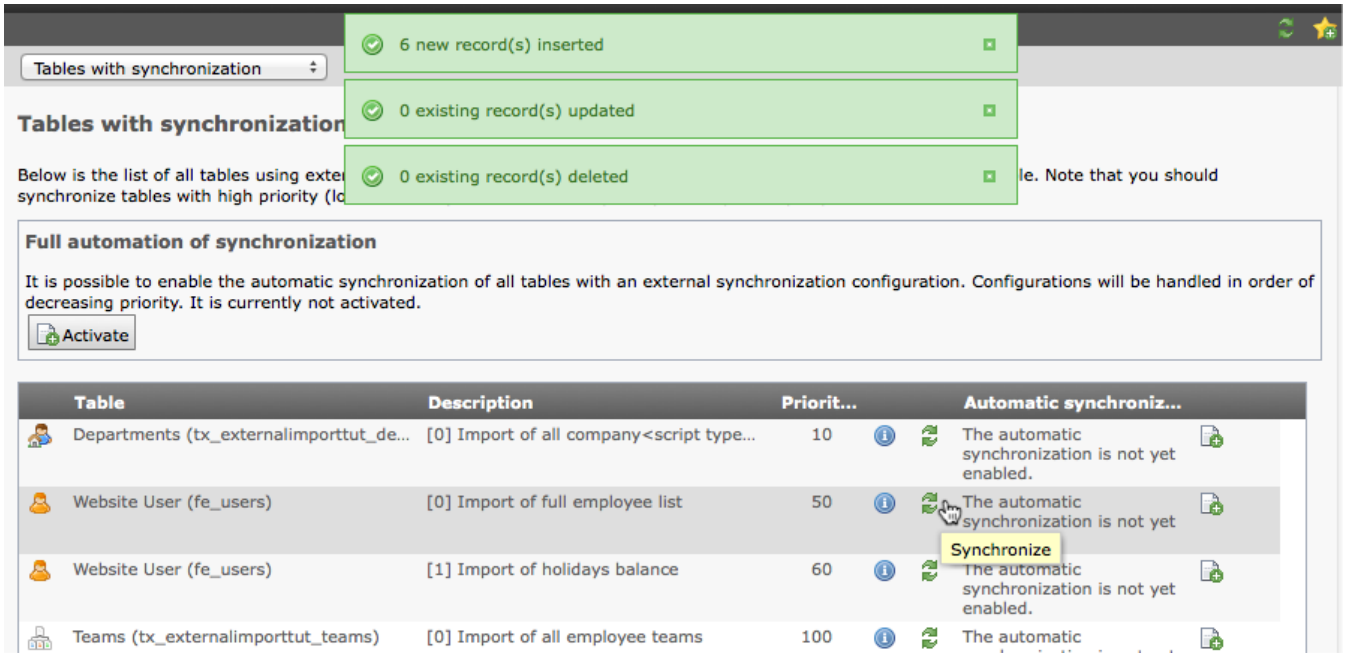


Note that some icons may not appear depending on user rights. Users without write access to a given table will not see the synchronize button, nor any of the actions related to the Scheduler.

Clicking on the information icon will open a pop-up window containing all the information about that particular configuration. The view consists of two tabs: the first one displays the configuration from the "ctrl" section of the TCA ("General information"), the second one displays the configuration for each column ("Columns mapping").



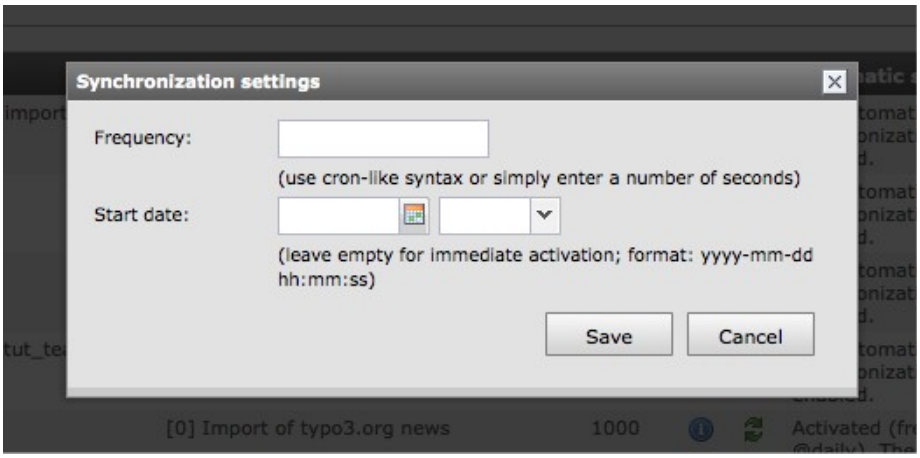
Clicking on the synchronize data button will immediately start the synchronization of the corresponding table. This may take quite some time if the data to import is large. If you move away from the BE module during that time, the process will still complete, but you will not get any feedback about the results. If you wait until the end of the process, pop-up messages will appear with the results:



