DOCUMENTS

Visual Studio Code

DOCUMENTS Extension

Christine Heller

Inhalt

Tools mit PortalScripting Unterstützung

Einführung in den Visual Studio Code

DOCUMENTS Extension

Live-Demos

Tools mit PortalScripting Unterstützung

- Notepad++
  - Einfacher Editor
  - DOCUMENTS 4 und 5
- Eclipse
  - Umfangreiche Entwicklungsumgebung
  - DOCUMENTS 4 und 5
- **Visual Studio Code**
  - Kombination: einfach mit guten Entwicklungswerkzeugen
  - DOCUMENTS 5.0b/c (DOCUMENTS 4 abgekündigt zum 31.12.2018)
  - Feature teilweise abhängig von der DOCUMENTS Version

otris software

Einführung in den Visual Studio Code

Visual Studio Code

Allgemein

- Kostenloser Open Source Editor von Microsoft
- Verfügbar für Windows, Linux, Mac
- Beliebteste Tool (Stack Overflow Umfrage 2018)

Installation und erste Schritte

- Free Download
- Monatliche Updates (automatische Hinweise, einfache Installation)
- Dokumentation und Erste Schritte
- VS Code auf Ordner öffnen! (Windows Explorer Kontextmenü)

Übersicht

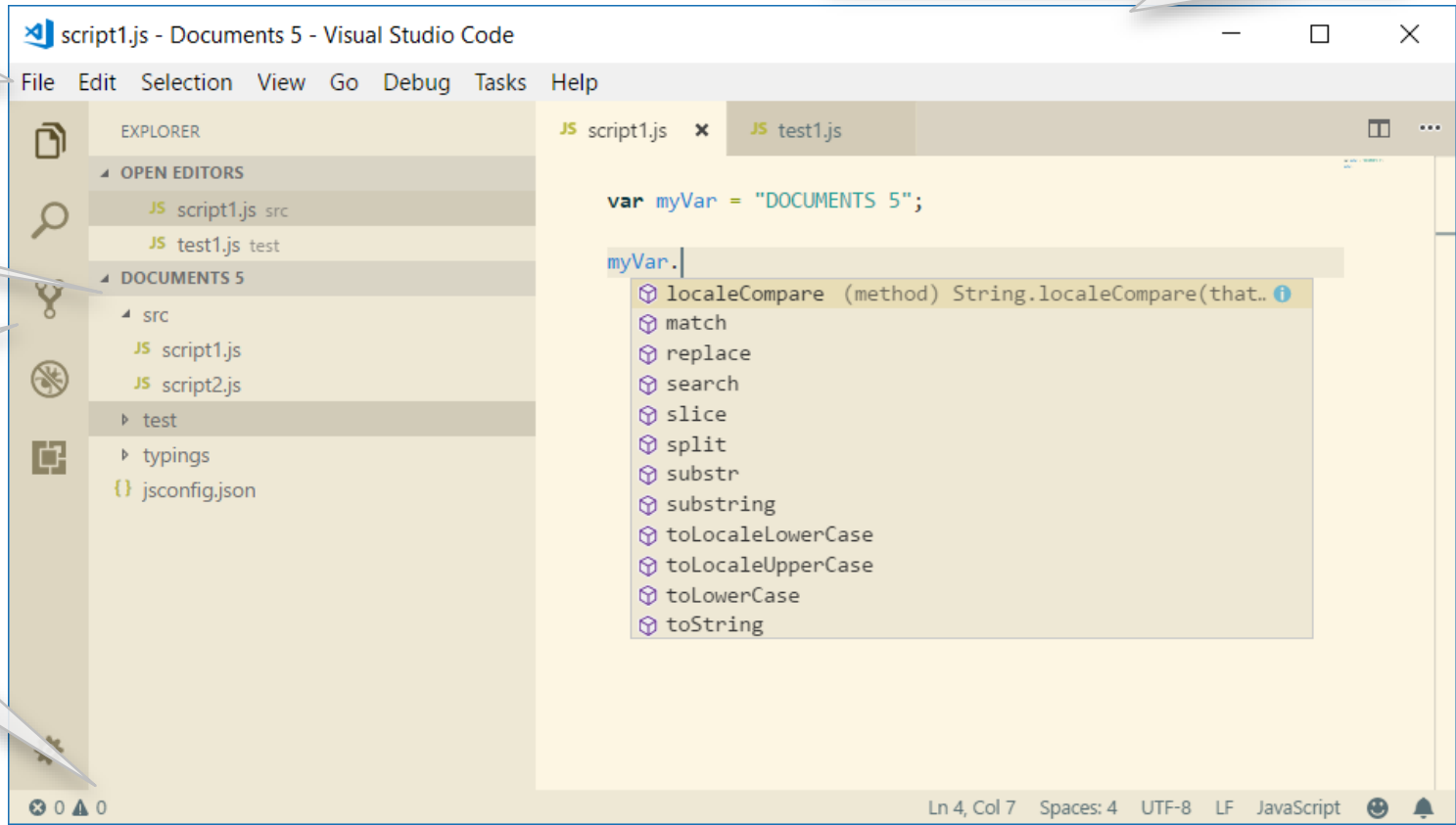
Editor
Unterstützung für JavaScript integriert

