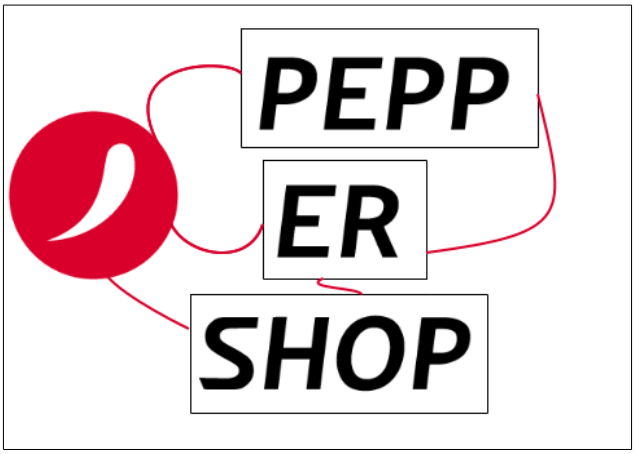
PEPPERSHOP

PepperShop Modulsysteme für Entwickler

Anleitung



Datum
20. Oktober 2016

Version
3.2

Inhaltsverzeichnis

PepperShop wird von Glarotech entwickelt und vertrieben. Seit 1998 ist das innovative Unternehmen im Internet tätig und auf E-Commerce spezialisiert. Sie als Kunde profitieren vom direkten Draht zu den Herstellern der Produkte.

Glarotech GmbH
Toggenburgerstrasse 156
CH-9500 Wil

info@glarotech.ch
Tel. +41 (0)71 923 08 58
www.glarotech.ch

glarotech

Module für den PepperShop

Früher war der PepperShop von der Architektur her vergleichbar mit einem monolithischen Betriebssystemkernel ohne Module. Dank dem modularen Ansatz ist der bisherige PepperShop Code - *Core* - mit sogenannten externen PepperShop Modulen einfach erweiterbar geworden.

Dank der Modularisierung erreicht man eine bisher unmögliche Code Kapselung. Weiter wird die benötigte Entwicklungszeit ungemein verkürzt. Der zu wartende Code wird separiert und vom Core entkoppelt. Dank einer (De-)Installationsroutine wird auch das Patchingverfahren und Know-How vom Modulanwender (Shopadministrator) hin zum Modulentwickler verlagert.

Die **wichtigsten Vorteile** kurz zusammengefasst: Module bieten modularer, in sich geschlossener Code. Der Modulentwickler muss sich nicht um die Interna des PepperShops kümmern, die PepperShop Core Entwickler können sich dem Core widmen und die Shopbetreiber haben keine mühsamen Patch-Sessions mehr vor sich.

1. Kurzübersicht über das PepperShop Modulsystem

Es gibt im PepperShop **zwei Arten der Modulintegration**, welche oft kombiniert werden und unterschiedliche Vor- / Nachteile haben:

1.1 A-Module

Ein älteres, sehr umfangreiches und wegen der starken Abstraktion eher langsames Modulsystem inkl. Administration, Sandbox und SQL-Parser, ... Details zu den A-Modulen werden wir in diesem Dokument noch weiter anschauen.

1.2 B-Module

Ein neueres System bestehend aus einer einfachen und sehr schnellen Funktion zum prüfen, ob ein Modul existiert (`modul_check(..)`) und einer Integration in den Kundenaccount (myAccount) des Shops.

Diese Art der Integration wurde B-Modul benannt, da es auch Module ohne Administrationsoberfläche gibt, welche ausschliesslich via modul_check Calls integriert wird. Es können also Module erstellt werden, die komplett unabhängig von der Datenbank sind.

Die Anwendung ist relativ einfach. Als Ablageorte wählt man wie bei den A-Modulen: Kundenseitig {shopdir}/shop/module/[modulname]/ und in der Administration {shopdir}/shop/Admin/module/[modulname]/. Die Einbindung erfolgt aber nicht über Interfaces, sondern mit einem modul_check Call, z.B.:

```
if (modul_check('module/[modulname]/[moduldatei_ohne_endung]','.def.php') {...}
```

1.3 Umsetzen, oft = Kombination von A + B Modul

Eigentlich ist der Weg der Integration aller neueren PepperShop Module. Sobald eine Administrationsmaske für das Modul benötigt wird, erstellt man diese mit einem klassischen A-Modul.

1.4 Modul Auflistung

Damit ein Modul in der Shop-Administration ⇨ Shop-Konfiguration ⇨ Module erkannt wird, kann man dessen Hauptdatei (oder die, welche sich am häufigsten bei Updates ändert) überwachen lassen. Dazu muss man folgenden Eintrag tätigen:

Datei: `{shop_verzeichnis}/shop/Admin/shop_update_module_check_local.php`
(hier einfach einen neuen Block einfügen)

Bisher verfügbare / vergebene Module als CSV-Tabelle anzeigen:
http://{ihredomain.com}/{pfad_zu_ihrem_shop}/shop/Admin/SHOP_KONFIGURATION.php?show_module_updates_csv=true& updcounter=1#tabs-3

2. A-Module im Detail

Wir haben PHP-Code, welcher die Intelligenz des Modulverwaltungsystems nachbildet und verschiedene Tabellen, damit das Modulsystem dauerhaft Daten speichern kann. Weiter gibt es Interfaces (Hooks), welche die Module in den normalen Programmablauf einbetten und natürlich die aufzurufenden Module selbst.

2.1 Schema

Das Modulsystem besteht aus mehreren ineinander greifenden Blöcken. Am einfachsten lässt sich die Funktionsweise mit einem kleinen Ablaufschema zeigen (siehe Abbildung auf nächster Seite).

Das Schema zeigt uns die Arbeitsweise des Modulsystems. Wann immer ein Interface im normalen Programmablauf des PepperShop Cores auftritt, werden augenblicklich alle Module aufgerufen, welche sich bei diesem Interface registriert haben. Jedem von ihnen wird dann zur Bearbeitung das vom Interface zur Verfügung gestellte **Interface-Objekt** übergeben. Das Interface-Objekt kann alles mögliche sein.

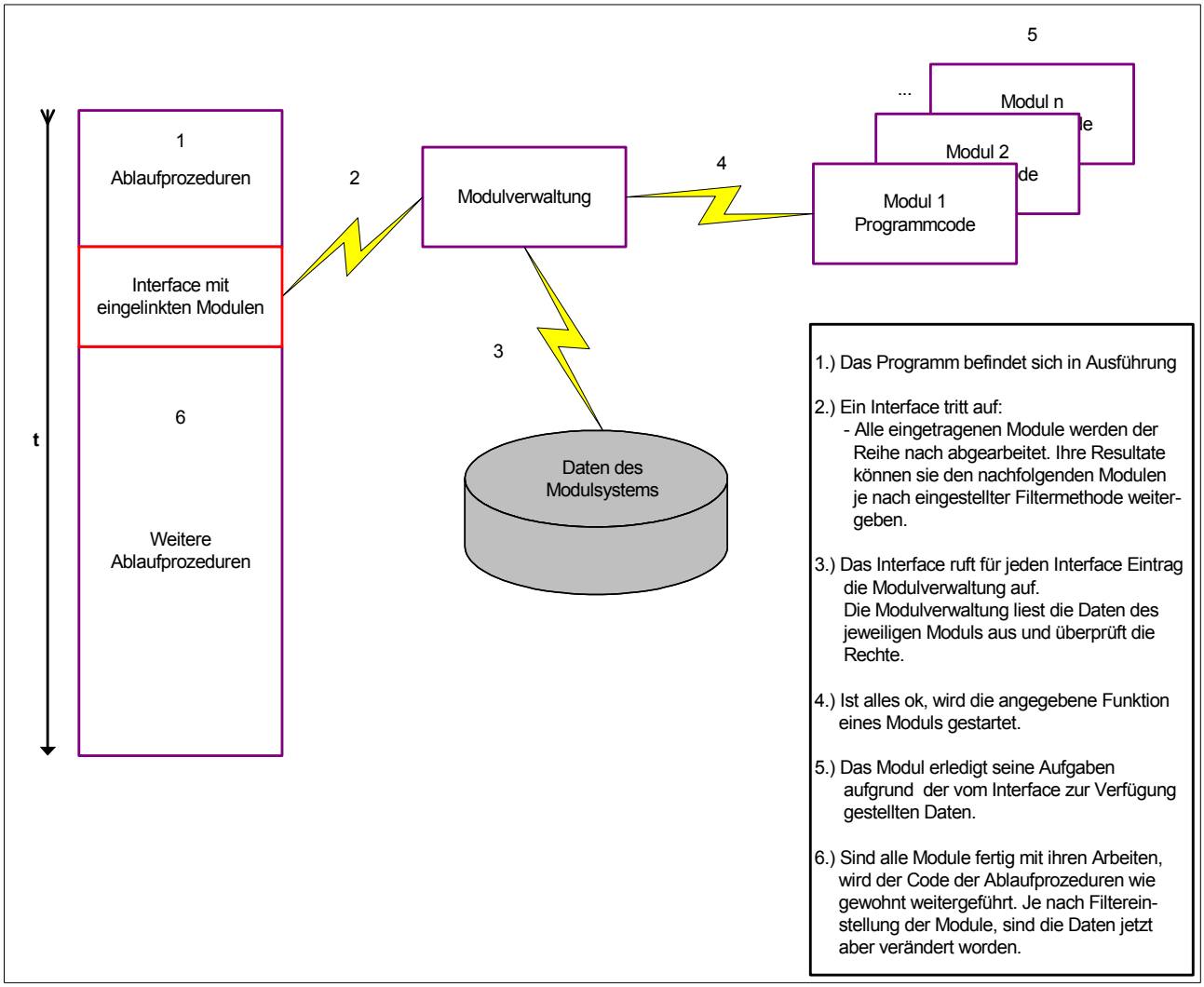


Abbildung 1: Schema des PepperShop A-Modulsystems

Wenn ein Interface z.B. am Ende des Bestellprozesses vorkommt, so wird es sich beim Interface-Objekt wahrscheinlich um ein Bestellungsobjekt handeln.