Setting up the automatic schedule

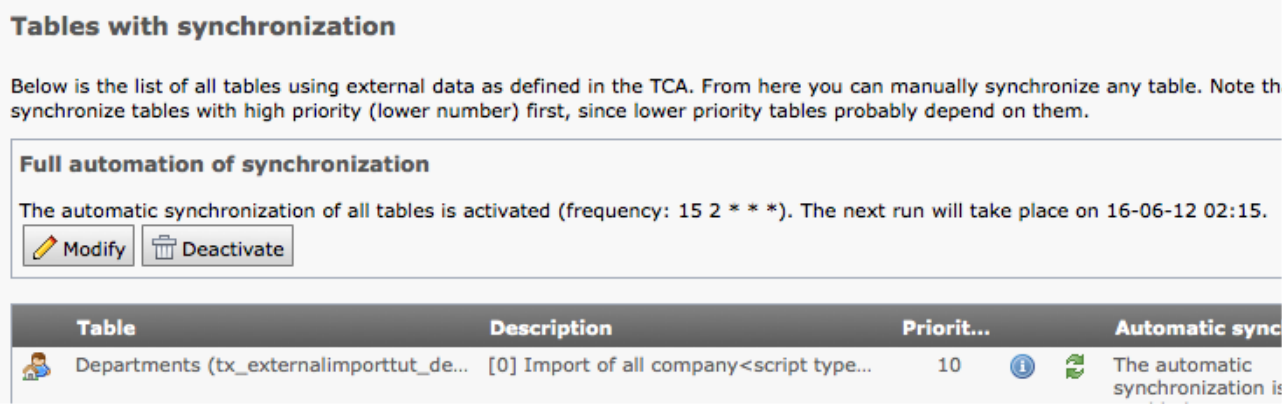
The automatic scheduling facility relies on the Scheduler to run. On top of the normal Scheduler setup, there are some points you must pay particular attention to in the case of external import.

As can be seen in the above screenshot, the information whether the automatic synchronization is enabled or not is displayed for each table. It is possible to add or change that schedule, by clicking on the respective icons. This triggers the display of a pop-up window with an input form where you can choose a start date (date of first execution; leave empty for immediate activation) and a frequency. The frequency can be entered as a number of seconds or using the same syntax as for cron jobs.



Clicking on the trash can icon cancels the automatic synchronization (a confirmation window will appear first).

At the top of the screen, before the list, it is possible to define a schedule for **all** tables. This means that all imports will be executed one after the other, in the order of priority.



Clicking on the "Activate" or "Modify" button will trigger the same window as for individual tables. Clicking on "Deactivate" will remove the scheduling.

Defining a schedule is not enough. Proper user rights must also be considered. See the "User rights" section in the "Administration" chapter.

Non-synchronizable tables

The second function of the BE module – called "Tables without synchronization" – displays a list of non-synchronizable tables. This view is purely informative as no action can be taken for these tables.

Mapping data

In the Administration chapter below, you will find explanations about how to map the data from the external source to existing or newly created tables in the TYPO3 database. There are two mandatory conditions for this operation to succeed:

- the external data **must** have the equivalent of a primary key
- this primary key **must** be stored into some column of the TYPO3 database, but **not** the uid column which is internal to TYPO3.

The primary key in the external data is the key that will used to decide whether a given entry in the external data corresponds to a record already stored in the TYPO3 database or if a new record should be created for that entry. Records in the TYPO3 database that do not match primary keys in the external data can be deleted if desired.

Clearing the cache

When data is imported into your TYPO3 installation, you may want to clear the cache for a number of pages in order for the new data to be displayed as soon as it is available. One way to achieve this is to rely purely on TYPO3 and use the TSconfig property:

```
TCEMAIN.clearCacheCmd = xx,yy
```

on the page(s) where the data is stored to automatically trigger the clearing of the cache for the given pages (xx and yy) when any record they contain is modified or deleted, or some new record inserted.

This works fine but has one big drawback: it is triggered for **each** record. If you manipulate a lot of records, the cache clearing may be called hundreds or thousands of times. This can be very bad for your site, especially if you have a very large cache.

Since version 2.0 of External Import, it is possible to trigger the clearing of the cache **after** the whole import process has completed for a given configuration. Instead of using TSconfig, the configuration would be something like:

```
$TCA['tt_news']['ctrl']['external']['0']['clearCache'] = 'xx,yy';
```

This will clear the cache for pages xx and yy, but only after all records have been inserted, updated and deleted. The process still relies on TCEmain for clearing the cache of each page, so you may rely on the usual clear cache hooks.

Debugging

There are many potential sources of error during synchronization from wrong mapping configurations to missing user rights to PHP errors in user functions. When a synchronization is launched from the BE module an ExtDirect call is made to the import script. The response is read and displayed in the BE module.