Menüzeile

Datei-Explorer

View Bar

Status Bar



Views ein- und ausschalten

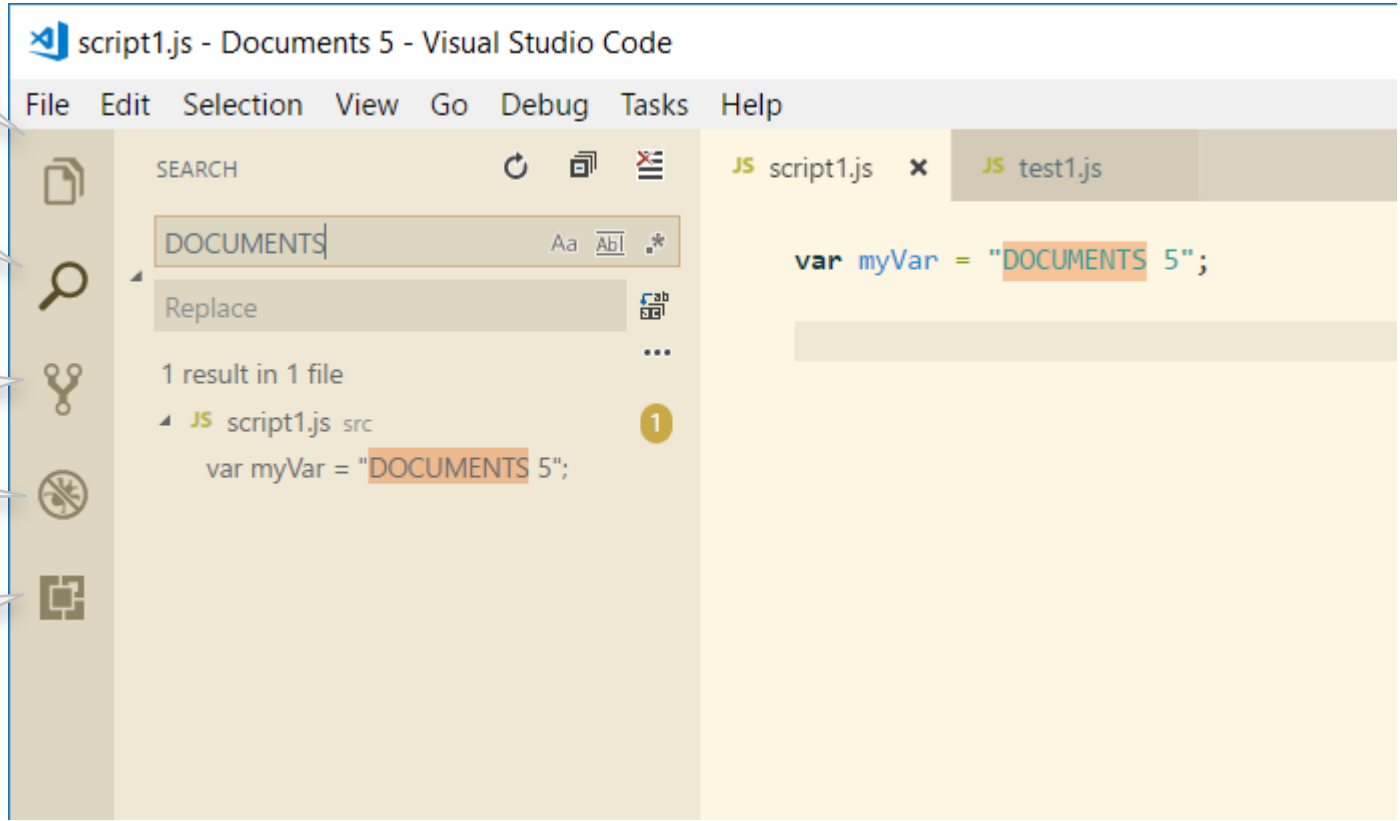
Explorer
(Default-Ansicht)

Suchen in Dateien
(ausgewählt)

Quellcodeverwaltung
(Git integriert)

Debugging

Extensions verwalten
(DOCUMENTS Extension)



Befehle

Tipp

Erstes Zeichen bestimmt den **Modus**

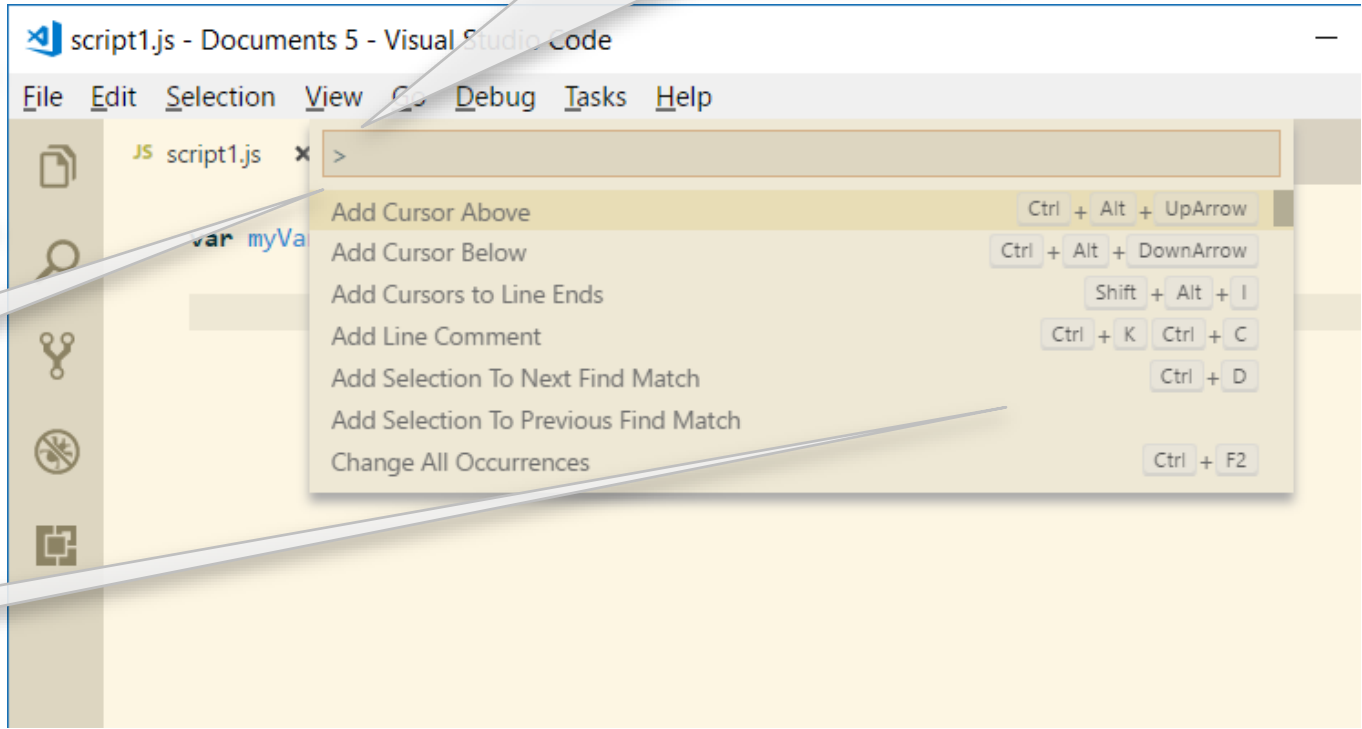
>	**Befehle** anzeigen/ausführen
@	**Symbole** anzeigen/hinspringen
.	**Dateien** anzeigen/öffnen
?	**Modi** anzeigen/wechseln

Menüzeile enthält die üblichen Editor-Befehle

Explorer und Editor enthalten Befehle in Kontextmenüs

Kommando-Palette (F1) enthält alle Befehle

Shortcuts für viele Befehle



Shortcuts

- Übersicht über alle Shortcuts
- Kommando-Palette
  - Befehle anzeigen/ausführen — **F1**
  - Dateien anzeigen/öffnen — **Ctrl + P**
  - Symbole der geöffneten Datei (sortiert) — **Ctrl + Shift + O (: in Palette)**
  - Symbole aus allen Dateien — **Ctrl + T (first Letter in Palette)**
  - Go to last/next position — **Alt + Left/Right Arrow**
- Editor
  - Suggestions (Code Completion) — **Ctrl + Space**
  - Kommentar einfügen/entfernen — **Ctrl + #**
- Shortcuts
  - Shortcuts ändern — **Ctrl + K S**

Konfiguration

- Keine Menüs oder Dialogfenster
- Konfigurationsdateien (JSON) direkt bearbeiten
  - VS Code bietet Informationen und Hilfen bei der Eingabe
- settings.json
  - Allgemeine Einstellungen
- keybindings.json
  - Shortcuts konfigurieren
- launch.json
  - Einstellungen für Debugging
- jsconfig.json
  - Einstellungen für JavaScript IntelliSense (Code Completion)

JavaScript Unterstützung

- Syntax Highlighting, Bracket Matching
- Code Navigation
  - Go to Definition
  - Go to last Position
- Refactoring
  - Code Bereich als Funktion extrahieren
  - Symbol an allen Stellen umbenennen
- **IntelliSense**
  - **Code Completion**, Member Listen, Parameter Info, Quick Info
  - PortalScripting-API: Projekt konfigurieren (Wizard!)

