FACT-Finder®
Europe's leading conversion engine

FACT-Finder

PHP Framework

FACT-Finder®
Europe's leading conversion engine

FACT-Finder Entwicklung

eMail: support@fact-finder.de
Tel.: 07231/12597-701

Version: 2.4

Aktualisierungsdatum: 13.03.2012

Inhaltsverzeichnis

Einleitung

Über dieses Dokument

Dies ist eine technische Dokumentation des FACT-Finder PHP Frameworks in der Version 2.4. Hier soll nun erklärt werden, wie dieses Framework aufgebaut ist, wie es funktioniert und wie man es am besten verwendet.

Grundgedanke

Das FACT-Finder PHP Framework (nachfolgend nur noch „Framework" genannt) wurde erstellt, um die grundlegenden Arbeiten bei einer Integration von FACT-Finder in ein PHP System zu ersparen und über eine komfortable API auf die Funktionalitäten von FACT-Finder zugreifen zu können. Das Framework garantiert keine vollständige Abdeckung aller FACT-Finder Features und dient lediglich als Unterstützung bei der Integration von FACT-Finder. Das Framework gilt somit als wiederverwendbarer Beispielcode.

Kern des Frameworks ist die Library, über welche große Teile der FACT-Finder Funktionen verwendet werden können. Daneben gibt es noch eine Demoimplementierung, welche anhand der Library die Suche über Demodaten demonstriert.

Voraussetzungen

Damit Sie das Framework in Ihrem Shop verwenden können, muss eine FACT-Finder Suchumgebung vorhanden sein. Damit die Suchumgebung eingerichtet werden kann, benötigen wir die Produktdaten Ihres Shops. In den meisten Fällen müsste es ein Tool im entsprechenden System geben. Ansonsten müssen Sie den Export auf einem anderen Weg zur Verfügung stellen. Das optimale Format für uns finden Sie im Dokument „*FACT-Finder Data Export*", welches Sie ebenfalls von uns anfordern bzw. über das Kundenportal herunterladen können.

Um das Framework in einen Shop implementieren zu können, sind Kenntnisse in PHP und Grundkenntnisse der objektorientierten Programmierung notwendig.

Das Framework selbst benötigt PHP ab der Version 5.1.2 mit den aktivierten Erweiterungen `SimpleXml` und `cURL`. Optimal wäre auch die `iconv`-Erweiterung, diese ist aber optional und ist nur dann notwendig, wenn auf der Seite nicht UTF-8 oder ISO-8859-1 verwendet wird.

Konventionen

Das Framework lehnt sich in den Konventionen an das Zend Framework an, vor allem was die Namespaceähnliche Benennung der Klassen anbetrifft: Ein Klassennamen muss mit dem Verzeichnispfad übereinstimmen, wobei analog zum Pfadtrenner der Unterstrich verwendet wird. Unterstriche sind ansonsten in Klassennamen nicht gestattet. Das Basisverzeichnis der Klassen heißt `FACTFinder`, somit beginnen alle Klassennamen der Library mit „`FACTFinder`".

Beispiel: Die Klasse `FACTFinder_Abstract_Adapter` befindet sich in der Datei `FACTFinder/Abstract/Adapter.php`.

Sind Shop spezifische Anpassungen erwünscht, sollten diese in Klassen parallel zum Basisverzeichnis `FACTFinder` im Verzeichnis `FACTFinderCustom` abgelegt werden. Die Klassennamen sollten ebenfalls der gleichen Namenskonvention entsprechen.

Beispiel: die Klasse `FACTFinderCustom_Result` befindet sich in der Datei `FACTFinderCustom/Result.php` und überschreibt die Klasse `FACTFinder_Result` welche sich in der Datei `FACTFinder/Result.php` befindet.

Den Grund für diese Konvention erfahren Sie im Kapitel „*Erweiterte Verwendung*" im Abschnitt „*Auto-Loading Mechanismus*".

Quickstart

Dieses Kapitel erklärt nun, wie Sie die Demo des Frameworks so ändern, dass die für Sie bereitgestellte Suchumgebung mit Ihren Daten korrekt verwendet wird. Sie können das Kapitel komplett überspringen, wenn Sie detaillierte Informationen über die Funktionsweise des Frameworks erhalten möchte.