Die Schema-Abbildung hat uns den Ablauf gezeigt, wie der PepperShop mit Modulen zusammen arbeitet. Es gibt aber noch mehr, als nur den Ablauf. Module müssen auch installiert und deinstalliert, sowie verwaltet werden können. Dies sind Administrationsaufgaben und werden grundsätzlich vom Ablaufsystem getrennt (aus Performance- und Sicherheitsgründen).

Sobald es mehr Flexibilität bei der Integration des Modulcodes an Stellen im Shop benötigt, kommt die Technik der **B-Module** zum Einsatz (siehe hierzu Kapitel 1.2). Man integriert (oft kundenseitig) den Code mit einem einfachen `if(modul_check(..)) {...}` direkt an die benötigte Stelle. Damit verliert man zwar etwas an Kapselung, wenn es keine Entwicklung ist, welche direkt ins Release fliesst, hat aber eine sehr hohe Performance, was insbesondere bei Integrationen im Artikelkatalog nötig ist und sich grundlegend von anderen Systemen abhebt.

2.2 Admin- und Userteil

Die Unterteilung von Ablaufsystem und Adminsystem sieht wie folgt aus:

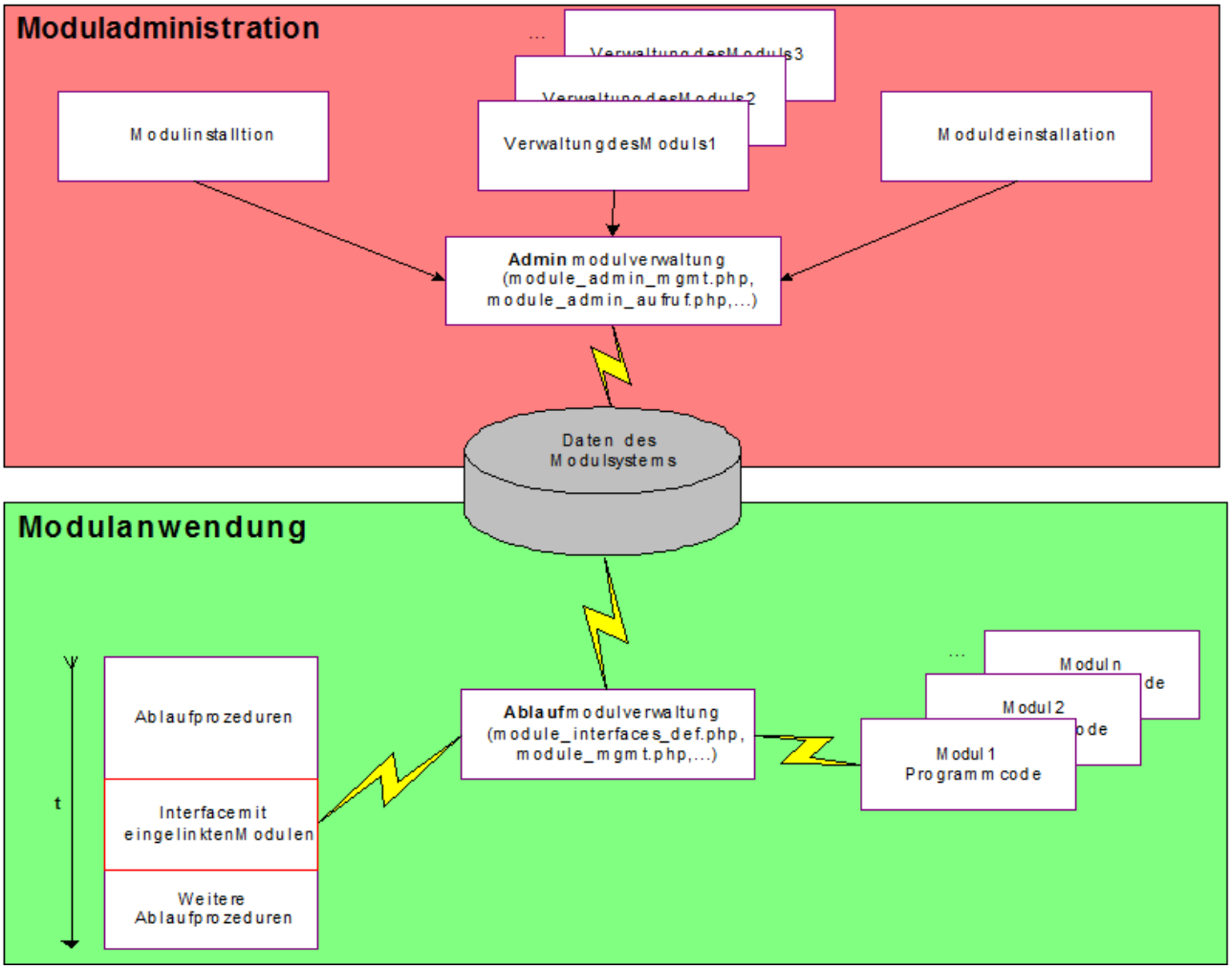


Abbildung 2: Unterteilung in Administration und Kundenseite

2.3 Zusammenfassung der Möglichkeiten

Damit man nun *alle* Teilbereiche *modular* entwickeln kann und die Module auch voneinander so gut wie möglich unabhängig sind, muss das Modulsystem mehrere Eigenschaften erfüllen und viele flexible Möglichkeiten bieten:

1.) Kapselung der Module: Die Entwicklung an Modulen muss komplett unabhängig von der Entwicklung des Gesamtsystems (PepperShop Coresystem) sein. Auch die Entwicklung von Modulen unter sich muss unabhängig sein (so sie nicht miteinander kommunizieren müssen).

2.) Der in sich geschlossene Modulcode kann sich auf mehrere Dateien und Unterverzeichnisse erstrecken.

3.) Sogenannte **persistente** (dauerhafte) Module können eigene Tabellen haben. Diese werden bei der Installation des Moduls angelegt und bei der Deinstallation wieder entfernt.

4.) Persistente Module dürfen auch bestehende Tabellen mit eigenen Attributen erweitern. Diese Attribute werden bei einer Modul*de*installation wieder entfernt.

5.) Module können sich bei Interfaces (Schnittstellen) registrieren, woraufhin sie aufgerufen werden, wenn dieses Interface im Programmablauf vorkommt. Auf diese Weise werden die Module in den normalen Programmablaufpfad integriert.

3. Was ist ein Modul

3.1 Einfache Module

Ein einfaches Modul ist ein Modul, welches keine dynamisch änderbaren Daten dauerhaft speichern muss. Dies können z.B. Filter sein, welche die ihnen übergebenen Daten aufgrund von fix eingestellten Regelwerken bearbeiten / erweitern und anschliessend wieder zurückgeben (z.B. Online-Datenexport, welcher die Bestelldaten anders formatiert übergibt).

Da das Modul weder vorhandene Tabellen erweitern muss, noch eigene Tabellen mitbringt, ist es ziemlich einfach so ein Modul in das System zu integrieren (es werden nur Dateien eingebunden).

3.1.1 Dateien und Verzeichnisse

Aufgrund der Sicherheitsrichtlinien im PepperShop gibt es zwei Verzeichnisse, welche von Modulen benutzt werden:

1.) `{shopdir}/shop/module` → Ablaufdateien (Security-Status: USER)

2.) `{shopdir}/shop/Admin/module` → Administration (Security-Status: ADMIN)

Module werden also in den entsprechenden Unterverzeichnissen `module` abgelegt. Hier wird pro Modul ein Unterverzeichnis angelegt. Erst da drin befinden sich die Dateien des Moduls.

Ein Modul besteht minimal aus drei Dateien und einem Verzeichnissen:

1.) `{shopdir}/shop/Admin/module/initialize_module.php` (Modulbeschreibung)

2.) Datei, welche die Administration des Moduls beinhaltet

3.) Dem Icon (Bild), welches den Button zur Moduladministration darstellt

Wie man sieht, kann es also reine Administrationsmodule geben (z.B. Log-Viewer, Statistik Auswertung, ...). Ein Modul, welches sich bei Interfaces registriert ist gleichzeitig auch ein Usermodul und hat demnach auch Dateien im Verzeichnis `{shopdir}/shop/module`. Zur Definition, wie welche Datei aufgebaut ist, kommen wir später.

Jedes Modul ist in ein *Unterverzeichnis* unterteilt. Dadurch herrscht Einheit im Modulverzeichnis und das automatisierte Installieren und Deinstallieren von Modulen ist einfach überschaubarer. Beispiel:

```
...{shopdir}/shop/module/
                          ausgabefilter (User Security Verzeichnis)
                          klick_statistik
                          rechnungsgenerator
                          warenwirtschaft_sap_upd
...{shopdir}/shop/Admin/module/
                              ausgabefilter (Admin Security Verzeichnis[1])
                              klick_statistik
                              rechnungsgenerator
                              warenwirtschaft_sap_upd
```

Regel: Die Namen der Ordner, in welchen die Module sind, müssen ausschliesslich Kleinbuchstaben und Underscores (_) zur Worttrennung verwenden. Keine weiteren Zeichen sind erlaubt und die maximale Namenslänge ist auf 40 Zeichen beschränkt.

3.1.2 Anmelden bitte

Damit jedes Modul sauber ins System eingebettet wird. Bringt es wie im letzten Kapitel erwähnt die Datei `initialize_module.php` mit. In dieser Datei befindet sich die *komplette Beschreibung des Moduls* durch den Modulentwickler. Alle Angaben werden in den mehrdimensionalen Array `$moduldef` (Moduldefinition) eingegeben. Diese Daten dienen als Grundlage für das geordnete Einbinden / Deinstallieren des Moduls über das Modulsystem. Im Moment ist die API Version 1.0 aktuell.

