

SQUEEZE 2

Customizing

Handbuch

Dieses Handbuch soll als Leitfaden für das Customizing dienen.

- [Einleitung User Exits](#)
- [Customizing in der Cloud](#)
- [Leitfäden](#)
 - [Best Practices Für Die User Exit Entwicklung](#)

Einleitung User Exits

User Exits dienen dem Erweitern / Customizing von SQUEEZE. Sollten Standard-Funktionen nicht ausreichen, kann mittels PHP-Code das Verhalten der Software angepasst werden.

Grundlagen

User Exits sind grundsätzlich mandantenspezifisch. D. h., dass jeder Mandant eines SQUEEZE Servers seine eigenen User Exits definieren und einsetzen kann. Daher befinden sich die jeweiligen User Exits im Repository des Mandanten.

Cloud vs. On Premise

Die Customizing-Optionen unterscheiden sich je nach Installationsart.

Bei On Premise Installationen stehen alle Customizing-Funktionen frei zur Verfügung. I. d. R. handelt es sich hier um Installationen bei einzelnen Endkunden. Mehr Details in: [Customizing On Premise](#)

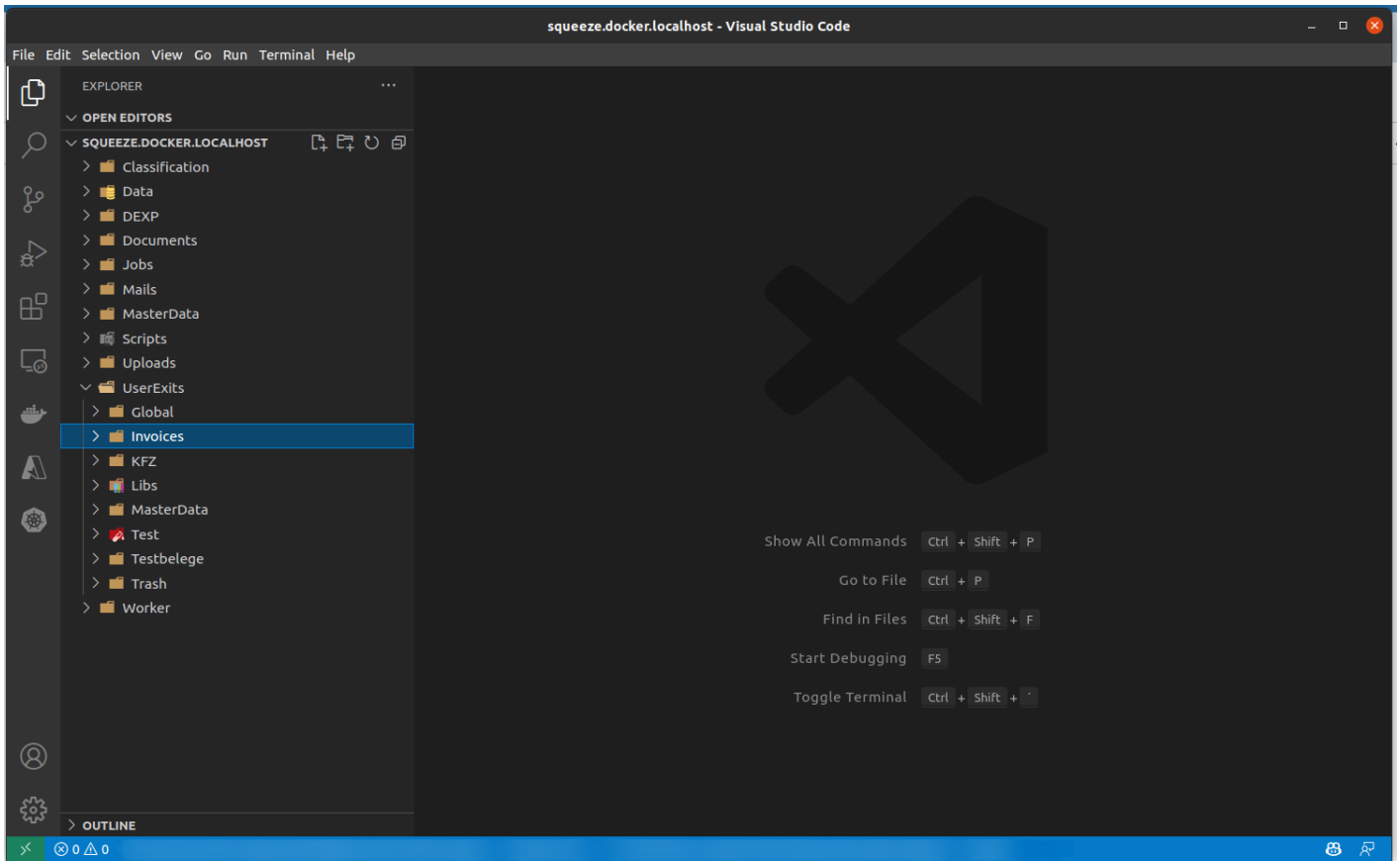
Bei Cloud-Mandanten ist die Bearbeitung von User Exits je nach Projekt bzw. Kunde eingeschränkt. Hier muss mit dem jeweiligen Ansprechpartner der DEXPRO oder des betreuenden Partners abgestimmt werden, ob ein Customizing möglich ist. Mehr Details in: [Customizing In Der Cloud](#)