Als ersten Schritt ist es wichtig, dass das Framework inklusiv `demo` Verzeichnis im Dokumentverzeichnis eines Webservers abgelegt wird. Schon an dieser Stelle sollte die `index.php` der Demo aufgerufen werden können und die Demo Suchumgebung sollte mit den Voreingestellten Demodaten funktionieren. Wenn dem nicht so ist, gibt es ein Problem mit der Server- oder PHP Installation.

Aufbau

Die Demo besteht aus den Dateien für die Webpage (CSS-, JavaScript- und Bilddateien) die sich im Verzeichnis `files` befinden. Im Verzeichnis `i18n` befinden sich zwei Sprachdateien. Damit ist es möglich, die Seite in beiden Sprachen anzuzeigen, je nachdem welche Sprache in der Konfiguration eingestellt ist. Außerdem gibt es noch das `templates`-Verzeichnis, welches die Templates für die Erzeugung der HTML Seiten enthält. Diese werden aus der Klasse `HtmlGenerator` angesprochen, welche im Verzeichnis `userdata` liegt. Darin ist auch die Konfigurationsdatei `local.config.xml` enthalten.

Im Demo-Verzeichnis selbst finden Sie noch die `index.php`, welche die Suche initialisiert, und die Proxy-Skripte `suggest.php` und `scic.php`. Diese werden für Ajax-Requests verwendet, welche aufgrund der Browser Cross-Site-Scripting-Sperre keine Requests an FACT-Finder direkt absenden dürfen.

Konfiguration

Zugangsdaten

Sobald eine FACT-Finder-Suchumgebung für Sie aufgesetzt wurde, sollten Sie die Zugangsdaten dafür erhalten. Tragen Sie diese in die Datei `local.config.xml` ein, welche Sie im Verzeichnis `demo/userdata/` finden.

Normalerweise müssen nur die Werte im Bereich `search` verändert werden. Die genauere Beschreibung der Konfiguration und der einzelnen Werte finden Sie im Kapitel „*Erweiterte Verwendung*“ im Abschnitt „*Konfiguration*“.

Es ist möglich, dass Sie vom entsprechenden Mitarbeiter von Omikron einfach nur einen Link zum Management Interface der FACT-Finder-Suchumgebung erhalten. Jedoch können Sie daraus die notwendigen Werte herauslesen.

Beispiel: Sie erhalten folgende URL:

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

Aus dieser URL können Sie folgende Werte ableiten:

- Address: `search123.fact-finder.de`
- Context: `IhreSuche`
- Port: `8080` (ist keine Nummer angegeben, handelt es sich um Port 80)

Über das Management können Sie dann auch die verfügbaren Channels einsehen und einen davon für den Wert `channel` eintragen.

Neben den Zugangsdaten sollten Sie auch die Info erhalten haben, welcher Authentifizierungstyp verwendet wird. Beim sichersten Typen `advanced` benötigen Sie außerdem den verwendeten Prefix und Postfix.

Feldnamen

Damit die Demo nun auch mit den Daten Ihrer Suchumgebung umgehen kann, müssen Sie noch die Feldnamen anpassen. Die Liste der Feldnamen Ihrer Suchumgebung finden Sie im FACT-Finder-Management in der Konfiguration unter „Relevanz".

Die Feldnamen müssen dann in der Datei `demo/templates/fieldnamesConfig.php` eingetragen werden. Bei dieser Datei handelt es sich um einfachen PHP Code. Auf der linken Seite stehen die benötigten Informationen (als Array-Keys) und rechts die Feldnamen, welche die entsprechenden Werte enthalten. Sollte eine Informationen in Ihren Daten nicht geliefert werden können, lassen Sie den entsprechenden Feldnamen einfach leer.

Abschluss

Nach diesen Anpassungen sollte die Demo nun mit Ihren Daten funktionieren. Sollte das nicht der Fall sein, dürfen Sie gerne Kontakt mit unserem PHP-Entwickler aufnehmen.