Modulbeschreibungsdaten, welche in der initialize_module.php definiert sind:

- Name des Moduls (= Ordner: Alphanummerische Zeichen + Underscore erlaubt, max. 40 Chars)
- Bezeichnung des Moduls (Name, für den Benutzer schön geschrieben)
- Zu welchen PepperShop Versionen ist dieses Modul kompatibel
- Kurzbeschreibung des Moduls
- Link zu weiteren Informationen zu diesem Modul
- Version des Moduls
- Release-Datum
- Entwickler-Informationen
- Angabe, ob dieses Modul nur ein Submodul eines weiteren Moduls ist
- Array_Interfaces (Bei welchen Interfaces wird dieses Modul wie aufgerufen)
- Array_eigene_Tabellen (Eigene, bei der Installation zu erstellende Tabellen des Moduls)
- Array_erweiterte_Tabellen (Bestehende Tabelle, welche vom Modul erweitert werden)
- Array_submodules (Falls dieses Modul Submodule hat)
- Array_required_modules (Module, welche vorausgesetzt werden)
- Security_ID (Sicherheitsstufe dieses Moduls)
- Angezeigter Name des Verwaltungstools
- Datei, welche das Verwaltungstool beinhaltet
- Icon (Bild) des Verwaltungstools

1 Das Administrationsverzeichnis ist vom Webserver her per .htaccess / Verzeichnisschutz Passwort geschützt.

Vorhanden, aber in der API 1.0 noch nicht benutzt:

- Fingerprint über die Moduldateien (MD5)
- Locale (durch das Modul Unterstützte Sprachen und Währungen)
- Valid Hosts (Rechner, von welchen das Modul starten darf)

Eine genauere Beschreibung findet man in der Beispieldatei der `initialize_module.php`. Zu finden in Kapitel 6.3.

3.2 Persistente Module

Persistenz erschliesst Modulen ganz neue Möglichkeiten. Ein Statistikmodul z.B. kann eigene Tabellen zur Verwaltung von Statistik Daten mitbringen. Module können ihre Variablen bequem in der Datenbank (zwischen)speichern und so komplexere Abläufe abarbeiten.

3.2.1 Module mit eigenen Tabellen

Module welche Daten / Metadaten dauerhaft speichern können / müssen, werden zwangsläufig eigene Tabellen in der Datenbank benötigen.

Das Array Element `$moduldef['eigene_tabellen']` in der Datei `initialize_module .php` liefert alle benötigten Angaben um die eigenen Tabellen des Moduls inkl. Attributen bei der Installation anzulegen. Bei der Deinstallation des Moduls werden die Tabellen wieder gelöscht (zuvor gespeicherte Daten gehen dann natürlich verloren).

Durch eine automatisierte, getestete und als zuverlässig arbeitend eingestufte Installations- / Deinstallationsroutine kann man eine Kapselung von persistenten Modulen erreichen.

Wir haben auf diese Weise Module, welche mehrere Dateien verwenden können, nach belieben Unterverzeichnisse und darin weitere Dateien ansprechen können und die sogar Daten persistent in eigene Tabellen ablegen können. Dank der automatisierten Installation, hat man dank der `initialize_module.php` Datei auch eine saubere Schnittstelle zu diesen Modulen. Man kann sie hinzufügen und wieder wegnehmen und wichtiger – wiederverwenden.

Tabellennamensregel: `<Modulname>_Tabellenname` (alles Kleinbuchstaben). Die angegebenen Tabellennamen werden also zuerst noch mit dem Präfix des Modulnamens erstellt → dies sollte man bei seinen SQLs miteinbeziehen.

3.2.2 Module erweitern bestehende Tabellen

Natürlich ist das immer noch nicht der Weisheit letzter Schluss. Je nach Komplexität eines Moduls muss es auch derart mit bestehenden Strukturen arbeiten können, dass gewisse Daten in schon bestehende Tabellen integriert werden müssen. Dank dem Konzept der '*erweiterten Tabellen*' kann dieses Modulsystem dynamisch, je nach Anforderung auch bestehende Tabellen erweitern.

Da dieses Thema von der Security her heikel ist, sind hier die Systemtabellen des Modulsystems und gewisse Coresystem Tabellen für Module gesperrt. Überhaupt geschieht der Datenbankzugriff von Modulen nur durch einen *DB-Object Wrapper*, welche die Einhaltung der Sicherheit durch die Module gewährleistet (zu finden in `module_database _def.php`).

Wie man im ER-Diagramm des Modulsystem sieht (siehe Kapitel 7), wird die ganze Verwaltung nicht in der Datenbank gespeichert. Die Tabellenverwaltung ist nur wichtig, wenn ein Modul installiert bzw. deinstalliert wird. Die Informationen werden also jeweils von der Datei `initialize_module.php` geholt.

Der ganze Ablauf ist der gleiche, wie bei den Modultabellen. Der einzige Unterschied besteht darin, dass die in den bestehenden Tabellen erstellten Attribute vor dem angegebenen Namen noch den Prefix des Modulnamens tragen:

Eine eindeutige Namensgebung verhindert Konflikte:

`<Modulname>_Attributname` (alles Kleinbuchstaben)

Bei der Deinstallation werden diese Attribute wieder entfernt.

Die Definition (Format) von zu erweiternden Tabellen entspricht exakt der Definition wie man eigene Tabellen des Moduls beschreibt (Kapitel 1.2.1).

4. Interfaces

Module können sich bei Schnittstellen registrieren und werden auf diese Weise aufgerufen. Das A und O dieses Modulsystems ist natürlich das Anlegen respektive das Platzieren der Schnittstellen. Dies wird von den Core-Entwicklern in Absprache mit den Modulentwicklern je nach Bedürfnis gemacht.

4.1 Wie sehen Interfaces aus

Interfaces sind Orte im Programmablauf des PepperShops, wo ein `Module_interface` Objekt erzeugt wird (Definition siehe `module_interface_def.php`) und via `call_modules` Memberfunction die bei diesem Interface registrierten Module aufgerufen werden.

Beispielsweise nach dem erfolgreichen Absenden einer Bestellung (Ende der Datei `USER_BESTELLUNG_1.php`) könnte man ein Interface wie folgt setzen:

```
/** Interface ID: 2, Name: bestell_obj_nach_erfolg_bestellung begin **/
    // Beschreibung: Dieses Modulinterface stellt das Bestellungsobjekt
    // nach einer erfolgreichen Bestellung zur Verfuegung
    $interface_id     = 2;
    $interface_object = &$meineBestellung; // Das Interface-Objekt ist hier also eine Bestellung
    $modul_interface  = new Module_interface($interface_id);
    $interface_object = &$modul_interface->call_modules(&$interface_object);
/** Interface ID: 2, Name: bestell_obj_nach_erfolg_bestellung end   **/
```

Zuerst wird hier die abgeschlossene Bestellung als Interface Objekt verwendet.

Wir instanzieren ein `Module_interface` Objekt und rufen danach die Funktion `call_modules` daraus auf. Dieser Funktion wird eine Referenz auf das Interface Objekt übergeben und sie liefert uns das (eventuell durch die Module modifizierte) Interface Objekt wieder zurück, nachdem alle bei diesem Interface registrierten Module diese Bestellung abgearbeitet haben.

4.2 Filtermodule und One-Way Module

Ein Modul kann sich auf zwei Arten bei einem Interface registrieren: `filter` oder `one_way`. Module welche im one_way Modus arbeiten, verändern das ihnen übergebene Interface-Objekt nicht. Sie benötigen lediglich die Daten daraus – Beispiel Statistikmodul. Andere wiederum arbeiten im `filter` Modus, filtern also das ihnen übergebene Interface Objekt → und somit die Daten des Programmablaufs. Sie geben die veränderten Daten wieder zurück. Ein Beispiel hierfür wäre ein Online Tracking Modul, welches den Status einer Bestellung ändern würde oder ein Darstellungsmodul, welches die Ausgabe der Anzeige aufgrund von übergebenen Daten übernimmt.

Module im `filter` Modus, werden nacheinander abgearbeitet. Es gibt eine Positionsangabe bei den Interfaces, welche aber erst später benutzt werden wird. Die Idee ist, dass man die relative Position der Abarbeitung mit einer Prioritätsangabe (Position) steuern kann. Das Schema wäre ähnlich der Startsequenz bei einem UNIX SysV Startvorgang. Ev. werden hier noch die Required-Module Abhängigkeiten mit

in die Auswertung einfliessen.

Wie man den Interface Tabellenattributen (siehe CREATE SQL-Statement der Tabelle `module_interfaces`) entnehmen kann, wird zwischen verschiedenen Interface Typen unterschieden. Vorläufig gibt es drei verschiedene Arten, wie sich ein Modul bei einem Interface registrieren kann.

4.3 interne Interfaces

Schnittstellen im Sinne von Wegscheidungen, wo Objekte z.B. fertig bearbeitet wurden und jetzt einen Weg gehen, welcher **im** PepperShop drin bleibt, also nicht zu einem anderen Programm auf z.B. einem anderen Server gesendet wird (vergleichbar mit dem `payment`-Interface im PepperShop). Beispiel wo Daten den Shop verlassen: Externe Clearingstelle für Kreditkartendaten (z.B. Saferpay/PostFinance/PayPal/...). Dies wird aber üblicherweise übers interne Payment Interface v.2 programmiert..

4.4 externe Interfaces

(Noch nicht implementiert) Hier werden die Daten anschliessend via ein POST-Formular zu einem anderen Programm, welches unter Umständen auf einem anderen Rechner läuft, gesendet.

4.5 modul Interfaces

(Wird wahrscheinlich in der nächsten API Version fallen gelassen!) Diese Interfaces werden vom eigentlichen Grundsystem des PepperShops nicht verwendet. Es sind aber Punkte im Programmablauf, bei welchen es Sinn machen könnte, später einmal die Daten abzugreifen und sie via ein Modul zu verarbeiten. Dieser Modus ist eventuell auch überflüssig. Er wurde damals erdacht, weil es in einem offenen System gut sein kann, dass man interne Interfaces anders schützen will, als öffentlich zugängliche.

5. Installation eines Moduls

Da die Installation und Deinstallation eines externen PepperShop Moduls automatisiert vom Modulsystem erledigt wird, können wir die Kapselung der Module garantieren, obwohl wir mehrere Klassen / Objekte / Dateien / Unterverzeichnisse und Tabellen verwenden.

5.1 Installation / Deinstallation

Administrator hat die Moduldateien heruntergeladen, entpackt und den Modulordner ins Unterverzeichnis `.../module` abgelegt.

In der PepperShop Administration gibt es neu einen Menüpunkt Module. Hier kommt man zu den Administrationspunkten der einzelnen Module und zur Verwaltung der Module.

Die Modulverwaltung sieht wie in der Abbildung auf der kommenden Seite aus.

Links sind die gefundenen, noch nicht installierten Module und rechts sind die schon fertig installierten Module.

