FACT-Finder®
Europe's leading conversion engine

FACT-Finder

PHP Framework

FACT-Finder®
Europe's leading conversion engine

FACT-Finder Development

eMail: support@fact-finder.com
Phone: +49 (0) 7231/12597-701

Version: 2.4

Last Update: 13.03.2012

Index

Introduction

About this document

This is the Technical Documentation for version 2.4 of the FACT-Finder PHP Framework. It explains how this Framework is constructed, how it works and how it is best used.

Basic principle

The FACT-Finder PHP Framework (referred to simply as the Framework in the rest of this document) has been created to take care of the basic tasks relating to the integration of FACT-Finder in a PHP-based system and to provide an easy-to-use API to access the functionality offered by the FACT-Finder system. The Framework does not provide complete access to all of the FACT-Finder features, and is simply intended as a means of supporting the integration of the FACT-Finder system. In this respect the Framework can be seen as a source of reusable sample code.

The core of the Framework is the library, through which the major proportion of FACT-Finder functions are used. However, it also provides a demonstration implementation that demonstrates the search process using the library and sample data.

Prerequisites

In order for you to be able to use the Framework in your web shop, you must have already configured a FACT-Finder Search Environment. To set up this search environment, we need the product data from your web shop. In most cases the corresponding shop system should provide a tool to do this. Otherwise, you will have to create and make available the data export by other means. You can find a description of the format in which you need to provide the data in our "*FACT-Finder Data Export*" documentation, which is available on request or can be downloaded from the Customer Portal.

Implementing the Framework in a web shop will require knowledge of PHP and a basic understanding of the concepts of object-oriented programming.

The Framework itself requires PHP version 5.1.2 or later with the SimpleXml and cURL extensions enabled. The iconv extension would also be very useful, but is actually optional and is only really needed if your web pages do not use the UTF-8 or ISO-8859-1 encodings.

Conventions

The Framework uses the conventions of the Zend Framework, especially in relation to the use of pseudo-namespaces in the naming of classes. A class name has to match the directory path, with an underscore character used to replace the path character. Underscores are otherwise not permitted in class names. The base directory for the classes is `FACTFinder`. As a result, names of all the classes in the library start with "`FACTFinder`".

Example: The class "`FACTFinder_Abstract_Adapter`" is located in the file "`FACTFinder/Abstract/Adapter.php`".

If shop-specific modifications are required, these should be created in classes that are located in a directory called "`FACTFinderCustom`", which exists in parallel to the "`FACTFinder`" directory. The class names here should also follow the same naming conventions.

For example the class "`FACTFinderCustom_Result`" is located in the file "`FACTFinderCustom/Result.php`" and overrides the class "`FACTFinder_Result`" which is located in the file "`FACTFinder/Result.php`".

The reasons for this convention are explained in the section "*Auto-loading mechanism*" of the "*Advanced use*" chapter.

Quick start

This chapter explains how to modify the Framework demonstration code so that it can be used with your data in your search environment. You can completely skip this chapter if you are looking for detailed information on how the Framework works.

The first step is that you must copy the Framework, including the `demo` directory, into the documents directory of the web server. At this point it should be possible to access the `index.php` file of the demo installation and the demo search environment should work with the preconfigured sample data. If this is not the case, there is a problem with the server installation or PHP installation.

Structure

The demonstration implementation comprises the files for the web page (CSS, JavaScript and image files) located in the "`files`" directory. The "`i18n`" directory contains two language files. This allows the pages to be displayed in either language, depending on which language is set in the configuration. There is also a "`templates`" directory, which contains the templates used to generate the HTML pages. These are accessed by the "`HtmlGenerator`" class, which is located in the "`userdata`" directory. This directory also contains the configuration file "`local.config.xml`".

The demo directory itself contains the index.php page which initialises the search engine and the proxy scripts "`suggest.php`" and "`scic.php`". These are used for Ajax requests, which cannot send requests directly to FACT-Finder due to browser cross-site scripting preventions.

Configuration

Access credentials

As soon as a FACT-Finder search environment has been set up for you, you should receive the access credentials for it. Enter these into the "`local.config.xml`" file located in the directory "`demo/userdata`".

Normally you only have to change the values in the "`search`" area. A more detailed explanation of the configuration and the values that you need to change can be found in the "Configuration" section of the "Advanced use" chapter.

It may be that your contact at Omikron simply provides you with a link to the Management Interface in the FACT-Finder search environment. However, you can also determine the values you need from this link.