Entwicklungsumgebung

IDE / Editor

Die Entwicklung von User Exits ist grundsätzlich mit jedem Text-Editor möglich. **Wir raten allerdings stark zur Verwendung von [Visual Studio Code](#).** Ergänzend setzen wir auf folgende Extensions:

- [PHP Intelephense](#)



Einrichten von Code Completion

Code Completion ist ab Squeeze 2.3 möglich.

Wir empfehlen die Einrichtung von Code Completion in ihrer IDE. Dieses Feature bietet ein wertvolles Werkzeug bei der Entwicklung *und dem Support* von User Exits. Vorteile der Verwendung sind:

- Ihre Entwicklungsumgebung zeigt Ihnen bessere (und vor allem korrektere) Fehlermeldungen an.
- Sie sehen direkt im Editor die Dokumentation von Methoden und Klassen, die Sie verwenden.
- Falls Sie Code nutzen, der in zukünftigen Versionen verändert oder entfernt wird, sehen sie das direkt im Editor.
- Ihr Editor schlägt ihnen Vervollständigungen vor, während Sie tippen. So sparen Sie sich (häufig) den Blick in eine Dokumentation.

```
, $creditor->description);
```

Squeeze\DocumentField::\$description
`<?php`
`public $description;`

@deprecated
Please use the setter, getter or adder to modify this field. This field will not be removed but could eventually be made private.

@var string

@internal

@OA\Property()

Squeeze\DocumentField::\$description
`<?php`

```
ment!";
```

Visual Studio Code

Dazu sind die folgenden Schritte notwendig:

1. Installieren sie [PHP Intelephense](#)
2. Legen sie eine Stubs-Datei (s. Abschnitt unten) im Ordner ab, den Sie mit Visual Studio Code öffnen. Es eignet sich die Ablage bspw. im Repository oder dem Mandantenordner im Repository.
3. Konfigurieren Sie folgende Einstellungen in Visual Studio Code ([Wie geht das?](#)). Passen Sie dabei ggf. die PHP Version auf die von ihnen eingesetzte PHP Version an.

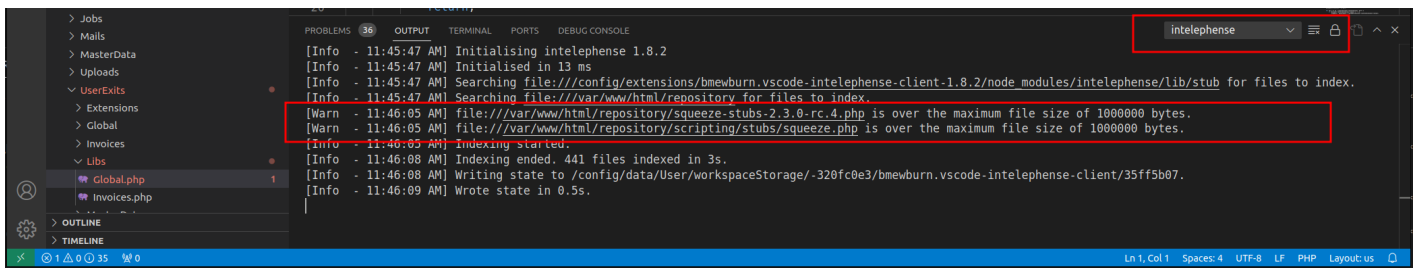
```
{  
  "php.suggest.basic": false,  
  "intelephense.files.maxSize": 1000000000,  
  "intelephense.environment.phpVersion": "7.4.22"  
}
```

Code Completion wird nicht angezeigt

Falls Autocomplete nicht funktioniert, dann prüfen Sie die Log-Ausgaben der Intelephense Extension:

- Output Konsole öffnen
- Im Dropdown auf Intelephense umstellen

Falls Sie hier einen Fehler sehen, hilft das ggf. bei der Problemlösung. Im Screenshot wurde bspw. die Einstellung bzgl. der Dateigröße nicht gesetzt.



Stubs-Datei

Die Stubs-Datei ist eine eine Datei mit PHP-Quellcode, die dafür sorgt, dass Visual Studio Code die Datentypen, Klassen und Funktionen von Squeeze kennt.