Wenn die Demo nun mit Ihren Daten funktioniert, gibt es folgende Möglichkeiten, um die Suche in Ihren Shop zu integrieren:

- Anhand der Demo haben Sie genug Beispielcode, um angepasst an Ihr Shopsystem eine komplette Implementierung von FACT-Finder vorzunehmen. Verwenden Sie das Framework in Ihrem Shop ähnlich wie in der `index.php` und der `userdata/HtmlGenerator.php`, um an die Daten von FACT-Finder zu gelangen und dann in Ihrem Shopsystem entsprechend darzustellen.
- Wenn nicht ganz so viel programmiert werden soll, kann man natürlich auch den Code der Demo weiterverwenden. Passen Sie das Layout und das Design Ihren Anforderungen und Wünschen an. Um das Suchergebnis dann ganz einfach in den Shop einzubinden, benennen Sie die `index.php` zum Beispiel in `search.php` um und binden Sie es an der entsprechenden Stelle in Ihrem Shop ein.

  Bitte beachten Sie, dass es sich dabei um Beispiel-Code handelt. Sie sind selbst dafür verantwortlich zu überprüfen, ob alle von Ihnen lizenzierten Module korrekt im Shop verwendet werden. Für eine Abnahme der Implementierung können Sie auch gerne Ihren zugeordneten Projektleiter seitens Omikron kontaktieren.
- Hat Ihnen das Framework nicht gefallen, steht es Ihnen frei, die FACT-Finder-Schnittstellen mit eigenem Code zu verwenden.

Alternativer Demoshop

Im Framework findet sich auch noch ein Verzeichnis `demo2`. Dieses beinhaltet einen zweiten, funktionsgleichen Demoshop, dessen Implementierung jedoch anders strukturiert ist. Während der Demoshop im Verzeichnis `demo` auf einem `HtmlGenerator` basiert, welcher den Shop aus `.phtml`-Template-Dateien aufbaut, verwendet der alternative Demoshop `render`-Funktion für die einzelnen Elemente der Seite. Wenn Ihr eigener Shop eher dieser Struktur ähnelt, basieren Sie Ihre Integration von FACT-Finder auf diesem zweiten Demoshop.

Erweiterte Verwendung

Dieses Kapitel beschreibt, wie das FACT-Finder Framework verwendet, verändert und erweitert werden kann. Es erklärt das Zusammenspiel der Konfiguration mit der API und wie die API intern funktioniert.

Auto-Loading-Mechanismus

Eine der Kernklassen ist der `FACTFinder_Loader`. Da diese Klasse oft verwendet wird, kann man Sie auch über den Alias `FF` ansprechen. (Dazu wurde eine finale Klasse mit dem Namen `FF` angelegt welche von der Klasse `FACTFinder_Loader` erbt, jedoch keine Methoden überschreibt und keine eigenen definiert).

Diese Klasse ist dafür verantwortlich, dass anhand des Klassennamens die korrekte Datei geladen wird. Daher ist es wichtig, dass die Namenskonventionen eingehalten werden (siehe *„Einleitung"*). Diese Klasse wird als `auto_loader` registriert, sodass für die Klassen des Frameworks keine includes notwendig sind.

Zusätzlich dient die Klasse zum dynamischen Instanziieren von Objekten. Dadurch ist es möglich, die Verwendung einer Klasse im kompletten Code zu ersetzen. Im Code wird beispielsweise ein Objekt vom Typ `record` erstellt: `$record = FF::getInstance("record");`. Standardmäßig findet die Loader-Klasse anhand des Namens die Datei `FACTFinder/Record.php` und die darin enthaltene `FACTFinder_Record`-Klasse. Soll diese nun überschrieben werden, muss einfach die Klasse `FACTFinderCustomer_Record` in der Datei `FACTFinderCustomer/Record.php` erstellt werden. Empfehlenswert ist es an dieser Stelle, von der `FACTFinder_Record` Klasse zu erben.

Des Weiteren kann diese Klasse dafür sorgen, dass Objekte einer Klasse nur einmal instanziiert werden (Singleton). Dafür kann die statische Methode `getSingleton` analog zur `getInstance` Methode verwendet werden.