Die Erkennung der schon installierten Module wurde früher schon angedeutet. Ein Prozess durchforstet das Admin-`module` Verzeichnis und überprüft ob die Module in den jeweiligen Verzeichnissen schon in der DB eingebunden wurden oder nicht. Dementsprechend, werden die Module im Module-Admin-GUI angezeigt. Die Überprüfung ist einfach.

Wenn es sich um ein Submodul eines anderen Moduls handelt, wird es nicht zur Auswahl angezeigt werden.

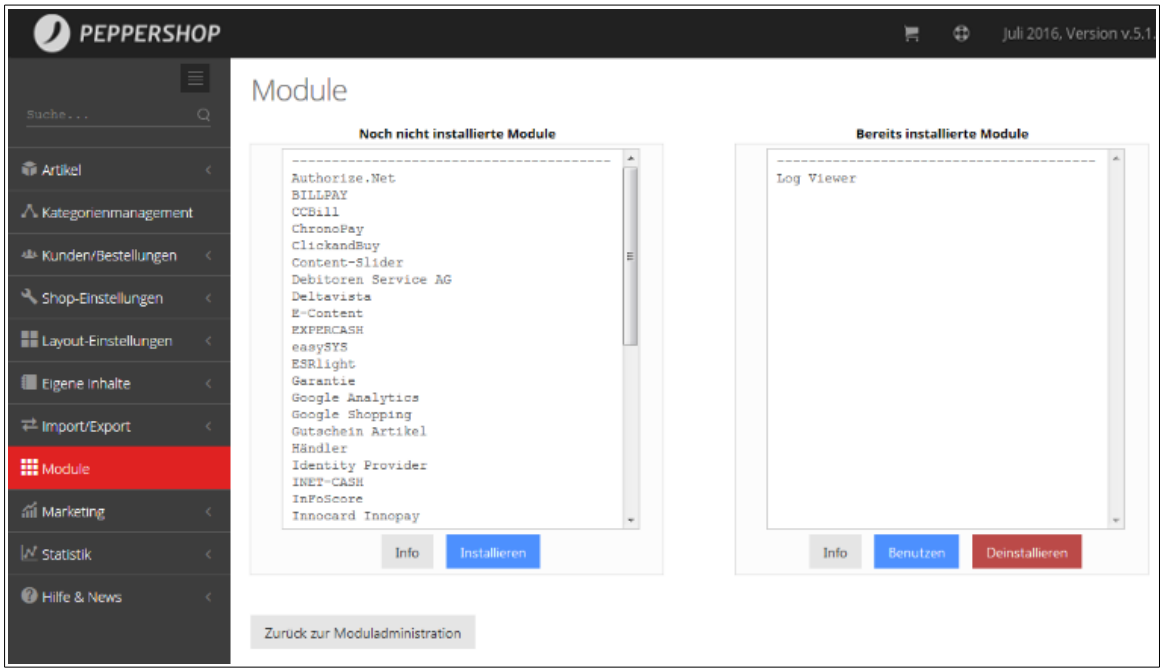


Abbildung 3: Modulverwaltung in der Shop-Administration

5.2 Der Installationsablauf

Die PepperShop Modulverwaltung beinhaltet die Funktion `install_module($modul_referenz)`. Diese Funktion geht wie folgt vor um ein Modul ins aktuelle System einzubinden.

5.2.1 Checks

Zuerst wird überprüft, ob der Installationstatus dieses Moduls auf not_installed gesetzt ist, resp. ob dieses Modul schon teilweise installiert wurde.

Danach folgt ein Test ob dieses Modul für die verwendete PepperShop-Version überhaupt zugelassen ist. Danach werden alle benötigten anderen Module (required Modules, Submodules) auf Anwesenheit überprüft. Falls ein benötigtes Modul fehlt, gibt es einen Hinweis und Abbruch.

Jetzt wird der Security Level des Moduls überprüft. Dazu werden die `security_id`'s der Interfaces, bei welchen sich dieses Modul registrieren will mit der eigenen `security_id` verglichen. Will das Modul sich bei einem Interface mit höherer `security_id` eintragen, so schlägt die Installation fehl. Die `security_id` ist kein Schutz um Hacker aus dem System zu sperren. Sie dient lediglich dazu Module, welche in Kontakt mit Kunden kommen von z.B. Backoffice Schnittstellen zu trennen. Auf diese Weise können ev. kompromittierte Module gar nicht erst in Bereiche eindringen, in welchen sie nichts zu suchen haben.

Weiter wird überprüft, ob das zu installierende Modul auch die vom PepperShop verwendeten Sprachen und Währungen (Locale) unterstützt. Wenn nicht – Meldung und Abbruch (Achtung: Dieser Check ist erst in einer kommenden API-Version vorgesehen).

5.2.2 Installationsarbeiten

In der Tabelle `module_mainmodule` wird für das Modul eine neue Zeile eingefügt mit allen skalaren Einträgen (siehe Kapitel 6.3).

Zwischen allen Schritten wird der `install_status` in der Tabelle `module_mainmodule` modifiziert, so dass jederzeit ein Rollback gemacht werden könnte.

Nun gibt es fünf Funktionen welche dem Namen entsprechend eine Aufgabe erfüllen:

1.) `install_module_mainmodule()`
Skalare Werte der Konfiguration in der Tabelle `module_mainmodule` speichern

2.) `install_module_tables()`: Eigene Modultabellen anlegen

3.) `extend_existing_tables()`
Check ob diese Tabelle erweitert werden darf (blacklist)
Tabellen erweitern / Abbruch

4.) `register_interfaces()`
Die Module werden bei den jeweiligen Interfaces registriert (Verbindungstabelle `module_interfaces_module`)

Der Fingerprint des Moduls wird in der Modultabelle gesetzt, indem man von den Moduldateien einen `md5`–Fingerprint erstellt und das aktuelle Datum hinzu codiert. (Kommt erst in einer späteren API Version hinzu).

Meldung, dass das Modul installiert sei.

6. Mein neues Modul - Quick Reference

Nun folgt eine kurze Anleitung, wie man am schnellsten ein eigenes Modul erstellen kann.

6.1 Modulentwicklung zusammengefasst

- Userdaten in `{shopdir}/shop/module/modul_name`
- Admindaten in `{shopdir}/shop/Admin/module/modul_name`
- Für jedes Interface gibt es eine eigene Handlerfunktion. Eine Handlerfunktion kann auch für mehrere Interfaces verwendet werden. Wenn das Interface sich im Userbereich befindet, muss auch die Handlerfunktion im Userbereich sein. Damit man eine Handlerfunktion im Userteil unterbringen kann, muss man dort mindestens eine Datei erstellt haben.
- Die Datei `initialize_module.php` liegt im Adminteil und muss korrekt ausgefüllt sein. Infos über 'sein' Modul, kann man in der Modulverwaltung im Administrationsmenü über den Klick auf 'Info' holen (ansehen, was man in der `initialize_module.php` konfiguriert hat).
- Damit die Administration vom Shopadministrator vorgenommen werden kann, benötigt man im Adminteil mindestens zwei Dateien:
    1. Icon (vorzugsweise im GIF-Format) mit 48 x 48px (96dpi Auflösung), möglichst mit transparentem Hintergrund (nicht einfach weiss!).
    2. Administration des Moduls in einer aufzurufenden Datei.
- Testing: Am besten man installiert das Modul zuerst selbst und testet seine Funktionalität.

Im nächsten Kapitel wird die Erstellung eines Moduls anhand des bestehenden Testmoduls '`persistent_interface_counter`' erläutert, dies hilft oft bei der Erstellung des ersten Moduls. Falls das Gutscheine Modul installiert ist, erhält man eine gute Vorlage mit moderner Widgets-Nutzung.

6.2 Erste Aufgaben

Das mitgelieferte Testmodul '`persistent_interface_counter`' dient als Grundlage für das neu zu erstellende Modul. Wir erstellen also zuerst mal eine Kopie des Moduls und benamsen die Ordner der

Kopie nach unserem Gusto:

`{shopdir}/shop/module/persistent_interface_counter/*` kopieren nach:
`{shopdir}/shop/module/mein_neues_modul`

Dasselbe mit dem Adminteil des `persistent_interface_counter` Moduls:

`{shopdir}/shop/Admin/module/persistent_interface_counter/*` kopieren nach:
`{shopdir}/shop/Admin/module/mein_neues_modul`

Wir haben nun in den beiden Ordnern ein neues Modul namens 'mein_neues_modul' Hier sollte natürlich eine treffendere Bezeichnung gewählt werden. Wir finden folgende Dateien in unserem neuen Verzeichnis:

```
{shopdir}/shop/module/mein_neues_modul/
                        interface_counter_functions.php
{shopdir}/shop/Admin/module/mein_neues_modul/
                        initialize_module.php
                        module_admin_img.gif
                        show_interface_counter.php
```

Fix bleibt der Name der Datei `initialize_module.php`. In dieser Datei müssen wir später Punkt für Punkt alle Einstellungen vornehmen, so dass unser neues Modul auch automatisch installiert und deinstalliert werden kann, doch dazu genaueres im nächsten Kapitel.

