EXT: External Import

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

Table of Contents

EXT: External Import....1
Introduction....3
Questions and support....3
Keeping the developer happy....3
Participating....3
Installation....4
Compatibility issues....4
Other requirements....6
Configuration....7
User manual....8
General considerations....8
Synchronizable tables....8
Non-synchronizable tables....9
Mapping data....10
Debugging....10
Troubleshooting....10
Process overview....11
Tutorial....11
Administration....12
User rights....12
General TCA configuration....12
Columns configuration....13
Mapping configuration....14
User functions configuration....15
MM-relations configuration....15
Developer's Guide....17
External Import API....17
User functions....17
Hooks....17
To-Do list....19
Changelog....20

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 synchronise any table manually or to defined a scheduling for all synchronisations. Automatic scheduling relies on Gabriel (TYPO3 4.2 or less) or the Scheduler (TYPO3 4.3+).

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

If you like this extension, do not hesitate to rate it. Go the Extension Repository, search for this extension, click on its title to go to the details view, then click on the "Ratings" tab and vote (note: you must be logged in to vote). Every new vote keeps the developer ticking. So just do it!

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.

Installation of Gabriel is not necessary, but is requested if you want to be able to define a schedule for automatic synchronisation. Note that you will need a recent version of Gabriel and not the one available in the TER. You can get the latest version of Gabriel from forge: http://forge.typo3.org/projects/show/extension-gabriel.

If you run TYPO3 4.3 or better, you must use the system extension Scheduler instead of Gabriel.

Compatibility issues

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 may be dropped from External Import at some point in the future).

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.

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 **Core APIs** 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 PHP time limit**: set a maximum execution time different of the default one. Setting this value to -1 preserves the default time limit. Setting to a number of seconds will change the maximum execution time to that value. This can be useful for large imports.
- **Email for reporting**: if an email address is entered here, a detailed report will be sent to this address after every automated synchronisation. Mails are not sent after synchronisations started manually from the BE module.
- **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).
- **Clean AJAX output**: this option is **enabled by default**. When a synchronisation is launched using the BE module, the import script is called using AJAX. This script may throw PHP errors or some other kind of debug output (e.g. failed SQL queries, if `SQLDebug` is enabled). This output will break the AJAX response and thus the display of the BE module. When this option is enabled, it will flush all output produced by external import, making it possible to receive a clean AJAX response, at the price of losing sight of possible errors. See "Debugging" for more details. Note: this option is ignored as of TYPO3 4.3 as the output is cleaned up anyway. The downside is that you will never see error output with TYPO3 4.3 or above.

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.

