ELO Debugging

[Stand: 13.05.2019 | Programmversion 12.00.000]

Diese Dokumentation beantwortet die Grundsätzlichen Fragen für den Einsatz von ELO Debuggern.



Info: Diese Dokumentation bezieht sich teilweise auf ELO 9.03. Die Angaben sind allerdings in weiten Teilen auf ELO 10 und höher übertragbar.

Inhalt

1 ELO Automation Services (ELOas) Debugger 2
1.1 Download und Installation 2
1.2 Den ELOas Debugger starten 3
1.3 Konfiguration 3
1.4 ELOas-Regeln debuggen 5
2 Java Client Debugger 9
3 Indexserver Debugger 14
4 Debugging im Browser 15
4.1.1 ELO API Apps Dokumentation 17

1 ELO Automation Services (ELOas) Debugger

1.1 Download und Installation

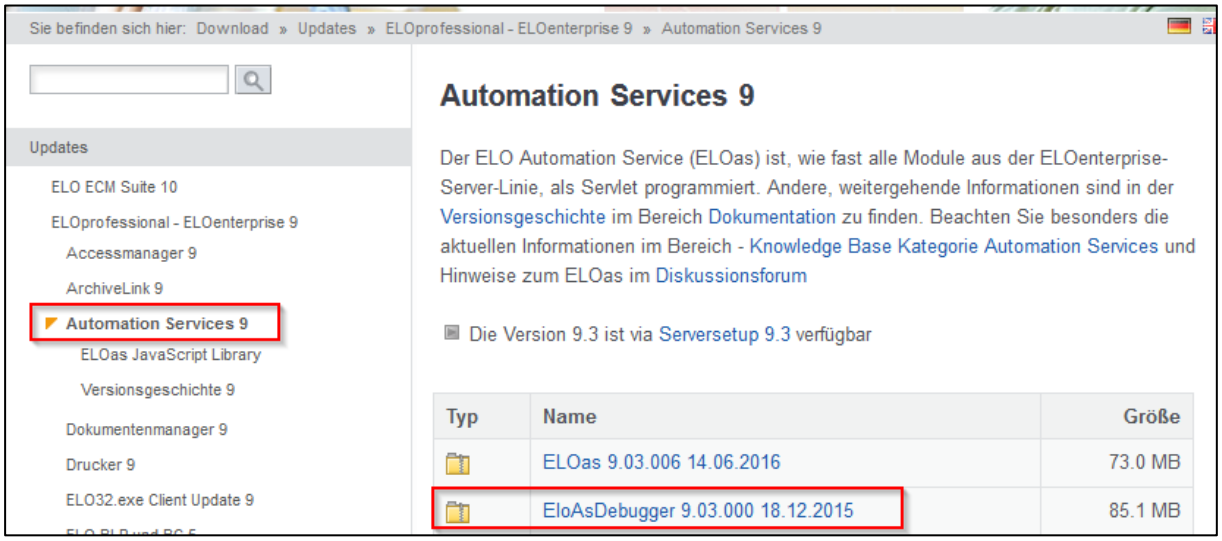


Abb. 1: Download des ELOas Debuggers

Der ELO Automation Services Debugger (kurz auch: ELOas Debugger) ist ein eigenständiges Programm, welches die gleichen Libraries wie der ELOas beinhaltet. Der ELOas Debugger steht im ELO Support Web zum Download zur Verfügung.

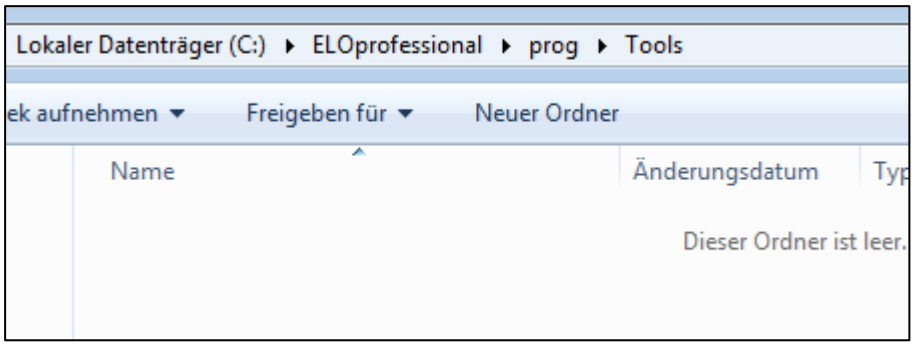


Abb. 2: Speicherort für den ELOas Debugger

Den Speicherort können Sie beliebig auswählen. Verwenden Sie zum Beispiel das Verzeichnis *Tools* im Serververzeichnis.

1.2 Den ELOas Debugger starten

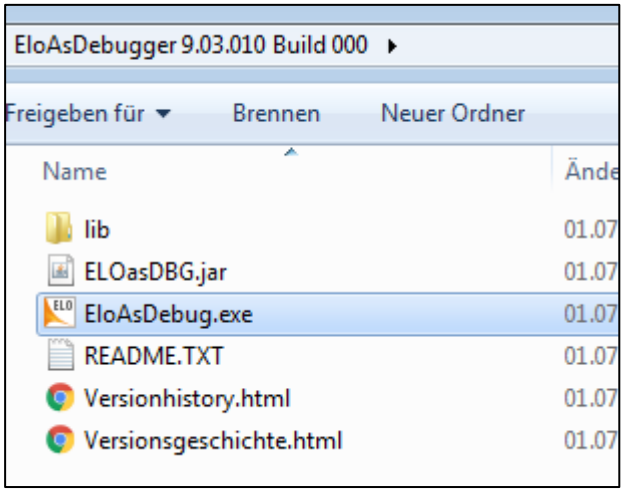


Abb. 3: Die Datei 'EloAsDebug.exe'

Mit einem Doppelklick auf die Datei *EloAsDebug.exe* starten Sie den ELOas Debugger.

1.3 Konfiguration

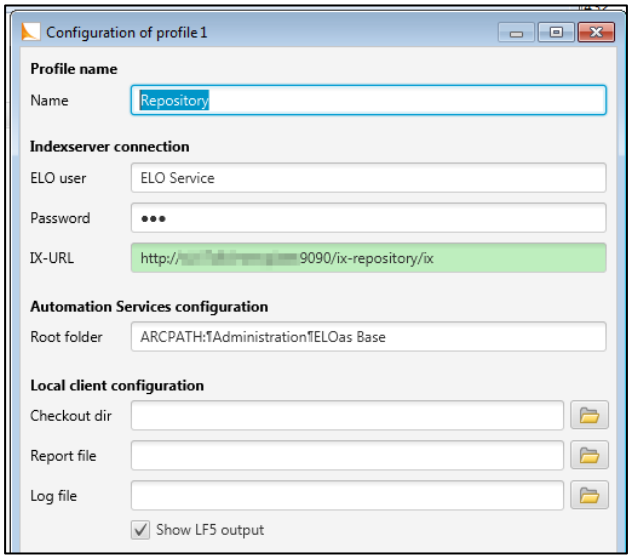


Abb. 4: Konfiguration des ELOas Debuggers

Initial müssen Sie den ELOas Debugger konfigurieren.

1. Über das Feld *Name* geben Sie dem Debugger-Profil einen beliebigen Namen.
2. Über das Feld *ELO user* tragen Sie den ELO-Benutzer ein, über den die ELOas-Regeln ausgeführt werden. Die ELOas Regeln werden im Normalfall über den ELO Service User ausgeführt.
3. Tragen Sie das entsprechende Passwort in das das Feld *Password* ein.
4. Tragen Sie die passende Indexserveradresse in das Feld *IX-URL* ein.