Die Datei `interface_counter_functions.php` befindet sich im User-Security Bereich und wird aufgerufen, wenn die registrierten Module durchlaufen werden (siehe dazu die Interface-Konfiguration in `initialize_module.php`). Wenn wir also z.B. beim Interface 2 vorbeikommen, so wird unsere Funktion welche wir angegeben haben aufgerufen - und zwar aus der Datei, die wir angegeben haben. Hierzu die Beispiele aus dem Testmodul:

Interface Konfiguration:

```
$moduldef['interfaces'][1]['interface_id'] = '2';
$moduldef['interfaces'][1]['file_with_function'] = 'interface_counter_functions.php';
$moduldef['interfaces'][1]['function_name'] = 'update_interface_counter';
$moduldef['interfaces'][1]['filter_typ'] = 'one_way';
```

Ausschnitt aus der Funktionsdefinition in der Datei `interface_counter_functions.php`:

```
// Die Funktion update_interface_counter fuehrt Buch ueber die Aufrufe der Interfaces, fuer
// welche dieses Modul (Persistent Interface Counter) registriert ist. Wenn noch kein Count-
// Eintrag in der Tabelle persistent_interface_counter_counts vorhanden ist, wird eine neue
// Zeile erstellt, ansonsten wird der bestehende Counterstand inkrementiert.
// Argumente: Datenarray (Array), Elementbeschreibung:
//            'interface         = Daten des Interfaces, welches diese Funktion aufgerufen hat
//            'interface_object' = Das Objekt, um welches es beim Interface gerade geht
//          [ 'session_id'       = Session-ID des Kunden, so sie vorhanden ist(!) ]
// Rueckgabe: Keine
function update_interface_counter($data_array) {

      // Beispiel eines Zugriffs auf das Interface-Objekt
      debug($data_array['interface_object']);
      debug($data_array['interface']->name);
      debug($data_array['session_id']);
```

Das Argument der aufgerufenen Funktion ist immer ein assoziativer Array. Unter 'interface' findet man das Interface selbst. Es kann benötigte Statusinformationen zum Interface angeben. Viel wichtiger ist aber das Interface-Objekt, welches immer als Element 'interface_object' im Array an die aufgerufene Funktion übergeben wird. Dieses Objekt enthält die vom Interface bereitgestellten Daten um die es eigentlich geht. Dies kann alles mögliche sein - ein Artikelobjekt, ein Bestellungsobjekt, ... Wenn man eine Filterfunktion implementiert, so kann man das Interface-Objekt am Ende wieder zurückgeben Wenn es hingegen eine one-way Funktion ist, ist es egal, was man am Ende zurück gibt. Wenn verfügbar wird unter 'session_id' noch die Session-ID des Kunden mit übergeben, z.B. so: `get_correct_session_id()`.

Nun können wir auch noch die Adminverwaltung machen - dies wäre im Moment noch die Datei `show_interface_counter.php`, welche man natürlich durch eine eigene Datei ersetzen kann. Dann darf allerdings nicht vergessen werden den entsprechenden Konfigurationseintrag in der `initialize_module.php` anzupassen (admin_menu_name). Nachdem dann auch noch das Bild (admin_menu_img) erstellt wurde, kann man zum nächsten Kapitel übergehen und alle Einstellungen angeben.

