Programming Reference

# Application development tools

a) for code

1) Hierarchy of classes both server and client side with main methods

2) Mark code that is a legacy and should be rewritten (how

3) Inspect all 3d party software and list where and how it is used

4) Missed parts (query builder

b) for configuration and usage (partly and only tech stuff)

1) Setup and getting started

2) Create your data model (define and import definitions)

3) Import and export data

4) Retrieve data - search and saved filters

5) Visualisation and analysis - mapping, smarty, graph

c) advanced and extensions

1) layouts

2) Fill layout with data/CMS

2.1) Static content from database

2.2) Dynamic content with smarty

2.3) Writing your own widget (extension)

* Functions and class methods
* Classes
* Class members (including properties and constants)
* Requires and includes
* Hooks (actions and filters)
* Inline comments
* File headers
* Constants
* **@ignore** Used when an element is meant only for internal use and should be skipped from parsing.
* **@since x.x.x:** Should always be 3-digit (e.g. @since 3.9.0). Exception is @since MU (3.0.0).
* **@access:** Only used for core-only functions or classes implementing “private” core APIs. If the element is private it will be output with a message stating its intention for internal use.
* **@see:** Reference to a function, method, or class that is heavily-relied on within. See the note above about inline @see tags for expected formatting.
* **@link:** URL that provides more information. This should never been used to reference another function, hook, class, or method, see @see.
* **@global:** List PHP globals that are used within the function or method, with an optional description of the global. If multiple globals are listed, they should be aligned by type, variable, and description, with each other as a group.
* **@param:** Note if the parameter is *Optional* before the description, and include a period at the end. The description should mention accepted values as well as the default. For example: *Optional. This value does something. Accepts ‘post’, ‘term’, or empty. Default empty.*
* **@return:** Should contain all possible return types, and a description for each. Use a period at the end. Note: @return void should not be used outside of the default bundled themes.

## Hierarchy of classes both server and client side with main methods

## Custom layout. Applications

Preface:

I am happy that we are nearly finished with database/definitions management/edit records parts and slowly going to practical part - creation of custom web sites based on heurist databases. On this way we have strong and weak/missed points.

Strong points are

**Customized Layout** - a scheme how to place widgets on page. It sets their initial options, applies css for particular widget or global css files for entire layout. It is possible to design and implement application any layout within couple of hours.

layout can be

1) cardinal - north/south/east/west/center resizeable panels (DH)

2) free - positions of panels are absolute and coordinates are fixed

3) template - layout manager loads html document and panels are placed into div with appropriate IDs (EN)

4) complex - layout is (nested) combination of all above approaches

Each panel can be tabbed. In other words it can contain many widgets.

**Widgets** - building blocks of web application - UI modules

We implemented nearly full set of widgets for all needs

menu and navigation: menu, expernation\_nav

search entry points: faceted search, saved searches (set of buttons or tree), dh\_search

lists: resultList and its descendants expertnation\_place and expertnation\_results

view: mapping, staticPage web content from record, smarty output

Widgets allows inheritance. So, for example, with overwriting of couple methods in resultList we get expertnation\_place and expertnation\_results.

Weak, unclear and missed points are

*Record renderer.* We have several options however there are lot of uncertainties.

1) Static content is stored in records (web content) or loaded from static html

2) Server side render (DH)

3) Smarty output

4) Client side javascript render (EN)

What method or combination of methods will be preferable depends on particular application. I’d prefer combination of 1 and 4 (as it was done for EN). In other words either user/customer can write html (with wyswyg editor) to describe particular record as he wishes or we implement a widget that render this record in fancy way.

*Missed widgets*

For good CMS we miss lot of required features

1) comments, discussion

2) polls and voting

3) multi-level catalogue (possible extension of faceted search)

4) callback forms, emails

5) payment integration

6) news line

Certainly most of them should be implemented later and gradually

*Menu. Navigation. Events and listeners*

I believe this is most vulnerable part. Widgets interact via global event listeners or widgets can be linked via event listeners of particular widgets. Alas we cannot predict the complete action scheme for every website/application. Moreover at the moment we have only one reliable/general chain of events/actions search->result list->record renderer.

I am not against to store ui configuration in records. They may be described and stored either as json data in layout file (as it currently done) or in form of heurist records. The problem is that we don’t have enough knowlege/experience what to store - what kind of formal data (configuration options for widget) will be stored in these records (or in layout). Definitions of links between widgets in the layout file will complicate it and will not cover all cases. So it would be better to implement special main control widget for every new heurist application.

So, at the moment it would be better to implement special menu and navigation widgets (or one widget). It should performs two tasks: loads/init menu, links all widgets in app

a) load menu items either from html (as for current heuris ui) or from json or from records or from combination of these sources.

b) navigation - links menu and widgets, links widgets to each other via events.

For every project we will create extension of this widget. As I said above, at the moment we cannot formalize all possible cases. Gradually we will see rules and patterns, so we will be able to implement record type that will describe it and end customer will be able to describe/configure his website with editing of heurist record.

* How long you think it would take to generate their requested website based on modifications to a Beyond1914 approach. IMPORTANT: please give me a range, and make sure the maximum really will cover getting such a site fully working, as I do not want to under-quote on the job. If we do it this way I will of course be padding the cost so that it is not jsut paying for itself but also generating us some additional funds;

Fo

* How long do you think it would take to set up a generic methodology for generating websites which would generate things like Beyond 1914 and this website (obviously DH has more specific functionality built into the map layer connections and the map popup, which is always going to have to be programmed separately). Again min and max, but make sure max is a realistic estimate of the time allowing for the unexpected ;-}

1) html template and css that fit specific styles guidelines

2) menu/navigation widget (similar to expernation\_nav)

Note: in EN expernation\_nav has different functionalities - navigation menu and result renderer (timeline page). It would be better to separate it to 2 widgets.

So, this widget must fulfill 2 tasks

a) initialize main menu. b) link all widgets in application