Konfiguration

Mit der Konfiguration kann man einen großen Einfluss auf die API nehmen und damit viele Probleme lösen. Die Konfigurationswerte werden standardmäßig mit der `Zend_Config` Klasse aus einer XML-Datei geladen. Die Klasse `FACTFinder_Configuration` dient als Wrapper für das `Zend_Config` Objekt und gibt die einzelnen Werte entsprechend dem implementierten Interface `FACTFinder_Abstract_Configuration` zurück. Eine andere Implementierung der Konfiguration ist von daher problemlos möglich.

In der Konfigurations-XML-Datei sind alle Werte im XML-Tag `production` eingeschlossen. Das liegt daran, dass die Zend-Konfiguration Werte gruppieren und vererben kann. Mehr Infos in der Zend-Framework-Dokumentation der Klasse `Zend_Config_Xml`.

Bedeutung und Verwendung der einzelnen Konfigurationswerte

In den folgenden Abschnitten werden nun die einzelnen Werte der `default.config.xml` erklärt. Die Methoden des Interfaces `FACTFinder_Abstract_Configuration` sind analog dazu aufgebaut und sollten von daher selbsterklärend sein. Ein Beispiel einer Konfiguration finden Sie im Verzeichnis `demo/userdata/` des Frameworks.

Allgemeine Einstellungen

version: Enthält die Version des Frameworks. Kann damit in der API überprüft und ausgegeben werden. Dieser Wert sollte nicht geändert werden.

revision: Enthält die Revisionsnummer des letzten internen SVN-Commits. Ist sozusagen eine genauere Versionsnummer, auch wenn große Sprünge zwischen den Nummern enthalten sind. Dieser Wert sollte nicht geändert werden.

debug: Kann auf `true` oder `false` gesetzt werden. Kann verwendet werden, um Debug-Ausgaben für FACT-Finder zu steuern. Es werden aktuell noch keine Debug-Ausgaben gemacht, da auch kein Debug-Tool in Verwendung ist.

Zugangsdaten

Damit spezifiziert wird, welche Datenbank genau für die Suche verwendet werden soll, sind einige Einstellungen im XML-Block `search` notwendig.

address: Enthält die Adresse (Domain oder IP) des Suchservers. Hier dürfen keine zusätzlichen Infos wie `http` oder Pfade angegeben werden.

port: Hier wird der verwendete Port des FACT-Finder Dienstes auf dem entsprechenden Server eingetragen.

protocol: Legt fest, welche Protokolle zur Abfrage verwendet werden können. Aktuell sind nur `http` und `https` möglich.

auth: Hier sind die Daten enthalten, um sich am FACT-Finder Service zu authentifizieren.

> **user**: Benutzername zur Anmeldung am FACT-Finder Service
>
> **password**: Passwort zur Anmeldung am FACT-Finder Service
>
> type: FACT-Finder unterstützt verschiedene Authentifizierungsvarianten: Bis zur FACT-Finder-Version 6.4 nur `http`, danach auch `advanced` und `simple` (womit `http` überflüssig

geworden ist). Worin sich die einzelnen Authentifizierungen unterscheiden, können Sie der allgemeinen Integrationsdokumentation entnehmen.

advancedPrefix / advancedPostfix: Diese Einstellungen werden beim Authentifizierungs-Typ `advanced` benötigt und auch nur dann berücksichtigt. Sie sorgen für eine sicherere Verschlüsselung.

context: Mit dieser Einstellung wird der Anwendungsname von FACT-Finder festgelegt. Sollten Sie von Ihrem zuständigen Omikron-Mitarbeiter nur eine URL zu FACT-Finder erhalten, ist dies der Pfad nach der Serveradresse.