VS Code DOCUMENTS Extension

Name

- JavaScript Remote Debugger for JANUS Apps

Technischer Name

- **`vscode-janus-debug`**

DOCUMENTS Extension Dokumentation

Dokumentation

- Auf GitHub
  - Wiki, `README` und `CHANGELOG`
- `View Bar → Extensions → JavaScript Remote Debugger for JANUS Apps`
  - `README` und `CHANGELOG`

Befehle und Settings

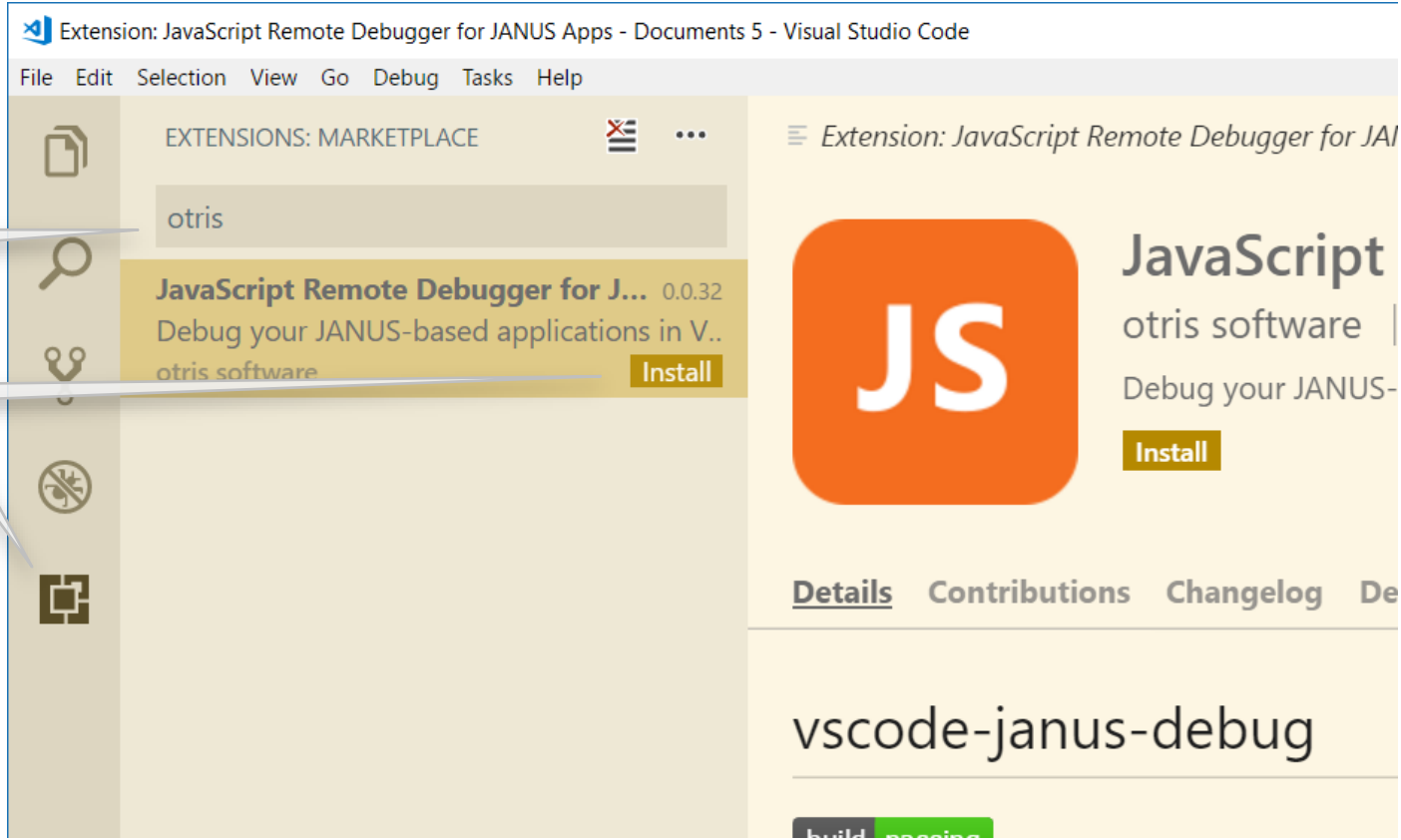
- Liste in `README`
- Kommando-Palette → Filter/Präfix: `DOCUMENTS`
- `settings.json` → Filter/Präfix: `vscode-janus-debug`

