Textpattern custom fields

Custom fields are changing. Here's the nitty gritty.

Features, constraints and conventions

- Custom fields per content type: article, image, file, link, category, section, and user.
- Unlimited number of fields.
- Field names and labels are available - labels are per-language. Names are enforced to a strict convention (no spaces!) which permits their use in <txp:article_custom> tags.
- Field names can be reused in different content types but must be unique *within* any given content type.
- Fields can be of varying types - text, number, date, checkbox, radio, checkbox set, radio set, select list, etc.
- Each field can have 'default' value(s).
- Field data is stored in the most efficient table structure based on its *datatype*. Fields will no longer be part of the 'textpattern' table.
- Storage and data input is abstracted away from the user. The administrator defines the available fields and their properties; the input/output mechanisms are predetermined based on sensible defaults. Field types and their storage mechanisms are extensible (or alterable) by plugin authors.
- Fields are presented to the front end as they are now ($thisarticle) but with additional capabilities.

Database tables

The underlying information is presented to users as a simple name-value metadata store. Hence all tables begin with txp_meta*. Under the hood it is more complex with a series of tables as follows:

Table: txp_meta

Custom field definitions.

Column	Notes
id	Numeric ID. Unique across ALL custom fields (auto-increment).
name	Field name. Uniqueness only enforced per content type. Can only contain regular characters - no spaces or punctuation beyond underscore. The human-visible label is managed in the txp_lang table.
content_type	The content type to which this field belongs. E.g. article, image, category, etc. Not 4NF, but makes it easier to extend.
data_type	The SQL field type that will be used to represent this field. An internal mapping of fields (e.g. checkbox_set, text_input, date) is preset to suitable data types. This mapping is exposed via callback.
render	How to render this particular field to users of admin / public sides, e.g. a textbox, select list, date picker, etc. Will tie in with the widget library.
family	Optional grouping to indicate that this field is part of a wider set of fields. Unused in core at present.
textfilter	Which textfilter to employ when storing the content. Only applicable to certain datatypes. The usual three (textile, line breaks, or untouched) are available by default, plus plugin-defined. If used, two values are stored: one 'raw' (as input) and one 'cooked' (as processed through the textfilter).
delimiter	The delimiter character(s) applied to this field when outputting as a delimited set (only applicable to certain data types).
ordinal	The numerical order in which to display this field, in relation to other fields. Freeform, for administrator (ab)use. Unnecessary (or could be hidden) if field order is determined via drag/drop or some other UI mechanism.
created	When the field is valid from, i.e. it will only appear on articles that have Posted dates *after* the field creation date.
modified	When the field structure was last modified (default: now).

expires	When the field will no longer become available. If a field has expired, its content remains visible for existing content, but the field is no longer presented on new content created after this date.

Table: txp_meta_options

Any field types that require options (such as checkbox sets, radio sets, or select list) or those that employ constraints will have entries in this table:

- One row per option, per type; or
- One row per constraint value, per constraint.

Column	Notes
meta_id	Foreign key to the txp_meta table's ID column so we know to which field this option belongs.
type	The type of data this row holds: option or constraint.
name	The option's value as seen by the system (e.g. the 'name' field in an <input> tag). The human-visible label is managed in the txp_lang table.
ordinal	The numerical order in which to display the option, in relation to others with the same meta_id.

Tables: txp_meta_value_*

Each datatype has a corresponding table in which data of that type is stored. The main types defined by core are: text, int, tinyint, varchar, datetime. For example, all varchar data for any fields of that datatype are stored in the txp_meta_value_varchar table. These tables are created on-the-fly as needed if one doesn't exist for a field of the given type.

Column	Notes
meta_id	Foreign key to the txp_meta table's ID column so we know to which field this data belongs.
content_id	Foreign key to the content type so we know to which record (the article, image, category, user, etc) this data belongs. Note: this will make section custom fields impossible until a section ID is introduced in that table. One special value exists: -1. This denotes the (optional) default value assigned to this field. Any row that contains a '-1' entry here has its content preset to this value when input fields are rendered.
value_id	Sequential indicator to denote multiple values stored for this field against the same content. Usually, this will be 0 as only one value will be stored per field, but if dealing with real-time input or multi-user systems, other values may be employed by plugins to distinguish/subdivide values.
value	The actual value of the nominated field. The SQL datatype assigned to this column is the most appropriate one for the type of data it is intended to store (for efficiency). Usually it matches the name of the table (e.g. txp_meta_value_datetime uses a DateTime field type).
value_raw	For any fields that employ a textfilter, this field holds the as-typed, raw input. The 'value' field holds the cooked content and is thus the one that is always output when displaying field contents.

PHP objects

Four main objects exist for manipulating and mapping custom field data. In the vendors\Textpattern\Meta namespace are:

ContentType

A collection of content type mappings for custom fields. Each entry in the (iterable) array stored in this object is a place in Textpattern where custom field content may be stored.

Extensible via reference using the txp.meta > content.types callback.

DataType

Default data field mapping and its capabilities.

Each entry in the (iterable) array stored in this object comprises:

- type (string) database field type
- size (int) database field size (width). null = system defined
- textfilter (bool) whether the field can accept Textfiltered content
- options (bool) whether the field is made of a set of options (e.g. select list)
- delimited (bool) whether the field can hold delimited content
- constraints (false or array) whether the field can be subject to constraints, and which ones to apply

Extensible via reference using the txp.meta > data.types callback.