Beispiel: aus der URL `http://search123.fact-finder.de/IhreSuche/Management.ff` ist `IhreSuche` der Anwendungsname und damit der Wert, der bei `context` eintragen wird.

channel: FACT-Finder unterstützt mehrere Channels, wobei normalerweise jeder Shop einen eigenen Channel zugeordnet bekommt. Hier können Sie den entsprechenden Channel-Namen eintragen. Über die FACT-Finder-Oberfläche können Sie unter „Konfiguration" alle Channels Ihrer FACT-Finder-Suche einsehen.

language: Stellen Sie hier die verwendete Sprache ein. Dieser Wert wird an FACT-Finder gesendet, was die Sprache einiger Textelemente beeinflusst. Dieser Wert kann auch für eine Internationalisierung verwendet werden, so wie das bei der Demo der Fall ist.

Parameter-Einstellung

Die Parametereinstellungen dienen dazu, die Unterschiede der Parameter zwischen Shop und FACT-Finder auszugleichen, Parameter hinzuzufügen oder zu entfernen. Innerhalb des XML-tags `params` gibt es zwei Unterbereiche: `server` und `client`. Der Bereich `server` enthält die Einstellungen für die Parameter die an den Server gesendet werden, der Bereich `client` für die Parameter die im Shop weiterverwendet werden. Es gibt drei verschiedene XML-Tags, die in diesen Bereichen beliebig oft verwendet werden können:

ignore: Parameter, die im entsprechenden Bereich ignoriert werden sollen. Das XML-Attribut „`name`" ist erforderlich und enthält den entsprechenden Parameter.

Beispiel: `<ignore name="channel" />` ignoriert den Parameter `channel`.

mapping: Damit lässt sich ein Mapping zwischen den Parametern von FACT-Finder und den Parametern des Shops herstellen. Generell ergänzen sich die Mapping-Einträge aus den beiden Bereichen (siehe Beispiel). Mit den beiden XML-Attributen `from` und `to` wird die Richtung der Transformation vorgegeben.

Beispiel: `<mapping from="query" to="keywords" />` würde so im Bereich `client` eingetragen werden und sorgt dafür, dass der Parameter `query`, der von FACT-Finder kommt, in den Parameter `keywords` umgewandelt wird, der so im Shop verwendet wird. Gleichzeit sollte im

Bereich `server` das XML-Tag `<mapping from="keywords" to="query" />` eingetragen werden, damit auch `keywords` wiederum in `query` umgewandelt wird.

required: Mit dieser Einstellung können Parameter erzwungen werden. Mit dem XML-Attribut „`name`" gibt man den entsprechenden Parameternamen an. Ist dieser Parameter im entsprechenden Bereich nicht vorhanden, wird er mit dem angegebenen Standardwert hinzugefügt, welcher über das Attribut „`default`" definiert wird. Es ist dabei nicht nötig Parameter zu erzwingen, die bereits durch die API gesetzt werden (zum Beispiel `format=xml` für die XML-Schnittstelle oder `username=xyz` etc. zur Authentifizierung).

Beispiel: `<required name="filterCustomerId" default="0" />` würde den Parameter `filterCustomerId=0` setzen, wenn dieser Parameter nicht vorhanden ist.

Encoding-Einstellungen

Mit den Einstellungen im XML-Tag `encoding` sorgt das Framework selbst dafür, dass die einzelnen Daten und Werte korrekt kodiert sind. Hier gibt es nur 3 Werte, die aber alle erforderlich sind. Wenn die verwendete PHP-Installation iconv unterstützt, können Sie alle dafür vorgesehenen Encodings einstellen. Ansonsten werden nur die Werte `UTF-8` und `ISO-8859-1` unterstützt.

pageContent: Definiert das Encoding der Shop-HTML-Seite

pageURI: Definiert das URL-Encoding des Shops. Entspricht meistens dem Encoding der Seite selbst.

serverURI: Definiert das URL-Encoding des FACT-Finder Servers. Wenn Sie sich unsicher sind, fragen Sie den FACT-Finder Support oder Ihren zuständigen Ansprechpartner bei Omikron.

Eigene Werte

Es können auch eigene Einstellungen in die Konfiguration eingetragen und innerhalb der Anwendung aus dem `Configuration`-Objekt über die Methode `getCustomValue($name)` ausgelesen werden. Grundsätzlich sind hier aber keine verschachtelten Strukturen möglich.