Die Stubs werden aktuell noch manuell erzeugt. Fragen Sie gern im Support nach, ob sie eine Stub-Datei erhalten können.

Customizing in der Cloud

Folgendes gilt nur bei Mandanten / Umgebungen, für welche Customizing in der Cloud freigeschalten wurde.
Ggf. treffen nur einige Funktionen in Ihren Kontext zu.

Web-IDE

Es ist möglich [Visual Studio Code](#) als Web-IDE für die Programmierung von User Exits einzusetzen. Insofern dies freigeschalten ist, erhalten Sie eine URL über welche auf die IDE im Browser zugegriffen werden kann.

Die Verwendung dieser IDE ist im Prinzip gleich mit der auf On Premise Umgebungen, allerdings sind einige Funktionen (bspw. das Terminal) gesperrt.

Leitfäden

Best Practices Für Die User Exit Entwicklung

Dieser Leitfaden dokumentiert Coding-Richtlinien, deren Einhaltung zu fehlerfreien und updatefähigen User Exits führen sollen.

Motivation hinter dem Leitfaden ist, dass in der Vergangenheit häufig User Exits programmiert wurden, welche bspw. auf Variablen oder Methoden zugegriffen haben, die eigentlich nicht für den Gebrauch in User Exits entworfen wurden. Das wiederum erhöht die Wahrscheinlichkeit für Fehler bei der Ausführung von User Exits, besonders nach Updates von SQUEEZE. Aus dieser Motivation wird perspektivisch eine [Scripting API](#) entwickelt und bereitgestellt.

Best Practices

Methoden vs. Klassen-Eigenschaften

Grundsätzlich ist die Verwendung von Methoden dem direkten Zugriff auf Klassen-Eigenschaften vorzuziehen.

```
$creditor = $xDoc->getFieldByName("Creditor");

// Bad
$creditor->value->value = "New Creditor";

// Good
$creditor->getValue()->setValue("New Creditor");
```

Der direkte Zugriff auf Eigenschaften, insbesondere bei Verkettung (`|$creditor->value->value|`) kann zu Exceptions führen, falls eine Eigenschaft null ist. Außerdem kann so nicht gewährleistet werden, dass die Werte, die gesetzt werden sollen, normalisiert werden sollen.

Deprecations - Veraltete APIs nicht verwenden

Wenn Sie ihre IDE korrekt konfiguriert haben, werden Ihnen veraltete Methoden, Felder und Klassen durchgestrichen dargestellt. Das ist im folgenden Screenshot zu sehen:

```
$tax2-> Squeeze\xDocumentField::$value
$tax2-> <?php
$tax2-> public $value;
$fieldL
}
if ($net3-> @deprecated
    $net3-> Please use the setter, getter or adder to modify this field. This field will not be removed but could eventually be made private.
    $net3-> @var \Squeeze\xDocValue
    $net3-> @internal
    $net3-> @OA\Property(type="object", ref="#/components/schemas/xDocValue") —,
    $fieldL '$value' is deprecated. intelephense(1007)
    $tax3-> View Problem (Alt+F8) No quick fixes available
    $tax3-> value->text = '0.00';
    $tax3-> state = "OK";
    $fieldList->updateField($tax3);
}
```

Hier ist bspw. das Feld `value` der Klasse `xDocValue` veraltet. Im Tooltip ist zu lesen:

“ Please use the setter, getter or adder to modify this field. This field will not be removed but could eventually be made private.

Sie sollten hier also versuchen Getter und Setter zu verwenden:

```
// Good
$val = $tax3->getValue();
$val->setValue("0.00");
$val->setText("0.00");

// Bad
$tax3->value->value = '0.00';
$tax3->value->text = '0.00';
$tax3->state = "OK";
$fieldList->updateField($tax3);
}
```

Deprecations werden phasenweise eingeführt als frühzeitige Warnungen, dass der Standard sich zukünftig verändern **kann**.

Solche Veränderungen werden in den Changelogs als Breaking Changes festgehalten, damit Sie bei einem Update prüfen können, ob es Handlungsbedarf gibt.

Fehlende Funktionalitäten

Sollten Funktionalitäten in Script-Libraries oder der [Scripting API](#) fehlen, ist es sinnvoll die fehlenden Funktionalitäten bspw. im Forum zu diskutieren. So können Funktionalitäten, die häufiger benötigt werden in den Standard aufgenommen werden.

Wiederverwendbare Methoden

Es bietet sich an einzelne Funktionen, die in User Exits umgesetzt werden müssen, als separate, wiederverwendbare Methoden umzusetzen. Der Vorteil davon ist, dass die Methoden einfacher kopiert und bei anderen Systemen erneut eingesetzt werden können, vorausgesetzt die Produktversionen sind kompatibel.

Die Zusammenfassung von gleichartigen Methoden in Klassen ist zu empfehlen.