Example:

You receive the following URL:

```
http://search123.fact-finder.de:8080/YourSearch/Management.ff
```

This URL provides you with the following values:

- Address: `search123.fact-finder.de`
- Context: `YourSearch`
- Port: `8080` (if no port is specified, it is assumed to be port 8080)

The Management Interface also gives you an overview of the available channels, one of which you can select and enter as the value for "channel".

Alongside the access credentials, you should also have received information about what authentication type is to be used. When using the most secure type "advanced" you also need the prefix and postfix that are to be used.

Field names

In order to allow the demo version to operate with the data in your search environment, you also have to adapt the field names. A list of field names in your search environment can be found by looking at the "Relevance" section of the Configuration page in the FACT-Finder Management Interface.

The field names should then be entered in the file "`demo/templates/fieldnamesConfig.php`". This file contains simple PHP code. The left-hand part contains the required information (as array keys) and the right-hand part contains the field names, which contain the corresponding values. If a given type of information cannot be supplied by your data, you can leave the corresponding field name blank.

Conclusion

Following these adjustments, this demonstration application should work with your data. If this is not working, please do not hesitate to contact our PHP developers.

If the demonstration application is now working with your data, you have the following options for integrating the search function into your web shop:

- The demonstration application should provide sufficient sample code to allow FACT-Finder to be implemented completely into your web shop, given a few adjustments. Use the Framework in your shop as it is used in "`index.php`" and "`userdata/HtmlGenerator.php`" in order to access the data from FACT-Finder and then display it in your shop system.

- If you want to avoid having to write too much of your own code, you are welcome to reuse the code provided by the demo. Simply adapt the layout and design according to your own needs and desires. A very simple method then of integrating the search results into your shop is to rename `index.php` as "`search.php`", for example, and then link to it from your shop as appropriate.

  Please remember that this code is provided simply as an example. You have the responsibility for ensuring that all of the modules you have licensed are used correctly in the shop system. You are welcome to contact your Project Manager at Omikron to help with the handover of the implementation.

- If you find the Framework is not suitable for your needs, you can also use the FACT-Finder interfaces with your own code.

Alternative Demoshop

The framework also contains another directory `demo2`. It contains a second, analogue demoshop whose structure differs from the one in `demo`. The latter is based on an `HtmlGenerator`, which constructs the page from `.phtml` template files, whereas the alternative demoshop uses `render` functions for the individual page elements. If your shop rather resembles this structure, you should base your integration of FACT-Finder on the second demoshop.

Advanced use

This chapter explains how you can use, modify and extend the FACT-Finder Framework. It describes how to configure the application using the API and explains the internal functionality of the API.

Auto-Loading Mechanism

One of the core classes is the `FACTFinder_Loader`. Since this class is used very frequently, it can also be accessed using the alias `FF`. (This has been done by creating a final class with the name `FF`, which inherits all of its methods and data from `FACTFinder_Loader` but which does not override any methods or create any of its own.

The class is responsible for loading the correct file based on the class name that is called. For this reason it is important that the naming convention described in the Introduction is upheld. The class is registered as an `auto_loader` so that no includes are required for the classes in the Framework.

The class is also used to instantiate objects dynamically. This means that you can replace the use of a class with your own code. For example, the code creates an object of the type "`record`": `$record = FF::getInstance("record");`. By default, the loader locates classes using the file name `FACTFinder/Record.php` and the class `FACTFinder_Record` that it contains. If you want to override this class, you simply create the class `FACTFinderCustomer_Record` in the file `FACTFinderCustomer/Record.php`. We recommend that you inherit the original class `FACTFinder_Record`.

In addition, you can use this class to ensure that objects of a given class are instantiated only once (known as singletons). You do this using the static method "`getSingleton`" which is used in a similar way to "`getInstance`".

Configuration

The configuration allows you to make major changes to the API behaviour and therefore resolve a large number of problems. By default, the configuration values are loaded from an XML file using the `Zend_Config` class. The `FACTFinder_Configuration` class is used as a wrapper for the `Zend_Config` object and returns the individual values according to the implemented interface `FACTFinder_Abstract_Configuration`. This means that it is simple to create another implementation of the configuration.

In the configuration XML file, all the values are contained in the "`production`" XML tag. The reason for this is that the Zend configuration is able to group and inherit values. More information can be found in the Zend Framework Documentation for the class `Zend_Config_Xml`.

Meaning and usage of the configuration values

The next few sections describe the individual values in `default.config.xml`. The methods of the `FACTFinder_Abstract_Configuration` interface are constructed in the same way and should therefore be self-explanatory. An example configuration can be found in the `demo/userdata` directory of the Framework.