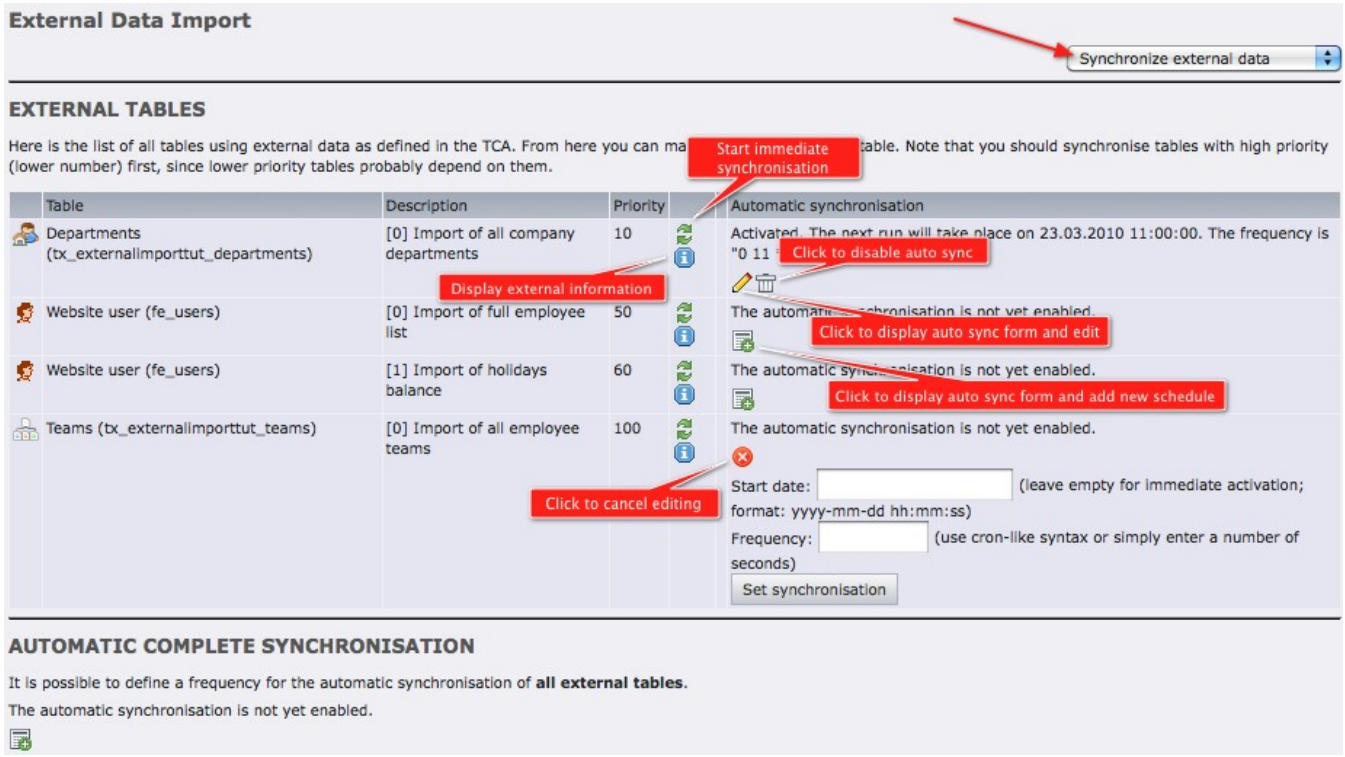
The extension can either fetch the data from some external source and receive data, passed to it by whatever script you can imagine. Fetching data from an external source goes through a standardised process. Connecting to an external source is achieved using connector services (see extension "svconnector"), 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 synchronised with the external source. Every time a synchronisation 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. 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 synchronise those tables from the BE module, nor to schedule their synchronisation. Such tables are referred to as "non-synchronizable tables". This type of action is called "pushing data".