5. Aktivieren Sie die Option *Show LF5 output*.
6. Starten Sie den ELOas Debugger neu.

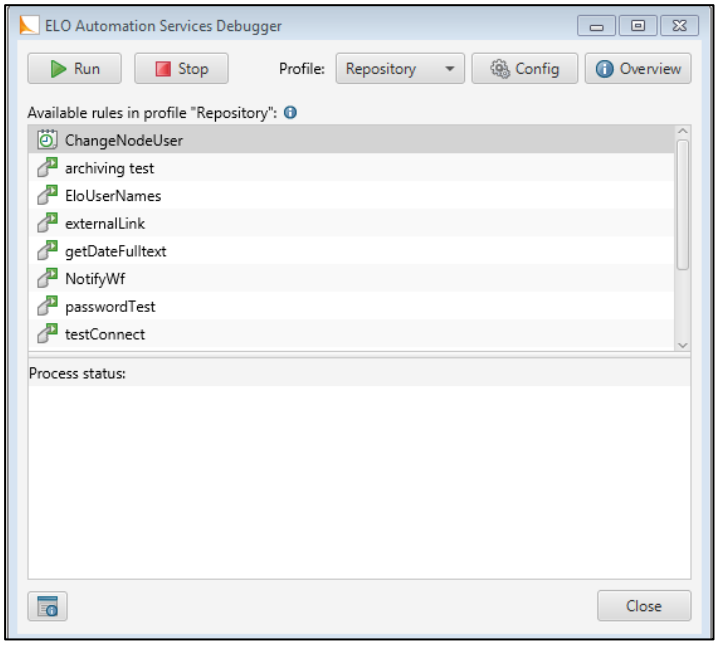


Abb. 5: ELOas-Regeln im Debugger

Nach der erfolgreichen Konfiguration sollten im Fenster *ELO Automation Services Debugger* Ihre Direct-Rules sowie zeitgesteuerten Regel aufgelistet werden.

1.4 ELOas-Regeln debuggen

Jetzt können Sie eine beliebige Regel debuggen.

> **Info:** Der Debugger wird bis ELO 10 in drei Fenstern geöffnet.

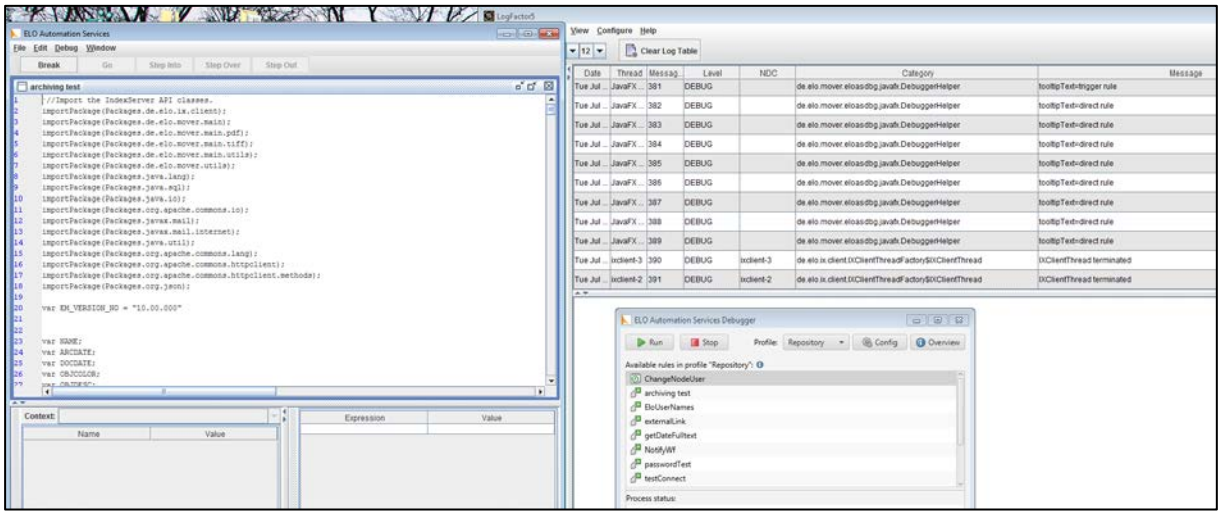


Abb. 6: Fenster des ELOas Debuggers

Das Fenster *ELO Automation Services* enthält die komplette Regel (Datenbankverbindung, Regel, weitere Libraries usw).

```
198  processRules: function (Sord) {
199    try {
200      sys.processRule1(Sord);
201    } catch (e) {
202      EM_ERROR = e;
203      log.info("Exception caught: " + EM_ERROR);
204      sys.processRule2(Sord);
205      return;
206    }
207  },
208
209  processRule1: function (Sord) {
210    // Rule: Rule1
211
212          log.info("First param=" + EM_PARAM1);
213          log.info("Second param=" + EM_PARAM2);
214          log.info("Third param=" + EM_PARAM3);
215
216
217  },
218
219  processRule2: function (Sord) {
220    //Generated Error Rule, clear all old destinations, reset ACL
221    EM_NEW_DESTINATION = new Array();
```

Abb. 7: Beispielsregel 'processRule1'

Ab Zeile 198 beginnen die Regeln.

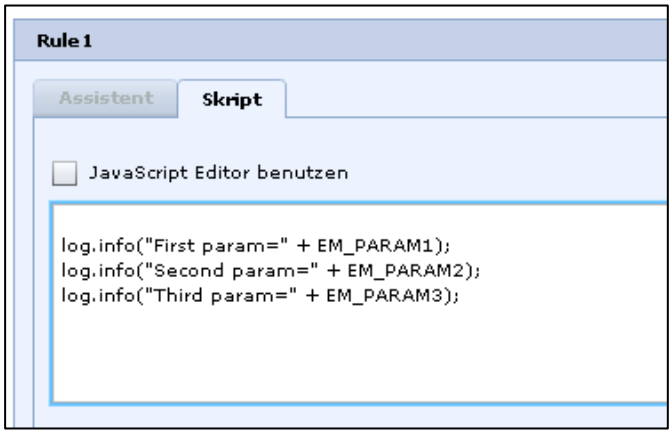


Abb. 8: Zeilen aus der Beispielsregel

Die eigentliche Testregel besteht nur aus drei Zeilen.

Im Beispiel oben wurden durch Klick in die graue Leiste Breakpoints in allen drei Zeilen gesetzt.

Nun kann die Regel gestartet werden. Dies geschieht im Fenster *ELO Automation Services Debugger*.

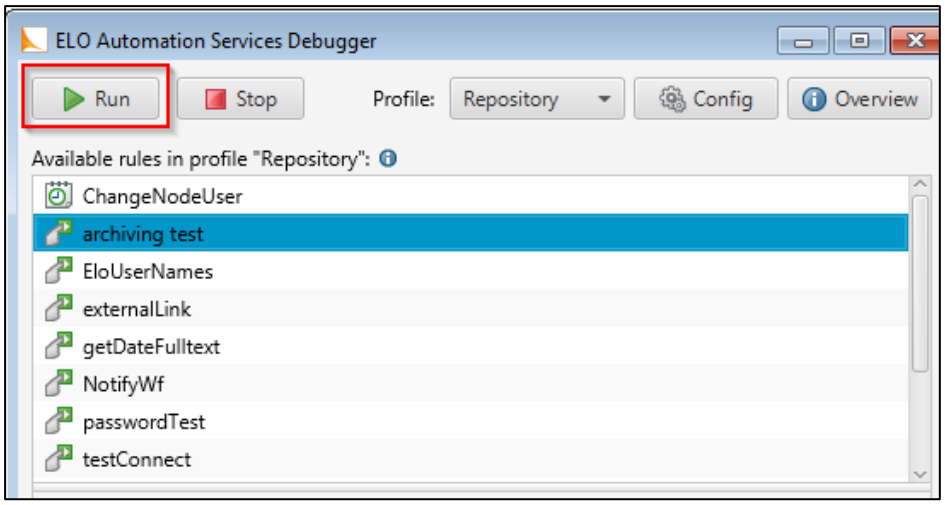


Abb. 9: Schaltfläche 'Run'

Wählen Sie dazu die entsprechende Regel aus und starten Sie den Debugging-Vorgang mit einem Klick auf *Run*.