However if some other errors happen, like a PHP error, or any type of output produced by calls to debug methods will not be visible because they are flushed before the ExtDirect response is sent. On way around this is to use an extension that writes to the devLog (e.g. "devlog") and activate `$TYPO3_CONF_VARS[SYS][enable_DLOG]`.

As described in "Configuration" above, it is also possible to receive a detailed report by email. It will contain a general summary of what happened during synchronization, but also all error messages logged by TCEmain, if any.

Troubleshooting

The automatic synchronization is not being executed

You may observe that the scheduled synchronization is not taking place at all. Even if the debug mode is activated and you look at the devLog, you will see no call to external_import. This may happen when you set a too high frequency for synchronizations (like 1 minute for example). If the previous synchronization has not finished, the Scheduler will prevent the new one from taking place. The symptom is a message like "[scheduler]: Event is already running and multiple executions are not allowed, skipping! CRID: xyz, UID: nn" in the system log (Admin Tools > Log). In this case you should delete the existing schedule and set up a new one.

The manual synchronization never ends

It may be that no results are reported during a manual synchronization and that the looping arrows continue spinning endlessly. This happens when something failed completely during the synchronization and the BE module received no response. See the advice in "Debugging" above.

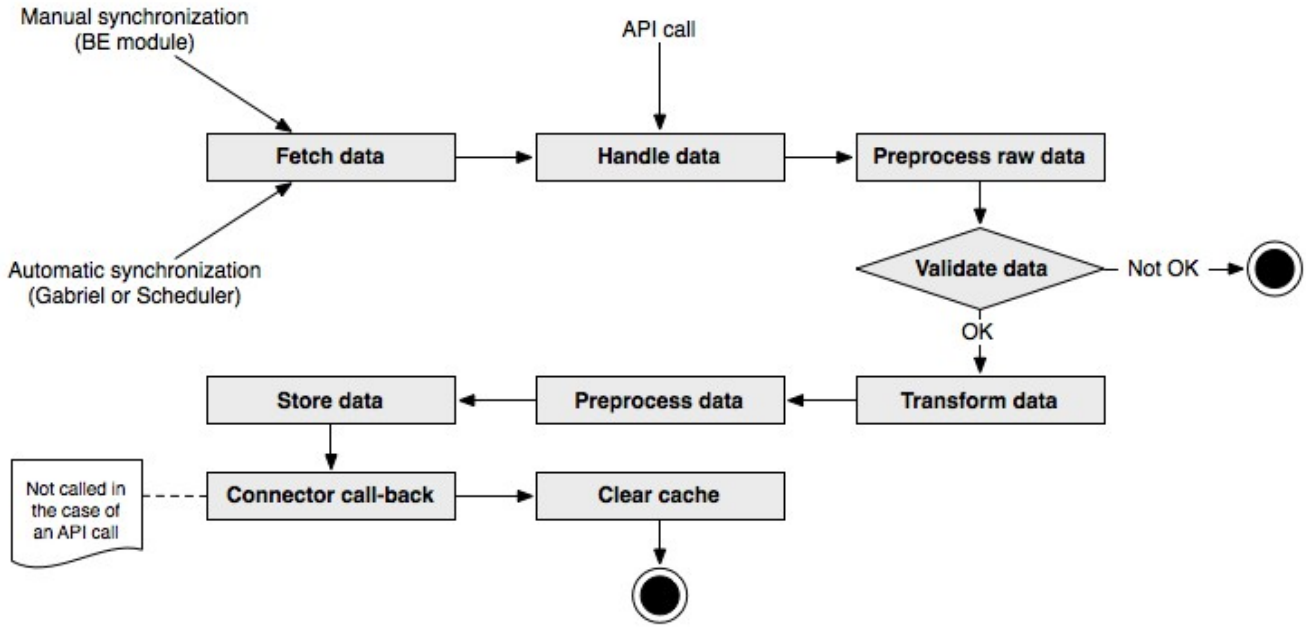
All the existing data was deleted

The most likely cause is that the external data could not be fetched, resulting in zero items to import. If the delete operation is not disabled, External import will take that as a sign that all existing data should be deleted, since the external source didn't provide anything.

There are various ways to protect yourself against that. Obviously you can disable the delete operation, so that no record ever gets deleted. If this is not desirable, you can use the "minimumRecords" option (see "General TCA configuration") below. For example, if you always expect at least 100 items to be imported, set this option to 100. If fewer items than this are present in the external data, the import process will be aborted and nothing will get deleted.

Process overview

The schema below provides an overview of the external import process:



When the external import is started from a synchronization operation (pull), data is first gathered from the external source (if some problem happens during this stage, the whole import process is aborted). This does not happen when the API is used, since the data is pushed into the import process. The next step is called "handle data". This is where the data that will be stored into the internal tables is filtered from all the data available from the external source. After this step, the external data is available inside the process as an associative PHP array with the keys matching the names of the database fields where the data will be stored.