General settings

version: Contains the version number of the Framework. This can be used for verification and output via the API. This value should not be changed.

revision: Contains the revision number of the most recent internal SVN commit. This is a more accurate version number, even if the gaps between revision numbers are quite large. This value should not be changed.

debug: Can be set to "`true`" or "`false`". You can use this to control whether FACT-Finder generates output for debugging. Currently no debug output is available since no debugging tool is being used.

Access credentials

The settings in the XML "`search`" section specifies which database is used for the search queries.

address: Contains the address (FQDN or IP address) of the search server. Do not add any other information such as "`http`" or paths.

port: The port used by the FACT-Finder service on the corresponding server is entered here.

protocol: This value specifies the protocol that is used to send and receive queries. Currently only "`http`" and "`https`" are available.

auth: This value contains information for authenticating with the FACT-Finder service.

> **user**: Username for logging on to the FACT-Finder service.
>
> **password**: Password for logging on to the FACT-Finder service.
>
> **type**: FACT-Finder supports a variety of authentication options. Until FACT-Finder version 6.4, only "`http`" was available. Since then we also have "`advanced`" and "`simple`" (which has rendered "`http`" superfluous). The general Integration Documentation explains the differences between the authentication options.

advancedPrefix / advancedPostfix: These settings are needed when using the authentication option "`advanced`". They are ignored when other authentication options are used. Their purpose is to increase security and the level of encryption.

context: This value sets the FACT-Finder application name. If you simply received a FACT-Finder URL from your Omikron partner, this value is the part of the path following the server address.

For example, in the URL `http://search123.fact-finder.de/YourSearch/Management.ff` the application name is "`YourSearch`" and this is therefore entered for "`context`".

channel: FACT-Finder supports multiple channels, with each web shop normally being assigned a discrete channel. You can enter the corresponding channel name here. Click "Configuration" in the FACT-Finder user interface to view all of the channels that have been registered in your FACT-Finder system.

language: Select the required language here. This value is sent to FACT-Finder to control the language used to generate certain text elements. This value can also be used for internationalisation (translation) of the search facility, which is the case in the demo application.

Parameter settings

The parameter settings are used to associate different parameters in the web shop and FACT-Finder, and to add or remove parameters. The "`params`" XML tag contains two sub-sections: "`server`" and "`client`". The "`server`" section contains the settings for the parameters that are sent to the server, while the "client" section contains settings for the parameters that are reused in the web shop. There are three different XML tags, which can be used as often as needed in these sections:

ignore: Parameters should be ignored in the corresponding area. The XML attribute "`name`" is required and contains the name of the corresponding parameter.

Example: `<ignore name="channel" />` ignores the "`channel`" parameter.

mapping: The mapping tag allows you to link FACT-Finder parameters with parameters in your web shop. In general the mapping entries complement the entries from each side (see example). The direction of the transformation is specified by the XML attributes "`from`" and "`to`".

Example: `<mapping from="query" to="keywords" />` was entered in the "`client`" section and ensures that the "`query`" parameter generated by FACT-Finder is transformed into the "`keywords`" parameter used in the shop. At the same time, the "server" section should contain an XML tag `<mapping from="keywords" to="query" />` that transforms "`keywords`" back to "`query`".

required: This setting allows you to force the use of specific parameters. The parameter names are specified using the "`name`" XML attribute. If this parameter is not supplied in the corresponding area, it is added along with the default value specified in the "`default`" attribute.

It is not necessary to force the use of parameters that are already set by the API (e.g. "`format=xml`" for the XML interface or "`username=xyz`" for authentication).

Example: `<required name="filterCustomerId" default="0" />` would set the parameter "`filterCustomerId=0`" if this parameter is otherwise not provided.

Encoding

The setting applied by the "encoding" XML tag is used by the Framework to ensure that the data and values are correctly encoded. This tag takes three values, all of which are mandatory. If the PHP installation supports iconv you can configure any of the available encodings . Otherwise, only the values "`UTF-8`" and "`ISO-8859-1`" are supported.

pageContent: Defines the encoding of the web shop HTML pages.

pageURI: Defines the encoding of the web shop URLs. This normally is the same as the encoding of the pages.

serverURI: Defines the encoding of the FACT-Finder server. If you are unsure of this setting, please check with FACT-Finder Support or your contact at Omikron.

Other values

You can also enter your own settings in the configuration file and access these from the application by calling the "`getCustomValue($name)`" method of the `Configuration` object. Nested structures are not permitted for these values, however.

Basic functionality

Main classes

The key classes are the `Adapter` class, which generates the result objects, `DataProvider`, which is responsible for obtaining the raw data, `ParametersParser`, which is responsible for generating the URLs, `EncodingHandler`, which encodes the data and `Configuration`, which provides the values of the individual configuration settings.

