d.velop

d.3 hook & server scripting api (groovy)

d.velop AG
Schildarpstraße 6–8
Germany, 48712 Gescher

Fon +49 2542 9307-0
Fax +49 2542 9307-20
www.d-velop.de
info@d-velop.de

d.velop

Inhaltsverzeichnis

1 Impressum/ Rechtliche Hinweise

Alle bisherigen Dokumentationen zu d.3 server scripting api (groovy) verlieren mit der Veröffentlichung dieser Dokumentation ihre Gültigkeit.

Die in dieser Dokumentation enthaltenen Informationen sind mit größter Sorgfalt erstellt und durch unsere Qualitätssicherung nach dem allgemeinen Stand der erprobten Technik geprüft. Dennoch sind Fehler nicht auszuschließen. Aus diesem Grund stellen die in der vorliegenden Dokumentation enthaltenen Informationen keine Verpflichtung, zugesicherte Eigenschaft oder Garantie dar. Die d.velop AG übernimmt auf Basis dieser Dokumentation keine Haftung oder Gewährleistung. Ansprüche nach dem Produkthaftungsgesetz sowie nach Deliktsrecht bleiben unberührt, sofern sie nicht individualvertraglich ausgeschlossen wurden.
Aussagen über gesetzliche, rechtliche und steuerliche Vorschriften und deren Auswirkungen haben nur für die Bundesrepublik Deutschland Gültigkeit.
Die d.velop AG behält sich vor, in ihrer Software vorhandene Komponenten von Drittanbietern durch funktionsadäquate Komponenten anderer Hersteller zu ersetzen. Die d.velop AG behält sich in Ausübung Ihrer jeweils gültigen Releasepolitik vor, Produktfeatures und einzelne Softwareprodukte nicht mehr durch Softwarepflege- und Supportleistungen zu unterstützen. Näheres dazu finden Sie im Supportlebenszyklus des d.velop-Service-Portals unter https://portal.d-velop.de.

Sofern Teile oder einzelne Formulierungen dieser Dokumentation der geltenden Rechtslage nicht, nicht mehr oder nicht vollständig entsprechen sollen, bleiben die übrigen Teile der Dokumentation in ihrem Inhalt und ihrer Gültigkeit davon unberührt.
In der Dokumentation können Sie über Links zu externen Internetseiten gelangen, die nicht von uns betrieben werden. Derartige Links werden von uns entweder eindeutig gekennzeichnet oder sind durch einen Wechsel in der Adresszeile Ihres Browsers erkennbar. Für die Inhalte dieser externen Internetseiten sind wir nicht verantwortlich.

Kontakt

d.velop AG

Schildarpstraße 6-8

48712 Gescher, Deutschland

Fon +49 2542 9307-0

d-velop.de oder info@d-velop.de

Vertreten durch den Vorstand: Christoph Pliete (Vorsitzender), Mario Dönnebrink

Vorsitzender des Aufsichtsrates: Dr. Helmut Bäumer

Handelsregister beim Amtsgericht Coesfeld, Nr. HRB 4903

Umsatzsteueridentifikationsnummer: DE 813062165

Bei Fragen zu dieser Dokumentation oder zur Software wenden Sie sich bitte an uns.

Fon +49 2542 9307-6000

support@d-velop.de

Kontakt

d.velop AG

Schildarpstraße 6-8
48712 Gescher, Deutschland

Fon +49 2542 9307-0

d-velop.de

info@d-velop.de

Vertreten durch den Vorstand: Christoph Pliete (Vorsitzender), Mario Dönnebrink

Vorsitzender des Aufsichtsrates: Dr. Helmut Bäumer

Handelsregister beim Amtsgericht Coesfeld, Nr. HRB 4903

Umsatzsteueridentifikationsnummer: DE 813062165

Bei Fragen zu dieser Dokumentation oder zur Software wenden Sie sich bitte an uns.

Fon +49 2542 9307-6000
support@d-velop.de

Dieses Dokument wurde zuletzt am 12.02.2019 überarbeitet und bezieht sich auf d.3 server scripting api (groovy) ab Version 8.1.0.
Name des Dokuments: d3serverscriptingapigroovy.pdf (Buildnummer: 20190212)

2 Einleitung

2.1 Über diese Dokumentation

Dies ist eine Dokumentation zum Groovy Skripting mit d.3 server ab Version 8.1.0.
Diese Dokumentation enthält Informationen, wie d.3-Hook-Funktionen und Skripte mit Groovy entwickelt werden können und welche Schnittstellen und Funktionen d.3 server dafür bereitstellt.

Diese Dokumentation steht Entwicklungspartnern der d.velop AG im Service Portal online bereit. Die Weitergabe dieser Dokumentation oder von Teilen daraus ist nicht gestattet. Bei Anfragen im Rahmen der Entwicklungspartnerschaft gilt stets nur die Onlinedokumentation.

Bitte beachten Sie, dass Ihre Software über diese Schnittstelle auch Zugriff auf die von Ihren Kunden im d.3ecm abgelegten und konfigurierten Daten erhält und Eingriff in die Abläufe im d.3ecm-System nimmt. Gehen Sie daher bitte sorgfältig vor und achten Sie darauf, dass Ihre Anwendung Teil eines bestehenden Zusammenspiels unterschiedlicher Anwendungen ist. Die unsachgemäße Verwendung dieser Schnittstelle kann veränderte Programmabläufe und Datenverlust zur Folge haben.

Die Software-Entwicklung mit dieser Programmierschnittstelle ist Individualentwicklung. Der von Ihnen erzeugte Programmcode fällt nicht unter die Pflege- und Supportbedingungen der Produkte der d.velop AG. Unser Support unterstützt Sie gerne, Ihre Anfragen sind jedoch kostenpflichtig, sofern sich die Anfrage nicht auf einen Fehler in unseren Produkten zurückführen lässt.

Alle Fragen zu den Voraussetzungen und zur Software-Entwicklung mit d.3ecm beantwortet Ihnen gerne das Technology Partner Management der d.velop AG.

2.2 Voraussetzungen

Voraussetzung für die Nutzung sämtlicher Java/ Groovy-Funktionalitäten ist Aktivierung des Java/ Groovy-Supports in d.3 config.

> **Hinweis**
>
> Der Java/ Groovy-Support ist in d.3 Server Version 8.0.x im Standard aktiviert.
>
> Weitere Informationen entnehmen Sie der Dokumentation zu d.3 admin.

Zu diesem Zweck bringt d.3 server die Java Laufzeitumgebung von Sun/Oracle in der Version 8 mit, deren Startmodul im gleichen Abschnitt voreingestellt ist.

2.3 Groovy

Groovy ist eine populäre, dynamische Skriptsprache für die Java Virtual Maschine.

Durch die enge Integration mit Java steht die ganze Java-Welt mit vielen umfangreichen Bibliotheken zur Verfügung.

Groovy besitzt einige Fähigkeiten, die in Java nicht vorhanden sind: Native Syntax für Maps, Listen und Reguläre Ausdrücke, ein einfaches Templatesystem, mit dem HTML- und SQL-Code erzeugt werden kann, eine XQuery-ähnliche Syntax zum Ablaufen von Objektbäumen, Operatorüberladung und eine native Darstellung für BigDecimal und BigInteger.

Groovy wird nicht wie andere Skriptsprachen über einen interpretierten Abstract Syntax Tree ausgeführt, sondern vor dem Ablauf eines Skripts direkt in Java-Bytecode übersetzt. Syntaktisch ist Groovy viel weiter von Java entfernt als BeanShell, dafür aber viel näher zu Ruby und Python.

Mehr Informationen finden Sie auf der offiziellen Groovy Website http://www.groovy-lang.org/. Dort gibt es Tutorials und eine Dokumentation für Anfänger, wie auch für fortgeschrittene Benutzer der Sprache.

3 Entwicklungsumgebung

Da es sich bei Groovy Code um schlichte Textdateien handelt, können diese jederzeit mit einem einfachen Texteditor bearbeitet werden.

Zur effektiven Hook-Entwicklung empfiehlt sich jedoch die Benutzung einer Entwicklungsumgebung wie Eclipse, welche den Benutzer durch Features wie automatische Code-Vervollständigung unterstützen.

Die Java-Klassen der d.3-Server-Schnittstelle und Groovy-Unterstützung sowie der Groovy-Interpreter selbst befinden sich im Java Archive **groovyhook.jar** im d.3 server-Programmverzeichnis (Standard: C:\d3\d3server.prg).

In Eclipse kann dieses in den **Project Properties** unter **Java Build Path als External JAR** eingebunden werden.

In dem Zusammenhang sollte auch gleich ein Groovy-PlugIn für Ecplise installiert werden, um direkt die bestmögliche Unterstützung für Syntax Highlighting und Kommandoergänzung nutzen zu können.

> **Hinweis**
>
> Bei der Entwicklung und dem Test von Groovy Programmcode sollte der d.3 config-Schalter **Neuladen von Groovy-Hookdateien bei Änderung aktivieren** (RELOAD_ON_CHANGE) aktiviert werden.
> Jedes Speichern von Groovy-Hookdateien führt dann dazu, dass diese von den Server Prozessen automatisch neugeladen werden und die Code-Änderungen sofort aktiv sind.

> **Wichtig**
>
> Das Aktivieren der "Neuladen-Option" sollte aus Sicherheitsgründen NICHT im Produktivsystem vorgenommen werden. Bei Änderungen an den Groovy-Skripten würden diese sofort produktiv aktiv!

3.1 Eclipse als Entwicklungsumgebung

Einrichtung von Eclipse als Entwicklungsumgebung für die Hook-Entwicklung

1. Laden und installieren Sie ein Java Development Kit (JDK) für Java.

   > **Wichtig**
   > Oracle hat seine Lizenzbedingungen bezüglich seiner Java Distribution überarbeitet, ab Februar diesen Jahres (2019) nicht mehr kostenlos nutzbar. Dies gilt für JDK/JRE-Updates die nach Januar diesen Jahres bei Oracle bezogen wurden.
   > Aus diesem Grund werden alle betroffenen Produkte künftig mit dem OpenJDK (https://openjdk.java.net/) ausgeliefert und bis dahin in Hotfixen maximal *"Oracle Java 8 Update 201"* verwendet. Bezüglich dieser Lizenz ist jedoch ab jetzt zwingend darauf zu achten, dass in der von uns ausgelieferten Distribution (jeweils der "jre"-Unterordner) keine Anpassungen unsererseits oder durch Kunden vorgenommen werden. Unsere Kunden können bestehende Installationen unserer Software weiterhin betreiben und werden beim nächsten Update (bis auf einen Hinweis in den Readme's) voraussichtlich nichts von diesem Wechsel mitbekommen.