The preprocess raw data step is just a container to call a hook. The next step validates the data. The base test is to check whether the minimum number of records is present in the external data or not. A hook is available for introducing more specific checks. The first check to fail (including the base check) triggers the abortion of the import process.

The transformation step is comprised of two important operations:

1. all simple (i.e. not MM) mappings are handled (or fixed values are applied).
2. declared user functions are called.

The preprocess step does nothing by itself, but provides a hook for manipulating the complete recordset of imported data.

Finally the data is actually stored to the database. Before this happens the MM-relationships are handled and hooks are available before each type of operation happens (insert, update and delete).

As a last step the connector is called again in case one wishes to perform some clean up operations on the source from which the data was imported (for example, mark the source data as having been imported). The postProcessOperations() method of the connector API is called. This will most probably just be a place for hooks as such post-processing operations are likely to be rather custom steps. Note that this step is not executed when the external import is started via an API call, as there is no connector involved in such a case.

Tutorial

Extension "externalimport_tut" provides an extensive tutorial about external import. It makes use of all possible configuration options. All examples are discussed in the extension's manual.

Administration

To start inserting data from an external source into your TYPO3 tables, you must first extend their TCA with a specific syntax, with general information in the "ctrl" section and specific information for each column. Obviously you can also create new tables and put your data in there.

User rights

Before digging into the TCA specifics let's have a look at the topic of user rights. Since External Import relies on TCEmain for storing data, the user rights on the synchronized tables will always be enforced. However additional checks are performed in both the BE module and the automated tasks to avoid displaying sensitive data or throwing needless error messages.

When accessing the BE module, user rights are taken into account in that:

- a user must have at least listing rights on a table to see it in the BE module.
- a user must have modify rights on a table to be allowed to synchronize it manually or define an automated synchronization for it.

DB mount points are not checked for at this point, so the user may be able to start a synchronization and still get error messages if not allowed to write to the page where the imported data should be stored.

When a synchronization runs automatically a check on user rights is also performed at the beginning, so that the synchronization can be skipped entirely if the CLI user does not have modify rights on the given table. This is reported in the mail report.

An automated synchronization will be run that the Scheduler. This means that the active user will be "_cli_scheduler", so this user needs to have enough rights to perform all expected operations, in particular:

- authorize this user to list and modify the tables that are going to be synchronized
- give this user access to the page(s) where the records are stored, i.e. pages must be in the DB Mounts of the user and user must enough rights on these pages, i.e. "Show page", "Edit content", "Edit page" and "Delete page" (Web > Access). Of course this can also be achieved via a group the user belongs to.

A good way to verify that the "_cli_scheduler" use has enough right is to use the "User Admin" module to switch to that user and perform manual synchronizations from there (this means giving access to the "External Import" BE module to the "_cli_scheduler" user).

General TCA configuration

Here is an example of a typical "ctrl" section syntax:

```
$TCA['tx_myext_mytable'] = array (
      'ctrl' => array (
            'title'     => ...,
            ...
            'external' => array(
                  0 => array(
                        'connector' => ...,
                        'parameters' => array(
                          ...
                        ),
                        'data' => 'xml',
                        'nodetype' => 'record',
                        'reference_uid' => ...,
                        'priority' => 10,
                        'pid' => 46,
                        'enforcePid' => true
                  )
            )
      ),
);
```

The "external" property is an indexed array. The following properties are available:

Key	Datatype	Description	Scope
connector	string	Connector service subtype Must be defined only for pulling data. Leave blank for pushing data.	Fetch data
parameters	array	Array of parameters that must be passed to the connector service. Not used when pushing data.	Fetch data
data	string	The format in which the data is returned by the connector service. Can be either "xml" or "array".	Fetch data
dataHandler	string	A class name for replacing the standard data handlers. See the "Developer's Guide" for more details.	Handle data
nodetype	string	Name of the reference nodes inside the XML structure, i.e. the children of these nodes correspond to the data that goes into the database fields.	Handle data (XML)
reference_uid	string	Name of the column where the equivalent of a primary key for the external data is stored.	Store data
priority	integer	A level of priority for execution of the synchronization. Some tables may need to be synchronized before others if foreign relations are to be established. This gives a clue to the user and a strict order for scheduled synchronizations. Not used when pushing data.	Display/Automated import process
pid	integer	ID of the page where the imported records should be stored. Can be ignored and the general storage pid is used instead (see configuration)	Store data
enforcePid	boolean	If this is set to true, all operations regarding existing records will be limited to records stored in the defined pid (i.e. either the above property or the general extension configuration). This has two consequences: a) when checking for existing records, those records will be selected only from the defined pid. b) when checking for records to delete, only records from the defined pid will be affected This is a convenient way of protecting records from operations started from within the external import process, so that it won't affect e.g. records created manually.	Store data
where_clause	string	SQL condition that will restrict the records considered during the import process. Only records matching the condition will be updated or deleted. This condition comes on top of the "enforcePid" condition, if defined. **Warning:** this may cause many records to be inserted over time. Indeed if some external data is imported the first time, but then doesn't match the "where_clause" condition, it will never be found for update. It will thus be inserted again and again. Whenever you make use of the "where_clause" property you should therefore watch for an unexpectedly high number of inserts.	Store data
additional_fields	string	Comma-separated list of fields from the external source that should be made available during the import process, but that will not be stored in the internal table. This is usually the case for fields which you want to use in the transformation step, but that will not be stored eventually.	Fetch data

Key	Datatype	Description	Scope

namespaces	array	Associative array of namespaces that can be used in XPath queries (see "Columns Configuration" below). The keys correspond to prefixes and the values to URIs. The prefixes can then be used in XPath queries.
		