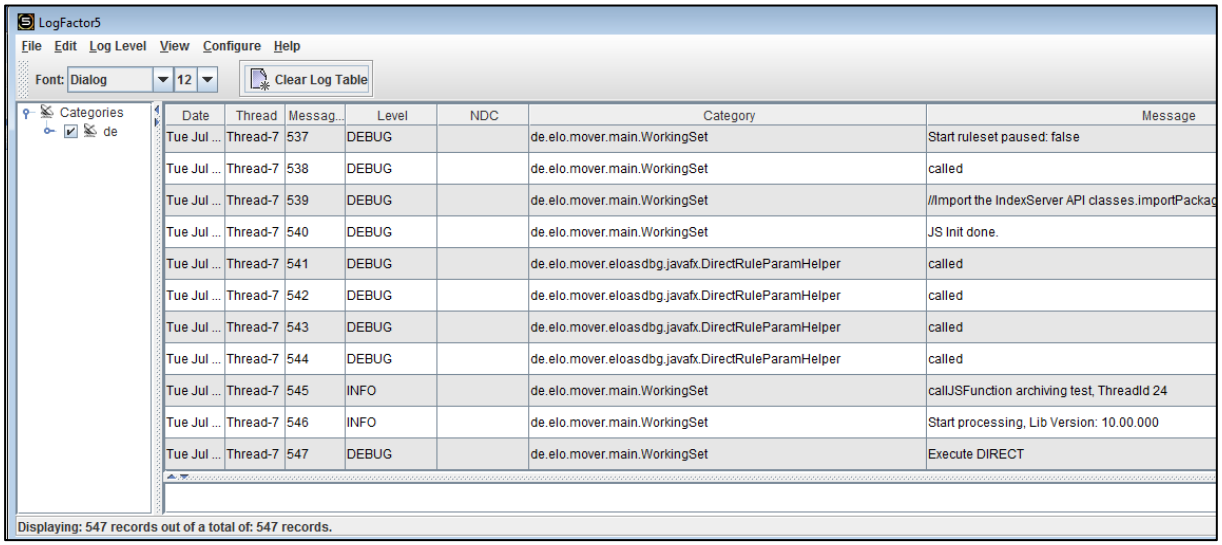


Abb. 10: Fenster 'LogFactor5'

Im Fenster *LogFactor5* werden die einzelnen Schritte dokumentiert.

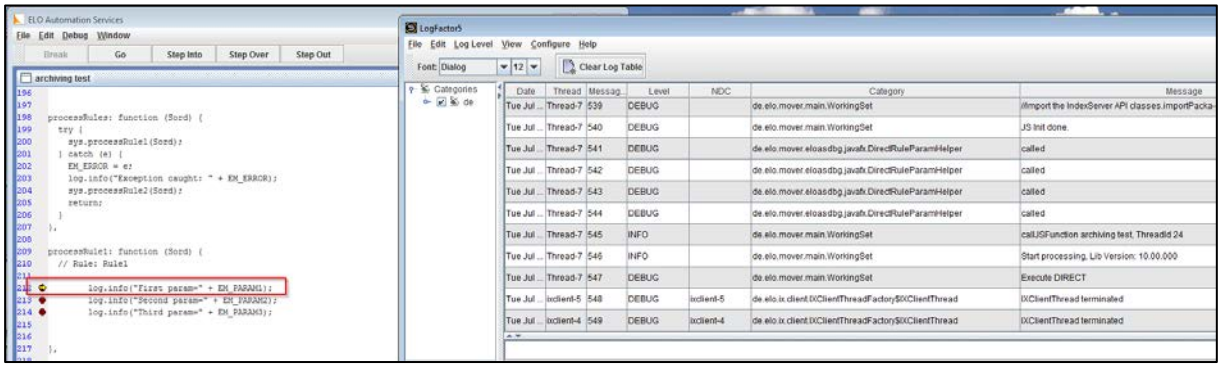


Abb. 11: Erreichen eines Breakpoints

Beim ersten Breakpoint bleibt das Log stehen.

Im Fenster *ELO Automation Services* klicken Sie sich über die Schalfläche *Step Into* durch die die einzelnen Zeilen und können im Fenster *LogFactor5* die entsprechende Logausgaben betrachten.

Mit einem Klick auf *Go* führen Sie die Regel bis zum nächsten Breakpoint bzw. bis zum Ende aus.

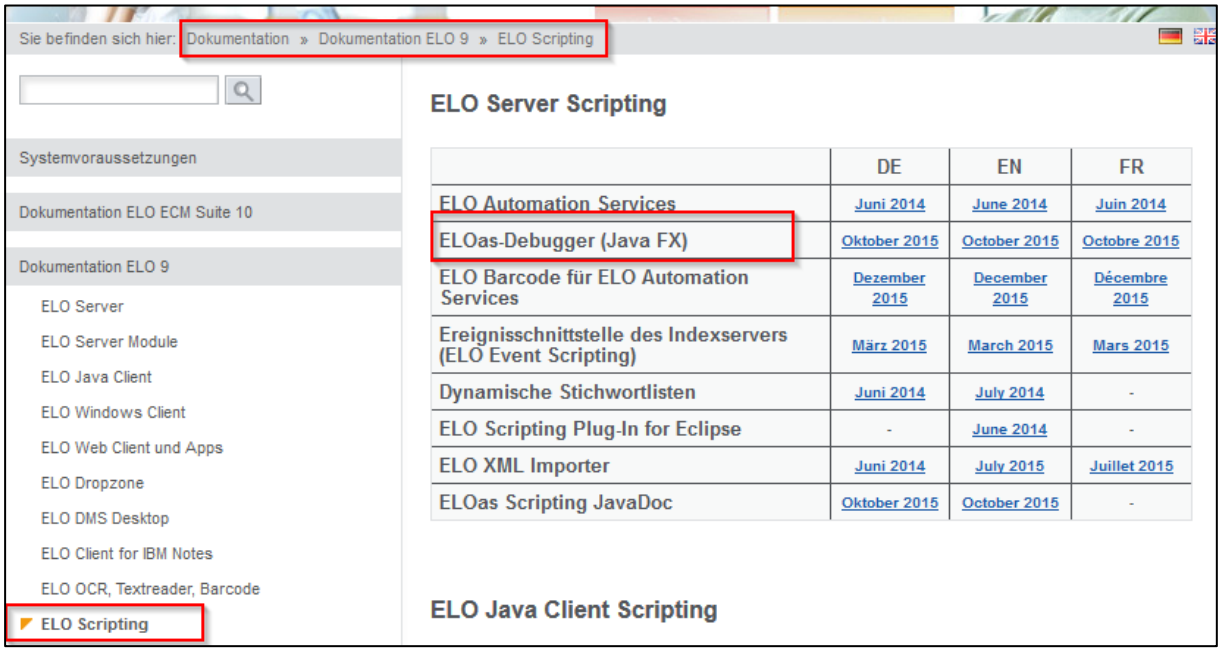
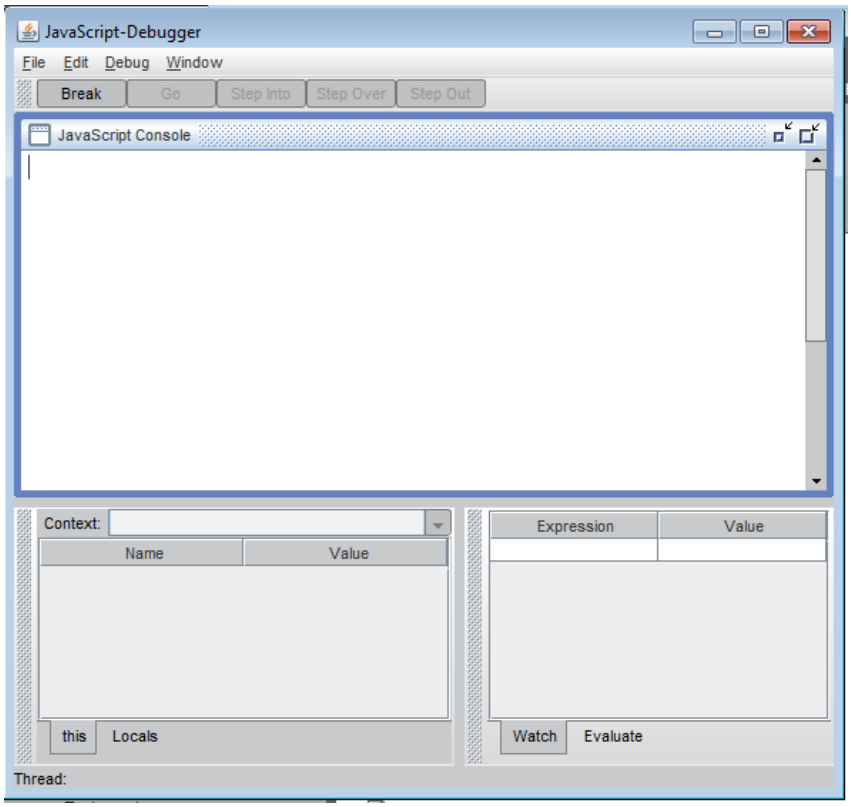