6.3 initialize_module.php

Hier wird der konfigurativ relevante Teil der `initialize_module.php` des Testmoduls 'Persistent Interface Counter' angezeigt. Man kann den Kommentaren auch gleich die jeweils benötigten Formatangaben entnehmen.

```
// ------------------------------------------------------------------------
// API Version der hier verwendeten PepperShop Modulschnittstelle:
$pps_module_api = '1.0';

// Informationen zu externen PepperShop Modulen:
// =============================================
// Module bestehen mindestens aus zwei Verzeichnissen: {shopdir}/shop/module/modul_name
// und {shopdir}/shop/Admin/module/modul_name. Im Admin-Modulverzeichnis muss mindestens
// diese Datei (initialize_module.php) vorhanden sein.
// - Ein Modul kann aber ohne weiteres auch eigene Unterverzeichnisse besitzen, sowie auch
//   Submodule haben.
// - Damit Module Daten persistent speichern koennen, duerfen sie waehrend der Installation
//   eigene Tabellen erstellen und bestehende um eigene Attribute erweitern. Bei einer De-
//   installation werden diese Datenbankerweiterungen wieder entfernt.
// - User-Security Scripte befinden sich in {shopdir}/shop/module/modul_name,
//   die Admin-Security-Scripts befinden sich im Admin-Pendant.
// - Module koenne (zumindest im Moment) noch keine eigenen Interfaces haben.
// - Module koennen weitere Module als Voraussetzung angeben.

// Definition der Variablen:
// =========================
// Die weiter unten definierten Variablen dienen der Beschreibung des Moduls. Diese muss
// sehr ausfuehrlich sein, damit das automatisierte Installations- und Deinstallationsscript
// durchlaufen kann. Nomenklatur:
// x.) Bezeichnung    : Leitet eine weitere Definition ein. x ist eine Laufnummer
// ! Beschreibung     : Beschreibung umschreibt Hinweise zum Thema der Bezeichnung
// --> Einschraenkung : Mit --> werden ZWINGEND ZU BEFOLGENDE Einschraenkungen der Bezeichnung genannt

// ------------------------------------------------------------------------------------
// ****************************** DEFINITION DES MODULS ******************************
// ------------------------------------------------------------------------------------

//  1.) Name des Moduls (entspricht dem Verzeichnisname des Moduls)
//      --> Der Name eines externen PepperShop Moduls darf hoechstens 40 Zeichen lang sein.
//      --> Der Name muss mindestens 3 Zeichen lang sein.
//      --> Er darf NUR aus alphanummerischen Zeichen und dem Underscore Zeichen (_) bestehen.
//      --> Dieser Name ist gleichzeitig auch der Name des Verzeichnisses des Moduls.
$moduldef['modulname'] = 'persistent_interface_counter';

// 2.) Bezeichnung des Moduls
//      ! Dies ist die Bezeichnung des Moduls und unterliegt somit weniger Restriktionen als der Modulname
//      --> Der Name darf hoechstens 40 Zeichen lang sein
//      --> Der Name muss mindestens 3 Zeichen lang sein.
$moduldef['modulbezeichnung'] = 'Persistenter Interface Counter';

// 3.) Versionschecknummern
//      ! Diese Nummern definieren die zu verwendende PepperShop Versionen. Die Versionisierung
//        ist wie folgt: Die erste und zweite Nummer (durch Punkt getrennt) ergeben ein Release.
//        Jedes unterstuetzte Release muss explizit angegeben werden. Die dritte (durch einen
//        Punkt getrennte Nummer (eigentlich ein String), definiert Versionen des Releases. Alle
//        Versionen eines Releases sind kompatibel, es sei denn man definiert auch die Versions-
//        Nummer, dann sind alle aelteren Versionen des angegebenen Releases inkompatibel.
//        Bsp. 1.4;1.5     = Das Modul ist kompatibel zu den Releases 1.4 und 1.5. Dies schliesst auch
//                           alle Versionen der beiden Releases mit ein: 1.4.003, 1.4.004, 1.5.1, ...
//        Bsp. 1.4.005;1.5 = Hier sind alle Versionen von 1.4 mit und nach 1.4.005 kompatibel und
//                           alle Versionen von 1.5.
//      ! Achtung: Man sollte keine zukuenftigen Releases angeben!
//      --> Einzelne Versionen via Strichpunkt getrennt eingeben.
$moduldef['versionschecknummern'] = '1.4.005;1.5;1.6;1.7;1.8;1.9;2.0;2.1;2.5;2.6;2.7;3.0;3.1;3.2;4.0;5.0';

//  4.) Kurzbeschreibung
//      ! Formatierungen sollen via HTML-Tags eingegeben werden.
$moduldef['kurzbeschreibung'] = '<b>ACHTUNG: DIES IST EIN TESTMODUL - NICHT IN PRODUKTIVEN SHOPS EINSETZEN!</b><br>
                                 Dieses Testmodul z&auml;hlt die Aufrufe der einzelnen Module, bei welchen
                                 es registriert ist. Es wird eine eigene Tabelle erzeugt und es gibt ein
                                 kleines Adminmen&uuml; um die Resultate ansehen zu k&ouml;nnen. Dies ist
                                 ein Testmodul, mit welchem man &uuml;ben kann, eigene externe PepperShop
                                 Module zu erstellen. Bitte nicht in produktiven Systemen installieren, da
                                 durch dieses Modul die Performance beim Bestellabschluss etwas vermindert wird.
                                ';

//  5.) Weiterfuehrender Link
//      ! Wenn dieser (optionale) Link angegeben ist, so kann der Shopadmin hier weitere Infos zum Modul holen.
//      --> Das Schema muss vor der URL angegeben werden (Schema = http:// oder https://, ...)
$moduldef['weitere_infos_link'] = 'http://www.peppershop.com/';
```

```
//  6.) Version dieses Moduls
$moduldef['modulversion'] = '1.0';

//  7.) Releasedatum dieser Modulversion
//      --> Format: TT.MM.JJJJ
$moduldef['releasedatum'] = '08.08.2012';

//  8.) Informationen zu den Entwicklern
//      ! Beispiel: JosÃ© Fontanil <fontajos@peppershop.com>. Strings in <> werden als E-Mail angezeigt.
$moduldef['entwickler_infos'] = 'Jos&eacute; Fontanil <fontajos@peppershop.com>';

//  9.) Ist Submodul von
//      ! Hier kann man den Modulnamen (nicht die Modulbezeichnung!) des Hauptmoduls angeben, falls dieses
//        Modul hier ein Submodul des Hauptmoduls ist.
//      --> Der Name eines externen PepperShop Moduls darf hoechstens 40 Zeichen lang sein.
//      --> Der Name muss mindestens 3 Zeichen lang sein.
//      --> Er darf nur aus alphanummerischen Zeichen und dem Underscore Zeichen (_) bestehen.
//      --> Dieser Name ist gleichzeitig auch der Name des Verzeichnisses des HAUPTmoduls.
$moduldef['submodule_of'] = '';

// 10.) Fingerprint
//      ! Im Moment noch nicht benutzt - spaeter wird hier ein MD5 Digest hinterlegbar sein, welcher dem
//        Shopadministrator erlaubt die Integritaet eines Moduls zu ueberpruefen.
//      --> MD5 Digest (32 Chars Laenge, Hexadezimales Alphabet)
$moduldef['fingerprint'] = '22b66409b0554640691d3d27541edfbb';

// 11.) Unterstuetzte Locales (Sprachen und optional Laender) - dient (vorerst) nur zur Anzeige fuer den Shopadmin
//      --> Format: ISO-639-1 fuer alleinstehende Sprachen (Bsp. de;en;fr;sp;...)
//      --> Format: ISO-639-2 fuer Sprachen inkl. Laender (Bsp. de_CH;de_DE;en_GB;en_US)
//      --> Wenn das Modul weder Sprach-, noch Laenderabhaengig ist kann all angegeben werden.
//      --> Die einzelnen Angaben koennen Strichpunkt separiert eingegeben werden. ISO-639-1 und -2 koennen gemixt werden.
$moduldef['locales'] = 'all';

// 12.) Interfaces, bei welchen sich das Modul registrieren soll
//      ! Dies ist ein etwas komplexerer Eingabetyp - es ist ein mehrdimensionaler Array - mehr nicht.
//      ! Pro Interface, bei welchem sich das Modul registrieren will, sind vier Angaben noetig:
//        (1) Interface_ID, (2) Datei, worin sich die auszufuehrende Funktion befindet,
//        (3) Name der auszufuehrenden Funktion, (4) Filtertyp
//      --> Format: array('i_id'=>'w','file'=>'x','func'=>'y','filter'=>'z')
//          --> w = Interface_ID, Format: positive Integerzahl (max. Digits == 11)
//          --> x = Dateiname, Format: Dateiname.Extension (kein Pfad)
//          --> y = Funktionsname, Format: Name der Funktion ohne Klammern mit Argumenten
//          --> z = Filtertyp, Format: one_way oder filter
// Registrierung beim ersten Interface:
$moduldef['interfaces'][0]['interface_id'] = '1';
$moduldef['interfaces'][0]['file_with_function'] = 'interface_counter_functions.php';
$moduldef['interfaces'][0]['function_name'] = 'update_interface_counter';
$moduldef['interfaces'][0]['filter_typ'] = 'one_way';
// Registrierung beim zweiten Interface:
$moduldef['interfaces'][1]['interface_id'] = '2';
$moduldef['interfaces'][1]['file_with_function'] = 'interface_counter_functions.php';
$moduldef['interfaces'][1]['function_name'] = 'update_interface_counter';
$moduldef['interfaces'][1]['filter_typ'] = 'one_way';

// 13.) Eigene Tabellen, welche angelegt werden sollen
//      ! Hier werden die eigens fuer dieses Modul zu erstellenden Tabellen angegeben
//      ! Wenn keine Tabellen erstellt werden muessen, einfach leerer Array definieren
//      --> Achtung: Eine Tabelle muss mindestens EIN Attribut besitzen, sonst wird sie nicht angelegt.
//      --> Format: array('table_name'=>'x','table_beschreibung'=>'y','attribute'=>z)
//          --> x = Name der Tabelle: MySQL Restriktionen (max. 64 Zeichen, keine Sonderzeichen, ...)
//          --> y = Beschreibung der Tabelle: Alphanummerische Zeichen, Kurzbeschrieb des Zwecks
//          --> z = Die Attribute der Tabelle, Format:
//                  array('name'=>a,'typ'=>b,'laenge'=>c,'zusatz'=>d,'null'=>e,'default'=>f,'extra'=>g,
//                        'primary'=>h,'index'=>i,'unique'=>j,'volltext'=>k,'beschreibung'=>l)
//                  --> a = Name des Attributs: (Alphanummerische Zeichen, siehe reservierte Woerter von MySQL)
//                  --> b = Typ: Datentyp dieses Tabellenattributs (z.B. int, varchar, text, ...)
//                  --> c = Laenge: Positive Integerzahl oder leer lassen (manchmal auch als maxlength interpretiert)
//                  --> d = Zusatz: '' | 'BINARY' | 'UNSIGNED' | 'UNSIGNED ZEROFILL'
//                  --> e = Null Setting: 'NULL | 'NOT NULL'
//                  --> f = Default: Defaultwert bei Neuerstellung in einer Zeile (max. Zeichenlaenge = 255)
//                  --> g = Extra: '' | 'auto_increment'
//                  --> h = Primary: '0' = ist NICHT Primary Key | '1' = IST Primary Key
//                  --> i = Index: '0' = Nein | '1' = Ja
//                  --> j = Unique: '0' = Nein | '1' = Ja
//                  --> k = Volltext Index: '0' = Nein | '1' = Ja (nicht bei allen Typen moeglich)
//                  --> l = Beschreibung: Wird nur hier und im Modulprozess verwendet (max. Chars = 255)
// Beschreibung der ersten eigenen Tabelle:
$moduldef['eigene_tabellen'][0]['table_name'] = 'counts';
$moduldef['eigene_tabellen'][0]['table_beschreibung'] = 'Enthaelt die Counter aller Interfaces';
// Attribut 1 der Tabelle:
$moduldef['eigene_tabellen'][0]['attribute'][0]['name'] = 'count_id';
$moduldef['eigene_tabellen'][0]['attribute'][0]['typ'] = 'int';
$moduldef['eigene_tabellen'][0]['attribute'][0]['laenge'] = '11';
$moduldef['eigene_tabellen'][0]['attribute'][0]['zusatz'] = '';
$moduldef['eigene_tabellen'][0]['attribute'][0]['null'] = 'NOT NULL';
$moduldef['eigene_tabellen'][0]['attribute'][0]['default'] = '';
$moduldef['eigene_tabellen'][0]['attribute'][0]['extra'] = 'auto_increment';
$moduldef['eigene_tabellen'][0]['attribute'][0]['primary'] = '1';
$moduldef['eigene_tabellen'][0]['attribute'][0]['index'] = '1';
$moduldef['eigene_tabellen'][0]['attribute'][0]['unique'] = '1';
$moduldef['eigene_tabellen'][0]['attribute'][0]['volltext'] = '0';
$moduldef['eigene_tabellen'][0]['attribute'][0]['beschreibung'] = 'Primary Key';
// Attribut 2 der Tabelle:
$moduldef['eigene_tabellen'][0]['attribute'][1]['name'] = 'interface_name';
$moduldef['eigene_tabellen'][0]['attribute'][1]['typ'] = 'varchar';
$moduldef['eigene_tabellen'][0]['attribute'][1]['laenge'] = '255';
$moduldef['eigene_tabellen'][0]['attribute'][1]['zusatz'] = '';
$moduldef['eigene_tabellen'][0]['attribute'][1]['null'] = 'NOT NULL';
$moduldef['eigene_tabellen'][0]['attribute'][1]['default'] = '';
```

```
$moduldef['eigene_tabellen'][0]['attribute'][1]['extra'] = '';
$moduldef['eigene_tabellen'][0]['attribute'][1]['primary'] = '0';
$moduldef['eigene_tabellen'][0]['attribute'][1]['index'] = '0';
$moduldef['eigene_tabellen'][0]['attribute'][1]['unique'] = '0';
$moduldef['eigene_tabellen'][0]['attribute'][1]['volltext'] = '0';
$moduldef['eigene_tabellen'][0]['attribute'][1]['beschreibung'] = 'Namen des Interfaces';
// Attribut 3 der Tabelle:
$moduldef['eigene_tabellen'][0]['attribute'][2]['name'] = 'fk_interface_id';
$moduldef['eigene_tabellen'][0]['attribute'][2]['typ'] = 'int';
$moduldef['eigene_tabellen'][0]['attribute'][2]['laenge'] = '11';
$moduldef['eigene_tabellen'][0]['attribute'][2]['zusatz'] = '';
$moduldef['eigene_tabellen'][0]['attribute'][2]['null'] = 'NOT NULL';
$moduldef['eigene_tabellen'][0]['attribute'][2]['default'] = '';
$moduldef['eigene_tabellen'][0]['attribute'][2]['extra'] = '';
$moduldef['eigene_tabellen'][0]['attribute'][2]['primary'] = '0';
$moduldef['eigene_tabellen'][0]['attribute'][2]['index'] = '0';
$moduldef['eigene_tabellen'][0]['attribute'][2]['unique'] = '0';
$moduldef['eigene_tabellen'][0]['attribute'][2]['volltext'] = '0';
$moduldef['eigene_tabellen'][0]['attribute'][2]['beschreibung'] = 'Interface_ID des Interfaces';
// Attribut 4 der Tabelle:
$moduldef['eigene_tabellen'][0]['attribute'][3]['name'] = 'counts';
$moduldef['eigene_tabellen'][0]['attribute'][3]['typ'] = 'int';
$moduldef['eigene_tabellen'][0]['attribute'][3]['laenge'] = '11';
$moduldef['eigene_tabellen'][0]['attribute'][3]['zusatz'] = '';
$moduldef['eigene_tabellen'][0]['attribute'][3]['null'] = 'NOT NULL';
$moduldef['eigene_tabellen'][0]['attribute'][3]['default'] = '';
$moduldef['eigene_tabellen'][0]['attribute'][3]['extra'] = '';
$moduldef['eigene_tabellen'][0]['attribute'][3]['primary'] = '0';
$moduldef['eigene_tabellen'][0]['attribute'][3]['index'] = '0';
$moduldef['eigene_tabellen'][0]['attribute'][3]['unique'] = '0';
$moduldef['eigene_tabellen'][0]['attribute'][3]['volltext'] = '0';
$moduldef['eigene_tabellen'][0]['attribute'][3]['beschreibung'] = 'Anzahl counts, wie oft dieses Interface schon aufgerufen wurde.';

// 14.) Zu erweiternde, schon bestehende Tabellen
//      ! Hier werden die Tabellen beschrieben, welche schon existieren und durch weitere Attribute
//        erweitert werden sollen.
//      ! Wenn keine Tabellen erweitert werden sollen, einfach leerer Array definieren
//      --> Format: Dasselbe Format wie bei $moduldef['eigene_tabellen']. Die Beschreibung
//                  einer schon bestehenden Tabelle wird ignoriert, der Name muss aber stimmen.
$moduldef['erweiterte_tabellen'] = array();

// 15.) Submodule
//      ! Wenn dieses Modul aus mehreren weiteren Modulen besteht, so koennen diese hier angegeben werden.
//      ! Wenn keine Submodule existieren, einfach einen leeren String angeben.  (Strichpunkt getrennte Liste)
//      --> Format: 'submodul_name_1;submodul_name2;...;submodul_namex'
$moduldef['submodule'] = '';

// 16.) Vorausgesetzte Module
//      ! Hier werden Module angegeben, welche korrekt installiert vorhanden sein muessen, damit dieses
//        Modul ueberhaupt erst installiert wird. (Strichpunkt getrennte Liste)
//      ! Wenn keine solchen Module gibt, einfach einen leeren String uebergeben
//      --> Format: 'required_modul_name_1;required_modul_name2;...;required_modul_namex'
$moduldef['required_modules'] = '';

// 17.) Security ID
//      ! Mit der Security_ID kann man dem Modul den Zugang zu verschiedenen Interfaces sperren.
//        Auf diese Weise kann ein kompromittiertes Modul nur begrenzt Schaden anrichten.
//      ! Die niedrigste Stufe der Security_ID ist = 1, die hÃ¶chste Stufe = 32768. Je hÃ¶her die
//        angegebene Security_ID ist, desto hÃ¶her ist auch die Zahl der erlaubten Interfaces
//      ! Welches Interface, welche minimale Security_ID erfordert um benutzt werden zu koennen,
//        ist in der Tabelle module_interfaces mit den Interfaceeintraegen ersichtlich.
//      --> Format: 'required_modul_name_1;required_modul_name2;...;required_modul_namex'
$moduldef['security_id'] = '2';

// 18.) Valid Hosts
//      ! Erweiterte Security wird es in der naechsten API-Version noch mit der valid_hosts Angabe geben.
//        Die Datenbank ist dafuer schon vorbereitet. (all = Alle Hosts, im Moment die Standardeinstellung)
//        Ausgewertet wird die Angabe aber noch nicht.
//      --> Format: all = Alle hosts | localhost = nur dieser Rechner | mehrere Rechner via ; getrennt angeben
$moduldef['valid_hosts'] = 'all';

// 19.) Name des Administrationsmenus
//      ! Im Administrationstool hat das Verwaltungsmenu dieses Moduls einen Namen, hier kann man
//        einen Namen definieren, wenn man keinen angibt, wird einfach die Modulbezeichnung verwendet
//      --> Format: Maximale Laenge 40 Zeichen, moeglichst keine Sonderzeichen verwenden
$moduldef['admin_menu_name'] = 'Interface Counter verwalten';

// 20.) URL zur Datei, wo das Admin-Verwaltungsmenu liegt
//      ! Diese URL ist entweder absolut oder (besser) relativ zum {shopdir}/shop/Admin/module/modul_name Verzeichnis
//        Die hier angegebene Datei wird 'verlinkt' und mit dem in 'admin_menu_name' Namen versehen.
//        Info: Achtung: Jeder Link in dieser Datei muss folgende GET-Parameter mitgeben:
//        - darstellen=".$HTTP_GET_VARS['darstellen']
//        - installed_selection=".$HTTP_GET_VARS['installed_selection']
//        - backlink=".$HTTP_GET_VARS['backlink']
//      --> Format: URL
$moduldef['admin_menu_link'] = 'show_interface_counter.php';

// 21.) URL zum Icon des Adminmenus
//      ! Diese URL ist relativ zum {shopdir}/shop/Admin/ Verzeichnis (sonst gibt es einen include-Fehler
//        Die hier angegebene Datei wird 'verlinkt' und mit dem in 'admin_menu_name' Namen versehen.
//      --> Format: URL
//      --> Format Icon: 48px x 48px, GIF oder PNG oder JPG.
```

```
$moduldef['admin_menu_img'] = 'modul_admin_img.gif';

// 22.) Pruefung, ob das Modul vorhanden ist
//      ! Hier wird zumeist mit einem modul_check Aufruf geprueft, ob dieses Modul verfuegbar ist. Das dritte Argument
//        sollte auf false (Boolean) gesetzt werden, so dass kein Include des Moduls gemacht wird.
//      --> Format: Boolean (true | false)
$moduldef['is_available'] = modul_check('Admin/module/persistent_interface_counter/show_interface_counter.php','',false,false);

// ----------------------------------------------------------------------------------
// *************************** ENDE DEFINITION DES MODULS ***************************
// ----------------------------------------------------------------------------------
```