		Example
		
		Given the following declaration:
		
		`'namespaces' => array( 'atom' => 'http://www.w3.org/2005/Atom' )`
		
		a Xpath query like:
		
		`atom:link`
		
		could be used. The prefixes used for XPath queries don't need to match the prefixes used in the actual XML source. The defaut namespace has to be registered too in order for XPath queries to succeed.
			Handle data (XML)

description	string	A purely descriptive piece of text, which should help you remember what this particular synchronization is all about. Particularly useful when a table is synchronized with multiple sources.	Display
disabledOperations	string	Comma-separated list of operations that should **not** be performed. Possible operations are insert, update and delete. This way you can block any of these operations. • **insert** is the operation performed when new records are found in the external source. • **update** is performed when a record already exists and only its data needs to be updated. • **delete** is performed when a record is in the database, but is not found in the external source anymore.	Store data
minimumRecords	integer	Minimum number of items expected in the external data. If fewer items are present, the import is aborted. This can be used – for example – to protect the existing data against deletion when the fetching of the external data failed (in which case there are no items to import).	Validate data
disableLog	boolean	Set to TRUE to disable logging by TCEmain. This setting will override the general "Disable logging" setting (see "Configuration" above for more details).	Store data
clearCache	string	Comma-separated list of pages whose cache should be cleared at the end of the import process. See "Clearing the cache" above.	Store data
deleteNotSynchedRecords	boolean	*This setting is deprecated. Please use **disabledOperations** instead.* Set to true if records that were not found during the synchronization (i.e. that do not exist in the distant source anymore) should be deleted. Set to false if they should be ignored.	Store data

Columns configuration

Then for each column, you also need an "external" syntax to define which external data goes into that column and any handling that might apply. This is also an indexed array. Obviously indices used for each column must relate to the indices used in the "ctrl" section. In its simplest form this is just a reference to the external data's name:

```
'field_name' => array (
      'exclude' => 0,
      'label' => '...',
      'config' => array(
            ...
      ),
      'external' => array(
            0 => array(
                  'field' => '...'
            )
```

```
        )
),
```

These are the parameters used in the column description:

Key	Datatype	Description	Scope
field	string	Name or index of the field (or node, in the case of XML data) that contains the data in the external source.	Handle data
attribute	string	If the data is of type XML, use this property to retrieve the value from an attribute of the node (selected with the "field" property above) rather than to the value of the node itself.	Handle data (XML)
xpath	string	This property can be used to execute a XPath query relative to the node selected with the "field" property. The value will be taken from the first node returned by the query. If the "attribute" property is also defined, it will be applied to the node returned by the XPath query. Please see the "namespace" option in the "General Configuration" above for declaring namespaces to use in a XPath query.	Handle data (XML)
fieldNS	string	Namespace for the given field. Use the full URI for the namespace, not a prefix.	Handle data (XML)
attributeNS	string	Namespace for the given attribute. Use the full URI for the namespace, not a prefix.	Handle data (XML)
MM	→ MM-configuration	Definition of MM-relations, see below for more details.	Transform data
mapping	→ Mapping configuration	This property can be used to map values from the external data to values coming from some internal table. A typical example might be to match 2-letter country ISO codes to the uid of the static_countries table.	Transform data
value	simple type	With this property, it is possible to set a fixed value for a given field. For example, this might be used to set a flag for all imported records.	Transform data
trim	boolean	If set ti TRUE, every value for this column will be trimmed during the transformation step.	Transform data
rteEnabled	boolean	If set to TRUE when importing HTML data into a RTE-enable field, the imported data will go through the usual RTE transformation process on the way to the database.	Transform data
userFunc	array	This property can be used to define a function that will be called on each record to transform the data from the given field. See example below. Note that the userFunc is called **after** the mapping.	Transform data
disabledOperations	string	Comma-separated list of database operations from which the column should be excluded. Possible values are "insert" and "update".	Store data
excludedOperations	string	**Deprecated** Replaced by "excludedOperations", see above.	Store data

Mapping configuration

The external values can also be matched to values from an existing TYPO3 table, using the "mapping" property.

