# Composing Shops

*Modulinstallation und andere coole Composer Features*

*(auch) für den OXID eShop ... und ein bisschen Shopware ;)*

*Stefan Moises - stefan@rent-a-hero.de Tobias Merkl - merkl@proudsourcing.de*

*@upsettweety @tabsl*

# Composer



> *Composer is a tool for dependency management in PHP. It allows you to declare the libraries your project depends on and it will manage (install/update) them for you.*

***getcomposer.org***

# Composer und OXID / Shopware

# Composer und OXID



- **Composer Installers** unterstützt OXID u. Shopware

# Composer und Shopware



- man kann Front-/Backend- u. Core-Plugins sowie Themes im Shop “installieren”

| | |
|---|---|
| Roundcube | `roundcube-plugin` |
| shopware | `shopware-backend-plugin`<br>`shopware-core-plugin`<br>`shopware-frontend-plugin`<br>`shopware-theme`<br>`shopware-plugin`<br>`shopware-frontend-theme` |
| SilverStripe | `silverstripe-module`<br>`silverstripe-theme` |

# OXID Modul installieren



```
1   "require": {
2     "shoptimax/custmod": "1.*",
3     "composer/installers": "~1.2"
4   },
5   "extra": {
6     "installer-paths": {
7       "modules/{$vendor}/{$name}/": ["type:oxid-module"]
8     }
9   }
10
```

# ... sogar aus einer ZIP-Datei



```
"repositories": [
  {
    "type": "package",
    "package": {
      "name": "shoptimax/custmod",
      "type": "oxid-module",
      "version": "1.0.0",
      "dist": {
        "url": "http://oxid.dev.local/custmod.zip",
        "type": "zip"
```

# … sogar aus einer ZIP-Datei



- wichtig ist hier die **“type”**-Variable, die auf einen unterstützten Installer-Typ gesetzt werden muss
- die ZIP-Datei muss nicht einmal ein gültiges Composer-Repository beinhalten, selbst eine *“composer.json”* ist nicht nötig
- **Repository-“type”** muss allerdings *“package”* sein
- ändert man die ZIP-Datei, so muss man beim Entwickeln den **Composer Cache leeren**, damit die Datei neu heruntergeladen wird (“composer clear-cache”) bzw. die “version” ändern, s. hier - generell empfiehlt Composer allerdings wg. dieser Einschränkungen, diesen Repository-Typ nur im Notfall zu nutzen

# OXID Themes installieren



```
"require": {
  "shoptimax/custmod": "1.*",
  "shoptimax/custtheme": "1.*",
  "composer/installers": "~1.2"
},
"extra": {
  "installer-paths": {
    "modules/{$vendor}/{$name}/": ["type:oxid-module"],
    "application/views/{$name}/": ["type:oxid-theme"]
  }
```

# OXID Themes installieren



- *"type"* muss hier **"oxid-theme"** sein:

```
 "name": "shoptimax/custtheme",
"type": "oxid-theme",
"version": "1.0.0",
"dist": {
  "url": "http://oxid.dev.local/custtheme.zip",
  "type": "zip"
}
```

# Composer Installers Extender



- Composer Installers Extender erweitert die Composer Installers, um beliebige andere/eigene Packages ausserhalb des “vendor”-Verzeichnisses installieren zu können

```
"extra": {
    "installer-types": ["library"],
    "installer-paths": {
      "special/package/": ["my/package"],
      "path/to/libraries/{$name}/": ["type:library"]
    }
  }
```

# Composer in OXID 6

# Composer in OXID 6



- OXID 6 kann komplett per Composer installiert werden [Blogbeitrag dazu]

```
composer create-project
oxid-esales/oxideshop-project project_name dev-b-6.0-ce
```

- der Shop landet dann im Unterverzeichnis “source”

# Composer in OXID 6



- auch Module können mit den neuen Composer Plugins installiert werden
- wichtig sind hier “type” und “target-directory” (in “extra”):

```
"type": "oxideshop-module",
...
"extra": {
 "oxideshop": {
   "target-directory": "{$vendor}/{$name}"
```

# Composer in OXID 6



- Modul (und auch Shop Source-Code) wird bei der Installation kopiert (von “vendor/…” nach “source/modules/“)
- darf kein “modules/vendor/name”-Unterverzeichnis beinhalten,

  sonst wird das doppelt angelegt
- *“type”* ist hier **“oxideshop-module”** (bzw. “oxideshop-theme”), nicht “oxid-module” wie beim “Composer Installers”-Plugin!