Abb. 12: Dokumentation zum ELOas Debugger

Eine detaillierte Beschreibung zum ELOas Debugger steht auch im ELO Support Web zur Verfügung.

2 Java Client Debugger



Der Java Client Debugger kann, falls er sich nicht automatisch mit dem Client öffnet, beim geöffnetem Client über die Tastenkombination STRG + ALT + D aufgerufen werden.

Im Java Client Debugger können alle Skripte, die sich im Ordner *Java Client Scripting Base* befinden, schrittweise debuggt werden.

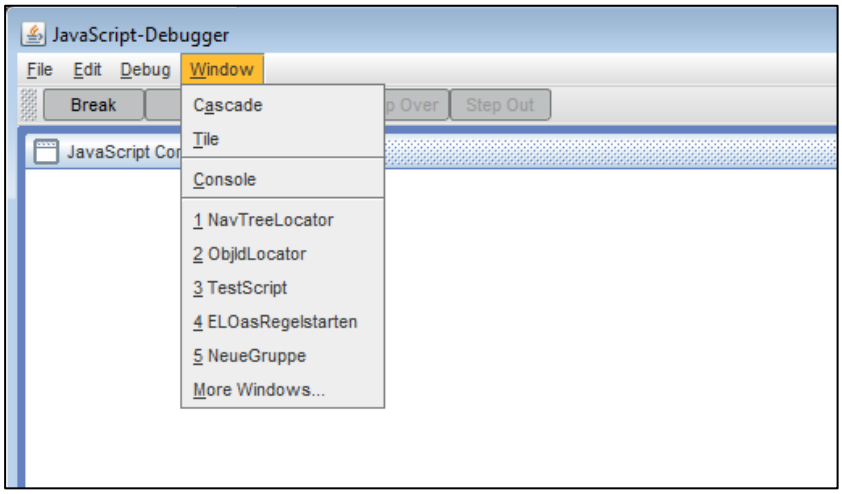


Abb. 13: Menüpunkt 'Window'

Über den Menüpunkt *Window* öffnen Sie ein Drop-down-Menü zum Auswählen von Skripten.

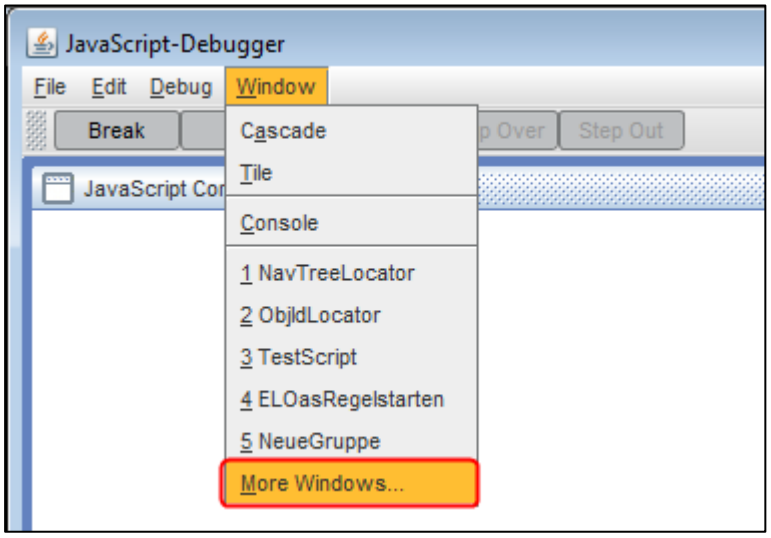


Abb. 14: Menüpunkt 'More Windows'

Über den Unterpunkt *More Windows* öffnen Sie die komplette Liste.

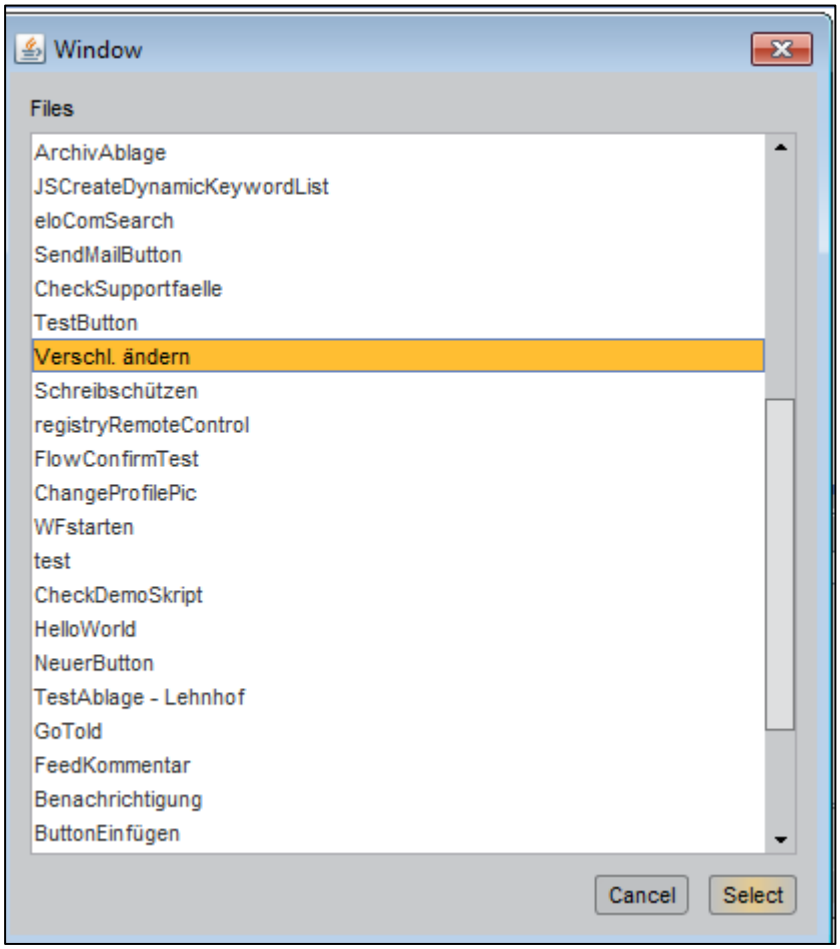


Abb. 15: Liste mit Skripten

Wählen Sie das gewünschte Skript aus und klicken Sie auf *Select*.