Key	Datatype	Description	Scope
table	string	Name of the table to read the mapping data from.	Transform data
reference_field	string	Name of the field against which external values must be matched	Transform data
value_field	string	Name of the field to take the mapped value from. If not defined, this will default to "uid".	Transform data

Key	Datatype	Description	Scope

where_clause	string	SQL condition (without the "WHERE" keyword) to apply to the referenced table. This is typically meant to be a mirror of the "foreign_table_where" property of the select-type fields. However it is not possible to use markers in this case. So if you have something like:
		
		`'foreign_table_where' => 'AND pid = ###PAGE_TSCONFIG_ID###'`
		
		in the TCA for your column, you should replace the marker by a hard-coded value instead, e.g.
		
		`'where_clause' => 'pid = 42'`
		
		Note that the clause must not start with a "AND" keyword either.
			Transform data

valueMap	array	Fixed hash table for mapping. Instead of using a database table to match external values to internal values, this property makes it possible to use a simple list of key-value pairs. The keys correspond to the external values.	Transform data
match_method	string	Value can be "strpos" or "stripos". Normally mapping values are matched based on a strict equality. This property can be used to match in a "softer" way. It will match if the external value is found inside the values pointed to by the "reference_field" property. "strpos" will perform a case-sensitive matching, while "stripos" is case-unsensitive. Caution should be exercised when this property is used. Since the matching is less strict it may lead to false positive. You should review the data after such an import.	Transform data
match_symmetric	boolean	This property complements "match_method" above. If set to true, the import process will not only try to match the external value inside the mapping values, but also the reverse, i.e. the mapping values inside the external value.	Transform data

Here's an example TCA configuration. A "value_field" property is defined, although it would be optional in this case, since its value is "uid", which is the default.

```
'field_name' => array (
      'exclude' => 0,
      'label' => '...',
      'config' => array(
            ...
      ),
      'external' => array(
            0 => array(
                  'field' => '...',
                  'mapping' => array(
                        'table' => name of foreign table,
                        'reference_field' => foreign MM key,
                        'value_field' => 'uid'
                  )
            )
      )
),
```

"Soft mapping" considerations

It is important to understand how the "match_method" property influences the matching process. Consider trying to map freely input country names to the `static_countries` table inside TYPO3. This may not be so easy depending on how names were input in the external data. For example, "Australia" will not strictly match the official name, which is "Commonwealth of Australia". However setting "match_method" to "strpos" will generate a match, since "Australia" can be found inside "Commonwealth of Australia"

User functions configuration

Here's an example setup for calling a user function.

```
'field_name' => array (
      'exclude' => 0,
      'label' => '...',
      'config' => array(
            ...
      ),
      'external' => array(
            0 => array(
                  'field' => '...',
                  'userFunc' => array(
                        'class' =>
'EXT:external_import/samples/class.tx_externalimport_transformations
.php:&tx_externalimport_transformations',
                        'method' => 'parseDate',
                        'params' => array(
                          'function' => 'date',
                          'format' => 'd.m.Y'
                        )
                  )
            )
      )
),
```

A user function requires three parameters. The first one ("class") is the name of the class to be instantiated. It can be prefixed by a path, in which case the file will be included automatically for you. Note the "&" before the class name. This will make the instance a singleton, avoiding too many instances. The next parameter ("method") defines which method of the class should be called. The third parameter ("params") is optional. It is an array and can contain any number of data. It will be passed to the method.

In the example above we are using a sample class provided with external import that can be used to parse a date and either return it as a timestamp or format it using either of the PHP functions `date()` or `strftime()`.

For more details about creating a user function, please refer to the Developer's Guide, below.

MM-relations configuration

It gets more complicated if there are MM-relations to rebuild after import:

```
'external' => array(
      0 => array(
            'MM' => array(
                  'mapping' => array(
                        'table' => name of foreign table,
                        'reference_field' => foreign MM key,
                  ),
                  'additional_fields' => array(
                        TYPO3 field name => external data field name
                  ),
                  'sorting' => 'field',
            )
      )
)
```

These are the parameters used in the MM configuration:

Key	Datatype	Description	Scope
mappings	array	*This property has been deprecated. See "mapping" below.*	Store data
mapping	→ Mapping configuration	This is similar to the "mapping" property described above. It is used to define which table to link to and which column in that table contains the external primary key.	Store data
additional_fields	array	List of fields that must be stored along the local and foreign keys in the MM table. For each such field, define which TYPO3 MM-table field corresponds to which external data field.	Store data
multiple	boolean	If some mm-relations exist several times in your external data (because they have various additional fields), you must set this property to 1, so that they are preserved (otherwise TCEmain will take only unique uid_local, uid_foreign pairs into account).	Store data

Key	Datatype	Description	Scope
sorting	string	Indicates that the data is to be sorted according to that particular field from the external data. Note that since the external import relies on TCEmain to store the data, TCEmain sets its own numbering for sorting, thus the value in sorting is never used as is, but just for ordering the records. So if the records in the external source are already sorted, there's no need to define the "sorting" property.	Store data

Note: when the "additional_fields" and/or "multiple" properties are used, additional database operations are performed to honour these settings, as it is not traditional behaviour for TYPO3 MM-relations. It should be possible with IRRE, but this isn't supported yet.