Figure 1 shows the dependencies between the various classes and objects. These dependencies reveal the following fixed sequence that should be followed when instantiating the core objects:

1. Configuration (has no dependencies)
2. EncodingHandler (requires Configuration for access to encoding settings)
3. ParamsParser (requires Configuration for access to parameter settings and EncodingHandler to encode the parameter values correctly)

4. DataProvider (requires Configuration for access to the FACT-Finder access credentials and ParametersParser for access to the server parameters)
5. Adapter (requires EncodingHandler in order to encode data correctly, ParametersParser in order to generate correct system URLs and DataProvider in order to get access to the raw data that can then be provided to the system in the corresponding objects)

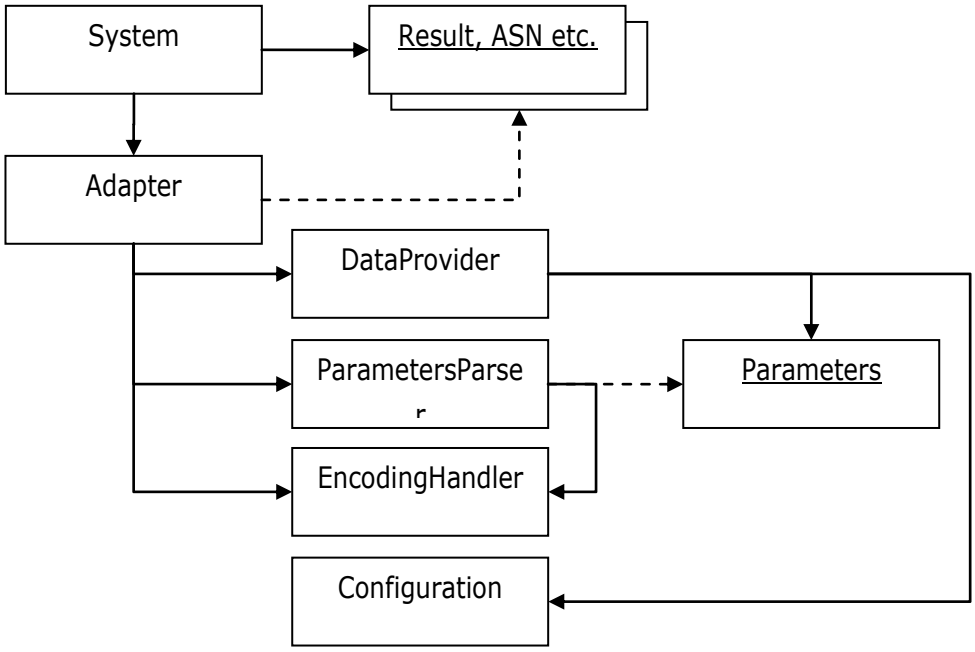

Figure 1: Class dependencies in the Framework

Solid lines indicate a dependency while dashed lines show where objects (underlined names) are generated.

The place at which the objects are generated basically depends on the system into which you wish to integrate FACT-Finder. In the Framework demonstration application the objects are instantiated directly in `index.php`.

Access to the actual results data

The Framework is constructed so that the required data is not generated until it is actually needed (a technique known as lazy loading). As soon as the corresponding Adapter class and therefore the remaining required classes have been instantiated, the Adapter class can start to deliver results. It is not necessary to start a search process explicitly – that is done in the background, for example, as soon as the `getResult()` method is called.

Whenever logical, the results classes inherit methods from the `ArrayIterator` and `IteratorAggregate` PHP classes as appropriate. This makes it possible to iterate through the

object of the corresponding class quite easily and thus access child objects during each iteration. The code documentation for these classes explains what data can be returned in each iteration.

Abstraction, inheritance and version management

Adapter

Since FACT-Finder offers other interfaces for various other functionality, there are corresponding other adapters that allow the data to be provided through the corresponding interface depending on the version of FACT-Finder.

The general adapter `FACTFinder_Abstract_Adapter` ensures a connection to `DataProvider`, `ParametersParser` and `EncodingHandler`. The abstracted functional adapters in the `FACTFinder/Abstract` directory then define the methods that allow you to access the corresponding data. These classes take care of the lazy loading, so that the generating methods simply have to be implemented for the appropriate data interface and version.

The usable adapters that understand the specific characteristics of the corresponding versions are then made available according to the version of FACT-Finder that is being used.