2. Laden Sie **Eclipse IDE for Java Developers** von www.eclipse.orgin herunter und entpacken Sie diese.
3. Starten Sie die entpackte **eclipse.exe**.
4. Wählen Sie ein geeignetes Verzeichnis für ihren `Workspace`. Der Workspace ist ein Verzeichnis, in dem Eclipse ihre Projekte verwaltet. Geben Sie hier KEIN Verzeichnis vom d.3-Server an, sondern in Ihren eigenen Dateien.

Nach dem Start von Eclipse muss noch das Groovy-Plugin für Eclipse installiert werden.

1. Unter **Help** wählen Sie **Install New Software...**.
2. Bestimmen Sie unter https://github.com/groovy/groovy-eclipse/wiki die für Ihre Eclipse-Version passende **Update Site**. Eine Update Site ist eine Webadresse, über die Eclipse auf automatisch Software installieren kann.
3. Unter **Work with** tragen Sie diese Update Site ein und bestätigen.
4. Aktivieren Sie das Feature **Groovy-Eclipse (Required)**.
5. Führen Sie die Installation durch.

Ihre Eclipse-Installation ist jetzt bereit zur Programmierung von d.3-Hooks mit Groovy.

Hinweis

Nach dem ersten Start von Eclipse in einem neu erstellten Workspace müssen Sie evtl. von der Willkommensseite zur Standardansicht wechseln, indem Sie die Schaltfläche **Workbench** in der oberen rechten Ecke anwählen.

Falls das Zielsystem keinen Internetzugang zulässt
Das oben beschriebene Vorgehen kann auch auf einem separaten Rechner durchgeführt werden, um die Eclipse-Umgebung einmal zusammenzustellen.

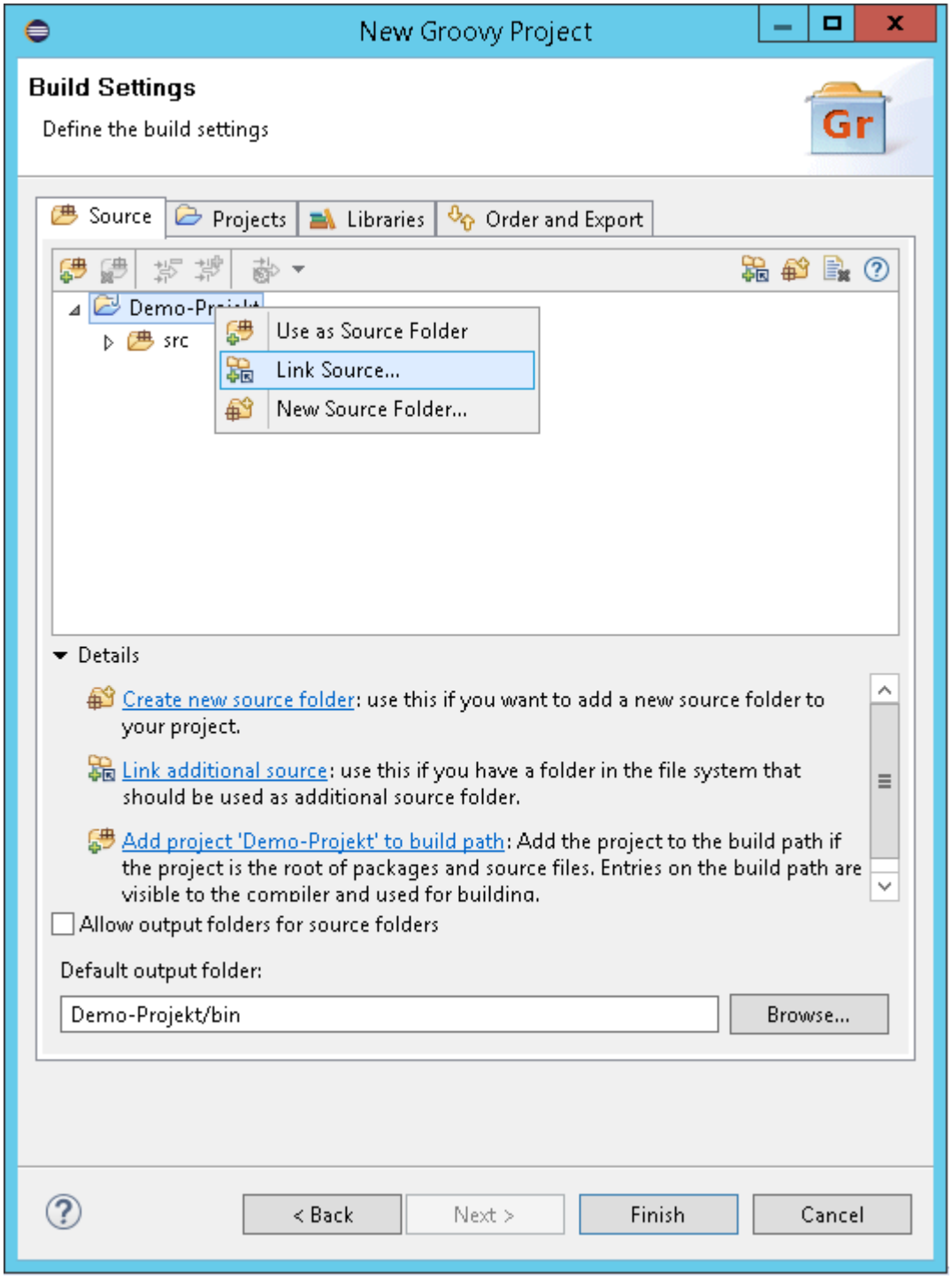
Sobald alles fertig ist, kann der Eclipse-Ordner einfach auf die Zielmaschine kopiert werden, es muss nicht explizit installiert werden.

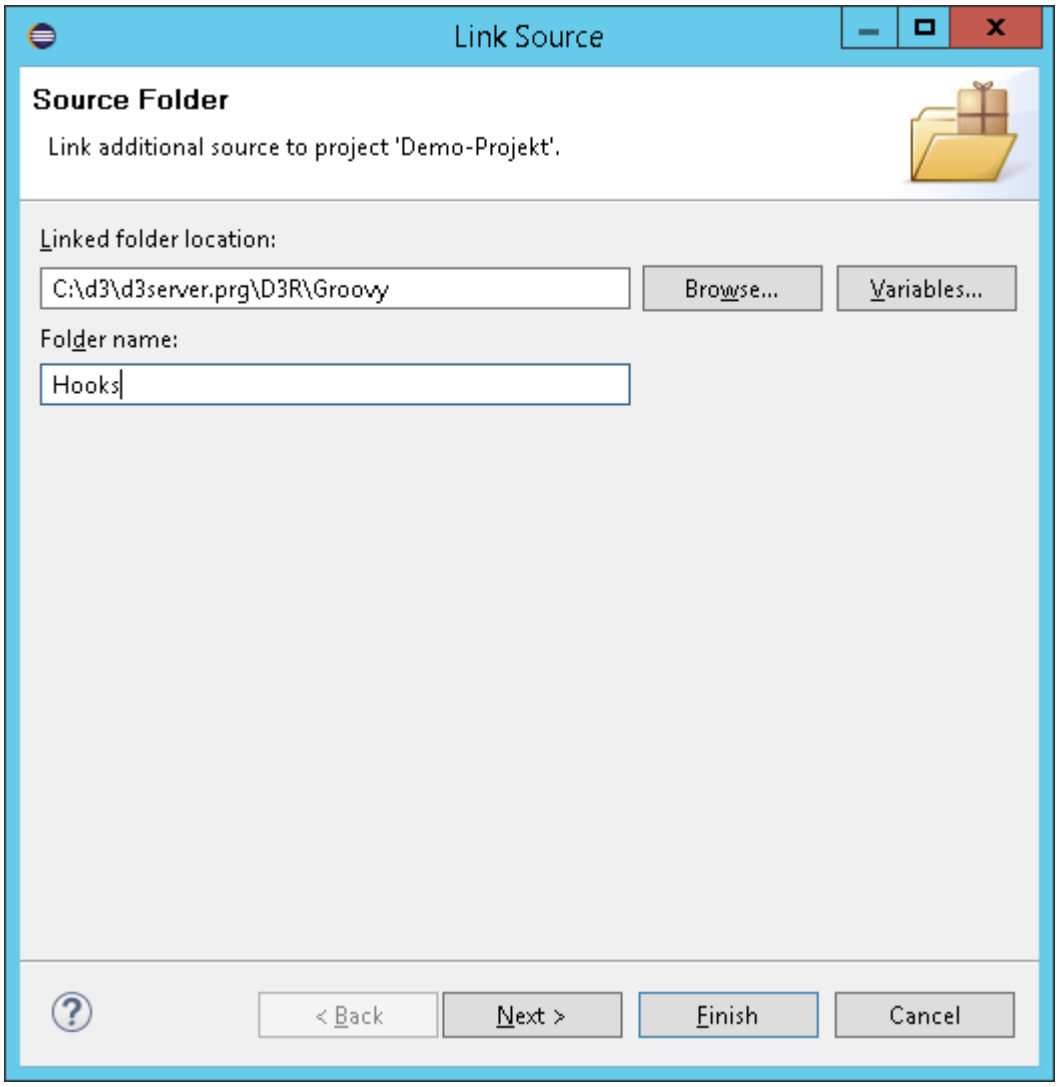
3.2 Erstellen von Hook-Projekten

Um Hooks zu programmieren, erstellen Sie ein neues Projekt: **File > New > Projekt**.