# Composer Merge-Plugin

# Composer-Merge-Plugin



- manchmal benötigt ein Modul weitere Abhängigkeiten, z.B. *“symfony/yaml”*
- diese sollten ebenfalls automatisch per Composer installiert werden
- eine Möglichkeit ist das **“composer-merge plugin”**
- dieses kann aus beliebigen Verzeichnissen weiter “composer.json”-Dateien in die Haupt-“composer.json” mergen und so alle benötigten Abhängigkeiten installieren

```
1 "require": {
2  "shoptimax/custmod": "1.*",
3  "wikimedia/composer-merge-plugin": "dev-master"
4  },
```

# Composer-Merge-Plugin



- in der “extras”-Sektion kann das Plugin konfiguriert werden:

```
"extra": {
 "merge-plugin": {
   "include": [
     "modules/**/*/composer.json"
   ],
   "recurse": true,
   "replace": true,
   "merge-dev": true,
   "merge-extra": false,
```

# Composer-Merge-Plugin



- das zu installierende Plugin (im Unterverzeichnis) definiert seinerseits Abhängigkeiten, die beim Composer-Run im Hauptverzeichnis brav mit installiert werden. z.B.:

```
"extra": {
 "name": "shoptimax/custmod",
 "description": "a module",
 "type": "oxid-module",
 "version": "1.0.0",
 "require": {
 "composer/installers": "~1.2",
 "symfony/yaml": "*"
```

# Composer-Merge-Plugin



- manchmal muss man allerdings zweimal “composer update” aufrufen (evtl. wegen ZIP-Resourcen?)

# Composer Script Hooks

# Composer Script Hooks



- Composer bietet eine ganze Reihe von Events, für die man Hooks definieren kann

```
"scripts": {
  "post-autoload-dump": [
    "ioly\\IolyInstaller::postAutoloadDump"
  ]
},
```

- z.B. um nach der Modulinstallation direkt Module zu aktivieren, Views zu generieren, tmp zu leeren o.ä.

# Composer Script Hooks



```
1
2       public static function postAutoloadDump(Event $event)
3       {
4           $vendorDir = $event->getComposer()
5           ->getConfig()->get('vendor-dir');
6           ...
```

# Composer Script Hooks



## Command Events #

- **pre-install-cmd**: occurs before the `install` comma
- **post-install-cmd**: occurs after the `install` commar
- **pre-update-cmd**: occurs before the `update` commar
  is executed without a lock file present.
- **post-update-cmd**: occurs after the `update` comman
  command has been executed without a lock file presen
- **post-status-cmd**: occurs after the `status` command
- **pre-archive-cmd**: occurs before the `archive` comm
- **post-archive-cmd**: occurs after the `archive` comma
- **pre-autoload-dump**: occurs before the autoloader is d

# Composer - custom installer

# Custom Composer Installers



- Composer API unterstützt eigene Installer
- z.B. OXID installer plugin für OXID 6
- Der Composer Installer für ioly / omc (OXID Module Connector) kann automatisch über composer.json-Einträge ioly/omc-Module installieren

# Custom Composer Installers



```
"omc-composer-installer": {
  "oxidversion": "4.10",
  "cookbooks": {
    "omc": "https://github.com/OXIDprojects/OXID-Modul-Connec
  },
  "modules": {
    "jkrug/ocbcleartmp": "1.0.0-v47",
    "oxcom/oxcom-omc": "latest"
  },
```

# Exkurs - ioly

# Exkurs: ioly



- ioly OpenSource-Modulinstaller
- ioly oder aioli?