Grundlegende Funktionsweise

Die Hauptklassen

Die wichtigsten Klassen sind: der `Adapter`, welcher letztendlich die Ergebnis-Objekte erzeugt; der `DataProvider`, welcher für das Beziehen der Rohdaten zuständig ist; der `ParametersParser`, welcher für das Erzeugen der URLs verantwortlich ist; der `EncodingHandler`, welcher für das Kodieren der Daten zuständig ist; und die `Configuration`, welche die einzelnen Konfigurationswerte zur Verfügung stellt.

In Abbildung 1 sehen Sie, wie die Klassen und Objekte voneinander abhängig sind. Aus diesen Abhängigkeiten ergibt sich folgende feste Reihenfolge der Instanziierung:

1. Configuration (hat keine Abhängigkeiten)
2. EncodingHandler (benötigt die Konfiguration für die Encoding-Einstellungen)
3. ParametersParser (benötigt die Konfiguration für die Parameter-Einstellungen und den EncodingHandler zur korrekten Kodierung der Parameter)
4. DataProvider (benötigt die Konfiguration um die FACT-Finder-Zugangsdaten auszulesen und die Parameter für den Server vom ParametersParser)
5. Adapter (benötigt den EncodingHandler, um die Daten korrekt zu kodieren, den ParametersParser, um die URLs für das System korrekt zu erzeugen, und den DataProvider, um letztendlich an die Rohdaten zu gelangen, die dann dem System in den entsprechenden Objekten zur Verfügung gestellt werden)

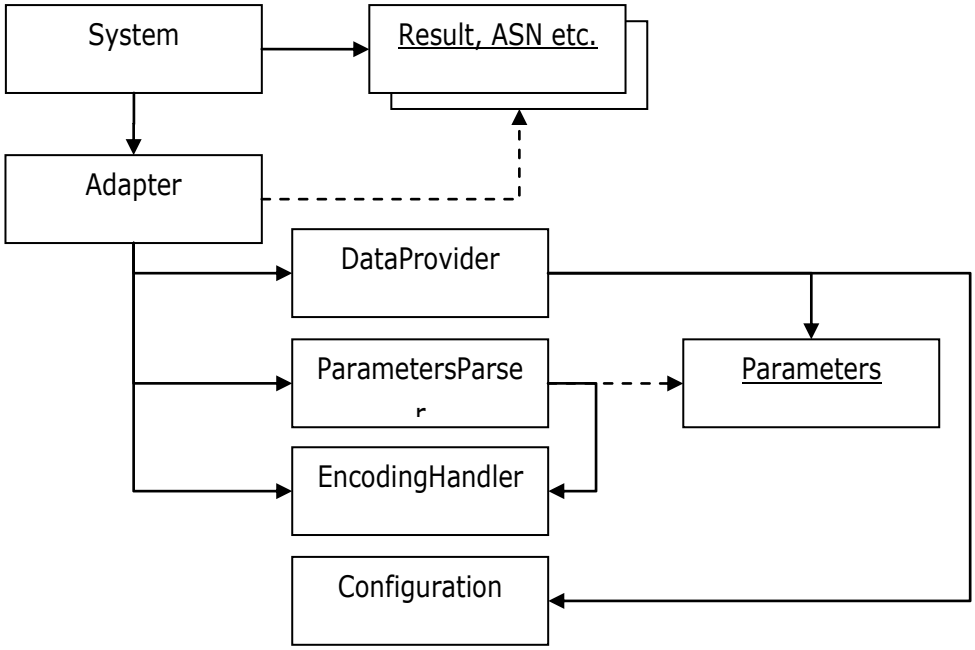

Abbildung 1: Abhängigkeiten der Klassen im Framework

Anmerkung zur Grafik: Durchgezogene Linien zeigen eine Abhängigkeit auf; gestrichelte Linien zeigen, wo die Objekte (mit unterstrichenem Namen) erzeugt werden.

Wo diese Objekte erzeugt werden, hängt letztendlich vom System ab, in welches FACT-Finder integriert werden soll. In der Demo des Frameworks geschieht die Instanziierung direkt in der `index.php`.