1. Wählen Sie **Groovy Projekt** und vergeben Sie auf der nächsten Seite einen sprechenden Namen.
2. Auf der Seite **Build Settings** fügen Sie ein neues Quelltext-Verzeichnis hinzu, indem Sie auf das Projekt rechtsklicken und **Link Source** auswählen.
3. In dem sich öffnenden Fenster geben Sie als **Linked folder location** das in d.3 admin definierte Verzeichnis für Ihre Groovy-Hooks an (siehe Groovy-Hook-Verzeichnisse für kundenspezifische Programmanpassungen).

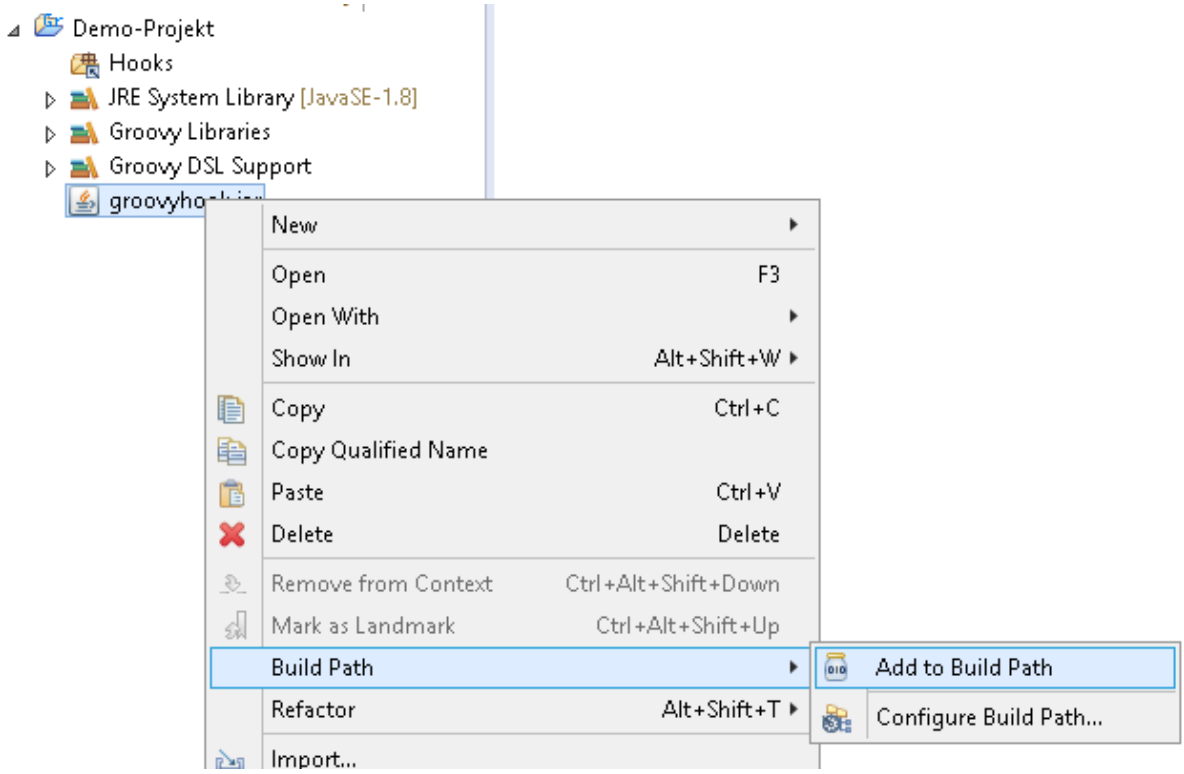
4. Als **Folder name** wählen Sie etwas Sprechendes wie z.B. "Hooks" und bestätigen Sie mit **Finish**.



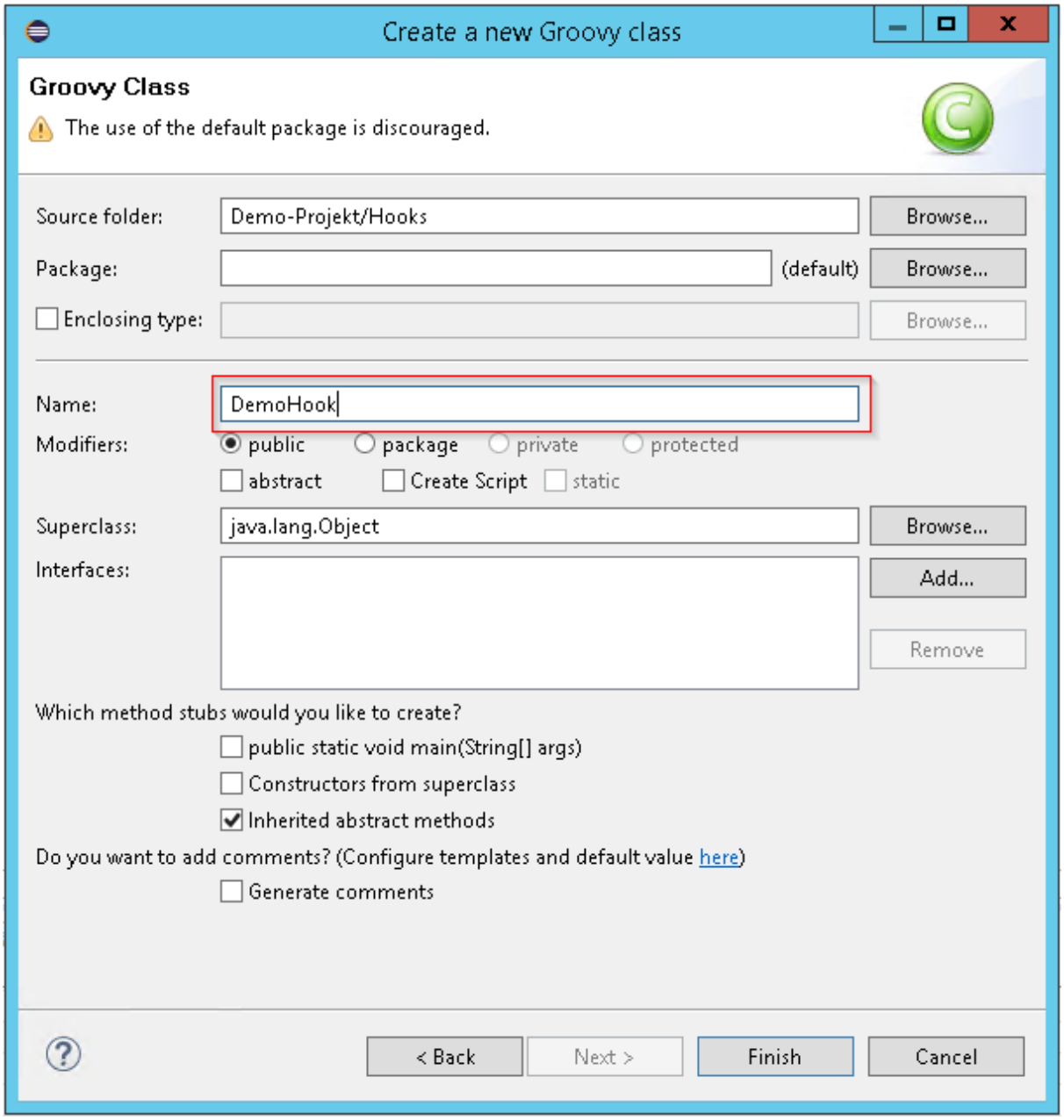


5. Erstellen Sie das Projekt mit **Finish**.
6. Sie können den Ordner **src** in Ihrem soeben erstellten Projekt nun löschen.
7. Suchen Sie im Installationsverzeichnis Ihres d.3-Servers (Standard: C:\d3\d3server.prg) die Datei **groovyhook.jar** und kopieren Sie diese in Ihr Eclipse-Projekt.
8. Fügen Sie die **groovyhook.jar** zum **Build Path** hinzu, indem Sie darauf rechtsklicken und aus Menü **Build Path > Add to Build Path** wählen. Wenn sich die Eclipse sowie d.3 server-Installation auf dem gleichen Rechner befinden, kann die **groovyhook.jar** auch direkt als **External JAR** eingebunden werden, ohne diese kopieren zu müssen.

> **Wichtig**
> Wenn die für das Groovy-Projekt automatisch in den Java Build Path aufgenommene Version der Groovy Libraries neuer ist, als die in der **groovyhook.jar** enthaltene, dann meldet Eclipse einen Versionskonflikt. In diesem Fall müssen im Tab **Libraries** die Einträge **Groovy DSL Support** und **Groovy Libraries** aus dem Java Build Path entfernt werden.



9. Erstellen Sie neue Hook-Klassen, indem Sie im Kontextmenü des Ordners **Hooks** unter **New | Other** den Typ **Groovy Class** auswählen.
10. In dem Assistenten vergeben Sie einen sprechenden Namen für Ihren Hook und bestätigen mit **Finish**.



Sie können nur Ihre Hook-Funktionen implementieren.

Sollten Sie in d.3 admin konfiguriert haben, dass bei Änderungen an Hooks diese automatisch neu geladen werden, müssen Sie in Eclipse nur speichern, um Änderungen zu testen. Nur wenn Sie das automatische Neuladen deaktiviert haben, müssen Sie Ihre Repositoryprozesse neustarten, um Änderungen zu übernehmen.

3.3 IntelliJ IDEA als Entwicklungsumgebung

Wenn statt eclipse die IntelliJ Idea genutzt werden soll, ist dies möglich. Hierzu sind ein paar Schritte notwendig.

