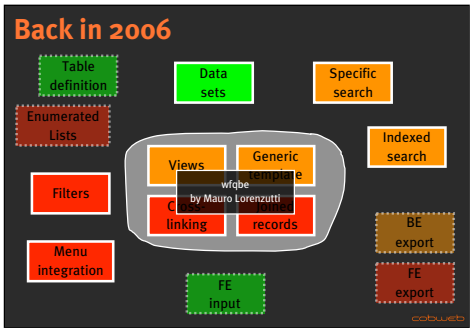
TYPO3 in the larger world

Interacting with third-party applications

François Suter

TYPO3 Developer Days '08, Hamburg, 10th May 2008

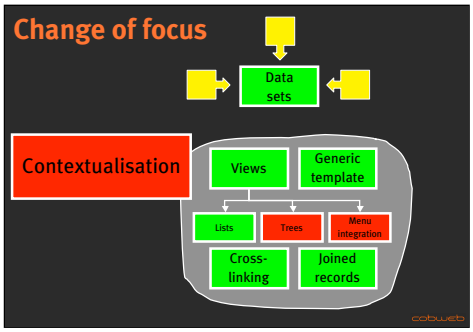
cobweb



Back at T3DD06 I held a session about the tools available or not for display any kind of data inside TYPO3 in a generic way.

I presented this jigsaw puzzle view that showed the various components and whether they existed or not. Not much has really changed since then and the main parts (highlighted in the center) have still not received a satisfying answer. Extension wfqbe by Mauro Lorenzutti has helped go a bit further. Its limitations is that it is geared towards displaying data from external tables and it cannot offer automatic handling of TYPO3 mechanisms (such as enable fields, languages overlays or versioning) when the data comes from TYPO3.

Furthermore, in the course of our projects, our focus has changed.

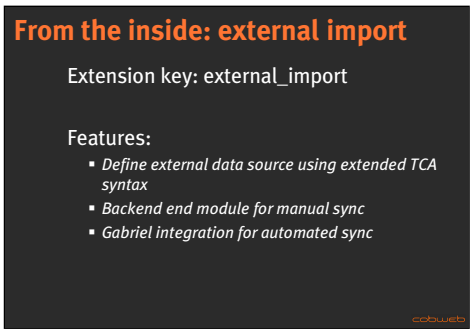


Our main goal is now to get data into TYPO3 and benefit from all the rendering possibilities of it, all the existing libraries, utility classes such as t3lib_div, tslib_content, etc.

The idea of Data Sets is to define a standard data structure that is then rendered by a set of plugins. Having a standard structure means that various extensions can provide Data Sets, and all can be rendered by the same set of plugins.

For rendering, it must be possible to have generic templates, in particular use the power of TypoScript.

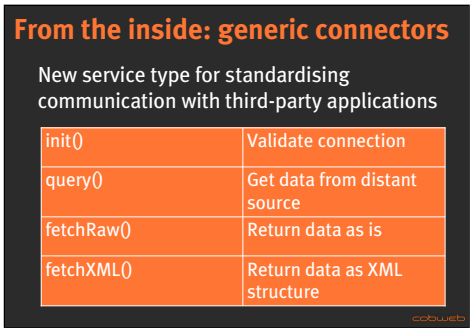
It must be possible to have different types of rendering for the data, including special views like tree or integration inside TYPO3 HMENUs.



A first goal was to get data from external source “from the inside”, that is data from third-party applications is fetched from within TYPO3 and stored in TYPO3 tables.

Tables for storage are defined with the TCA as usual, but use an extended syntax to define how to connect to the external application, which data to fetch and where and how to store it.

A backend module is available for starting manual synchronisations or to define a schedule for automated synchronisations using Gabriel.



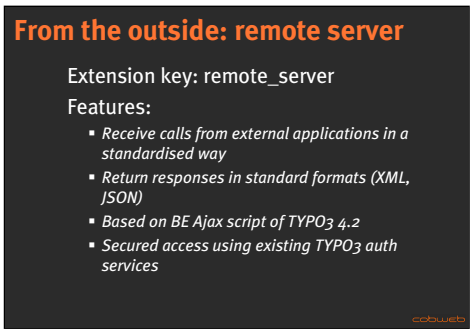
To make the external import easier to use, I introduced a new class of services called “connectors”. The idea is to provide a standard way of communicating with third-party distant applications. Connector services define an API for such work.