Das Skript wird in den Debugger geladen.

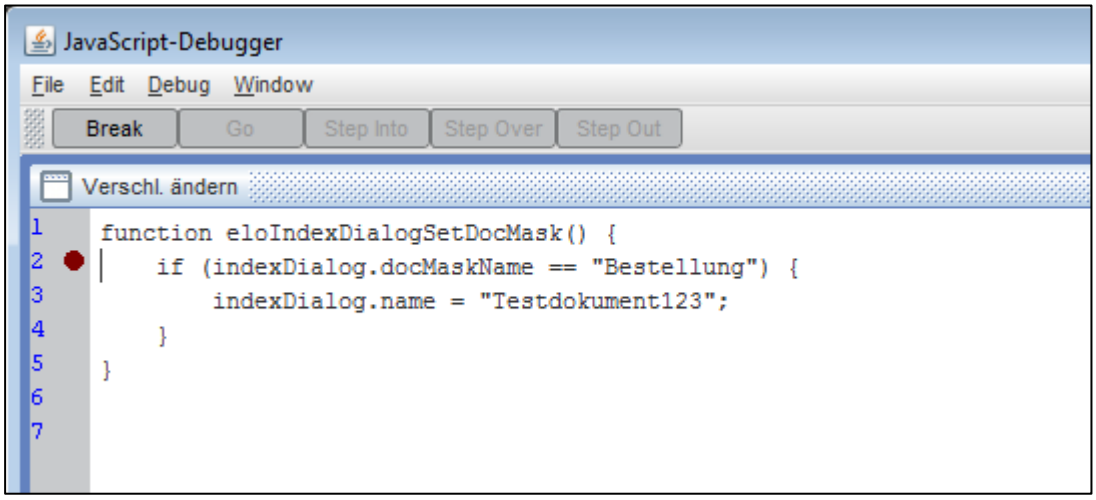


Abb. 16: Beispiel mit Breakpoint

Um bestimmte Ausgaben im Skript zu überprüfen können Sie über die Schaltfläche *Break* Breakpoints setzen.

Um den Debugging-Vorgang zu starten, muss das jeweilige Skript über den Client ausgeführt werden.

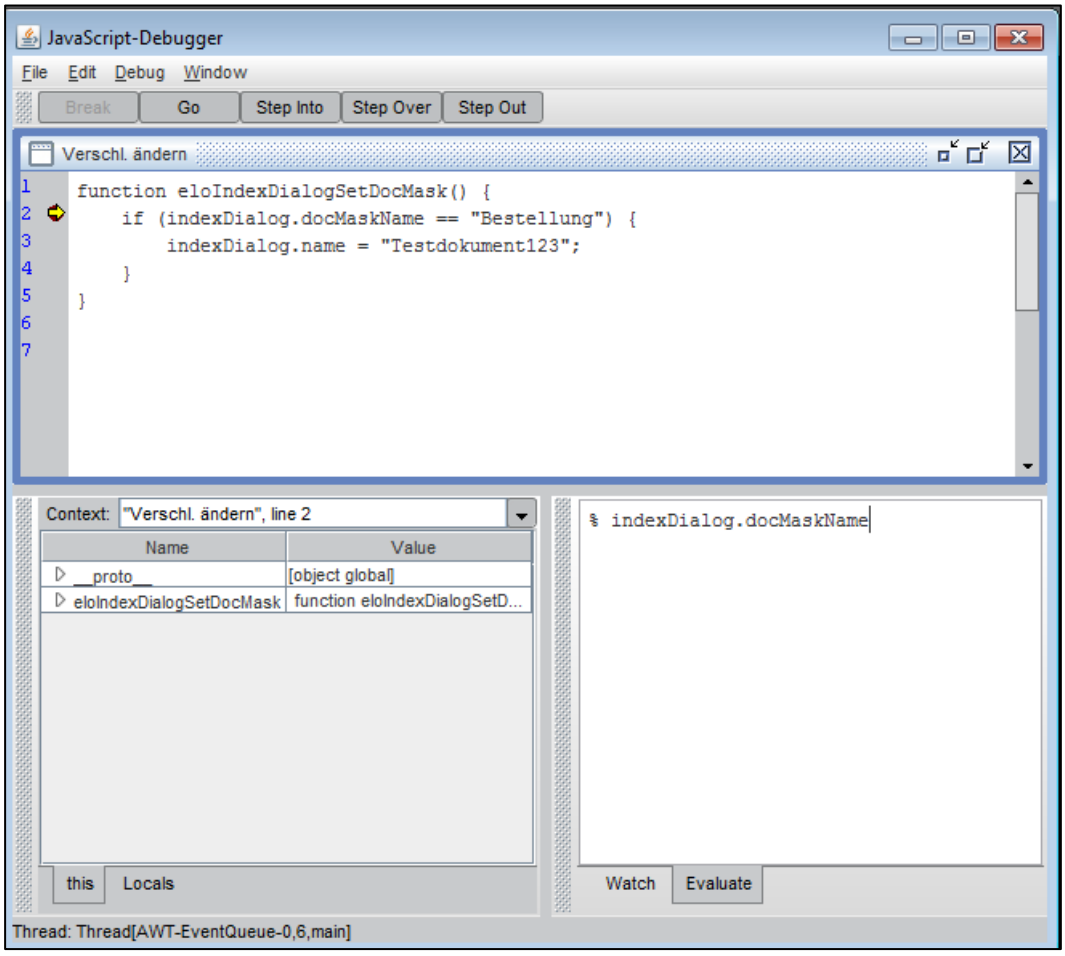


Abb. 17: Erreichter Breakpoint

In unserem Beispiel öffnen wir im Archiv die Verschlagwortung eines Dokuments, sodass die Methode getriggert wird. Wenn die Methode an der If-Abfrage ankommt, dann wird am Breakpoint angehalten.

Nun kann abgefragt werden, welche Maske dieses Dokument hat, um zu sehen, ob die Abfrage richtig ausgeführt und ein Ergebnis geliefert wird.