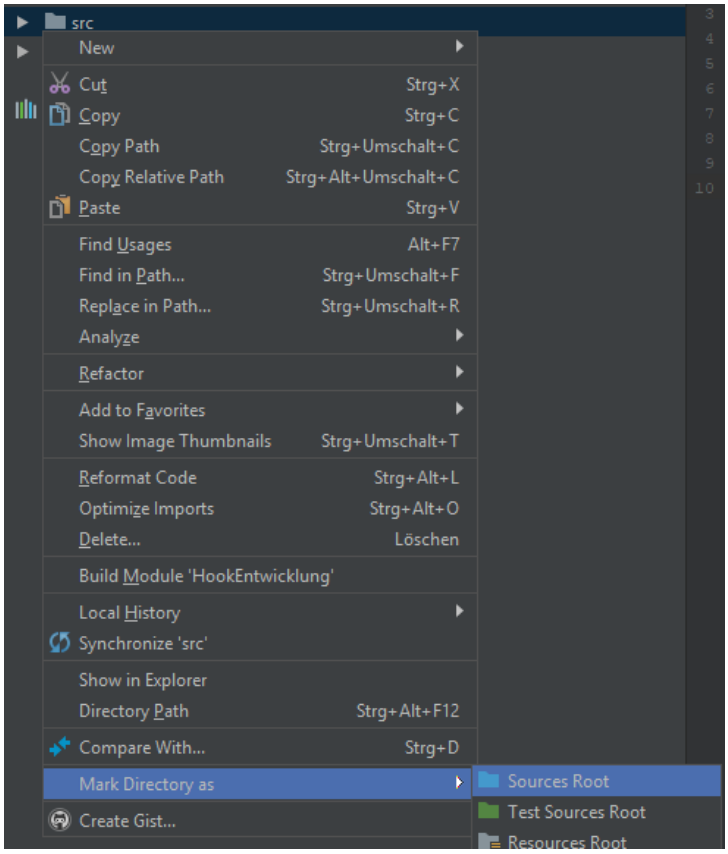
Einrichtung von IntelliJ als Entwicklungsumgebung für die Hook-Entwicklung

1. Laden und installieren Sie ein Java Development Kit (JDK) für Java.

   > **Wichtig**
   > Oracle hat seine Lizenzbedingungen bezüglich seiner Java Distribution überarbeitet, ab Februar diesen Jahres (2019) nicht mehr kostenlos nutzbar. Dies gilt für JDK/JRE-Updates die nach Januar diesen Jahres bei Oracle bezogen wurden.
   > Aus diesem Grund werden alle betroffenen Produkte künftig mit dem OpenJDK (https://openjdk.java.net/) ausgeliefert und bis dahin in Hotfixen maximal Oracle Java 8 Update 201 verwendet. Bezüglich dieser Lizenz ist jedoch ab jetzt zwingend darauf zu achten, dass in der von uns ausgelieferten Distribution (jeweils der **jre**-Unterordner) keine Anpassungen unsererseits oder durch Kunden vorgenommen werden. Unsere Kunden können bestehende Installationen unserer Software weiterhin betreiben und werden beim nächsten Update (bis auf einen Hinweis in den Readme's) voraussichtlich nichts von diesem Wechsel mitbekommen.

2. Laden Sie IntelliJ IDEA© herunter (die kostenfreie Community Edition reicht hier aus) und installieren Sie diese: https://www.jetbrains.com/idea/download/#section=windows
3. Laden Sie danach die Groovy-Bibliothek herunter und installieren diese: http://groovy-lang.org/download.html
4. Starten Sie danach IntelliJ.
5. Wählen Sie ein geeignetes Verzeichnis für ihr Projekt.

6. Markieren des Verzeichnis als **Sources Root**.

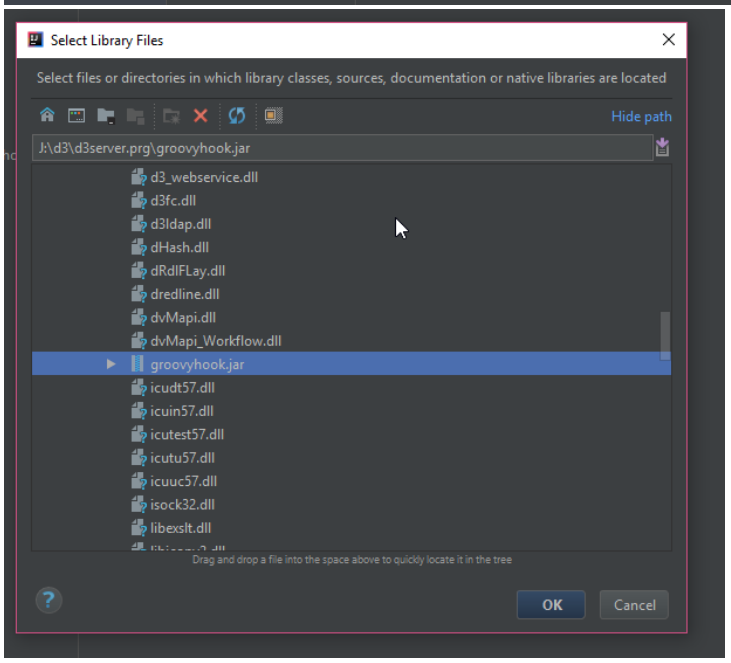


Bekanntgeben der Groovy-API von d.3

Danach müssen Sie noch die Groovy-API Ihrem Projekt bekannt geben.

1. Öffnen Sie hierzu **File** > **Project Structure**.
2. Im neuen Fenster müssen Sie dann unter Libraries die **groovyhook.jar** aus dem Programmverzeichnis von d.3 server hinzufügen.
   Klicken Sie hierzu auf das **+**-Zeichen und wählen **Java**.

3. Wählen Sie schließlich die **groovyhook.jar** aus dem Programmverzeichnis von d.3 server aus.



4. Es erscheint nun eine Meldung, dass die Bibliothek zu Ihrem Projekt hinzugefügt wird. Bestätigen Sie diese Meldung mit **OK**.
5. Die Bibliothek wird nun angezeigt. Damit die Einstellung gespeichert wird, wählen Sie **Apply** oder **OK**.

Nun steht Ihnen die Groovy-API von d.3 zur Verfügung und Sie können Groovy-Hooks mit Hilfe der IntelliJ-Entwicklungsumgebung entwickeln.

Weitere Informationen zu IntelliJ IDEA

Eine ausführliche Dokumentation, sowie Tutorials finden sich beim Hersteller JetBrains auf der Homepage: https://www.jetbrains.com/idea/documentation/.

4 Groovy Hook-Typen

Was ist ein Hook?

Mittels sogenannter Hooks kann in unserem System an vielen definierten Schnittstellen, sogenannten Einsprung- bzw. Eintrittspunkten, eigene Funktionen mittels Groovy-Skript hinterlegt werden. Damit kann auf User-Interaktionen mittels Skripten reagiert und individuelle Anpassungen im System hinterlegt werden.

Unterschiedliche Typen der Hook-Schnittstellen

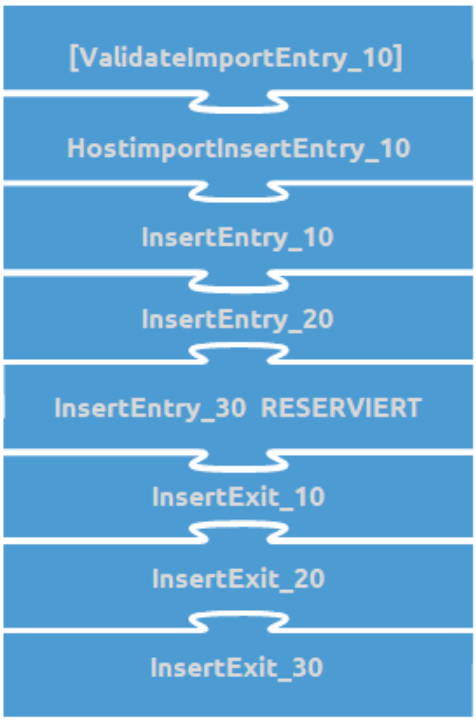
Es stehen unterschiedliche Integrations-Typen zur Auswahl:

- Eintrittspunkte sind Event-gesteuerte Skript-Schnittstellen
- Validierungshooks zur erweiterten Validierung von Werten
- Bereitstellung von dynamischen Wertemengen
- Erweiterte Dokumentklassen
- [Aktenplan] - aktuell leider nicht in Groovy unterstützt.

4.1 d.3-Eintrittspunkte

Allgemein

Die Eintrittspunkte werden durch eine User-Aktion getriggert und werden dann, wie die Perlen auf einer Kette, nach einander abgearbeitet. Wird dabei eine Hook-Funktion innerhalb dieser Kette mit einem Wert ungleich 0 verlassen, wird damit die Kette unterbrochen. Wird zum Beispiel während des manuellen Imports einer Rechnung festgestellt, dass die Kunden-Nr. einen falschen Wert enthält, kann damit die Ablage der Rechnung im System verhindert werden.



Alle verfügbaren Eintrittspunkte für Hook-Funktionen im d.3 werden in den folgenden, untergeordneten Seiten im Einzelnen beschrieben.

Für die beschriebenen Parameter einer Hook-Funktion werden die im Kontext des Aufrufs verfügbaren d.3 Archiv-Objekte an die Groovy-Hook-Funktionen übergeben.

Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer die d.3 Schnittstelle übergeben.

```
import com.dvelop.d3.server.Entrypoint
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;

class MyHooks{
   @Entrypoint(entrypoint="hook_insert_entry_10")
   public int checkIncommingDocuments(D3Interface d3, User user, DocumentType docType, Document doc){
       println "groovy hook insert_entry_10";
       doc.field[1] = "<New value from hook function>";
       return 0;
   }// end of checkIncommingDocuments
} // end of MyHooks
```

Hinweis

Es können auch mehrere Methoden pro Eintrittspunkt registriert werden.

Ist dabei die Reihenfolge des Aufrufs relevant, kann diese mit dem numerischen Attribut `order` in der Annotation festgelegt werden.
Der Defaultwert für Attribut `order` ist dabei "1".