DOCUMENTS Extension installieren

1. Bereich für Erweiterungen auswählen

2. `otris` eintippen

3. Install



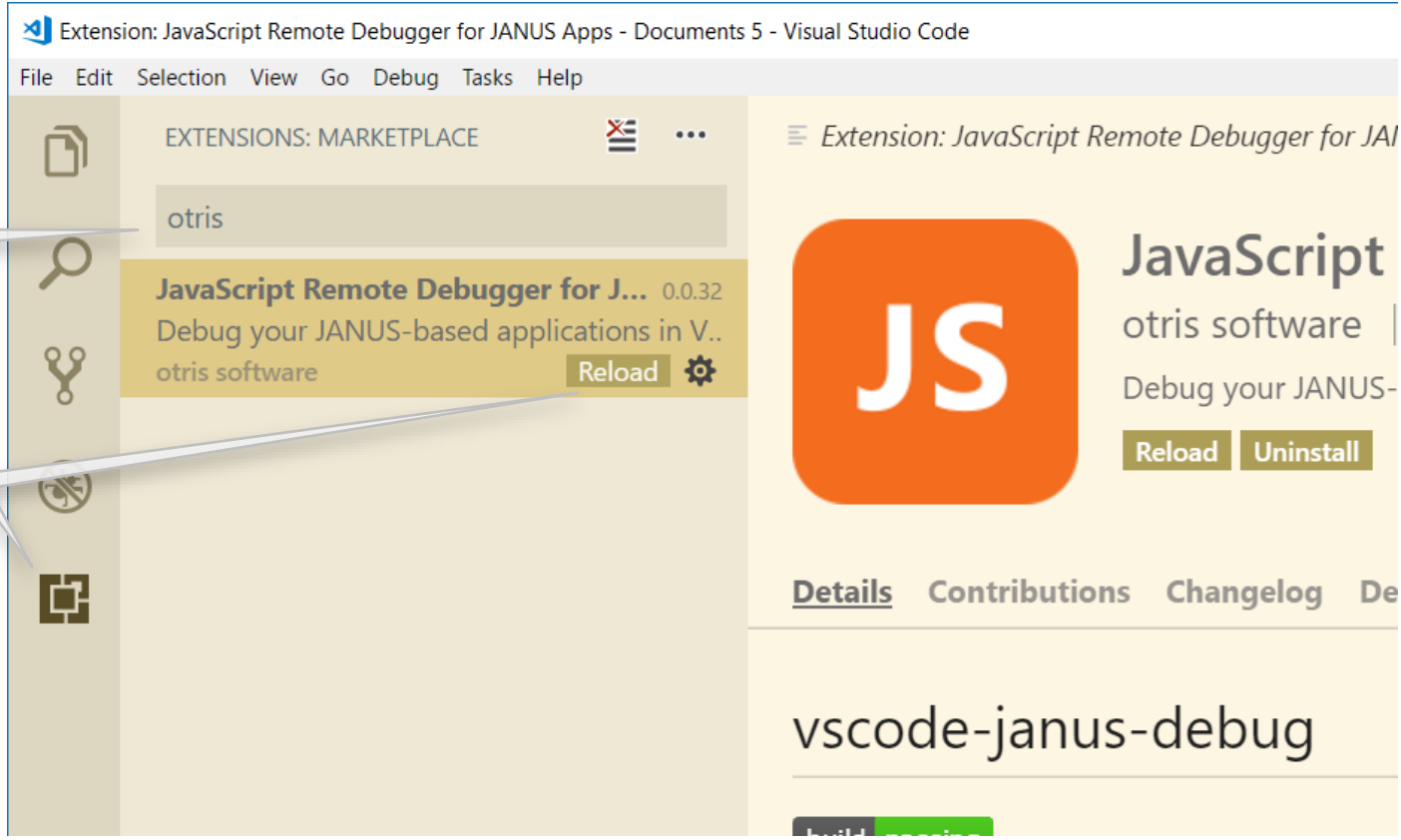
DOCUMENTS Extension installieren

1. Bereich für Erweiterungen auswählen

2. `otris` eintippen

3. Install

4. Reload



DOCUMENTS Extension installieren

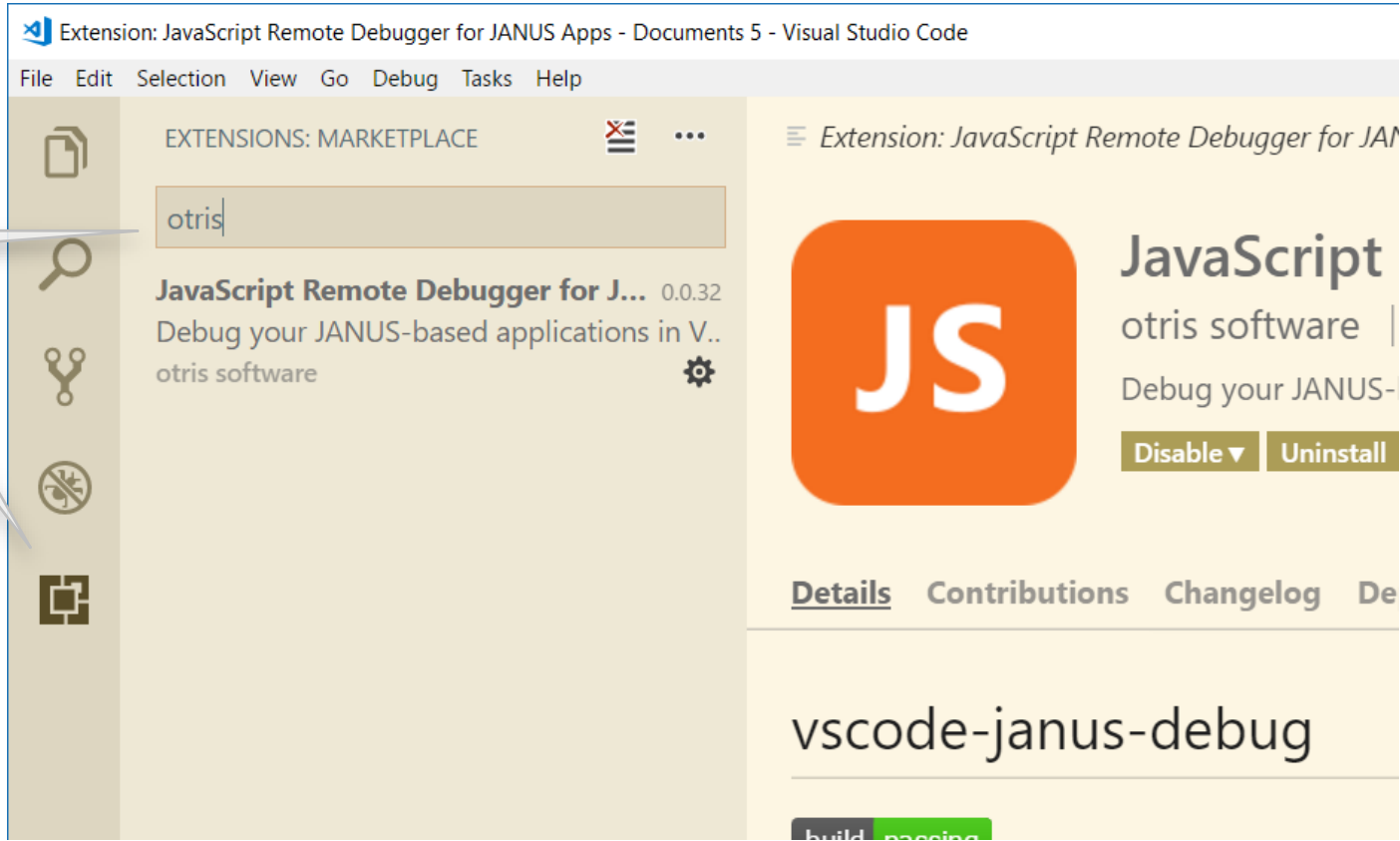
1. Bereich für Erweiterungen auswählen

2. `otris` eintippen

3. Install

4. Reload

Updates werden automatisch installiert!



Erste Schritte

Scripte aus dem Manager mit VS Code „extern bearbeiten“

- Umgebungsvariable setzen
  - `SCRIPTEDITOR = C:\Program Files (x86)\Microsoft VS Code\Code.exe –w -n`
- Nur editieren, keine Features der DOCUMENTS Extension!