Der Modulname (hier `persistent_interface_counter`) ist also eigentlich der Name des Verzeichnisses. Der Name im üblichen Sinne ist die Modulbezeichnung (Persistenter Interface Counter). Dies sollte man nie verwechseln, da der Modulname sehr restriktive Formatierungsregeln zu folgen hat und intern oft zur Referenzierung verwendet wird.

Achtung: Speziell bei der Definition der Tabellen muss man selbst auf die Einhaltung von SQL-Spezifikationen achten (nicht mehr als zwei Primary Keys pro Tabelle, u.a.)... MySQL wird sonst beim Versuch des CREATEs eine Fehlermeldung ausgeben.

Achtung: Im Moment sollte man möglichst auf Submodule verzichten, da die Implementation der Submodule-Installation und Deinstallation sehr einfach gehalten ist. Die Prozeduren funktionieren im Prinzip, aber sie sind nicht sehr fehlertolerant.

Beim Punkt '*is_available*' sollte man rechts daneben einen modul_check Call aufrufen. Bitte nicht vergessen hinten anfügen, dass kein Include der Datei erfolgen soll.

Achtung: Wenn man locales verwendet, gilt zu beachten, dass die Installationsroutine im Moment keine Einträge in die Verbindungstabelle `module_locale` vornimmt.

6.4 deinstall_module.php

Diese Datei beherbergt die Definition einer einzelnen Funktion. Wenn diese Datei und die Funktion somit existiert, wird diese vor dem Deinstallationsprozess aufgerufen. Hier kann man also Deaktivierungs- und Aufräumarbeiten durchführen. Die Funktionsdefinition sieht wie folgt aus:

```
// Folgende Methode wird bei einer Deinstallation eines externen PepperShop
// Moduls automatisch aufgerufen, wenn die Deinstallation gestartet wurde.
// ACHTUNG: Die Rueckgabe wird vom Modulsystem ausgewertet. Bei false wird
// der Deinstallationsprozess unterbrochen!
// Argumente: &$module (Referenz auf ein Module Objekt)
// Rueckgabe: true | false (Boolean)
function deinstall_peppershop_module(&$module) {
    // Rueckgabewert initialisieren
    $result = true;

    // Hier koennen z.B. noch Aufraeumarbeiten durchgefuehrt werden...

    return $result;
}// End function deinstall_peppershop_module
```

6.5 Links innerhalb der Modul-Verwaltung

Wenn man innerhalb des Moduls mehrere HTML-Seiten hat, welche miteinander interagieren sollen, so kann man beim Modul `log_viewer` (nur im Adminbereich vorhanden) nachsehen, wie hier vorzugehen ist. Die Besonderheit liegt darin, dass die eigenen Dateien jeweils included werden und immer gewisse GET-Parameter mitgeben müssen. Siehe dazu auch die Funktion: `get_admin_link_addon()`.