DataProvider

The abstract DataProvider class `FACTFinder_Abstract_DataProvider` also simply takes care of dependencies with other classes. The class enables the data to be accessed in a variety of ways, depending on which data interfaces are being used. Currently only `FACTFinder_Http_DataProvider` and `FACTFinder_Http_ParallelDataProvider` are implemented, but it is conceivable that another DataProvider could be created for the WebService.

These two DataProviders differ in how they process requests to the FACT-Finder server. The simple `DataProvider` issues an individual (sequential) request for every adapter, whereas the `ParallelDataProvider` collects the requests from all adapters and sends them out in parallel. Thus, the usage of both DataProviders differs slightly. The simple (sequential) `DataProvider` is initialized and used as follows:

```
... // Initialize $paramsParser, $config and $encodingHandler
$dataProvider_s      = FF::getInstance('http/dataProvider',
                $paramsParser->getServerRequestParams(), $config);
$dataProvider_tc     = FF::getInstance('http/dataProvider',
                $paramsParser->getServerRequestParams(), $config);

$searchAdapter       = FF::getInstance('xml67/searchAdapter',
                $dataProvider_s, $paramsParser, $encodingHandler);
$tagCloudAdapter     = FF::getInstance('xml67/tagCloudAdapter',
                $dataProvider_tc, $paramsParser, $encodingHandler);

$result              = $searchAdapter->getResult();
$tagCloud            = $tagCloudAdapter->getTagCloud();
... // Read and output $result and $tagCloud
```

Internally, both `getResult()` and `getTagCloud()` issue a request to the FACT-Finder server through their corresponding `DataProvider`. When using the `ParallelDataProvider` you have to trigger the requests explicitly. In exchange, they will be sent out in parallel, which might save a lot of waiting time, especially when many adapters are in use:

```
... // Initialize $paramsParser, $config and $encodingHandler
$dataProvider_s      = FACTFinder_Http_ParallelDataProvider::
                getDataProvider($paramsParser->getServerRequestParams(),
                $config);
$dataProvider_tc     = FACTFinder_Http_ParallelDataProvider::
                getDataProvider($paramsParser->getServerRequestParams(),
                $config);

$searchAdapter       = FF::getInstance('xml67/searchAdapter',
                $dataProvider_s, $paramsParser, $encodingHandler);
$tagCloudAdapter     = FF::getInstance('xml67/tagCloudAdapter',
                $dataProvider_tc, $paramsParser, $encodingHandler);

FACTFinder_Http_ParallelDataProvider::loadAllData();

$result              = $searchAdapter->getResult();
$tagCloud            = $tagCloudAdapter->getTagCloud();
... // Read and output $result and $tagCloud
```

Here, calling `loadAllData()` triggers the parallel requests, so that the data is already available when calling `getResult()` for instance. Also, notice the difference in initializing both DataProviders.

Versions and interfaces

Currently the Framework differentiates between the interfaces `Http`, `Xml64` and `Xml65`. There is no real logical separation here, it is more simply to do with the naming of the classes contained in each version. Despite this, there is a certain logical structure even though the concepts of interface and FACT-Finder version are mixed up.

The `Http` directory contains the classes that work via HTTP independently of the data format (DataProvider) or that support a proprietary or no data format.

The XML interfaces expect XML, as the name suggests, and can therefore use the HTTP DataProvider class for the request to FACT-Finder, which will return XML provided the corresponding parameter is set. The task of parsing the XML data is then up to the appropriate Adapter class.

Since the FACT-Finder interfaces (may) change from one version to the next, in certain circumstances there are new Adapter classes for the same interface, which have been adapted with respect to specific modifications. Normally these Adapter classes simply inherit their methods, overriding those that need it as appropriate.

Functionality of the individual classes

The functionality of each class, including the methods that are exposed and what they do, can be found from the documentation that has been generated and is located in the `doc/phpdoc/` directory. To read this documentation, simply open up the `index.html` in that path in your web browser.

FAQ

Questions relating to the Framework

How can the Framework be adapted to reflect modified code or custom functionality?

You can override almost any class so that this is replaced by your own version across the system. More information can be found in the section "*Auto-Loading Mechanism*".

Not all of our licensed FACT-Finder modules are covered by the Framework, what should we do?

In general, the licensee is responsible for implementing FACT-Finder in their web shop system. This Framework is intended as a tool and to prevent users having to reinvent the wheel. We will gladly respond to requests to expand the Framework in some situations, or you can modify it, send us the code and contribute to the further development of the module.

Other questions?

If you have any other questions, please call: +49 (0)7231 12597 701 or send a mail to support@fact-finder.com. A qualified assistant will be happy to help you.