The GemsTracker Core

GemsTracker is a library that extends the Zend Framework libraries, but also supplies a standard new project template.

A new project

GemsTracker supplies a default **new_project** installation that allows you to quickly setup a standard project. The package contains these directories:

- `application` Project specific code and settings
  - `classes` Project specific classes
  - `configs` Project specific Zend Framework, GemsTracker and database configuration files
  - `controllers` (Optional) Project specific Zend FW controllers
  - `events` (Optional) Project specific track engine events
  - `languages` (Optional) Overrule the default translations
  - `layouts` (Optional) Project specific Zend FW layouts
  - `snippets` (Optional) Project specific code that generates Html
  - `views` (Optional) Project specific Zend FW views
- `htdocs` The webroot of the project
  - `gems` Default GemsTracker css, images and jQuery
- `library` The location for the GemsTracker library and optionally the Zend FrameWork?
  - `Gems` The GemsTracker library directory, link to a stable release or to our development branch
- `var` A writable directory for files that can change after installation
  - `cache` Cache and sessions
  - `logs` Error logs
  - `settings` File locks and other installation dependent settings
  - `uploads` Pdf's ant other uploadable files

The library

GemsTracker is implemented as a separate library. You should either link to a stable tagged release and not change the code, or if you are involved in the project you can develop against the GemsTracker trunk.

The directory structure is based on the standard Zend Framework project directory structure with some extensions.

- `classes` The main project code
- `configs` Database definitions for project types
- `controllers` Zend FW Controllers stubs that can be overruled by a project
- `docs` Some documentation
- `languages` Default translations (currently Dutch and English)
- `layouts` GemsTracker default Zend FW layouts
- `snippets` Code that generates reuseable Html fragments / snippets

- `views` GemsTracker default Zend FW views (but very empty for a Zend project)

GemsTracker builds on the Zend Framework but does not follow in slavishly. Some changes are caused by the requirement that standard code can be overruled at the project level. Other changes were made because we want to extend the Zend Framework (but did not yet get round to the extensive documentation and unit testing required by the framework). These two types of extensions are easily distinguished by their parent directories within the classes directory:

- `Gems` The core of GemsTracker
- `MUtil` Possible extensions to Zend, should not be Gems specific
- `ZFDebug` A Google build debug extension for Zend
- `Zend` Those extreme exceptions where we really had to fix a Zend bug (currently only one)
- `GemsEscort.php` The Zend Bootstrap object for GemsTracker. This object handles initialization, layout and security. The GemsEscort? object must be overloaded at the project level.

For Gems and MUtil we generated API documentation, but here we will describe their effect on Zend Framework development.

The MUtil extensions

MUtil stand for MagnaFacta? Utilities. MagnaFacta is one of the companies hired to develop GemsTracker.

When a MUtil directory has the same name as an existing Zend directory it usually concerns a simple extension of that Zend component. E.g. the `Markup` directory contains an extension that renders e.g. BB or Wiki code to flat text. Always useful if you want to include a text version for smartphone with an HTML e-mail message. The Potemkin `Translate` adapter allows you to act as if there is a translator, without defining any.

However some other default extensions are more important:

- `Application` Extends the Zend `Bootstrap` object to an `Escort` object that allows .NET like event function use during the whole application request cycle.
- `Controller` Extends the standard `Action` to include the use of new `Html`, `Models` and `Snippets` components
- `Form` Provided extensions for using both tabbed and tabled forms, improved focus handling and use of the `Html` component.

Other directories extend the framework. These can be divided in two sets, high-level and low-level. We start with the low level extensions. The low-level do not adapt the Zend Framework, but enable the other extensions:

- `Lazy` Delayed execution, think callable with parameters and repetition
- `Parser` An SQL parser for SQL script processing
- `Ra` Array and parameter processing functions
- `Registry` Automated object parameter loading using a registry
- `Util` Programmer extendable functionality (e.g. for factory functions in Ra, Html and Snippets)

The high level packages are the ones that make a GemsTracker a non standard Zend Application:

- `Html` Simple extendable HTML classes using `Lazy` repetition and Ra parameters
- `Model` The M in Model View Controller. Describe labeling, display formatting, validating, etc... for non-db or db-based data sets
- `Snippets` Quickly create reusable HTML fragments from code. Use of `Html`, `Model` and `Registry` is prepared but not required

The Gems extension

Most components in `Gems` extend a `Zend` or `MUtil` component adding functionality specific to GemsTracker, without adding any significant changes to the existing workings of those components. The exceptions fall in two broad categories: those that enable extensions and changes at the project level and those that form the core of GemsTracker.

1. **Project level extensibility**
   GemsTracker tries to give a programmer as much freedom as necessary to change the core workings at the project level, without the programmer having to change or copy the core GemsTracker library.
   - `Default` Standard controllers, for easy overloading of controllers at the project level
   - `Loader` Allow 'mirrored' project level objects to be loaded instead of Gems level objects
   - `Project` Choose multi-layout, multi-organization, logging level and track types
2. **GemsTracker functionality**
   - `Communication` (SOAP) communication with external applications
   - `Event` Survey or track specific code triggered before or after a survey is taken
   - `Export` Data export for scientific analysis
   - `Menu` The application menu, of course adaptable at the project level
   - `Tracker` **THE CORE**: integration with survey sources, track engines and token display
   - `User` Extensible, role based user authentication and authorization