Vor dem Aufruf aller Methoden einer Klasse für denselben Eintrittspunkt werden diese nach Attribut `order` aufsteigend sortiert.
Soll also eine Methode nach einer anderen für den gleichen Eintrittspunkt aufgerufen werden, muss deren `order` Wert größer sein.

```
// ...
   @Entrypoint(entrypoint="hook_insert_entry_10", order = 2)
   def checkIncommingDocuments_2(D3Interface d3, User user, DocumentType docType, Document doc){
       println "Zweite Methode für Eintrittspunkt hook_insert_entry_10";
       doc.field[2] = "<New value from second hook function>";
       return 0;
   }// end of checkIncommingDocuments
}/ end of MyHooks
```

Hinweis

Ein zweiter Annotations-Typ ist @Condition. Mit diesem können Bedingungen für den Aufruf der damit annotierten Methode definiert werden.
Per Eigenschaft doctype können IDs von d.3-Dokumentarten angegeben werden.
Besitzt dieser Eintrittspunkt einen Parameter vom Typ Document oder ein DocumentType, werden die enthaltenen Dokumentart-IDs mit der Bedingung verglichen.
Bei mindestens einer Übereinstimmung wird die Methode aufgerufen.

```
// ...
   @Entrypoint(entrypoint="hook_insert_entry_10", order=2)
   @Condition(doctype= [ "DA1" ] )
   def insertEntry10_2(D3Interface d3, User user, DocumentType docType, Document doc)
   {
// ...
```

Beispiel - Angabe mehrerer Dokumentart-IDs

```
// ...
   @Condition(doctype=["DA1", "DA2", "DA3"])
   def insertEntry10_2(D3Interface d3, User user, DocumentType docType, Document doc)
   {
// ...
```

Beispiel - mehrere IDs per konstanter Variablen

```
// ...
 static final String PHOTO              = "DFOTO";  // Document type for photos
 static final String CURRICULUM_VITAE   = "DLELA";  // Document type for curriculum vitae
 static final String PERSONAL_MASTAER_DATA = "DPSB";   // Document type for personnel master data
 static final String CERTIFICATE        = "DZEUG";  // Document type for certificates
 @Condition(doctype=[PHOTO, CURRICULUM_VITAE, PERSONAL_MASTAER_DATA, CERTIFICATE])
// ...
```

4.1.1 Abhängige Dateien

4.1.1.1 hook_dep_doc_entry_10

```
int hook_dep_doc_entry_10(D3Interface d3, Document doc, String status, Integer fileId, UserOrUserGroup editor, String depExt, Integer transfer)
```

Aufrufzeitpunkt:

Vor dem Eintragen einer abhängigen Datei in die Datenbank

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das Dokument, zu dem die abhängige Datei gehört
status	aktueller Status des Dokuments ("Be", "Pr", "Fr", "Ar")
fileId	Version des Dokuments
editor	Bearbeiter oder Prüfer-Gruppe des Dokuments bei Status Bearbeitung bzw. Prüfung
depExt	Dateierweiterung der abhängigen Datei
transfer	1: Aufruf während eines Statustransfers

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_dep_doc_entry_10" )
  // (4)
  public int doSomething (D3Interface d3, Document doc, String status, Integer fileId, UserOrUserGroup editor,
String depExt, Integer Transfer ){
    // (5)
    d3.log.error("Hello world!");
    // (6)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.1.2 hook_dep_doc_exit_10

```
int hook_dep_doc_exit_10(D3Interface d3, Document doc, String status, Integer fileId, UserOrUserGroup editor,
String depExt, Integer transfer)
```

Aufrufzeitpunkt:

Nach Eintrag der abhängigen Datei in die Datenbank.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das Dokument, zu dem die abhängige Datei abgelegt wurde
status	aktueller Status des Dokuments
fileID	Version des Dokuments
editor	Bearbeiter oder Prüfer-Gruppe des Dokuments bei Status Bearbeitung bzw. Prüfung
depExt	Dateierweiterung der abhängigen Datei
transfer	1: Aufruf während eines Statustransfers

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_dep_doc_exit_10" )
  // (4)
  public int doSomething(D3Interface d3, Document doc, String status, Integer fileId, UserOrUserGroup editor,
String depExt, Integer Transfer){
    // (5)
    d3.log.error("Hello world!");
    // (6)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.2 Aktualisieren der Eigenschaftswerte (UpdateAttributes)

4.1.2.1 hook_upd_attrib_entry_20

```
int hook_upd_attrib_entry_20(D3Interface d3, Document doc, User user, DocumentType docType,
DocumentType docTypeNew)
```

Verfügbare Felder:

Alle beim API-Call übergebenen Felder.

Änderbar sind jedoch nur die **dok_dat_**-Felder und das Feld **text**.

Aufrufzeitpunkt:

Es wurden lediglich die neuen Attribute empfangen, jedoch noch nicht auf Plausibilität geprüft.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	Dokument-Objekt, mit den zu aktualisierenden Eigenschaftswerten. Diese können in dieser Hook-Funktion geändert werden.
user	der ausführende Benutzer
docType	Dokumentart vor der Eigenschaftsaktualisierung
docTypeNew	Dokumentart nach der Eigenschaftsaktualisierung

> **Hinweis**
>
> Die Werte docType und docTypeNew unterscheiden sich nur, wenn tatsächlich ein Dokumentartwechsel durchgeführt wurde.

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.Condition;
import com.dvelop.d3.server.DocumentType;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_upd_attrib_entry_20" )
  // (4)
  @Condition( doctype = [ "XXXX" ] )
  // (5)
  public int doSomething( D3Interface d3, Document doc, User user, DocumentType docType, DocumentType
docTypeNew ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.2.2 hook_upd_attrib_exit_10

> **Hinweis**
>
> Diese Hook-Funktion wird nur ausgeführt, wenn zuvor kein Fehler aufgetreten ist. Liefert diese Funktion einen Wert ungleich 0, wird die Aktualisierung abgebrochen und die Änderungen rückgängig gemacht.

```
int hook_upd_attrib_exit_10(D3Interface d3, Document doc, Integer errorCode, User user, DocumentType docType, DocumentType docTypeOld)
```

Aufrufzeitpunkt:

Direkt vor Beendigung der DB-Transaktion.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	Dokument-Objekt mit den zu aktualisierenden Eigenschaftswerten Diese können in dieser Hook-Funktion noch geändert werden.
errorCode	0: Aktualisierung erfolgreich <> 0: Fehlernummer; i. a. vom DB-Server geliefert
user	der ausführende Benutzer
docType	Dokumentart nach der Eigenschaftsaktualisierung
docTypeOld	Dokumentart vor der Eigenschaftsaktualisierung

> **Hinweis**
>
> Die Werte docType und docTypeOld unterscheiden sich nur, wenn tatsächlich ein Dokumentartwechsel durchgeführt wurde.

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.Condition;
import com.dvelop.d3.server.DocumentType;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_upd_attrib_exit_10" )
  // (4)
  @Condition( doctype = [ "XXXX" ] )
  // (5)
  public int doSomething( D3Interface d3, Document doc, Integer errorCode, User user, DocumentType docType,
DocumentType docTypeOld ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ `public`, wobei `public` im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.2.3 hook_upd_attrib_exit_20

> **Hinweis**
>
> Diese Hook-Funktion wird immer ausgeführt, auch wenn zuvor ein Fehler aufgetreten ist. Es wird daher empfohlen, den Parameter **errorCode** auszuwerten.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	Dokument-Objekt mit den aktualisierten Eigenschaftswerten
errorCode	0: Aktualisierung war erfolgreich <> 0: Fehlernummer; i. a. vom DB-Server geliefert
user	der ausführende Benutzer
docType	Dokumentart nach der Eigenschaftsaktualisierung
docTypeOld	Dokumentart vor der Eigenschaftsaktualisierung

```
int hook_upd_attrib_exit_20(D3Interface d3, Document doc, Integer errorCode, User user, DocumentType docType, DocumentType docTypeOld)
```

Aufrufzeitpunkt:

Direkt nach Beendigung der DB-Transaktion.

> **Hinweis**
>
> Die Werte docType und docTypeOld unterscheiden sich nur, wenn tatsächlich ein Dokumentartwechsel durchgeführt wurde.

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.Condition;
import com.dvelop.d3.server.DocumentType;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_upd_attrib_exit_20" )
  // (4)
  @Condition( doctype = [ "XXXX" ] )
  // (5)
  public int doSomething( D3Interface d3, Document doc, Integer errorCode, User user, DocumentType docType,
DocumentType docTypeOld ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.3 Dokumentanlage (ImportDocument)

4.1.3.1 hook_hostimp_entry_10

> **Wichtig**
>
> Dieser Einsprungpunkt ist aktuell nicht mit der Groovy-Hookschnittstelle kompatibel und kann deshalb noch nicht genutzt werden.
> Bis zur Herstellung der Kompatibilität muss auf die JPL-Variante des Einsprungpunkts zurückgegriffen werden.

```
int hook_hostimp_entry_10(D3Interface d3,  String importDir, String fileName, Document doc, DocumentType docType, String newImport)
```

Aufrufzeitpunkt:

Wird nur beim Hostimport aufgerufen. Direkt nach dem Einlesen der `default.ini` und der `JPL`-Attributdatei.

Die Unicode-Konvertierung wurde an dieser Stelle noch nicht durchgeführt. Auch die Werte der übersetzbaren Wertemengen wurden noch nicht konvertiert.

Hier ist eine Änderung der übergebenen Eigenschaftswerte möglich.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
importDir	Verzeichnis, aus dem die Datei importiert wird
fileName	Dateiname der zu importierenden Datei
doc	das zu importierende Dokument-Objekt
docType	Dokumentart des zu importierenden Dokuments
newImport	"1": import eines neuen Dokuments "0": Import einer neuen Dateiversion zu einem existierenden Dokument

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.Condition;
import com.dvelop.d3.server.DocumentType;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_hostimp_entry_10" )
  // (4)
  @Condition( doctype = [ "XXXX" ] )
  // (5)
  public int doSomething(D3Interface d3,  String importDir, String fileName, Document doc, DocumentType
docType, String newImport){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.3.2 hook_insert_entry_10

```
hook_insert_entry_10 (D3Interface d3, User user, DocumentType docType, Document doc)
```

Aufrufzeitpunkt:

- Vor dem Import. Es wurde lediglich getestet, ob die Verbindung zur DB in Ordnung ist. Hier ist eine Änderung der übergebenen Eigenschaftswerte möglich.
- Bei der Datenvalidierung für einen anschließenden Dokumentenimport (`API ValidateAttributes` mit Parameter `"function" = "Insert"`)

Die Dokumenteigenschaften sind über das Dokumentobjekt zugreifbar. Die Werte der erweiterten Eigenschaften können im Hook geändert werden.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
user	der ausführende Benutzer
docType	Dokumentart des neu zu importierenden Dokuments
doc	das neue Dokument-Objekt

> **Hinweis**
>
> Szenario:
>
> Wird ein Dokument im d.3-System abgelegt, soll im d.3 Log-File einfach nur eine Fehlermeldung "Hallo Welt" angezeigt werden.