Arbeiten im „Projekt“

- Projekt mit Wizard erstellen
- Scripte bearbeiten
  - Extension: Dokumentation, Befehle und Settings
  - PortalScripting API: IntelliSense, Dokumentation

Wizard

1. Windows Explorer Kontextmenü auf leerem Ordner → `mit VS Code öffnen`
2. Befehl `Wizard: Download / Create Project`
3. Unterordner für Kategorien erstellen?
4. Credentials eingeben
   - launch.json
5. Browser auswählen
   - → `View Documentation in Browser`
6. Projekt wird erstellt
   - Alle PortalScripte des DOCUMENTS Servers im Ordner `src`

Credentials eingeben

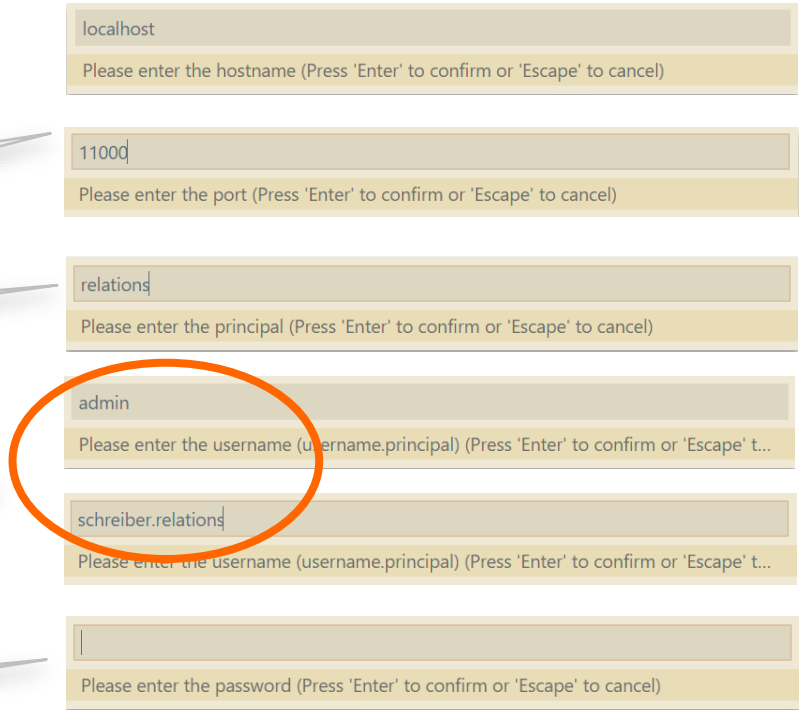
Der erste Befehl, der eine Verbindung zum DOCUMENTS Server aufbaut, fragt alle benötigten Informationen ab

Hostname und Port des Servers

Mandant

Benutzername (Redakteur):
- `admin` oder
- **Benutzername und Mandant getrennt durch einen Punkt**

Passwort



Live Demo

Projekt mit Wizard

Befehle und Settings

Wizard – Projektstruktur

Anmelde-Informationen
Konfiguration / launch.json

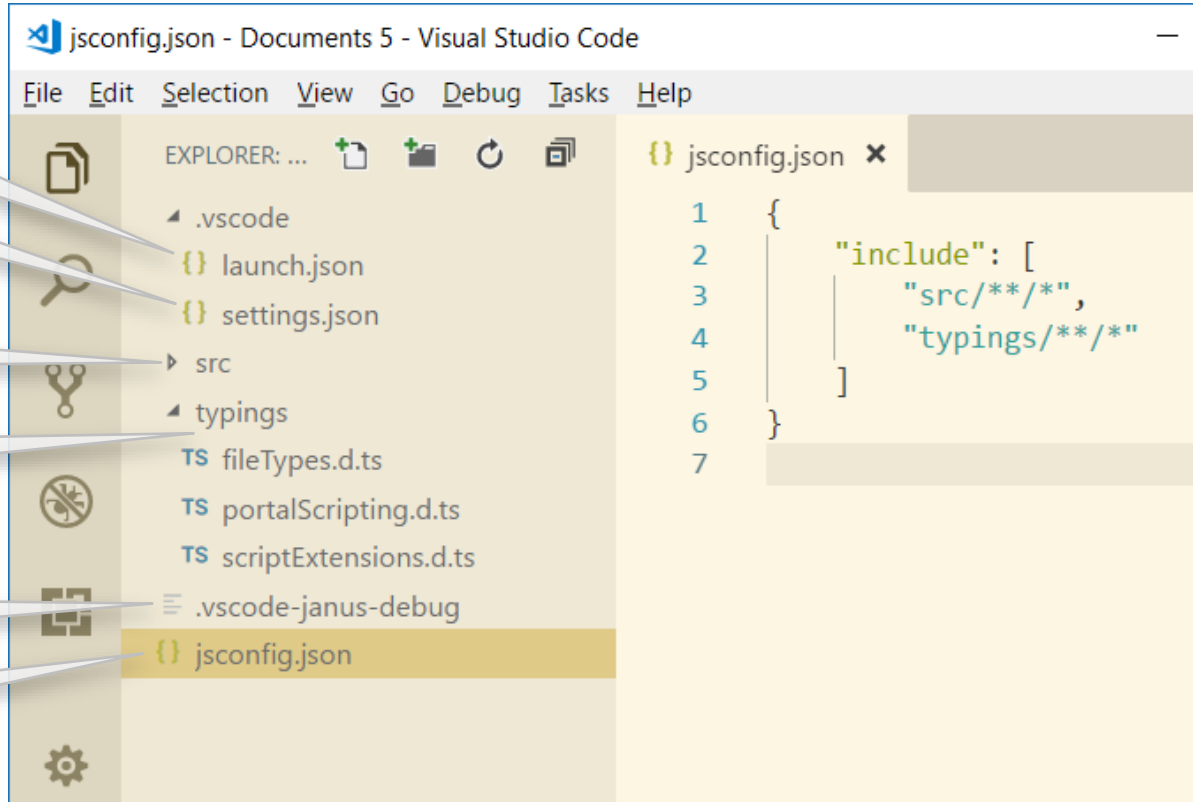
Allgemeine Einstellungen
Konfiguration / settings.json