This API is not definite yet. Possible additions may be to return the data as a PHP array or as an object. Maybe XML is enough.

Beyond standardisation, the advantage is also reuse. For example, one specific connector service was used both for an auth service (FE users) and the external data import.

External import: extended TCA

Demo



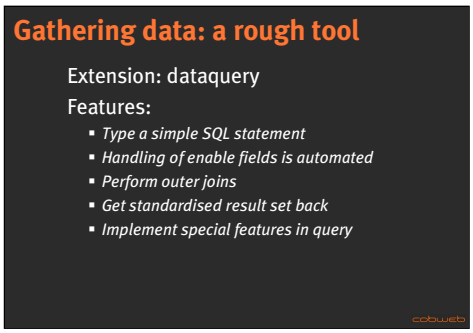
Another possibility of getting external data into TYPO3 is to push that data from the outside into TYPO3.

To do this I wanted to be able to receive remote calls from distant applications into the TYPO3 backend. The inspiration was provided by the work done by Benjamin Mack for the AJAX calls inside the TYPO3 backend. I copied part of his code and adopted the same behavior for remote calls. The main difference is that AJAX calls are made inside TYPO3 with a user session already running. In the case of remote calls, such a session does not yet exist. What's more going through the standard BE startup script (typo3/init.php) is not an option because of numerous checks that don't apply to remote calls (for example, client compatibility). So the extension includes a copy of init.php, which makes it possible to skip a number of operations. But it does go through the standard authentication process, which makes it as secure as the BE can be. I just have some trouble with challenged and super-challenged authentication.

The remote service can return its response in a number of formats, including JSON, XML and plain text. All necessary stuff from 4.2 has been copied into the extension so it should be able to run on TYPO3 < 4.2, although this is as yet untested.

Remote server

Demo



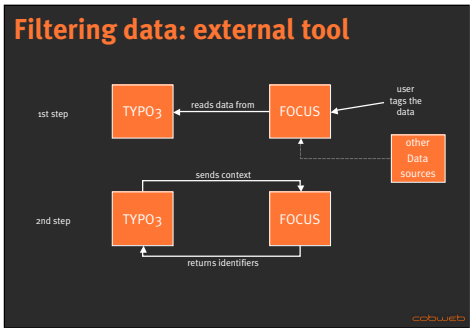
Now that we've got all this data into TYPO3, we still need to display it some way or other. This part is as yet still pretty experimental, although already used on one site in production.

This is a simple tool where you can input a SQL query and get a record set as a result. It can handle inner and outer joins. Its main advantage is that it parses the query and assembles it again. This makes it possible to add elements to query dynamically for example search criteria. It also automatically manages enable fields (although not quite completely yet). Automatic support for language overlays and workspaces is planned but not done yet. Implementation will depend on future projects.

Most importantly this simple extension was a first exploration of the definition of a standardised structure for result sets, that could be passed to various FE plug-ins capable of understanding that structure and produce output. It makes it possible to have various sources that produce datasets and various tools that produce views out of them.

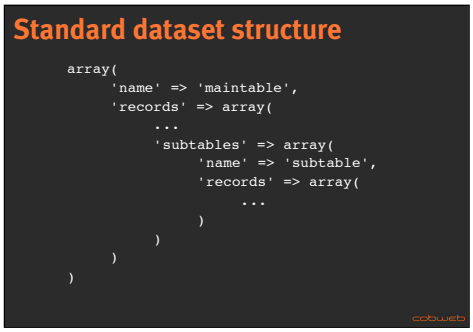
Having a specific tool to get the data makes it possible to introduce special mechanisms, for example related to "external" data. One could imagine that some fields would be marked as being "real-time" in the extended TCA syntax. For such fields we would not rely on the value stored during the last synchronisation, but fetch the "live" value directly from the third-party application (think stock level in an ERP for a e-commerce solution).

This tool is more flexible than the RECORDS object or the CONTENT object. CONTENT can query only a single table and RECORDS cannot handle outer joins very well (where clauses for joined tables are added to WHERE instead of



Among the other sources of data structures is an application we are currently developing for putting elements in relationship to each other. It works by gathering date items from one or more sources, in particular pages, content elements or DAM records from TYPO3. Some auto-tagging can take place, for example by using keywords existing in TYPO3. These elements can be tagged and thus put in relationship with each other. Tagging is not simply done using keywords, but can use rules.