7. ER-Diagramm des Modulsystems

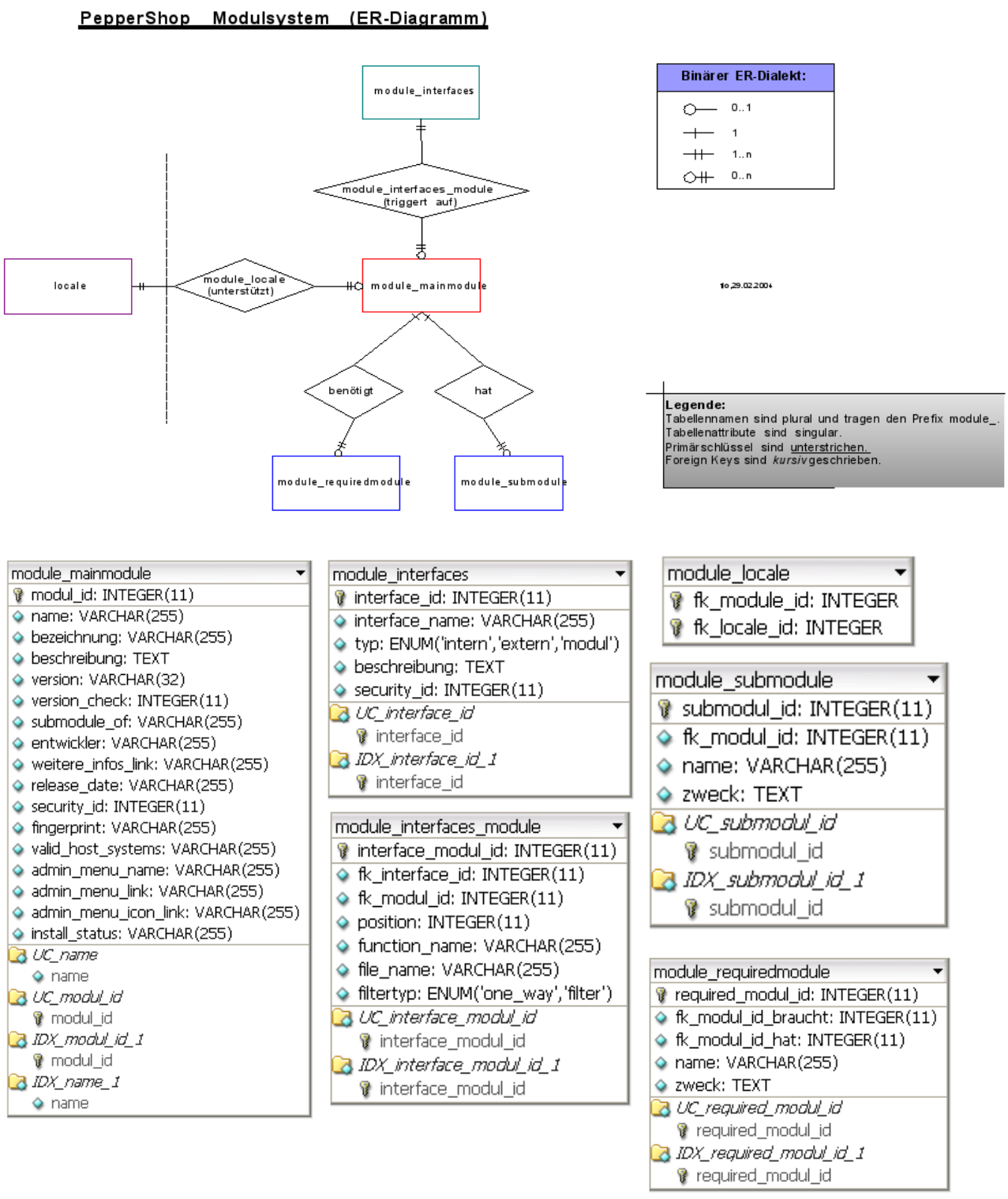


Abbildung: Datenbanktabellen des A-Modul Subsystems

8. CREATE SQLs des Modulssystems

Folgende CREATE Statements befinden sich am Ende der Datei `template_create.sql`:

```
CREATE TABLE module_mainmodule
(
    modul_id INT(11) NOT NULL AUTO_INCREMENT,
    name CHAR(255) NOT NULL DEFAULT 'undefined',  # /* Entspricht auch dem Ordner des Moduls */
    bezeichnung VARCHAR(255) NOT NULL DEFAULT 'undefined',
    beschreibung TEXT NOT NULL,
    version CHAR(32) NOT NULL DEFAULT 'undefined',
    version_check INT(11) NOT NULL DEFAULT 0, # /* Int. um aelter als ... zu bestimmen*/
    submodule_of VARCHAR(255) NOT NULL DEFAULT '',
    entwickler VARCHAR(255) NOT NULL DEFAULT 'undefined', # /* Name | Firma | E-Mail | Page | ... */
    weitere_infos_link varchar(255) NOT NULL default '',
    release_date VARCHAR(255) NOT NULL DEFAULT '01.01.1970', # /* Veroeffentlichungsdatum des Moduls */
    security_id INT(11) NOT NULL DEFAULT 0, # /* Security Level des Moduls */
    fingerprint CHAR(255) NOT NULL DEFAULT 'undefined', # /* md5-Hash der Dateien | Link zum hash */
    valid_host_systems VARCHAR(255) NOT NULL DEFAULT 'all', # /* Fuer welche Systemvers. zugelassen. Delimiter ;*/
    admin_menu_name VARCHAR(255) NOT NULL DEFAULT 'undefined',
    admin_menu_link VARCHAR(255) NOT NULL DEFAULT 'undefined',
    admin_menu_icon_link VARCHAR(255) NOT NULL DEFAULT 'undefined',
    install_status VARCHAR(255) NOT NULL DEFAULT 'undefined',
    PRIMARY KEY (modul_id),
    UNIQUE UC_modul_id (modul_id),
    INDEX IDX_modul_id_1 (modul_id),
    UNIQUE UC_name (name),
    INDEX IDX_name_1 (name)
);

CREATE TABLE module_submodule
(
    submodul_id INT(11) NOT NULL AUTO_INCREMENT,
    fk_modul_id INT(11) NOT NULL DEFAULT 0, # /* Fremdschluessel aus Tabelle module_module */
    name CHAR(255) NOT NULL DEFAULT 'undefined', # /* Kopie - vereinfacht SQL-Abfragen */
    zweck TEXT NOT NULL, # /* Beschreibung: Wieso dieses Modul benoetigt wird */
    PRIMARY KEY (submodul_id),
    UNIQUE UC_submodul_id (submodul_id),
    INDEX IDX_submodul_id_1 (submodul_id)
);

CREATE TABLE module_requiredmodule
(
    required_modul_id INT(11) NOT NULL AUTO_INCREMENT,
    fk_modul_id_braucht INT(11) NOT NULL DEFAULT 0, # /* referenziert das Modul, welches ein weiteres benoetigt */
    fk_modul_id_hat INT(11) NOT NULL DEFAULT 0, # /* referenziert das benoetigte Modul */
    name CHAR(255) NOT NULL DEFAULT 'undefined', # /* Kopie: Name des benoetigten Moduls - vereinfacht SQL-Abfragen */
    zweck TEXT NOT NULL, # /* Beschreibung: Wieso dieses Modul benoetigt wird */
    PRIMARY KEY (required_modul_id),
    UNIQUE UC_required_modul_id (required_modul_id),
    INDEX IDX_required_modul_id_1 (required_modul_id)
);

CREATE TABLE module_interfaces
(
    interface_id INT(11) NOT NULL AUTO_INCREMENT,
    interface_name CHAR(255) NOT NULL DEFAULT 'undefined',
    typ ENUM('intern','extern','modul') NOT NULL DEFAULT 'intern', # /* Typ des Interfaces */
    beschreibung TEXT NOT NULL,
    security_id INT(11) NOT NULL DEFAULT 0, # /* Security Level des Interfaces */
    PRIMARY KEY (interface_id),
    UNIQUE UC_interface_id (interface_id),
    INDEX IDX_interface_id_1 (interface_id)
);

CREATE TABLE module_interfaces_module
(
    interface_modul_id INT(11) NOT NULL AUTO_INCREMENT,
    fk_interface_id INT(11) NOT NULL DEFAULT 0, # /* Fremdschluessel: Tabelle module_interfaces */
    fk_modul_id INT(11) NOT NULL DEFAULT 0, # /* Fremdschluessel: Tabelle module_mainmodule */
    position INT(11) NOT NULL DEFAULT 0,
    function_name VARCHAR(255) NOT NULL DEFAULT '',
    file_name VARCHAR(255) NOT NULL DEFAULT '',
    filtertyp ENUM('one_way','filter') NOT NULL DEFAULT 'filter', # /* Wie das Interface benutzt wird */
    PRIMARY KEY (interface_modul_id),
    UNIQUE UC_interface_modul_id (interface_modul_id),
    INDEX IDX_interface_modul_id_1 (interface_modul_id)
);

CREATE TABLE module_locale
(
    fk_module_id INT(11) NOT NULL,
    fk_locale_id INT(11) NOT NULL,
    PRIMARY KEY (fk_module_id, fk_locale_id),
    INDEX (fk_module_id, fk_locale_id)
);
```

9. INSERT Statements zweier Interfaces

Die Interfaces werden vom PepperShop Core Entwicklungsteam ausgearbeitet und ein-gefügt. Die INSERTS werden in der Datei `template_insert.sql` - am Ende - gespeichert.

```
# /* Insert der Modulinterfaces */

INSERT INTO module_interfaces (interface_id, interface_name, typ, beschreibung, security_id)
    VALUES (1, 'artikelobj_vor_artikel_ausgabe', 'intern',
    'Liefert ein Artikelobjekt vor dem Anzeigen im Content-Frame.', 1
);

INSERT INTO module_interfaces (interface_id, interface_name, typ, beschreibung, security_id)
    VALUES (2, 'bestellungsobj_nach_erfolgreicher_bestellung', 'intern',
    'Liefert ein Bestellungsobjekt nach dem Abschliessen einer erfolgreich verarbeiteten Bestellung.', 2
);
```

10. INSERT Statements eines Moduls

Folgende Insert-Statements werden bei der Installation automatisch erzeugt. Sie sollen hier nur eine Übersicht geben, welche Daten in etwa wohin gespeichert werden.

```
# /* INSERT Statements des Testmoduls Persistent Interface Counter */

INSERT INTO module_mainmodule (modul_id, name, subfolder, version, version_check, entwickler,
    release_date, beschreibung, security_id, fingerprint, valid_host_systems, admin_menu_name,
    admin_menu_link, admin_menu_icon_link)
    VALUES (1, 'Interface Counter', 'persistent_interface_counter', '0.1', 1,
    'José Fontanil <fontajos@peppershop.com>, PepperShop Projekt http://www.peppershop.com',
    '2003-09-09', 'Dieses Modul zählt die Aufrufe der einzelnen Interfaces.', 0, 'undefined',
    'all', 'Interface Counter', 'show_interface_counter.php', 'modul_admin_img.gif'
);

INSERT INTO module_interfaces (interface_id, interface_name, typ, beschreibung, security_id)
    VALUES (1, 'artikelobj_vor_artikel_ausgabe', 'intern',
    'Liefert ein Artikelobjekt vor dem Anzeigen im Content-Frame.', 0
);

INSERT INTO module_v_interfaces_module (interface_modul_id, fk_interface_id, fk_modul_id, position,
    function_name, file_name, filtertyp)
    VALUES (1, 1, 1, 0, 'update_interface_counter', 'update_interface_counter.php', 'one_way'
);

INSERT INTO module_tabellen (modul_tabelle_id, name, beschreibung)
    VALUES (1, 'persistent_interface_counter_counts', 'Enthält die Interface Counterstände.'
);

INSERT INTO module_tabellen_attribute (modul_tabellen_attribut_id, fk_modul_tabelle_id, name,
    typ_name, typ_laenge_set, typ_attribut, typ_null, typ_default, typ_extra, typ_primary,
    typ_index, typ_unique, typ_volltext, beschreibung)
    VALUES (1, 1, 'count_id', 'int', '11', '', 'NOT NULL', '', 'auto_increment', '1', '1', '1', '0', 'Primary Key'
);
INSERT INTO module_tabellen_attribute (modul_tabellen_attribut_id, fk_modul_tabelle_id, name,
    typ_name, typ_laenge_set, typ_attribut, typ_null, typ_default, typ_extra, typ_primary,
    typ_index, typ_unique, typ_volltext, beschreibung)
    VALUES (2, 1, 'interface_name', 'varchar', '255', '', 'NOT NULL', '', '', '0', '0', '0', '0', 'Namen des Interfaces'
);
INSERT INTO module_tabellen_attribute (modul_tabellen_attribut_id, fk_modul_tabelle_id, name,
    typ_name, typ_laenge_set, typ_attribut, typ_null, typ_default, typ_extra, typ_primary,
    typ_index, typ_unique, typ_volltext, beschreibung)
    VALUES (3, 1, 'fk_interface_id', 'int', '11', '', 'NOT NULL', '', '', '0', '0', '0', '0', 'Interface-ID des Interfaces'
);
INSERT INTO module_tabellen_attribute (modul_tabellen_attribut_id, fk_modul_tabelle_id, name,
    typ_name, typ_laenge_set, typ_attribut, typ_null, typ_default, typ_extra, typ_primary,
    typ_index, typ_unique, typ_volltext, beschreibung)
    VALUES (4, 1, 'counts', 'int', '11', '', 'NOT NULL', '0', '', '0', '0', '0', '0', 'Anzahl counts, wie oft dieses Interface schon
            aufgerufen wurde'
);

# /* Anlegen der von Persistent Interface Counter benoetigten Tabelle (sollte sonst bei der Installation des Moduls gemacht werden */

CREATE TABLE persistent_interface_counter_counts
(
    count_id INT(11) NOT NULL AUTO_INCREMENT,
    fk_interface_id INT(11) NOT NULL DEFAULT 0,
    interface_name VARCHAR(255) NOT NULL DEFAULT '',
    counts INT(11) NOT NULL DEFAULT 0,
    PRIMARY KEY (count_id),
    UNIQUE UC_count_id (count_id),
    INDEX IDX_count_id_1 (count_id),
    INDEX IDX_fk_interface_id_2 (fk_interface_id)
);
```

11. Fragen?

Bei Fragen bitte direkt im Forum auf http://www.peppershop.com posten. Kommerzieller Support kann in Form von Supportpaketen eingekauft werden.