Alle Scripte
Kategorien: Wiki, Folie

IntelliSense Metadaten
IntelliSense, *.d.ts

Extension Informationen
.vscode-janus-debug

IntelliSense Projektinformationen
IntelliSense / jsconfig.json



settings.json

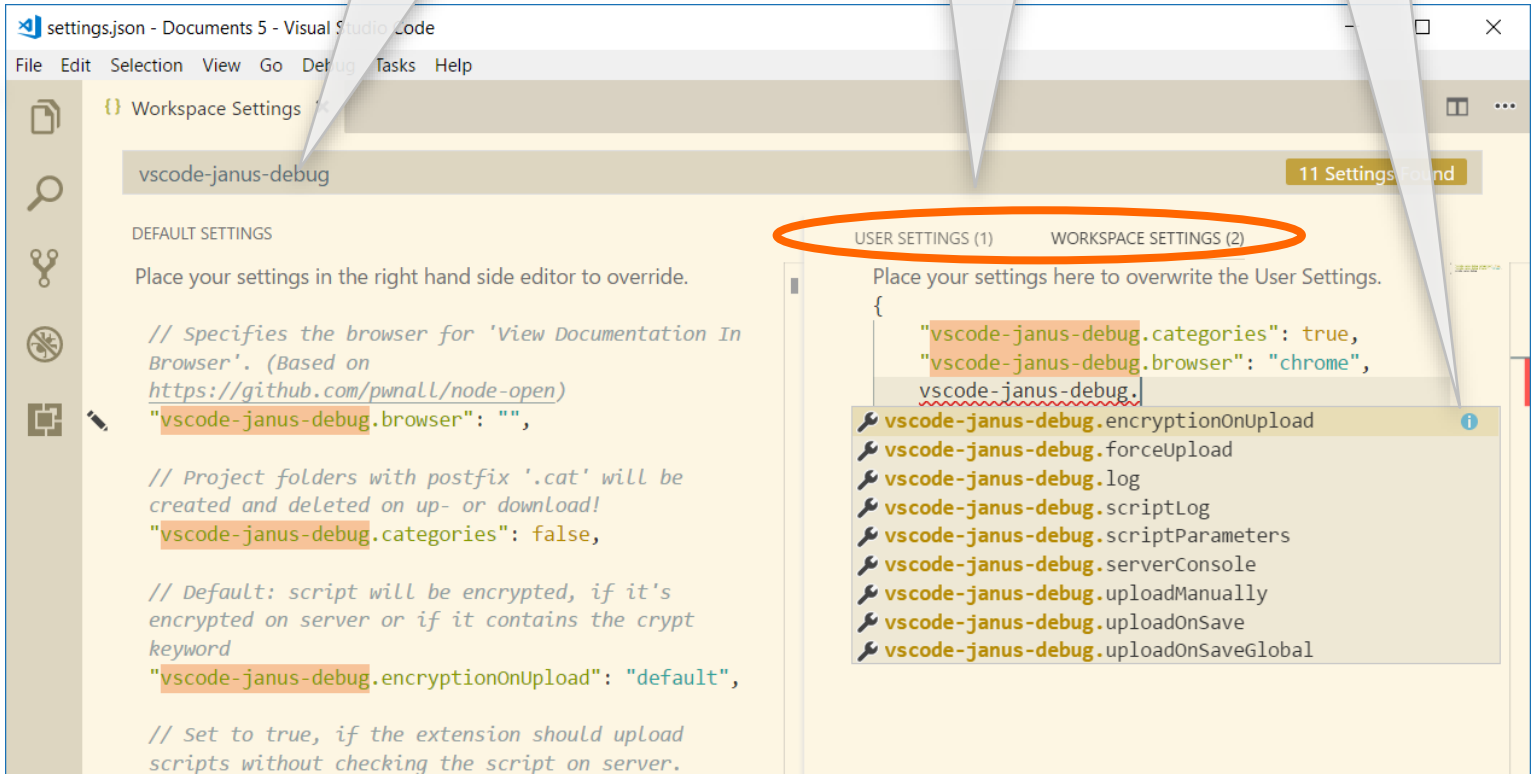
Default Settings mit Doku und Suche

User und Workspace Settings umschaltbar

Dokumentation anzeigen

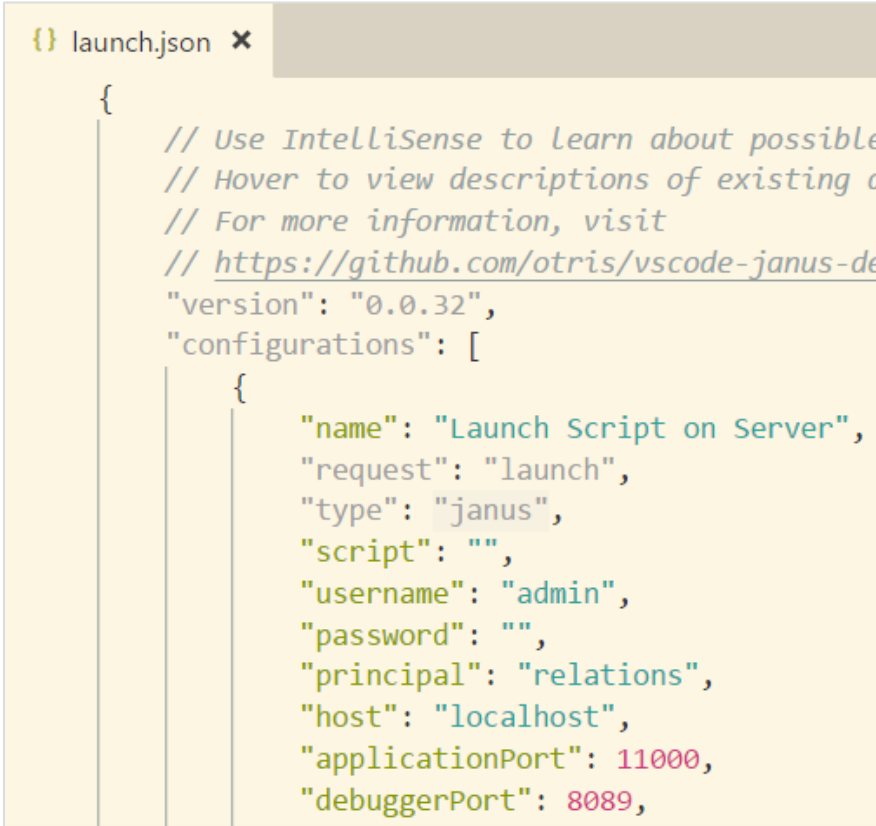
File → Preferences → Settings:

Default-, User- und Workspace-Settings



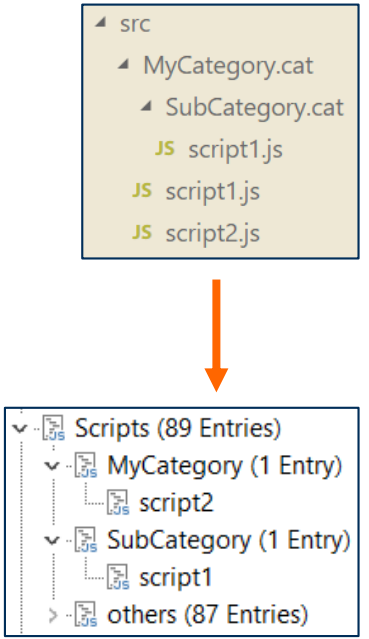
launch.json

- Credentials werden in der launch.json gespeichert
- Passwort (plain!) speichern ist optional
  - Ansonsten Abfrage einmal nach Starten der Extension
- launch.json kann manuell geändert oder erstellt werden
  - Änderungen werden sofort übernommen
  - Benutzername beachten
- Credentials über Eingabebox eingeben
  - → launch.json löschen