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.Condition;
import com.dvelop.d3.server.DocumentType;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_insert_entry_10" )
  // (4)
  @Condition( doctype = [ "XXXX" ] )
  // (5)
  public int doSomething( D3Interface d3, User user, DocumentType docType, Document doc ) {
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.3.3 hook_insert_entry_20

```
hook_insert_entry_20 (D3Interface d3, Document doc, DocumentType docType, User user)
```

Aufrufzeitpunkt:

Vor dem Import. Die Nutzdatei wurde bereits in das Zielverzeichnis übertragen.

Die SQL-Kommandos für die Speicherung der Dokument-Metadaten wurden generiert.

Die übergebenen Dokumenteigenschaften können nicht mehr geändert werden.

Die Eigenschaftswerte sind noch nicht auf Gültigkeit (Wertebereich, reg. Expression, Min.-Max.-Bereich, ...) geprüft worden.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das zu importierende Dokument
docType	Dokumentart des zu importierenden Dokuments
user	der ausführende Benutzer

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.Condition;
import com.dvelop.d3.server.DocumentType;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_insert_entry_20" )
  // (4)
  @Condition( doctype = [ "XXXX" ] )
  // (5)
  public int doSomething( D3Interface d3, Document doc, DocumentType docType, User user ) {
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.

3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.3.4 hook_insert_exit_10

```
int hook_insert_exit_10 (D3Interface d3, Document doc, String fileDestination, Integer importOk, User user, DocumentType docType)
```

Aufrufzeitpunkt:

Nach dem Import. Die Datenbank-Transaktion wurde noch nicht geschlossen. Somit kann man hier noch eine letzte Zurücknahme erzwingen und damit den Import rückgängig machen.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das neue Dokument
fileDestination	Pfad und Name der Zieldatei (Angabe, wohin die Zieldatei geschrieben wurde)
importOk	1: bisher kein Fehler ausgetreten 0: Es trat ein Fehler beim Importieren des Dokuments auf
user	der ausführende Benutzer
docType	Dokumentart des neuen Dokuments

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.Condition;
import com.dvelop.d3.server.DocumentType;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_insert_exit_10" )
  // (4)
  @Condition( doctype = [ "XXXX" ] )
  // (5)
  public int doSomething(D3Interface d3, Document doc, String fileDestination, Integer importOk, User user,
DocumentType docType ) {
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.3.5 hook_insert_exit_20

```
int hook_insert_exit_20(D3Interface d3, Document doc, String fileDestination, Integer importOk, User user, DocumentType docType)
```

Aufrufzeitpunkt:

Nach dem Import. Die Datenbank-Transaktion wurde geschlossen (`COMMIT` oder `ROLLBACK`). Somit kann ein erfolgreicher Import nicht mehr rückgängig gemacht werden.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das neue Dokument
fileDestination	Pfad und Name der Zieldatei (Angabe, wohin die Zieldatei geschrieben wurde)
importOk	1: bisher kein Fehler ausgetreten 0: Es trat ein Fehler beim Importieren des Dokuments auf
user	der ausführende Benutzer
docType	Dokumentart des neuen Dokuments

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.Condition;
import com.dvelop.d3.server.DocumentType;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_insert_exit_20" )
  // (4)
  @Condition( doctype = [ "XXXX" ] )
  // (5)
  public int doSomething( D3Interface d3, Document doc, String fileDestination, Integer importOk, User user,
DocumentType docType ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.3.6 hook_insert_exit_30

```
int hook_insert_exit_30 (D3Interface d3, Document doc, String fileDestination, Integer importOk, User user, DocumentType docType)
```

Aufrufzeitpunkt:

Nach dem Import. Die Datenbank-Transaktion wurde bereits geschlossen.

> **Hinweis**
>
> Die Funktion kann genauso verwendet werden wie hook_insert_exit_20.

4.1.4 Dokumente freigeben (ReleaseDocument)

4.1.4.1 hook_release_entry_10

> **Hinweis**
>
> Falls diese Hook-Funktion einen Wert ungleich 0 liefert, wird die Freigabe abgebrochen.

```
int hook_release_entry_10(D3Interface d3, Document doc, User user, DocumentType docType, String unblock)
```

Aufrufzeitpunkt:

Direkt vor Start der Dokumentfreigabe. Es wurde ermittelt, dass der Benutzer das Recht hat, das Dokument freizugeben.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das freizugebende Dokument
user	der ausführende Benutzer
docType	Dokument des freizugebenden Dokuments
unblock	gleich "1", wenn das Dokument entsperrt wird gleich "" sonst

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.Condition;
import com.dvelop.d3.server.DocumentType;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_release_entry_10" )
  // (4)
  @Condition( doctype = [ "XXXX" ] )
  // (5)
  public int doSomething( D3Interface d3, Document doc, User user, DocumentType docType, String unblock ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.4.2 hook_release_exit_10

```
int hook_release_exit_10(D3Interface d3, Document doc, User user, Integer errorCode, DocumentType docType, String unblock)
```

Aufrufzeitpunkt:

Nach Durchführung der Freigabe, nach Beendigung der Datenbank-Transaktion.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das freigegebene Dokument
user	der ausführende Benutzer
errorCode	0: Freigabe war erfolgreich sonst: Fehlercode
docType	Dokumentart des freigegebenen Dokuments
unblock	gleich "1", wenn das Dokument entsperrt wird ungleich "1" bei normalen Freigaben

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.Condition;
import com.dvelop.d3.server.DocumentType;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_release_exit_10" )
  // (4)
  @Condition( doctype = [ "XXXX" ] )
  // (5)
  public int doSomething( D3Interface d3, Document doc, User user, Integer errorCode, DocumentType docType,
String unblock ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.5 Dokument prüfen (VerifyDocument)

4.1.5.1 hook_verify_entry_10

> **Hinweis**
>
> Falls diese Hook-Funktion einen Wert ungleich 0 liefert, wird die Prüfung abgebrochen.

```
int hook_verify_entry_10(D3Interface d3, Document doc, Integer versionId, User user)
```

Aufrufzeitpunkt:

Direkt vor Start der Datenbank-Transaktion. Es wurde ermittelt, dass der Benutzer das Recht hat, das Dokument zu prüfen.

Eingabeparameter:

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das zu prüfende Dokument
versionId	Nummer der zu prüfenden Dokumentversion
user	der ausführende Benutzer

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_verify_entry_10" )
  // (4)
  public int doSomething( D3Interface d3, Document doc, Integer versionId, User user ){
    // (5)
    d3.log.error("Hello world!");
    // (6)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.5.2 hook_verify_exit_10

```
int hook_verify_exit_10(D3Interface d3, Document doc, Integer versionId, User user, Integer errorCode)
```

Aufrufzeitpunkt:

Nach Durchführung der Prüfung. Nach Beendigung der Datenbanktransaktion.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das geprüfte Dokument
versionId	Nummer der geprüften Dokumentversion
user	der ausführende Benutzer
errorCode	0: Prüfung erfolgreich sonst: Datenbank-Fehlernummer beim Eintrag der Prüfung in die Datenbank

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_verify_exit_10" )
  // (4)
  public int doSomething( D3Interface d3, Document doc, Integer versionId, User user, Integer errorCode ){
    // (5)
    d3.log.error("Hello world!");
    // (6)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.6 Dokumentsuche (GetDocumentList/SearchDocument)

4.1.6.1 hook_search_entry_05

```
int hook_search_entry_05(D3Interface d3, User user, DocumentType docType, Document searchContext)
```

Aufrufzeitpunkt:

Vor dem Zugriff auf die Volltext-Engine über **d.3 search**. Hier kann über den searchContext auch der Volltext-Suchbegriff noch geändert werden.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
user	der ausführende Benutzer
docType	falls angegeben, die Dokumentart der gesuchten Dokumente
searchContext	die Suchbegriffe über ein Dokument-Objekt zugreifbar

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_search_entry_05" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething( D3Interface d3, User user, DocumentType docType, Document searchContext ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss

nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.

6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.6.2 hook_search_entry_10

```
int hook_search_entry_10(D3Interface d3, User user, DocumentType docType, Document searchContext)
```

Aufrufzeitpunkt:

- Vor der Suche nach Dokumenten: Die übergebenen Suchkriterien sind noch nicht auf Plausibilität geprüft worden. Eine ggf. aktivierte Konvertierung der Suchkriterien nach Klein- bzw. Großschrift wurde noch nicht durchgeführt.
- Bei der Datenvalidierung für eine anschließende Suche (API ValidateAttributes mit Parameter "function" = "Search").

Die Suchbegriffe wurden übernommen. Diese können im Hook über den 'searchContext' geändert werden.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
user	der ausführende Benutzer
docType	falls angegeben, die Dokumentart der gesuchten Dokumente
searchContext	die Suchbegriffe über ein Dokument-Objekt zugreifbar

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_search_entry_10" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething( D3Interface d3, User user, DocumentType docType, Document searchContext ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.6.3 hook_search_entry_20

```
int hook_search_entry_20(D3Interface d3, User user, DocumentType docType)
```

Aufrufzeitpunkt:

Vor der Suche nach Dokumenten: Der SELECT-Befehl für die Suche nach den Dokumenten ist entsprechend den Suchkriterien schon zusammengesetzt worden.