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

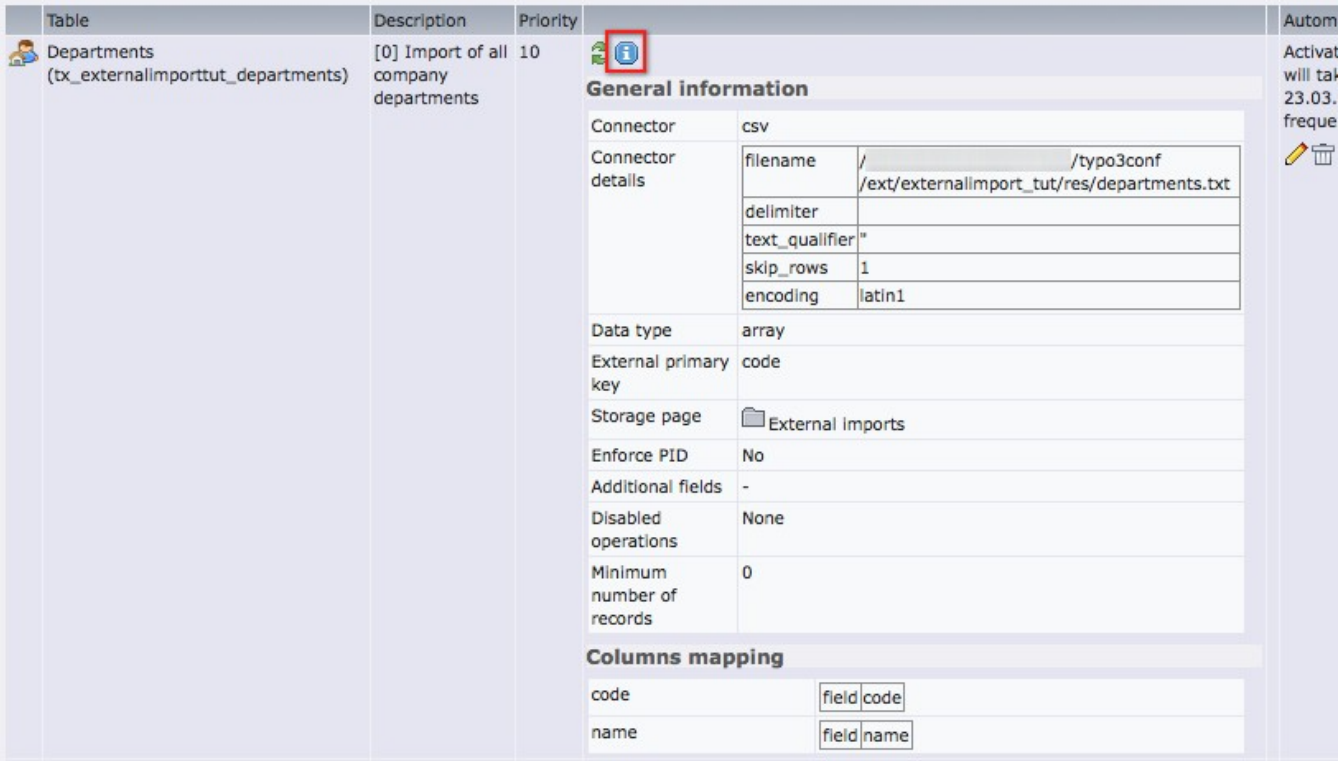
Synchronizable tables

The first function of the BE module – called "Synchronize external data" – displays a list of all synchronizable tables. The various features are summarized in the picture below. Most importantly clicking on the looping arrows icon will immediately start the synchronisation of the corresponding table. It is also possible to automate the synchronisation of each table. This process is described in more details below.



When a manual execution is finished, a report is printed out in the column next to the icon:

Clicking on the "info" icon displays all the TCA information related to the external import process:



Setting up the automatic schedule

The automatic scheduling facility relies on Gabriel (TYPO3 4.2 or less) or the Scheduler (TYPO3 4.3+) to run. On top of the normal Gabriel or 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 synchronisation is enabled or not is displayed for each table. It is possible to add or change that schedule, by clicking on the "new" or "pencil" icon respectively). This triggers the display of 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 synchronisation.

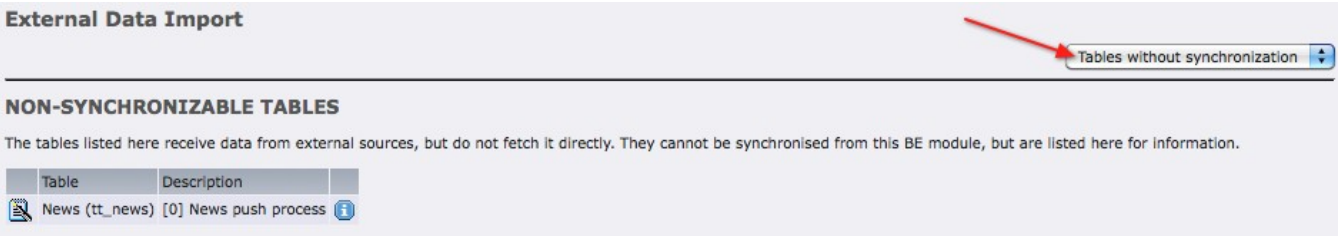
At the bottom of the screen, a schedule can be defined for **all** tables. This means that all imports will be executed one after the other, in the order of priority.