Kategorien als Unterordner

- `settings.json`: `vscode-janus-debug.categories = true`
- Wird im Wizard abgefragt!
- Download
  - Scripte aus Kategorie **`MyCategory`** → Unterordner **`MyCategory.cat`**
- Upload
  - Scripte aus Unterordner **`MyCategory.cat`** → Kategorie **`MyCategory`**
  - Warnung (optional) wenn Kategorie geändert wird
- Kategorien ≠ Unterordner
  - Kategorienamen können Sonderzeichen enthalten – Ordnernamen nicht
  - Kategorien haben nur eine Ebene – Ordner haben mehrere Ebenen
  - Scripte mit gleichem Namen in Unterordnern möglich – in Kategorien nicht



Live Demo

PortalScripting API:

- IntelliSense
- Dokumentation

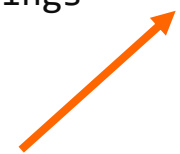
Type Declaration Files – Typings

Allgemein

- Enthalten nur Deklarationen und Typ Informationen
- Ermöglichen IntelliSense für „fremde“ Module
- Liegen meistens im Ordner `Typings`

Befehl `Install IntelliSense`

- Erstellt **PortalScripting Typings** passend zum Server
- Wird vom Wizard aufgerufen!



PortalScripting Typings

`portalScripting.d.ts`

- Allgemeine PortalScripting API
- Abhängig von DOCUMENTS Version

`fileTypes.d.ts`

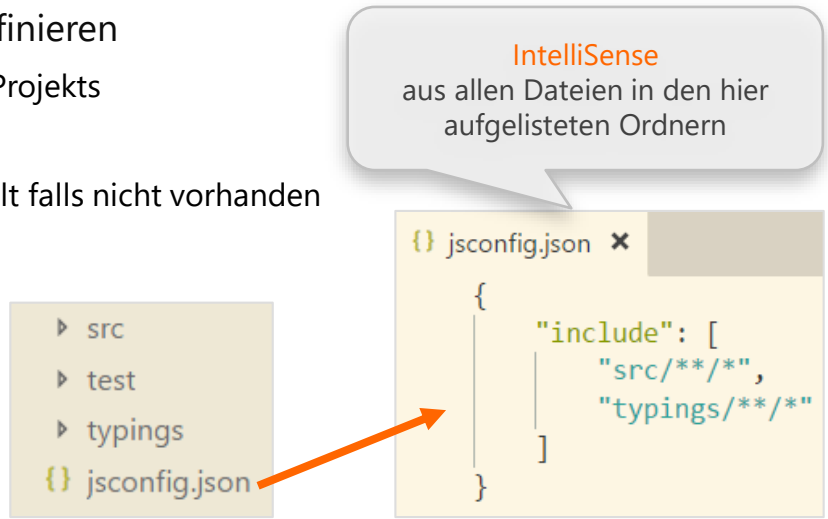
- Mappentypen auf dem Server
- JSDoc Kommentare!

`scriptExtensions.d.ts`

- Scriptlist, ScriptTree, Guided Tour

jsconfig.json

- Default (keine `jsconfig.json`):
  - Alle Dateien unabhängig von einander
  - Kein IntelliSense für Dateien des Arbeitsbereichs
- Mit `jsconfig.json` JavaScript Projekt definieren
  - IntelliSense aus allen Dateien des JavaScript Projekts
  - Projekt definieren über `include` Porperty
  - In **`Install IntelliSense / Wizard`** erstellt falls nicht vorhanden



JSDoc

- Typ einer Variable/Funktion in JSDoc Kommentar angeben
  - → IntelliSense an der Variable/Parameter für angegebenen Typ
- Tipp: über einer Funktion `/**  Enter` eingeben → alle `@param` automatisch

Typ durch Definition klar

```
var myVar = "DOCUMENTS 5";
myVar.
```

- localeCompare
- match
- replace
- search
- slice
- split
- substr
- substring

Return-Typ von getString() nicht dokumentiert

```
/** @type {string} */
var myVar = anyModule.getString();
myVar.
```

- localeCompare
- match
- replace
- search
- slice
- split
- substr
- substring

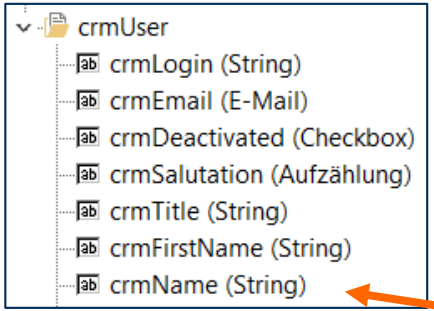
JSDoc Kommentar für Parameter

```
/**
 * @param {string} param
 * @returns {string}
 */
function myFunction(param) {
    param.
}
```

- localeCompare
- match
- replace
- search
- slice

IntelliSense für Mappentypen

crmUser
Mappentyp im Manager



context.createFile
gibt den Typ
crmUser zurück

```
var myDocFile1 = context.createFile("crmUser");
myDocFile1.
```
- crmDeactivated?
- crmEmail?
- crmFax?
- crmFirstName?
- crmLogin?
- crmMobilePhone?
- crmName?

context.file
gibt den Typ
DocFile zurück

```
var myDocFile2 = context.file;
myDocFile2.
```
- abort
- addDocumentFromFileSystem
- addPDF
- archive
- archiveAndDelete
- cancelWorkflow
- changeFiletype

context.file
mit JSDoc Kommentar
gibt den Typ **crmUser** zurück

```
/**
 * @type {crmUser}
 */
var myDocFile2 = context.file;
myDocFile2.
```
- crmDeactivated?
- crmEmail?
- crmFax?
- crmFirstName?
- crmLogin?
- crmMobilePhone?
- crmName?

crmName
hat den Typ
String

```
/**
 * @type {crmUser}
 */
var myDocFile2 = context.file;
var myName = myDocFile1.crmName;
myName.
```
- charAt
- charCodeAt
- concat
- indexOf
- lastIndexOf
- length

Zusammenfassung IntelliSense

Type Declaration Files

- Enthalten Deklarationen und Typ-Informationen
- IntelliSense für externe Module

jsconfig.json

- JavaScript Projekt definieren
- IntelliSense für Dateien innerhalb des JavaScript Projekts