Developer's Guide

External Import API

It is very simple to use the external import features. You just need to assemble data in a format it can understand (XML structure or recordset) and call the appropriate method. You will need to include calls class.tx_externalimport_importer.php and do the following call:

```
$importer = t3lib_div::makeInstance('tx_externalimport_importer');
$importer->importData($table, $index, $rawData);
```

The call parameters are as follows:

Name	Type	Description
$table	string	Name of the table to store the data into
$index	integer	Index of the relevant external configuration
$rawData	mixed	The data to store, either as XML or recordset

This is particularly useful in conjunction with the Remote Server extension (key: remote_server). With this in place you can call the TYPO3 BE and send data to it, then handle and store this data into local tables using the external import API.

User functions

The external import extension can call user functions for any field where external data is imported. A sample function is provided in `samples/class.tx_externalimport_transformations.php`. Basically, the function will receive three parameters:

Name	Type	Description
$record	array	The complete record being handled. This makes it possible to refer to other fields of the same record during the transformation, if needed.
$index	string	The key of the field to transform. Modifying other fields in the record is not possible since the record is passed by value and not by reference. Only the field corresponding to this key should be transformed and returned.
$params	array	Additional parameters passed to the function. This will be very specific to each function and can even be complete omitted. External import will pass an empty array to the user function if the "params" property is not defined.

The function is expected to return only the value of the transformed field.

Warning: the record received as input into the user function has already gone through renaming the fields. That means the names of the fields are not those of the external data, but those of the TYPO3 fields.

Hooks

The external import process contains the following hooks:

- **preprocessRawRecordset**: this hook makes it possible to manipulate the data just after it was fetched from the remote source, but already transformed into a PHP array, no matter what the original format. The hook receives the full recordset and a back-reference to the calling object (an instance of class tx_externalimport_importer) as parameters. It is expected to return a full recordset too.
- **validateRawRecordset**: this hook is called during the data validation step. It is used to perform checks on the nearly raw data (it has only been through "preprocessRawRecordset") and decide whether to continue the import or not. The hook receives the full recordset and a back-reference to the calling object (an instance of class tx_externalimport_importer) as parameters. It is expected to return a boolean, true if the import may continue, false if it must be aborted.
  Note the following: if the minimum number of records condition was not matched, the hooks will not be called at all. Import is aborted before that. If several methods are registered with the hook, the first method that returns false aborts the import. Further methods are not called.
- **postprocessRecordset**: similar to "preprocessRawRecordset", but after the transformation step, so just before it is stored to the database. The hook receives the full recordset and a back-reference to the calling object (an instance of class tx_externalimport_importer) as parameters. It is expected to return a full recordset too.
- **updatePreProcess**: this hook can be used to modify a record just before it is updated in the database. The hook is

called for each record that has to be updated. The hook receives the complete record and a back-reference to the calling object (an instance of class tx_externalimport_importer) as parameters. It is expected to return the complete record.

- **insertPreProcess**: similar to the "updatePreProcess" hook, but for the insert operation.
- **deletePreProcess**: this hook can be used to modify the list of records that will be deleted. As a first parameter it receives a list of primary key, corresponding to the records set for deletion. The second parameter is a reference to the calling object (again, an instance of class tx_externalimport_importer). The method invoked is expected to return a list of primary keys too.
- **datamapPostProcess**: this hook is called after all records have been updated or inserted using TCEmain. It can be used fo any follow-up operation. It receives as parameters the name of the affected table, the list of records keyed to their uid (including the new uid's for the new records) and a back-reference to the calling object (an instance of class tx_externalimport_importer). Each record contains an additional field called "tx_exteralimport:status" which contains either "insert" or "update" depending on what operation was performed on the record.
- **cmdmapPostProcess**: this hook is called after all records have been deleted using TCEmain. It receives as parameters the name of the affected table, the list of uid's of the deleted records and a back-reference to the calling object (an instance of class tx_externalimport_importer).

Custom data handlers

It is possible to use a custom data format instead of the standard `tx_externalimport_importer::handleArray()` and `tx_externalimport_importer::handleXML()`. The value declared as a custom data handler:

```
$TCA['some_table']['ctrl']['external'][0]['data'] = 'tx_foo_bar';
```

is a class name. The corresponding class file should be declared with the autoloader.

The class itself **must** implement the `tx_externalimport_dataHandler` interface, which contains only the `handleData()` method. This method will receive two arguments:

- an array containing the raw data returned by the connector service
- a reference to the calling `tx_externalimport_importer` object

The method is expected to return a simple PHP array, with indexed entries, like the standard methods (`tx_externalimport_importer::handleArray()` and `tx_externalimport_importer::handleXML()`).