Defining a schedule is not enough. Proper user rights must also be considered. During the installation of Gabriel, you will have created a "_cli_gabriel" user (or "_cli_scheduler" for the Scheduler). This is the profile that will be used during the scheduled synchronisations, so you must make sure that this user has enough rights to perform such operations. Basically, this is what you should do:

- authorise this user to list and modify the tables that are going to be synchronised
- 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.

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.

Debugging

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

When PHP errors or other debug output are produced they corrupt the AJAX response (which is expected to be in JSON format). To avoid this corruption, the import script will flush all output before sending back its response. The downside of this method is that the error or debug output is then lost. Note that this information is hard to read anyway. You need to be able to read the raw response to view what error or debug output may be in it (for example by using the combination of Firefox and its Firebug add-on). *This is true only for TYPO3 4.1 or 4.2. As of TYPO3 4.3 the output is always flushed so there's no way errors can be seen that way.*

One particular type of debug output is the `$TYPO3_CONF_VARS[SYS][sqlDebug]` setting. If you activate this, TYPO3 will produce a full debug stack trace whenever a SQL query fails. This information will be removed if you have chosen to clean the output. One way around this is to also use an extension that writes to the devLog (e.g. "devlog") and activate `$TYPO3_CONF_VARS[SYS][enable_DLOG]`. With this failed queries output will be written to the devLog so the information is not lost.

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 synchronisation, but also all error messages logged by TCEmain, if any.

Troubleshooting

The automatic synchronisation is not being executed

You may observe that the scheduled synchronisation 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 synchronisations (like 1 minute for example). If the previous synchronisation has not finished, Gabriel will prevent the new one from taking place. The symptom is a message like "[gabriel]: 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 synchronisation never ends

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

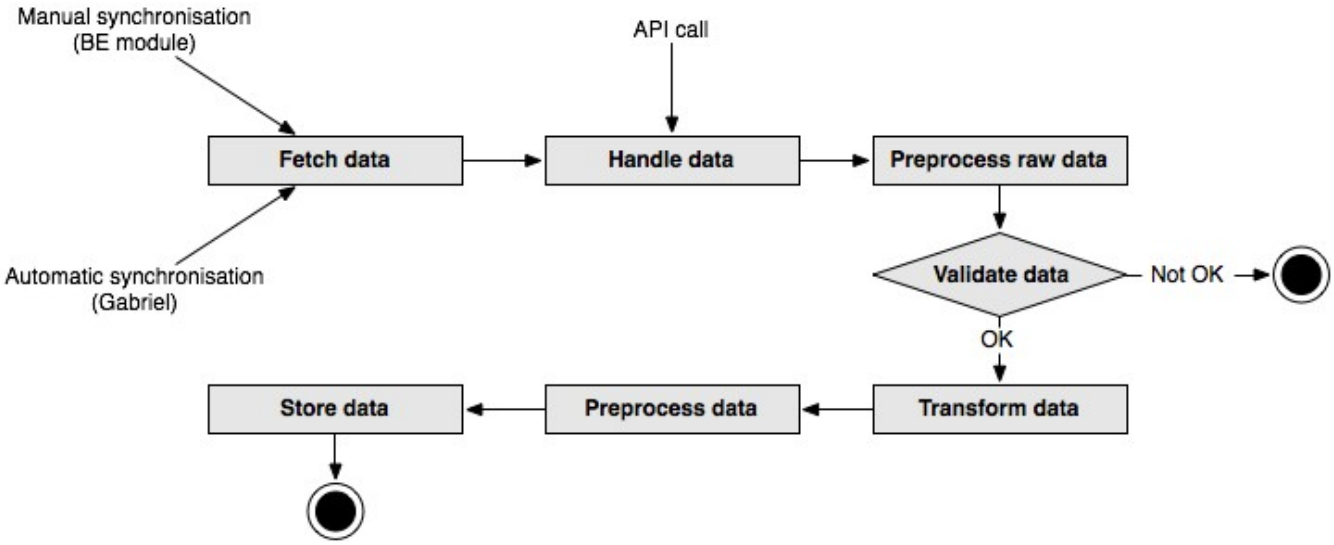
All the existing data was deleted

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 synchronisation operation (pull), data is first gathered from the external source. 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).