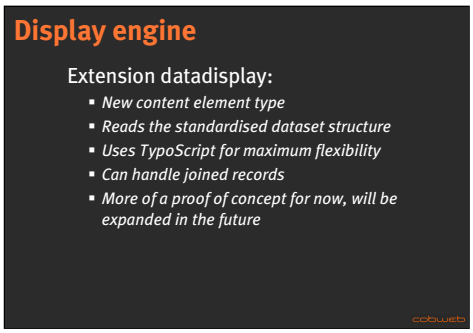
A dialogue then takes place between TYPO3 and this application (called Focus for now). TYPO3 requires data from Focus related to a certain context, which is created on the TYPO3-side and can contain specific keywords, user data, navigation data, etc. Focus then replies with a list of IDs (possibly nested or joined), which are transformed into a standardised data structure and displayed.



To be able to get data from various sources and minimise the number of tools necessary for display, it appeared necessary to define a standard dataset structure to share among all those components. This is a simple PHP array so it is mostly a question of adhering to certain conventions.

This structure can recurse indefinitely.

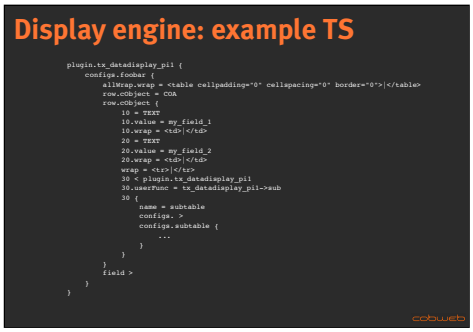
In future developments we could imagine having a helper class to create such structures.



For the moment, there is a single FE plug-in for rendering called “datadisplay”. It adds a new content element (was that a wise choice?) and the idea is to have many types for different rendering possibilities (lists, single views, tree-views, etc.). It may be that we also end up with several plug-ins. This is really more of a proof of concept for now, although it is used in a production environment already for simple lists.

The records from the standardised dataset are loaded into cObj making TypoScript available for rendering. This makes this type of rendering very flexible, although it may be a bit complicated at times. One possible evolution would be to have different rendering services which would be based on different techniques (e.g. TypoScript, HTML templates, etc.).

The plug-in can handle joined records by being called recursively in TypoScript. In this case, another userFunc is called to avoid duplication all the initialisation stuff.



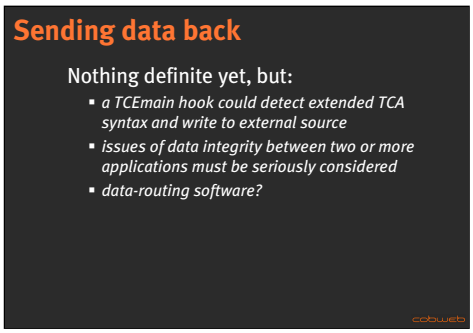
This sample template shows how data can be formatted using TypoScript. There's an "allWrap" stdWrap that will wrap the whole structure, a "row" stdWrap for each record and a "field" stdWrap for each column of the record. The "field" property can be ignored and all the rendering done at "row"-level instead, for increased flexibility.

In this example, you can see how joined records are called. The "30" property calls the plugin again, but with method sub() instead of main() to avoid repeating the initialisation process.

Those rendering templates are stored in a property called plugin.tx_datadisplay_pi1.configs. The name of the configuration is supposed to match the alias of the main table queried in the dataquery extension. This way it is possible to have many different configuration existing side by side, especially useful if you want to insert more than one datadisplay plug-in on the same page.

Display engine

Demo



It may become necessary to send back data to the third-party application. The extended TCA syntax could be used here as well. One can imagine using hooks in TCEmain to detect if a table has an external property defined and start whatever process is necessary to send the data to the third-party application.

Of course this raises issues of checking the data integrity, if the distant data was changed in the meantime. Data could be fetched and compared, but this task could maybe be devolved to a third-party application that would be use to route data back and forth and enforce any rules at the same time (i.e. who is allowed to write where and what?).