Zugriff auf die eigentlichen Ergebnis Daten

Das Framework ist grundsätzlich so aufgebaut, dass die benötigten Daten immer dann erzeugt werden, wenn diese auch benötigt werden (LazyLoading). Sobald der entsprechende Adapter und damit auch die restlichen notwendigen Klasse instanziiert sind, kann der Adapter auch Ergebnisse liefern. Es ist nicht nötig, eine Suche explizit zu starten: dies geschieht im Hintergrund sobald zum Beispiel die Methode `getResult()` aufgerufen wird.

Die Ergebnis-Klassen erben meistens bzw. immer dann, wenn es sinnvoll ist, von der PHP-Klasse `ArrayIterator` oder `IteratorAggregate`. Damit ist es möglich, einfach über das Objekt der entsprechenden Klasse zu iterieren und somit auf die untergeordneten Objekte in der Schleife zuzugreifen. In der Code-Dokumentation wird bei diesen Klassen erklärt, welche Daten in einer Schleife geliefert werden.

Abstraktion, Vererbung und Versionierung

Adapter

Da FACT-Finder für die diversen Funktionalitäten teilweise andere Schnittstellen anbietet, gibt es entsprechend unterschiedliche Adapter um die Daten der jeweiligen Schnittstelle abhängig von der verwendeten FACT-Finder-Version bereit zu stellen.

Der allgemeine Adapter `FACTFinder_Abstract_Adapter` sorgt dafür, dass die Verbindung zu `DataProvider`, `ParametersParser` und `EncodingHandler` vorhanden ist. Die abstrakten Funktions-Adapter im Verzeichnis `FACTFinder/Abstract` definieren dann die Methoden, mit denen es möglich ist auf die entsprechenden Daten zuzugreifen. In diesen Klassen geschieht dann auch schon das LazyLoading, die erzeugenden Methoden müssen dann noch für die jeweilige Datenschnittstelle und Version implementiert werden.

Je nach der verwendeten Version von FACT-Finder sind dann die verwendbaren Adapter verfügbar, die dann auf die Eigenarten der jeweiligen Version eingehen.

DataProvider

Auch die abstrakte DataProvider-Klasse `FACTFinder_Abstract_DataProvider` sorgt nur für die Abhängigkeiten zu den anderen notwendigen Klassen. Abhängig davon, welche Daten-Schnittstelle verwendet wird, ist es dann möglich auf unterschiedliche Art und Weise an die Daten zu gelangen. Momentan sind nur der `FACTFinder_Http_DataProvider` und der `FACTFinder_Http_ParallelDataProvider` implementiert. Es ist jedoch möglich, dass auch noch ein DataProvider für den RESTful-Service und den WebService erstellt wird.

Die beiden vorhandenen DataProvider unterscheiden sich dahingehend, dass der einfache `DataProvider` für jeden Adapter einen einzelnen (sequentiellen) Request an den FACT-Finder-Server schickt, während der `ParallelDataProvider` die Requests aller Adapter sammelt und

dann parallel abschickt. Dementsprechend unterscheidet sich auch die Verwendung der DataProvider. Der einfache (sequentielle) `DataProvider` wird wie folgt initialisiert und verwendet:

```
... // Initialisierung von $paramsParser, $config und $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();
... // Auslesen und Ausgabe von $result und $tagCloud
```

Hierbei wird sowohl durch `getResult()` als auch durch `getTagCloud()` intern durch den jeweils zugehörigen `DataProvider` ein Request an den FACT-Finder-Server abgeschickt. Bei Verwendung des `ParallelDataProviders` muss man das Abschicken der Requests explizit selbst auslösen. Dafür werden diese parallel ausgeführt, was insbesondere bei der Verwendung vieler Adapter Wartezeiten stark verringern kann:

```
... // Initialisierung von $paramsParser, $config und $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();
... // Auslesen und Ausgabe von $result und $tagCloud
```

Der Aufruf von `loadAllData()` löst dabei die parallelen Requests aus, sodass beim Aufruf von z.B. `getResult()` die Daten bereits vorliegen. Zu beachten ist außerdem der Unterschied in der Initialisierung der DataProvider.