Eingabeparameter:

Parameter	Beschreibung
d3	die d.3-Schnittstelle
`user`	der ausführende Benutzer
`docType`	alles angegeben, die Dokumentart der gesuchten Dokumente

```groovy
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_search_entry_20" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething( D3Interface d3, User user, DocumentType docType ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.

3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.6.4 hook_search_exit_30

```
hook_search_exit_30(D3Interface d3, User user, Integer errorCode, Integer noResults, Integer noRefused, DocumentType docType)
```

Aufrufzeitpunkt:

Ganz am Ende der Suche, direkt bevor die Ergebnisse an den Client geliefert werden

Parameter	Beschreibung
d3	die d.3-Schnittstelle
`user`	Benutzer, der die Suche ausführt
`errorCode`	0 bei Erfolg ansonsten ein Fehlercode
`noResults`	Anzahl der Treffer
`noRefused`	Anzahl verweigerter Treffer
`docType`	falls angegeben, die Dokumentart der gesuchten Dokumente

Rückgabewert:

wird ignoriert

> **Hinweis**
>
> Über die Hook-Eigenschaft **no_results_refused** kann die Rückgabe der Anzahl der verweigerten Treffer an den Aufrufer deaktiviert werden.
> Diese kann in Kontexten genutzt werden, sofern Benutzer gar nicht sehen dürfen, dass es überhaupt Treffer gibt.
> Aufruf: **d3.hook.setProperty("no_results_refused", "0")**

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_search_exit_30" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething( D3Interface d3, User user, Integer errorCode, Integer noResults, Integer noRefused,
DocumentType docType ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem

Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.

6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.7 Einspielen einer neuen Version (ImportNewVersionDocument)

4.1.7.1 hook_new_version_entry_10

> **Hinweis**
>
> Wird beim **ValidateAttributes nextcall=ImportNewVersionDocument** übergeben, wird die Funktion auch aufgerufen.
>
> Durch die Neustrukturierung der Dokumentablage mit d.3-Version 8 haben die Parameter **fileSource** und **fileDestination** keinen sinnvollen Inhalt mehr.
> Die Parameter sind weiterhin vorhanden, damit sich die Hook-Schnittstelle nicht ändert und müssen somit weit entgegengenommen werden. Als Wert wird aber nur Leerstring übergeben.

```
int hook_new_version_entry_10(D3Interface d3, Document doc, String fileSource, String fileDestination, User user, DocumentType docType)
```

Aufrufzeitpunkt:

Es wurde geprüft, ob das Dokument bereits in **d.3** existiert. Die Dokumenteigenschaften können hier noch geändert werden.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	Dokument, zu dem eine neue Dateiversion eingespielt werden soll
fileSource	abgekündigt mit Version 8.0
fileDestination	abgekündigt mit Version 8.0
user	der ausführende Benutzer
docType	Dokumentart der zu importierenden Dateiversion

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_new_version_entry_10" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething( D3Interface d3, Document doc, String fileSource, String fileDestination, User user,
DocumentType docType ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.7.2 hook_new_version_entry_20

> **Hinweis**
>
> Durch die Neustrukturierung der Dokumentablage mit d.3-Version 8 haben die Parameter **fileSource** und **fileDestination** keinen sinnvollen Inhalt mehr.
> Die Parameter sind weiterhin vorhanden, damit sich die Hook-Schnittstelle nicht ändert und müssen somit weit entgegengenommen werden. Als Wert wird aber nur Leerstring übergeben.

```
int hook_new_version_entry_20(D3Interface d3, Document doc, String fileSource, String fileDestination, User user, DocumentType docType)
```

Aufrufzeitpunkt:

Es wurde erfolgreich geprüft, ob die neue Dateiversion existiert. Diese wurde noch nicht eingespielt.

Die Datenbanktransaktion wurde noch nicht gestartet. Die Dokumenteigenschaften sind validiert worden und können nicht mehr geändert werden.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	Dokument, zu dem eine neue Dateiversion eingespielt werden soll
fileSource	abgekündigt mit Version 8.0
fileDestination	abgekündigt mit Version 8.0
user	der ausführende Benutzer
docType	Dokumentart der zu importierenden Dateiversion

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_new_version_entry_20" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething( D3Interface d3, Document doc, String fileSource, String fileDestination, User user,
DocumentType docType ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.7.3 hook_new_version_entry_30

> **Hinweis**
>
> Durch die Neustrukturierung der Dokumentablage mit d.3-Version 8 haben die Parameter **fileSource** und **fileDestination** keinen sinnvollen Inhalt mehr.
> Die Parameter sind weiterhin vorhanden, damit sich die Hook-Schnittstelle nicht ändert und müssen somit weit entgegengenommen werden. Als Wert wird aber nur Leerstring übergeben.

```
int hook_new_version_entry_30(D3Interface d3, Document doc, String fileSource, String fileDestination, User user, DocumentType docType)
```

Aufrufzeitpunkt:

Wird sofort nach hook_new_version_entry_20 ausgeführt.

Alle Angaben analog zu hook_new_version_entry_20.

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_new_version_entry_30" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething( D3Interface d3, Document doc, String fileSource, String fileDestination, User user, DocumentType docType ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.

3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.7.4 hook_new_version_exit_10

```
int hook_new_version_exit_10(D3Interface d3, Document doc, Integer errorCode, User user, DocumentType docType)
```

Aufrufzeitpunkt:

Die Datenbanktransaktion wurde gestartet. Alle Eigenschaften, auch die Mehrfacheigenschaften (60er-Felder) wurden aktualisiert.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	Dokument, zu dem eine neue Dateiversion importiert wurde
errorCode	1: Beim Aktualisieren der Eigenschaften ist ein Fehler aufgetreten 0: Aktualisieren der Kenndaten erfolgreich
user	der ausführende Benutzer
docType	Dokumentart der importierten Dateiversion

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_new_version_exit_10" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething(D3Interface d3, Document doc, Integer errorCode, User user, DocumentType docType)
{
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.7.5 hook_new_version_exit_20

> **Hinweis**
>
> Durch die Neustrukturierung der Dokumentablage mit d.3-Version 8 hat der Parameter **fileDestination** keinen sinnvollen Inhalte mehr und ist deshalb abgekündigt.

```
int hook_new_version_exit_20(D3Interface d3, Document doc, String fileDestination, Integer importOk, User user, DocumentType docType)
```

Aufrufzeitpunkt:

Auch die Mehrfacheigenschaften (60er-Felder) wurden aktualisiert. Die neue Dateiversion wurde, ggf. zusammen mit zugehörigen, abhängigen Dateien, importiert.

Die Datenbanktransaktion wurde beendet (COMMIT oder ROLLBACK).

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	Dokument, zu dem eine neue Dateiversion importiert wurde
fileDestination	abgekündigt mit Version 8.0
ImportOk	1: Einspielung der neuen Version erfolgreich 0: Einspielung der neuen Version mit Fehler abgebrochen
user	der ausführende Benutzer
docType	Dokumentart der importierten Dateiversion

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_new_version_exit_20" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething( D3Interface d3, Document doc, String fileDestination, Integer importOk, User user,
DocumentType docType ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.7.6 hook_new_version_exit_30

```
int hook_new_version_exit_30(D3Interface d3, Document doc, Integer importOk, Integer errorCode, User user, DocumentType docType)
```

Aufrufzeitpunkt:

wie bei **hook_new_version_exit_20**

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	Dokument, zu dem eine neue Dateiversion importiert wurde
ImportOk	1: Einspielung der neuen Version erfolgreich 0: Einspielung der neuen Version mit Fehler abgebrochen
errorCode	Im Fehlerfall (importOk=0): Fehlercode des zuvor aufgetretenen Fehlers
user	der ausführende Benutzer
docType	Dokumentart der importierten Dateiversion

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_new_version_exit_30" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething( D3Interface d3, Document doc, Integer importOk, Integer errorCode, User user,
DocumentType docType ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.8 Erzeugen/ Bearbeiten von TIFF- oder PDF-Dokumenten

4.1.8.1 hook_rendition_entry_10

```
int hook_rendition_entry_10(D3Interface d3, Document doc, User user)
```

Aufrufzeitpunkt:

Vor dem Start der Abbildungserstellung, wenn diese über einen **d.3-**Benutzer aufgerufen wurde (über **d.3-**API oder Server-API). Wird nicht aufgerufen bei automatischem Aufruf über hinterlegte Regeln.

> **Hinweis**
>
> Der Aufruf kann durch Returnwert ungleich 0 abgebrochen werden.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das Dokument, von dem ein TIFF- oder PDF-Abbild erstellt werden soll
user	der **d.3**-Benutzer, der die Erstellung angefordert hat

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_rendition_entry_10" )
  // (4)
  public int doSomething( D3Interface d3, Document doc, User user ){
    // (5)
    d3.log.error("Hello world!");
    // (6)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.8.2 hook_rendition_entry_20

```
int hook_rendition_entry_20(D3Interface d3, Document doc, DocumentType docType, String sourcePath, String sourceFilename, String destFilename)
```

Aufrufzeitpunkt:

Direkt vor dem Senden des Erstellungsjobs an den **d.ecs rendition service**.

> **Hinweis**
>
> In dieser Hook-Funktion können über die Hook-Eigenschaftsfelder **rendition_parameter_name** und **rendition_parameter_value** Render-Optionen an den Rendition Service übergeben werden.
>
> Beispiel:
>
> ```
> d3.hook.setProperty("rendition_parameter_name", 1, "PRINT_FORMAT")
> d3.hook.setProperty("rendition_parameter_value", 1, "A2")
> ```

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das Dokument von dem ein TIFF- oder PDF-Abbild erstellt werden soll
docType	Dokumentart dieses Dokuments

Parameter	Beschreibung
sourcePath	Quellpfad der Stammdatei
sourceFilename	Dateiname der Stammdatei
destFilename	Dateiname der fertigen Abbilddatei

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_rendition_entry_20" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething( D3Interface d3, User user, DocumentType docType, Document searchContext ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.

6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.8.3 hook_rendition_exit_30

```
int hook_rendition_exit_30(D3Interface d3, Document doc, String destStatus, String tiffFilename, Integer errorCode, String fileType)
```

Aufrufzeitpunkt:

Nach dem Abholen der fertigen TIFF-/PDF-Datei vom Rendition Server.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das Dokument, von dem ein TIFF- oder PDF-Abbild erstellt wurde
destStatus	Zielstatus des Dokumentes ("B", "P", "F", "A")
tiffFilename	Zielpfad + Dateiname der Abbild-Datei
errorCode	0 = ok -1 = Fehler beim Abholen der Datei vom Rendition Service -> siehe **d.3**-Logdatei
fileType	Dateityp, der gerendert wurde ("P1", "T1", "TXT")

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_rendition_exit_30" )
  // (4)
  public int doSomething( D3Interface d3, Document doc, String destStatus, String tiffFilename, Integer errorCode, String fileType ){
    // (5)
    d3.log.error("Hello world!");
    // (6)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.9 Login

4.1.9.1 hook_val_passwd_entry_10