Field

A custom field definition. A representation of a row in the txp_meta table with a few additional conventions built in:

- Can be constructed with a numeric id or array('name', 'type') as argument, in which case the associated field values/options will be loaded into the object as well. Otherwise, a blank object will be instantiated ready for population.
- Field name sanitization is handled automatically.
- If a field type is altered after it has been in use, data is migrated from that field type to another, within reason. If you try and change a varchar field to an int, data loss is likely but it's up to the administrator if they wish to take the risk or migrate by hand first.
- Localised labels are managed automatically in txp_lang by prefixing the field name with txpcf_<content-type>_.
- Localised help text is managed automatically in txp_lang by prefixing the field name with txphlp_<content-type>_.

- Localised option text is managed automatically in txp_lang by prefixing the field value with txpopt_<content-type>_.
- Data is ordered by ordinal when loaded into the object.
- A render() method is available which will construct an input field of the correct type based on the 'render' database field, populated with default value(s) if appropriate. Used on the admin side to permit input into associated fields, but could also be used by entrepreneuring public plugins to permit public input into designated fields.

FieldSet

An (iterable) collection of Field objects, filtered or arranged by given criteria. The primary filter is 'type' - a collection MUST have a content type (default = article). After that you can rejig the output by:

- id, ordered by custom field number.
- name.
- field, as a backwards-compatible set of custom_N items.
- date, as a set of fields that are "valid" at a particular datestamp. Useful when setting up input fields on an article, for example, as it will only fetch the custom fields that were in use when the article was Posted (i.e. the article date falls between the field's created and expires dates).

Data workflow

- Read the id of the content being edited/viewed.
- Obtain the custom field FieldSet for the content.
- Look up that content id in the meta store (including defaults) for each field.
- Admin side:
    - Render appropriate fields, content, defaults and constraints for the collection.
    - Handle saving custom field content when the main content is saved.
- Public side:
    - Load $thisarticle with content.
    - Permit <txp:custom_field /> and <txp:if_custom_field> tags to interrogate and retrieve data.

Tag considerations

(none of this is done yet - feel free to implement!)

Public custom_field tags need to support:

- A 'type' to select which flavour of field to display (content type). Bonus points for automatically detecting the type based on context, and simply permitting overrides if necessary.
- Grabbing groups of fields to display and using break/wraptag to separate them. e.g. <txp:custom_field name="field1, field2, field3" break="p" wraptag="div" class="field-group" />. Use inputLabel() to render these if possible, so that callbacks are automatically available per field. Figure how to best utilise core features like "<+>".
- Possible ordering of groups when displaying the fields and their content.
- Ability to use the field names in <txp:article_custom> as content filters.
- Retain ability to fallback on <txp:custom_field name="any-field"> so that hacks such as obtaining the raw article image data are still possible. Should be easy if all data remains in $thisarticle.
- Possibly beef up <txp:if_custom_field> so it can compare sensible things when different field types are in use. For example:
    - Testing if a field is greater than/less than a date/value.
    - Testing if one of the options in a checkbox set is selected.
    - Testing if all of the options in a set are selected.
    - Testing multiple fields: if field A is something AND field B is something else? Or leave this to plugins?
    - …
- When the widget library is available, ability to render an appropriate input widget for each field.
- Backwards compatibility with existing user templates on upgrade.

Field handling - admin side

Custom fields cannot remain tucked away in their own Custom fields twisty on the Write panel. The big question is how to build them into the fabric of the Write panel (and other panels) and what customizations we offer to authors.

Things to consider:

- How and where will the fields appear on first load - before any customization has been done?
- Should we consider permitting block drag/drop on the Write panel? And on other panels? Each field would need to be treated as its own block and/or be permitted to be dropped into existing blocks to become part of that group. Can all this be handled on mobile devices too?
- How far would block customization go? Would we only permit some to be moved? Or everything? What about things like Title that are essential? Should they be unmovable?
- Can blocks be hidden completely or removed, e.g. 'Recent articles'. And if they are removed, how would someone get them back?
- What happens to data on save if a block is not visible? Do we need to maintain hidden fields for such content so the Save process doesn't complain at missing content fields? If so, hidden fields might be a way of maintaining field visibility so they can be reinstated via some mechanism.
- How can fields be grouped into logical blocks, and have names assigned to the groups? Think i18n. Dragging things into an already-existing group makes sense, but I don't much care for the Android concept of dragging one first-class item on top of another to pop up a dialog to permit a new group to be named. Doesn't seem intuitive to me, but maybe people are used to that?
- Field ordering: the custom fields have the notion of 'ordinal' which is the order they appear when displayed as sets, and is configured when the fields are defined. But maybe this database feature is no longer required and we can drop the concept if we permit drag/drop. Or maybe we internally keep this and use the drag/drop to silently set ordinality behind the scenes.
- Instead of drag/drop, could customization be done in some other way: on a layout builder panel, for example, like bot_wtc does? Would the Widget library help us here to permit blocks and fields to be reordered and grouped more easily?
- Should customization be permitted per user or per group? Should it be an admin-only task, for example, so an admin (Managing Editor or higher) sets up the 'workflow' for all users of each particular privilege level? Or should all users be able to set it up to their liking?
- If we lose the Custom fields block, the callback will disappear, which kills off third party CF solutions like glz_customfields. This may not be an issue if we can figure out how to migrate users somehow, because I think it's the only plugin that uses the callback. Need

to find a way to figure this out as we only officially upgrade the 10 text fields out of the gate so some "pre-parsing" plugin or a way of exporting the current glz_custom_fields field types so they can be imported into the new structure needs to be considered.
- What about custom scripts that glz_custom_fields offers? Do we consider offering the same feature in core? Maybe just a callback when each field is displayed/saved is enough?

Please put ideas, solutions and/or further concerns for discussion here...

- Need to carefully define allowed characters for field name (assuming alphanumeric Latin and underscore at this stage), especially since this has i18n implications with multi-byte characters.