Versionen und Schnittstellen

Momentan wird zwischen den Schnittstellen Http, Xml64, Xml65, Xml66 und Xml67 unterschieden. Eine wirkliche logische Trennung ist das nicht, sondern hat mehr mit der Benennung der darin enthaltenen Klassen zu tun. Trotzdem gibt sich ein gewisser logischer Aufbau, auch wenn hier die Kriterien „Schnittstelle" und „FACT-Finder Version" gemischt werden.

Im http-Verzeichnis sind die Klassen enthalten, die unabhängig vom Daten-Format über HTTP arbeiten (der DataProvider) oder ein proprietäres, versionsunabhängiges Daten-Format unterstützen.

Die XML-Schnittstellen erwarten, wie der Name schon sagt, XML als Ergebnisformat und können daher für den Request an FACT-Finder den HTTP-DataProvider verwenden, der mit einem entsprechenden Parameter XML liefert. Das Parsen der XML-Daten ist dann wieder Aufgabe des jeweiligen Adapters.

Da sich die FACT-Finder-Schnittstellen von Version zu Version ändern (können), gibt es unter Umständen neue Adapter der gleichen Schnittstelle, die auf die speziellen Änderungen eingehen. Normalerweise erben diese Adapter voneinander und überschreiben nur die notwendigen Methoden.

Funktionsweise der einzelnen Klassen

Wie genau die einzelnen Klassen funktionieren, welche Methoden angeboten werden und was diese machen, können Sie der generierten Dokumentation im `doc/phpdoc/`-Verzeichnis entnehmen. Damit diese Doku lesbar ist, muss die darin enthaltene index.html in einem Browser geöffnet werden.

FAQ

Fragen zum Framework

Wie wird das Framework um angepassten Code oder eigene Funktionalitäten erweitert?

Sie können nahezu jede Klasse so überschreiben, dass diese systemweit ersetzt wird. Mehr Informationen dazu finden Sie im Abschnitt *„Auto-Loading-Mechanismus*".

Es sind nicht alle unsere lizenzierten Module von FACT-Finder durch das Framework abgedeckt, was nun?

Generell ist der Lizenznehmer selbst für die Implementierung von FACT-Finder im Shop zuständig. Dieses Framework dient nur als Hilfe und soll verhindern, dass Probleme erneut gelöst werden. Auf Nachfrage können wir unter Umständen das Framework erweitern oder Sie erweitern es, lassen uns den entsprechenden Code zukommen und tragen damit zur Weiterentwicklung bei.

Alle Zugangsdaten sind korrekt angegeben, es wird aber immer noch kein Ergebnis angezeigt.

Am besten Sie senden uns in so einem Fall die verwendete Konfigurations-XML-Datei zu, sodass wir alle Einstellungen nochmal mit den Einstellungen von FACT-Finder abgleichen können.

Einen häufigen Grund für diesen Fehler stellen jedoch die unterschiedlich eingestellten Uhrzeiten auf den Servern dar, sodass beim Authentifizierungstyp `advanced` das Passwort als abgelaufen gilt. Überprüfen Sie in diesem Fall, ob die Uhrzeit korrekt eingestellt ist. Funktioniert es trotzdem nicht, handelt es sich vermutlich um eine geringe jedoch ausschlaggebende Abweichung zwischen den Uhrzeiten auf den beteiligten Rechnern. In diesem Fall informieren Sie bitte einen FACT-Finder-Mitarbeiter, dieser überprüft dies anhand der FACT-Finder-Fehlermeldungen und kann bei geringen zeitlichen Abweichungen die Toleranz etwas erhöhen.

Spezielle Fragen

Weitere Fragen werden im Kundenportal unter http://support.fact-finder.de im FAQ-Bereich beantwortet. Wenn Sie auch dort keine Antwort auf Ihre Fragen finden, dann rufen Sie einfach an: 07231/12597-701 oder senden eine Mail an support@fact-finder.de. Ein kompetenter Mitarbeiter hilft Ihnen weiter.