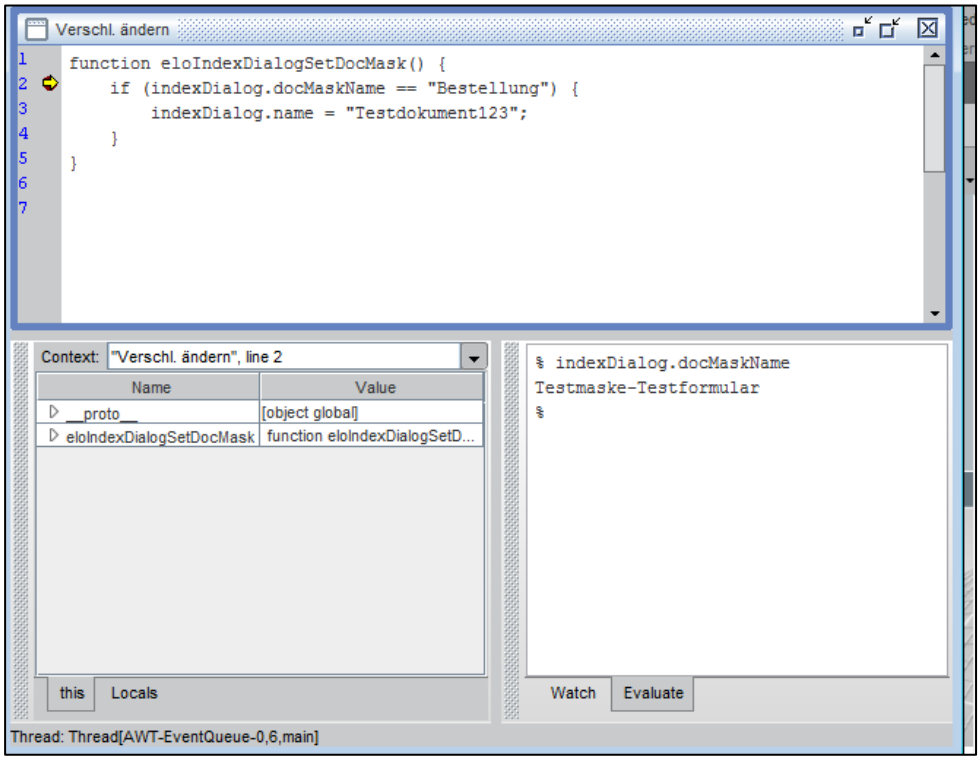


Abb. 18: Auswertung des Testskripts

In diesem Fall hat das Dokument die Maske *Testmaske-Testformular* und somit stimmt die if-Abfrage nicht. Die Kurzbezeichnung wird somit nicht geändert.

Im Fenster *this* werden die Methoden dieses Skripts aufgelistet. Das Fenster *Evaluate* dient dazu um Anfragen zu tätigen.

- Go = Führt das Skript ohne Unterbrechung komplett aus.
- Step Into = Mit *Step Into* wird in dem Methodenaufruf reingesprungen
- Step Over = Mit *Step Over* wird der aktuelle Methodenaufruf übersprungen
- Step Out = Man befindet sich derzeit in einem Methodenaufruf und springt zum Ende bzw. raus aus dem aktuellen Methodenaufruf.

Eine detaillierte Beschreibung des Java Client Debuggers befindet sich im Buch „Skriptprogrammierung mit ELOprofessional“ im Kapitel „Fehlersuche in Skripten“.

3 Indexserver Debugger

Indexserver Skripte können über den Eclipse Debugger mit dem ELO Scripting Plug-In for Eclipse debugged werden. Hierzu gibt es für den Debugger eine Dokumentation auf der ELO Support Web Seite:

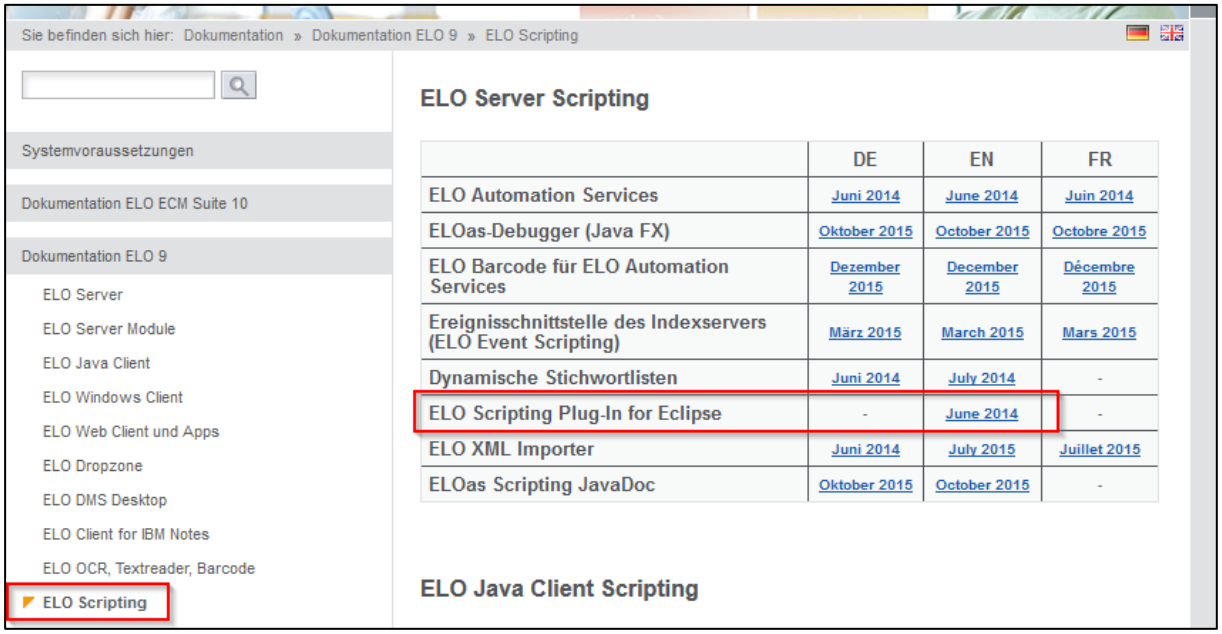


Abb. 19: Dokumentation für das 'ELO Scripting Plug-In for Eclipse'

4 Debugging im Browser

Formulare können über den Browser debugged werden.

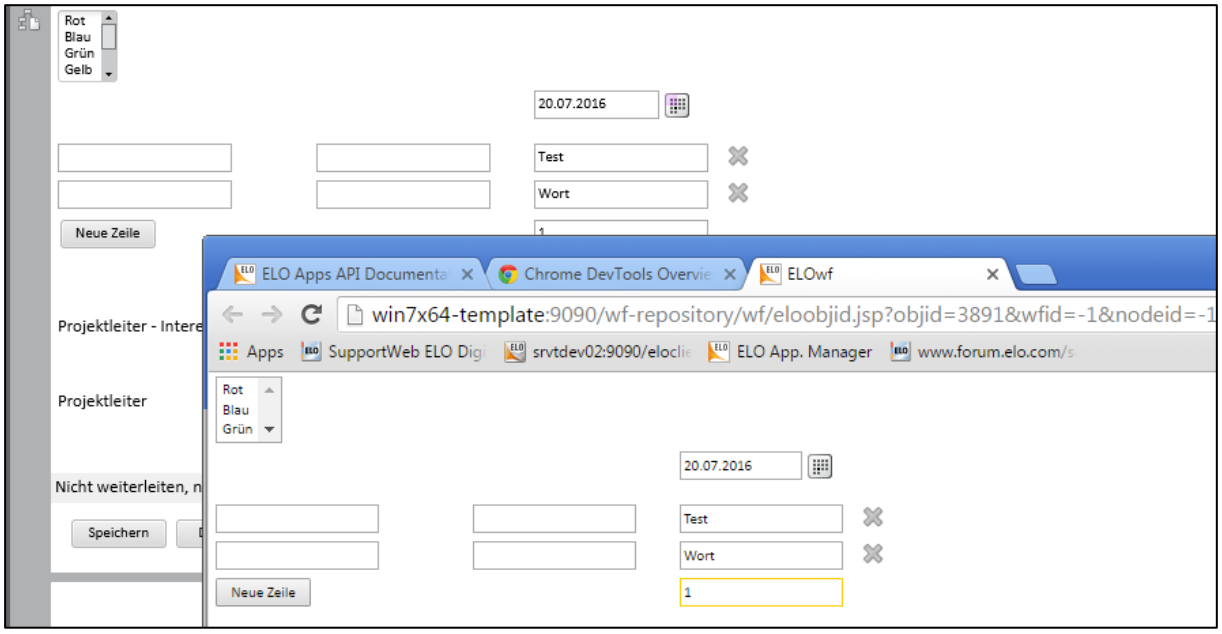


Abb. 20: Formular im Browser

Dazu müssen Sie das Formular im Browser öffnen (Hierzu die STRG-Taste gedrückt halten und im Formular in eine freie Stelle klicken). Nachfolgend verwenden wir den Chrome-Browser.