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 synchronised 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 synchronise it manually or define an automated synchronisation for it.

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

When a synchronisation runs automatically a check on user rights is also performed at the beginning, so that the synchronisation can be skipped entirely if the CLI user does not have modify rights on the given table. This is reported in the mail report. For more on setting up proper rights for automatic synchronisation, please refer to "Setting up the automatic schedule".

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.	External import
parameters	array	Array of parameters that must be passed to the connector service. Not used when pushing data.	External import
data	string	The format in which the data is returned by the connector service. Can be either "xml" or "array".	External import
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.	External import
reference_uid	string	Name of the column where the equivalent of a primary key for the external data is stored.	External import

Key	Datatype	Description	Scope
priority	integer	A level of priority for execution of the synchronisation. Some tables may need to be synchronised before others if foreign relations are to be established. This gives a clue to the user and a strict order for scheduled synchronisations. Not used when pushing data.	External import
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)	External import
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.	External import
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.	External import
description	string	A purely descriptive piece of text, which should help you remember what this particular synchronisation is all about. Particularly useful when a table is synchronised with multiple sources.	External import
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.	External import
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).	External import
deleteNotSynchedRecords	boolean	*This setting is deprecated. Please use **disabledOperations** instead.* Set to true if records that were not found during the synchronisation (i.e. that do not exist in the distant source anymore) should be deleted. Set to false if they should be ignored.	External import

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 that contains the data in the external source.	External import
MM	→ MM-configuration	Definition of MM-relations, see below for more details.	External import
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.	External import
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.	External import
excludedOperations	string	Comma-separated list of database operations from which the column should be excluded. Possible values are "insert" and "update".	External import
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.	External import

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.	External import
reference_field	string	Name of the field against which external values must be matched	External import
value_field	string	Name of the field to take the mapped value from. If not defined, this will default to "uid".	External import

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.
			External import

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.	External import

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'
                )
        )
    )
),
```

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.*	External import
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.	External import

Key	Datatype	Description	Scope
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.	External import
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).	External import
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.	External import

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.

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.
- **preprocessRecordset**: 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 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).

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 localised records
- Handle self-referencing tables, when inserting new records
- Look at IRRE for handling MM-relations that use additional fields or are repeated several times.

Changelog

Version	Changes:
1.1.0	Added support for additional where clause in foreign mappings
1.0.0	Added early check of user rights in automated synchronisation Added check of user rights in BE module display Added option for limiting preview/debug output size Added TCA property to exclude some fields from insert or update operations Added display of TCA external configuration in BE module
0.11.2	Fixed abusive display of validation error message
0.11.0 – 0.11.1	Added reporting by email Added automatic synchronisation per import configuration Made it possible to delete a defined automatic synchronisation Added support for the Scheduler (TYPO3 4.3+) Added a process to abort the import
0.10.0	Added fixed values and fixed value maps
0.9.0	Added "clear output" option
0.8.1	Added preprocessing hook on raw recordset
0.8.0	Introduced possibility to synchronise a table with multiple external sources Added support for user functions for transforming external data before storage Introduced API for pushing data into the external import Added hooks for preprocessing records before insert, update and delete Added property for limiting records manipulation to a given pid Added property for limiting allowed operations Cleaned up MM-mapping syntax Corrected bugs in mapping feature Updated manual with instructions for new features and some troubleshooting help Raised status to beta
0.7.0 – 0.7.x	Internal releases
0.6.2	Updated manual with missing notes about array data format
0.6.1	Added array data format handling Added user rights setup instructions in the manual
0.6.0	Introduced use of TCEmain for proper data manipulation Cleaned up extended TCA syntax Added hook for pre-processing data before storage
0.5.0	First public release