# Exkurs - OXID Modul Connector (OMC)

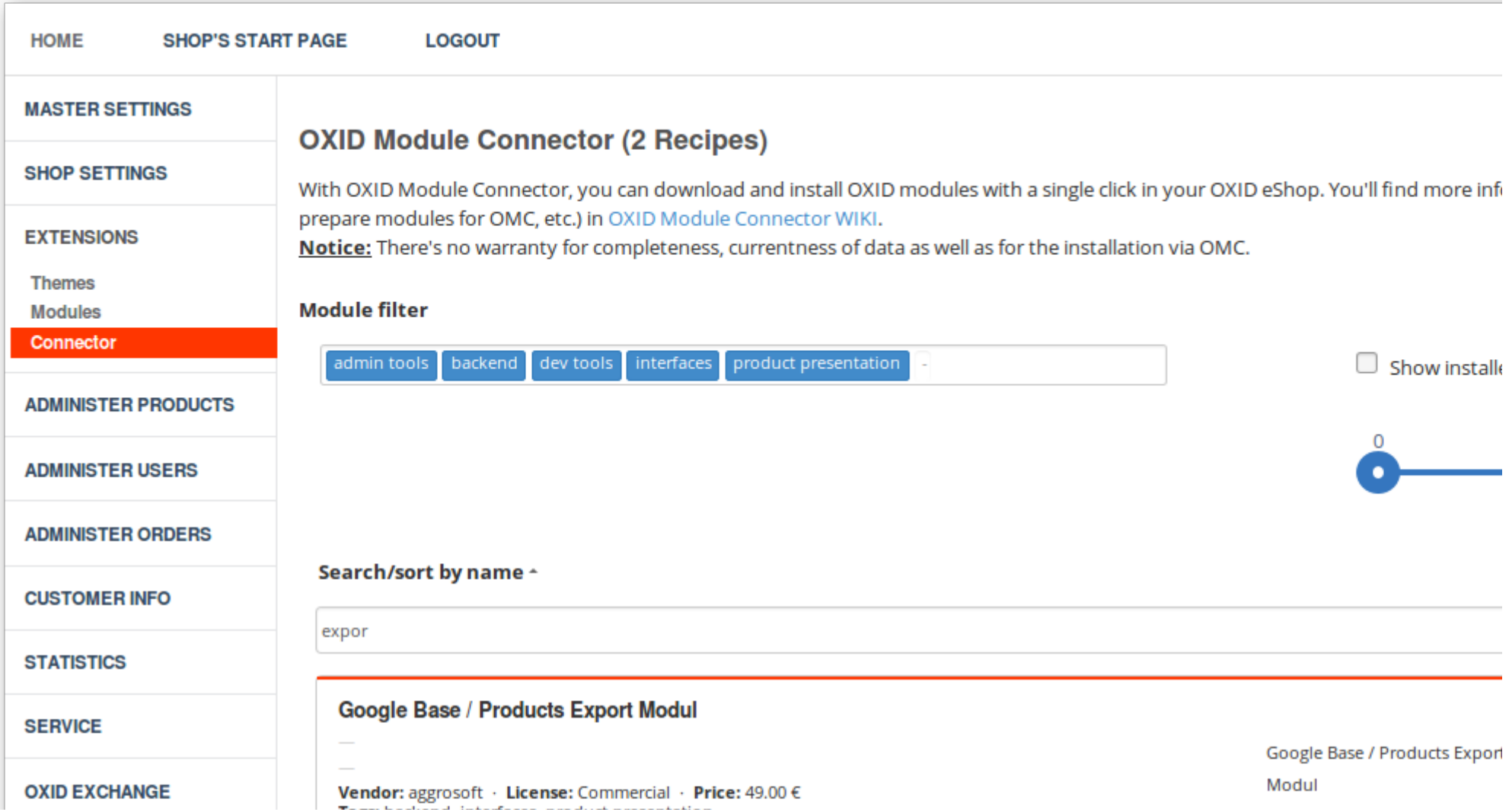
# Exkurs: OXID Module Connector (OMC)



- Communityprojekt (OXID Hackathon 2016)
- OMC ermöglicht es, im Backend oder per Konsole Module aus einem Modulkatalog (“Rezepte”) zu installieren
- basiert auf dem “ioly”-Core, der (alternativ zu Composer) über “Rezepte” aus unterschiedlichen Quellen (versch. ZIP-Dateien) die gewünschten Module installieren kann

# Exkurs: OXID Modul Connector (OMC)





# OMC / ioly oder composer

# OMC / ioly oder composer



**Vorteile / Unterschiede zu Composer**

- die Module selbst können (z.B. als ZIP) an beliebigen Orten liegen und benötigen selbst keinerlei Meta-Info (z.B. composer.json)
- es gibt dadurch auf Github eine zentrale Sammelstelle für alle “Rezepte” (JSON-Dateien), d.h. es entsteht dadurch ein “Modul-Katalog” aller verfügbaren Module (aktuell nur OXID)
- die Dateien aus den Modul-Packages können über das zugehörige “Rezept” in beliebigen Verzeichnissen installiert und auch wieder deinstalliert werden
- es gibt Callbacks, die Module können auch über das Backend direkt aktiviert werden
- es gibt auch Support für kommerzielle Module etc.

# composer update vs. composer install

# Composer install vs. update



- “composer install” installiert alle nötigen Packages und schreibt das “composer.lock”
- wird dann für weitere “composer install” Aufrufe genutzt, d.h. der Stand der Composer-Pakete bleibt erhalten und der Prozess ist wesentlich kürzer
- “composer update” muss man nur machen, wenn man tatsächlich alle Pakete auf Updates prüf und diese ggf. in neuer Version haben möchte
- *ergo:* **composer.lock mit in GIT einchecken!**

# Resources



- Composer Installers
- Composer Installers Extender
- OXID Modul Connector, OMC/ioly Composer Installer
- ioly
- Module in OXID 6 über Composer installieren, OXID 6 Installation, OXID 6 Neuerungen
- OXID 6 Composer Plugin
- Composer-Merge-Plugin
- Eigene Composer Installer
- Composer Script Hooks
- Composer install vs. update