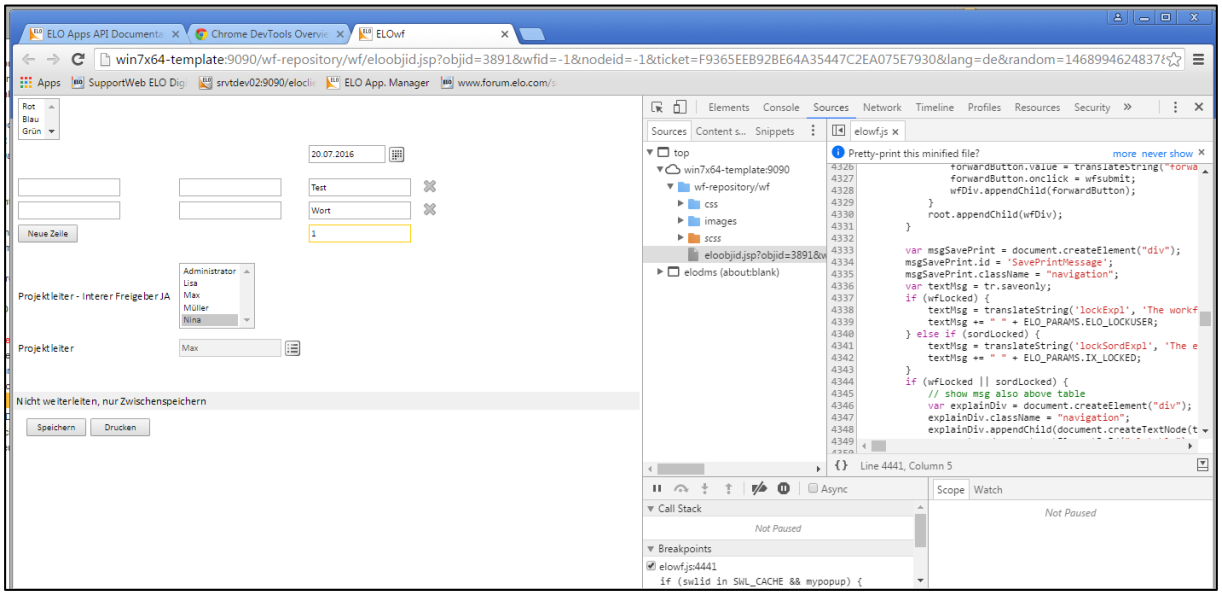


Abb. 21: Entwicklertools

Rufen Sie mit F12 die Entwickler-Tools auf und öffnen Sie den Tab *Sources*.

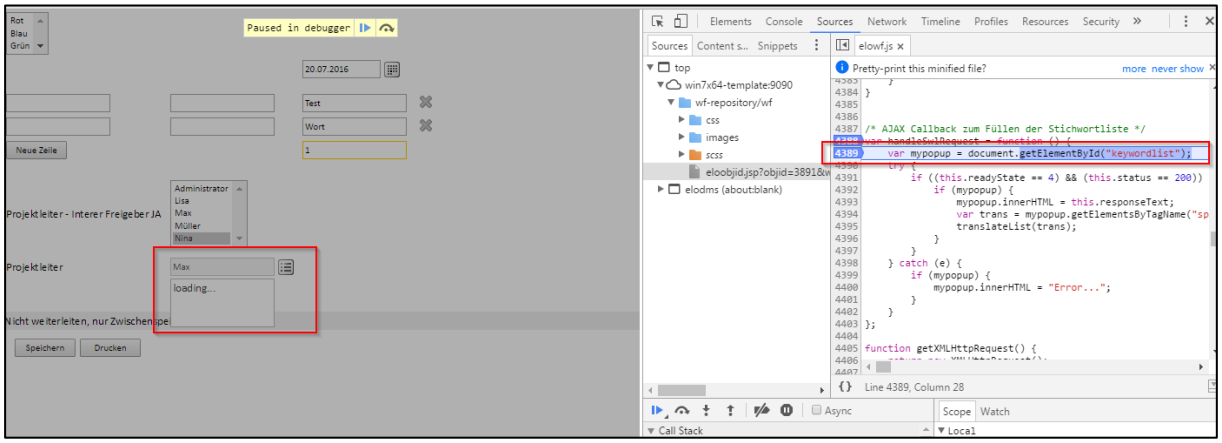


Abb. 22: Breakpoint in den Entwicklertools

Mit Klick in die Zeilennummer können Sie Breakpoints setzen. Wenn nun die Stelle mit dem Breakpoint im Formular betätigt wird, dann wird die entsprechende Zeile blau markiert.

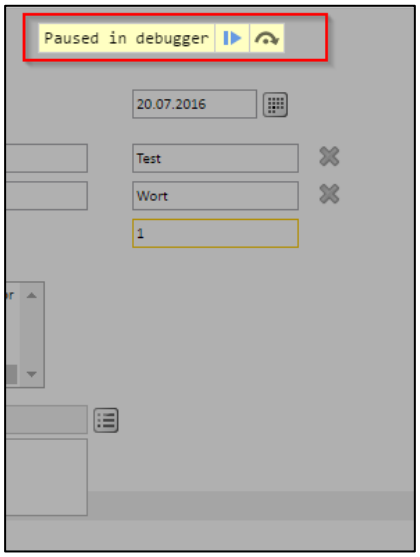


Abb. 23: Bedienelemente im Browser

Der Browser-Debugger bietet Bedienelemente zum Steuern des Debuggers (Im Chrome-Browser in einer gelben Leiste).

4.1.1 ELO API Apps Dokumentation

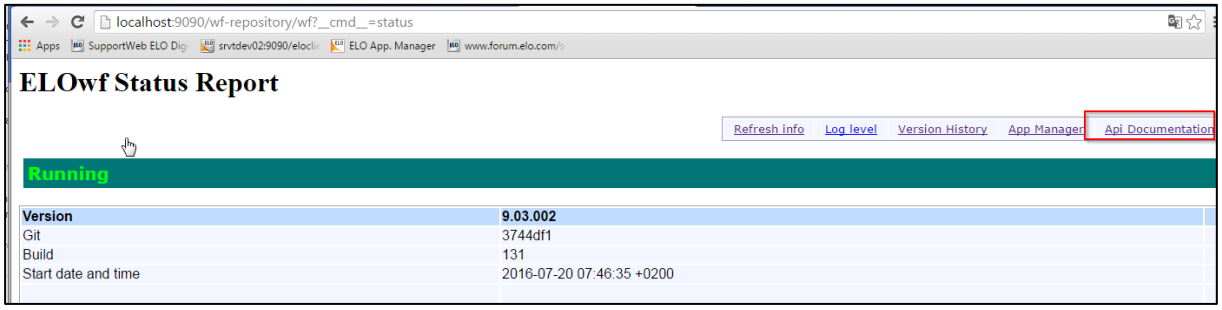


Abb. 24: Link zur ELO API Apps Dokumentation

Weitere Informationen zum Debugging im Browser sind in der ELO Apps API Dokumentation zu finden.