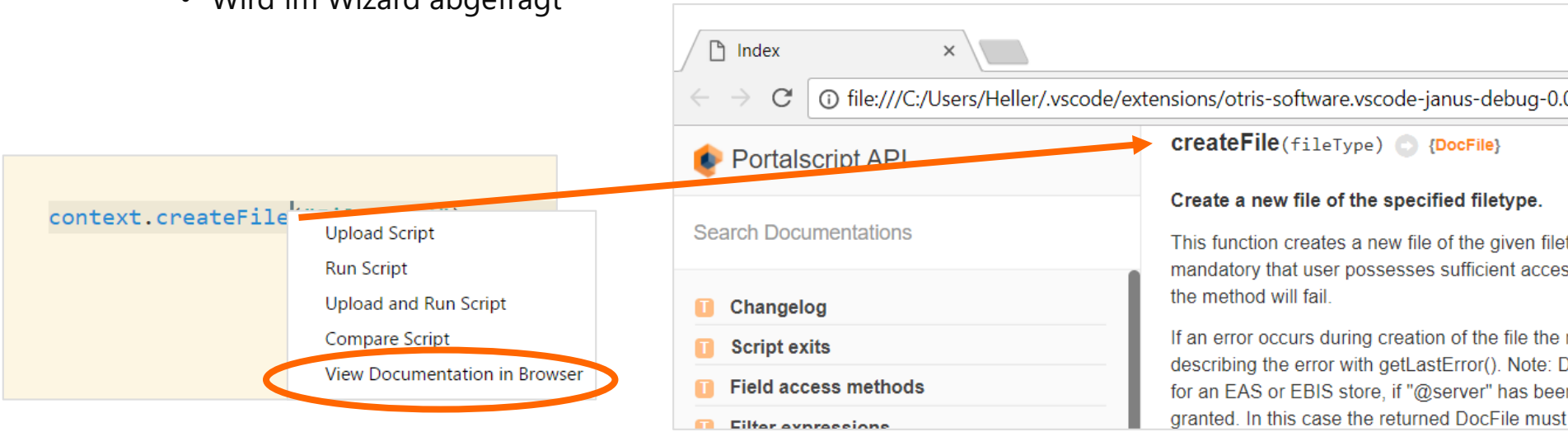
JSDoc

- Dokumentation und Typ-Informationen für JavaScript
- IntelliSense verbessern durch Angabe von Datentypen

PortalScripting Dokumentation

Dokumentation über Kontextmenü im Editor öffnen

- Befehl: `View Documentation in Browser`
- `settings.json:` `vscode-janus-debug.browser` setzen!
  - → Sprung direkt zu Member/Funktion
- Wird im Wizard abgefragt



Fazit

VS Code ist ein schöner Editor mit…

- guten Entwicklungswerkzeugen
- DOCUMENTS Anbindung
  - Upload/Download (mit Kategorien), Run, Compare, …
- IntelliSense für PortalScripting, Mappentypen, …
- Integrierter Dokumentation für PortalScripting
- Integrierter Server Konsole
- weiteren Features zu Verschlüsselung, Script-Parameter, Auto-Upload

und direkt aus dem otrisLAB  **ab auf die DOPaK…**

DOCUMENTS

VS Code
> JS Debugger

otrisLAB

JS Debugger: Zielsetzung

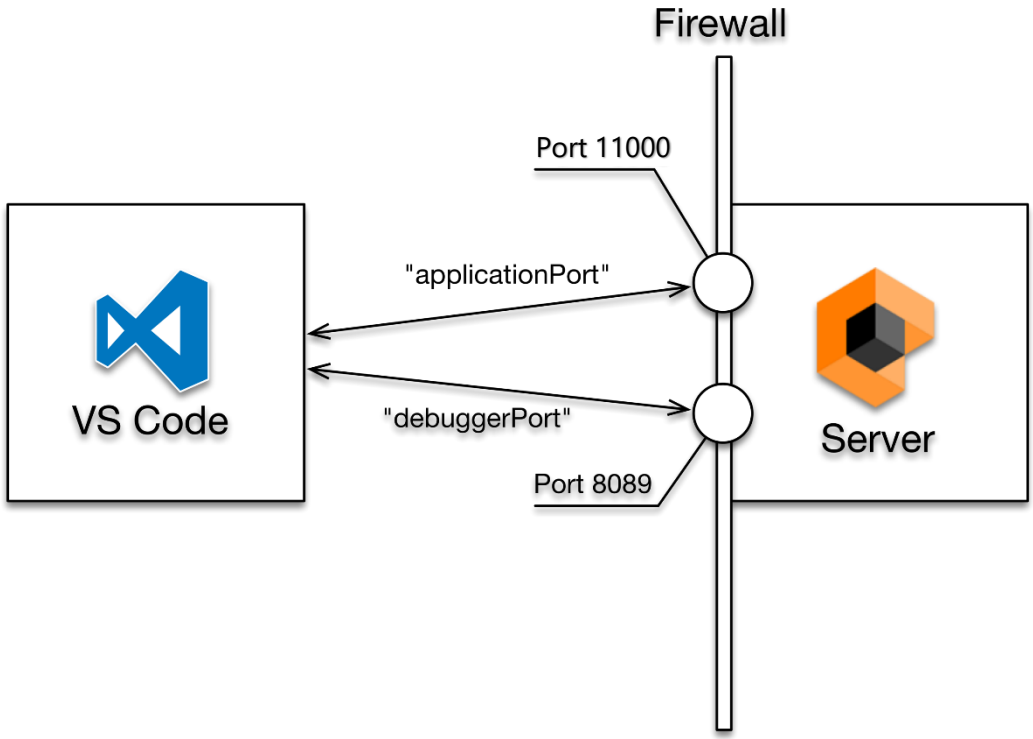
PortalScripte direkt aus dem VS Code heraus debuggen

Attach an laufende PortalScripte in DOCUMENTS 5

Wie ein „normaler“ moderner Debugger

- Während der Ausführung Breakpoints setzen
- Continue (F5)
- Next (F10)
- Step-in (F11)
- Variablen auslesen
- Support für das `debugger;` - Statement

JS Debugger: Architektur



JS Debugger: aktivieren

Unterstützt ab DOCUMENTS 5.0d

In die documents.ini eintragen

- JSDebugger yes
- [JSDebuggerPort 8089]
  - oder beliebigen anderen TCP Port, optional

Danach evtl. Firewall-Regeln und Security Groups anpassen

DOCUMENTS Server neu starten

Nicht in Live-Umgebungen verwenden!

Live Demo
JS Debugger

JS Debugger: Aktueller Stand

PortalScripte aus VS Code starten

- Breakpoint setzen
- Upload und Debug Script

Nicht durch VS Code gestartete PortalScripte debuggen (attach)

- Statement **`debugger;`** als „Breakpoint“ in das Script einfügen und hochladen
- Scripting-Engine stoppt implizit bei Erreichen des Statement **`debugger;`**
- VS Code: Debuggen > Attach to Server
  - ggfs Context auswählen, falls mehrere Scripte laufen

JS Debugger: Kommunikationsprotokoll

```
contextId/{ JSON payload }\n
```

Request

```
1/{"type":"command","name":"step","id":"0815"}
1/{"type":"command","name":"set_breakpoint","breakpoint":{"url":"test.js","line":22},"id":"4711"}
```

Response

```
1/{"type":"info","subtype":"paused","url":"test.js","line":22,"source":"util.out(\"Hello, world\");","id":"4711"}
```

otris software
vereinfacht Verantwortung.

Vielen Dank!

Christine Heller

Thomas Richter

www.otris.de

otris software AG
Königswall 21
44137 Dortmund