Note: this was not tested by myself (the extension author). It was introduced to answer the particular need to parse large arrays using method similar to XPath. This would have relied on a library which was not considered stable enough. Having custom data handlers makes it possible

Known problems

Namespace support for XML data type was introduced for a project within a very precise scope. It was tested only within that scope and should be considered as being in "beta" state at best. Namespace support is not trivial and can certainly be improved.

Please report bugs and improvements as usual to: http://forge.typo3.org/projects/extension-external_import/issues

To-Do list

There is a roadmap on Forge for the continuing development of this extension:

http://forge.typo3.org/projects/roadmap/extension-external_import

Below are some other ideas that have no priority for now:

- Handle localized records, when translations are in the same table record
- Look at IRRE for handling MM-relations that use additional fields or are repeated several times.

Appendix A – Old upgrade processes

This chapter describes potential upgrade issues for older versions of External Import (branches 0.x and 1.x).

Upgrade to TYPO3 4.3 and the Scheduler

If you already have a complete setup using Gabriel on a TYPO3 4.2 or less box, the upgrade process will not be completely smooth. Indeed TYPO3 4.3 provides a Core integration of Gabriel called "Scheduler". This comes as a system extension and represents a serious improvement on Gabriel.

So if you upgrade to TYPO3 4.3, you should really drop Gabriel and use the Scheduler instead. The drawback is that you will lose the currently scheduled imports as it is not possible to transfer Gabriel information to the Scheduler (too much changed between the two tools). That should not keep you from switching though, as the Scheduler offers far more control and reporting on scheduled jobs (and Gabriel support was dropped from External Import as of version 2.0.0).

Upgrade to 0.8.0

With version 0.8.0 it became possible to define multiple external sources for a given table. This implied changing the extended TCA syntax. When upgrading to version 0.8.0 you must also change all your "external" TCA properties. All such properties have become indexed arrays. So if you had the following:

```
$TCA['tx_myext_mytable'] = array (
	'ctrl' => array (
		'title'     => ...,
		...
		'external' => array(
			'connector' => ...,
			'parameters' => array(
				...
			),
			'data' => 'xml',
			'nodetype' => 'record',
			'reference_uid' => ...,
			'priority' => 10,
			'deleteNonSynchedRecords' => 1
		)
	),
);
```

You must change it to:

```
$TCA['tx_myext_mytable'] = array (
	'ctrl' => array (
		'title'     => ...,
		...
		'external' => array(
			0 => array (
				'connector' => ...,
				'parameters' => array(
				  ...
				),
				'data' => 'xml',
				'nodetype' => 'record',
				'reference_uid' => ...,
				'priority' => 10,
				'deleteNonSynchedRecords' => 1
			)
		)
	),
);
```

The same goes for the columns definitions which should be changed from:

```
'field_name' => array (
	'exclude' => 0,
	'label' => '...',
	'config' => array (
		...
	),
	'external' => array (
		'field' => '...',
	)
),
```

to:

```
'field_name' => array (
      'exclude' => 0,
      'label' => '...',
      'config' => array (
            ...
      ),
      'external' => array (
            0 => array (
                  'field' => '...',
            )
      )
),
```

Furthermore the MM-mappings syntax has been simplified. So the following configuration:

```
'external' => array(
      0 => array(
            'MM' => array(
                  'mappings' => array(
                        'uid_foreign' => array(
                          'table' => name of foreign table,
                          'reference_field' => foreign MM key,
                          'value_field' => 'uid'
                        )
                  ),
                  'additional_fields' => array(
                        TYPO3 field name => external data field name
                  ),
                  'sorting' => 'field',
            )
      )
)
```

can be rewritten to:

```
'external' => array(
      0 => array(
            'MM' => array(
                  'mapping' => array(
                        'table' => name of foreign table,
                        'reference_field' => foreign MM key,
                        'value_field' => 'uid'
                  ),
                  'additional_fields' => array(
                        TYPO3 field name => external data field name
                  ),
                  'sorting' => 'field',
            )
      )
)
```

although the old syntax is still supported.

Also note that the "deleteNonSynchedRecords" property was deprecated in favour of the more flexible "disabledOperations" property (see Configuration below). It is still supported though.

These are expected to be the last major syntax changes which why the extension status was raised to beta.

Upgrade from 0.5.0

If you were using version 0.5.0, you may have some surprises as the extended TCA syntax has been modified for MM-relations:

- in MM mappings, the "uid_local" mapping no longer needs to be defined. Indeed the local uid is considered to be always "uid", since the whole point of this extension is to store the data into database tables that respect the TYPO3 standards.
- The "reference_field" for the "uid_foreign" mapping now uses the name of the field in the local database table. This is matched to the field name in the external data by reading to what external field that column is matched.
- The "update" property has been removed, since TCEmain deletes existing MM-relations anyway.
- The "sorting_data" field has been removed. The "sorting" property now stores what was in "sorting_data" and there are no other options for sorting.