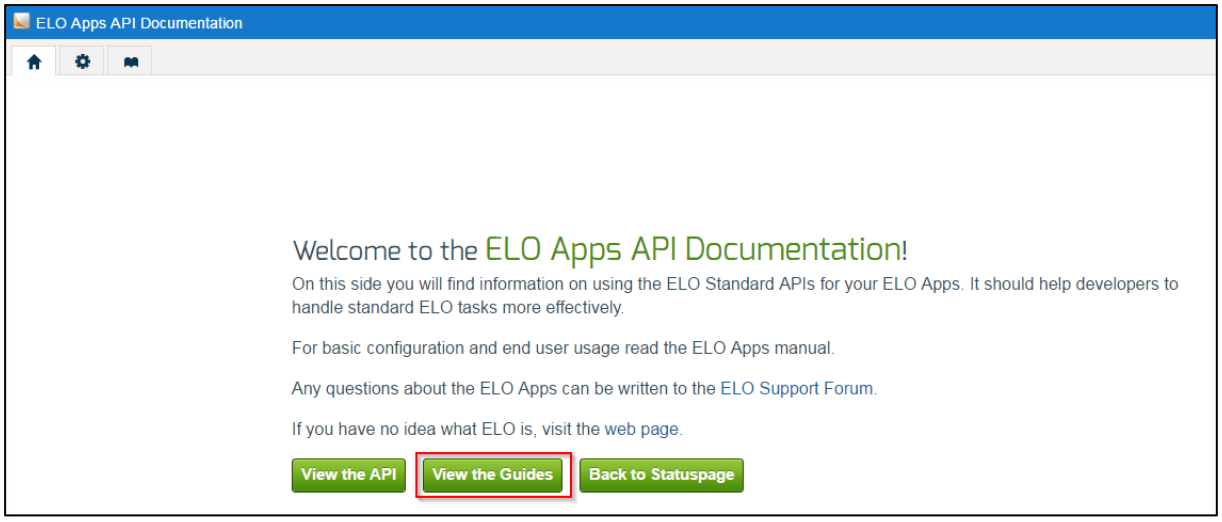


Abb. 25: Schaltfläche 'View the Guides'

Klicken Sie auf *View the Guides*.

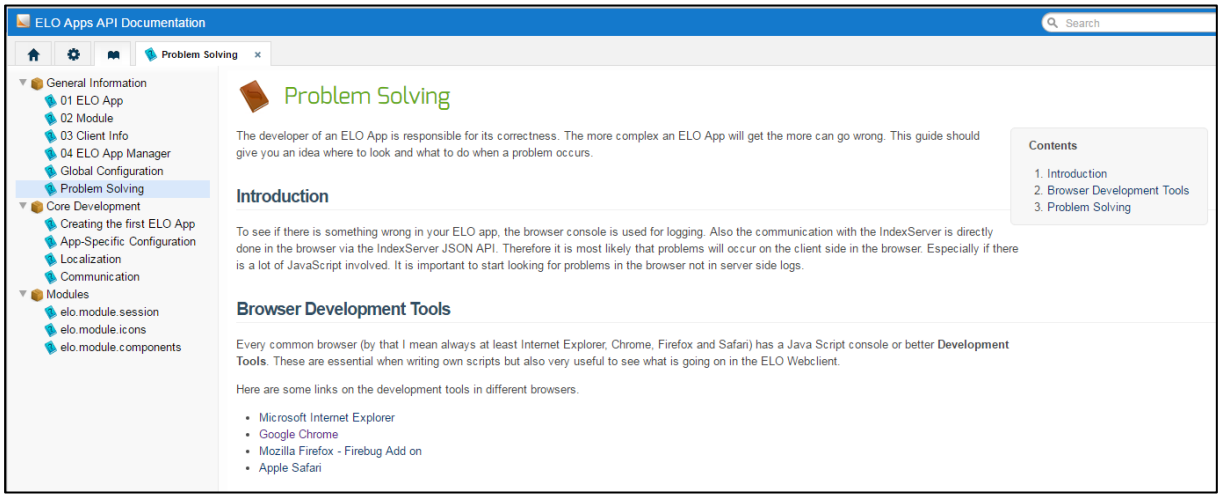


Abb. 26: Guides in der ELO Apps API Dokumentation

Unter dem Punkt *Browser Development Tools* kann die Beschreibung der entsprechenden Browser aufgerufen werden.

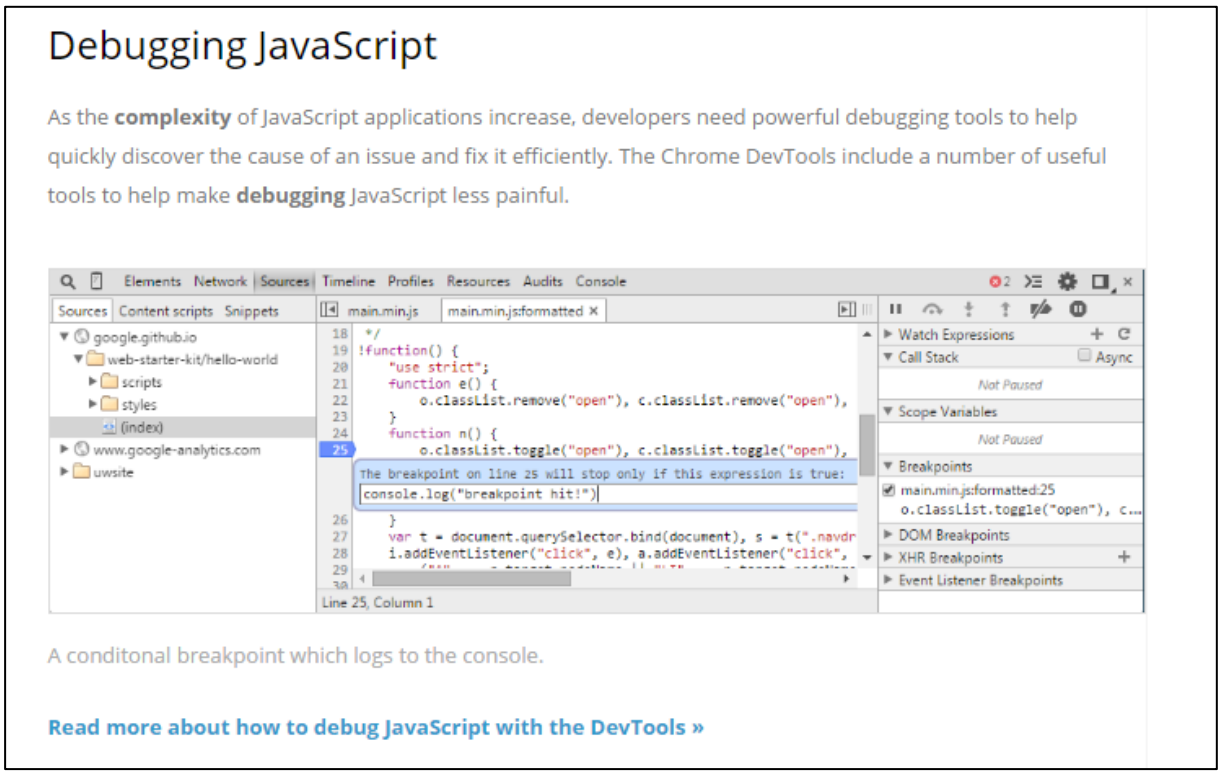


Abb. 27: Beispiel für einen Guide

Darunter wird für den Browser das Debugging (mit weiterführenden Links) vorgestellt.