```
int hook_val_passwd_entry_10(D3Interface d3, User user, String appLanguage, String appVersion)
```

Aufrufzeitpunkt:

Hook-Funktion vor der Prüfung von Benutzername und Passwort durch API-Funktion **ValidatePasswordForUser**.

Ein langer Benutzername ist bereits gegen den internen Namen getauscht worden.

Anmeldedaten können nicht verändert werden.

Parameter	Beschreibung
`d3`	die d.3-Schnittstelle
`user`	anzumeldender Benutzer (**d.3-**Kurz- oder Langname; LDAP-Benutzername)
`appLanguage`	SprachID, die von der Anwendung übergeben wurde, z.B. "049"=deutsch, "001"=englisch
`appVersion`	Versionsstring, der von der Anwendung übergeben wurde Zeichen 1..3 Modulkennung z.B. 200 für d.xplorer Zeichen 4..6: Version des Moduls z.B. 800 für Version 8.0.0 Zeichen 7..8 Loglevel, z.B. 9 für DEBUG

Rückgabe:

Ein Wert != 0 führt zur Änderung des Rückgabewertes von API-Funktion ValidatePasswordForUser und somit zum Abbruch des Login.

Der Rückgabewert des Hook wird von 9500 abgezogen, d.h. -1 => 9500-(-1) = 9501.

Diese Zahl wird an den Client zurückgegeben und ist somit in der msglib.usr zu hinterlegen.

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_val_passwd_entry_10" )
  // (4)
  public int doSomething( D3Interface d3, User user, String appLanguage, String appVersion ){
    // (5)
    d3.log.error("Hello world!");
    // (6)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.9.2 hook_val_passwd_exit_10

```
int hook_val_passwd_exit_10(D3Interface d3, int errorCode, User user, String appLanguage, String appVersion)
```

Aufrufzeitpunkt:

Test von Benutzername und Passwort gegen d.3-Benutzerstamm oder auch ggf. gegen einen Directory Server (per LDAP/Kerberos) sind gelaufen.

Das Ergebnis steht fest und wird als Parameter "error" übergeben.

Eingabeparameter:

Parameter	Beschreibung
`d3`	die d.3-Schnittstelle
`errorCode`	Fehlercode der Benutzername/Passwort Prüfung; z.B. 0002 = Benutzername/Passwort ungültig; 0 = Erfolg
`user`	anzumeldender **d.3**-Benutzer
`appLanguage`	Sprach-ID, die von der Anwendung übergeben wurde, z.B. "049"=deutsch, "001"=englisch
`appVersion`	Versionsstring, der von der Anwendung übergeben wurde

Rückgabe:

Ein Rückgabewert `!= 0` führt zum Abbruch (siehe hook_val_passwd_entry_10).

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_val_passwd_exit_10" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething( D3Interface d3, int errorCode, User user, String appLanguage, String appVersion ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.10 Löschen eines Dokuments (DeleteDocument)

4.1.10.1 hook_delete_entry_10

```
int hook_delete_entry_10(D3Interface d3, Document doc, User user, DocumentType docType)
```

Aufrufzeitpunkt:

- Vor dem Löschen des Dokuments. Es wurde erfolgreich geprüft, ob der Benutzer das Dokument löschen darf.
- Der Löschvorgang kann hier noch abgebrochen werden durch Rückgabe eines Returncodes ungleich 0.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das zu löschenden Dokument
user	der Benutzer, der das Dokument löschen möchte
docType	Dokumentart des zu löschenden Dokuments

Privilegiertes Löschen:

Das Löschen von Dokumenten erfolgt per Default immer durch Verschieben in den internen Papierkorb. Mit Hilfe des privilegierten Löschens kann eine Dokumentversion allerdings sofort vollständig und unwiederbringlich aus dem System (Datenbank, Dokumentenbaum und in einigen Fällen auch vom Sekundärspeichersystem) gelöscht werden. Hierfür muss eine separate Lizenz bei der **d.velop** AG erworben und gegebenenfalls auch Beratung beauftragt werden. In diesem Einsprungpunkt kann das privilegierte Löschen dann aktiviert werden durch setzen der Hook-Eigenschaft "DELETE_PRIVILEGED".

Eigenschaft	Beschreibung
DELETE_PRIVILEGED	"0" : Löschen durch Verschieben in den Papierkorb (Standardwert) "1" : Privilegiertes Löschen. Entfernt Dokumente unwiederbringlich aus dem System.

Beispiel:

```
d3.hook.setProperty("DELETE_PRIVILEGED", "1")
```

> **Hinweis**
>
> Weitere Informationen zum Löschen von Dokumenten finden sie im Kapitel Aufbewahrungsfristen und Löschen von Dokumenten

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_delete_entry_10" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething( D3Interface d3, Document doc, User user, DocumentType docType ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.10.2 hook_delete_exit_10

```
int hook_delete_exit_10(D3Interface d3, Document doc, User user, Integer errorCode, DocumentType docType)
```

Aufrufzeitpunkt:

- Nach dem Löschen des Dokuments. Entsprechend kann der Löschvorgang in diesem Einsprungpunkt nicht mehr abgebrochen werden.
- Ob der Löschvorgang erfolgreich war, kann über den Parameter **errorCode** geprüft werden.

Parameter	Beschreibung
`d3`	die d.3-Schnittstelle
`doc`	das zu löschende Dokument
`user`	Benutzer, der das Dokument löscht
`errorCode`	0: Dokument wurde erfolgreich gelöscht sonst: Löschen fehlgeschlagen; Fehlercode
`docType`	Dokumentart des zu löschenden Dokuments

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_delete_exit_10" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething( D3Interface d3, Document doc, User user, Integer errorCode, DocumentType
docType ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.

3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.11 Löschen von Verknüpfungen (Unlink)

4.1.11.1 hook_unlink_entry_30

```
int hook_unlink_entry_30(D3Interface d3, Document docFather, Document docChild)
```

Aufruf:

Direkt vor Ausführung des Datenbankbefehl zum Lösen der Verknüpfung.

Parameter	Beschreibung
`d3`	die d.3-Schnittstelle
`docFather`	das übergeordnete Dokument
`docChild`	das untergeordnete Dokument

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_unlink_entry_30" )
  // (4)
  public int doSomething( D3Interface d3, Document docFather, Document docChild ){
    // (5)
    d3.log.error("Hello world!");
    // (6)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.11.2 hook_unlink_exit_10

```
int hook_unlink_exit_10(D3Interface d3, Document docFather, Document docChild, Integer unlinkErrorCode, Integer errorCode)
```

Aufruf:

Nach der Verknüpfungslösung zweier Dokumente.

Parameter	Beschreibung
`d3`	die d.3-Schnittstelle
`docFather`	das übergeordnete Dokument
`docChild`	das untergeordnete Dokument
`unlinkErrorCode`	0: Verknüpfungslösung war erfolgreich -1: Vater und Kind sind identisch bzw. einer der beiden existiert gar nicht -2: Die beiden Dokumente sind nicht verknüpft -4: Beim Entfernen der Verknüpfung trat ein Datenbankfehler auf (s. dazu `errorCode`)
`errorCode`	0 = Ok sonst Datenbank- oder Hook-Fehler

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_unlink_exit_10" )
  // (4)
  public int doSomething( D3Interface d3, Document docFather, Document docChild, Integer unlinkErrorCode,
Integer errorCode ){
    // (5)
    d3.log.error("Hello world!");
    // (6)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.

4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.12 Postkorb (SendHoldFile)

4.1.12.1 hook_ack_holdfile_exit_10

```
int hook_ack_holdfile_exit_10(D3Interface d3, User user, Document doc, Integer holdfileId)
```

Quittieren eines Postkorbeintrags.

Aufrufzeitpunkt:

Nach dem Quittieren eines Postkorbeintrages durch Aufruf der API-Funktion **AcknowledgeReceivedHoldFile**.

Verhindern lässt sich ein Quittieren nicht mehr, da der Aufruf nach dem Quittieren stattfindet.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
user	Benutzer, der die Quittierung ausgelöst hat
doc	das quittierte Dokument
holdfileId	ID des Postkorbeintrags

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_ack_holdfile_exit_10" )
  // (4)
  public int doSomething( D3Interface d3, User user, Document doc, Integer holdfileId ){
    // (5)
    d3.log.error("Hello world!");
    // (6)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.13 Redlining (WriteRedline)

4.1.13.1 hook_write_redline_entry_10e

```
int hook_write_redline_entry_10(D3Interface d3, Document doc, User user, DocumentType docType)
```

Aufrufzeitpunkt:

Die Funktion wird vor dem Schreiben einer Redlining-Datei ausgeführt.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das Dokument zu dem die Redlining-Datei abgelegt wird
user	der ausführende Benutzer
docType	Dokumentart des Dokuments

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_write_redline_entry_10" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething( D3Interface d3, Document doc, User user, DocumentType docType ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss

nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.

6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.13.2 hook_write_redline_exit_30

```
int hook_write_redline_exit_30(D3Interface d3, Document doc, User user, DocumentType docType)
```

Aufruf:

Nach dem Schreiben einer Redlining-Datei (per **d.3**-API-Call **WriteRedline**).

Parameter	Beschreibung
d3	die d.3-Schnittstelle
`doc`	das Dokument zu dem eine Redlining-Datei abgelegt wurde
`user`	der ausführende Benutzer
`docType`	Dokumentart des Dokuments

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_write_redline_exit_30" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething( D3Interface d3, Document doc, User user, DocumentType docType ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.14 Senden einer Wiedervorlage (SendHoldfile)

4.1.14.1 hook_holdfile_entry_10

```
int hook_holdfile_entry_10(D3Interface d3, Document doc, UserOrUserGroup recipient, UserOrUserGroup
sender, Integer chainId, String notice, String wvType)
```

Aufrufzeitpunkt:

Wird aufgerufen, bevor die Übergabeparameter geprüft werden.

Die Werte der Übergabeparameter sind auch noch in den folgenden Hook-Eigenschaftsfeldern verfügbar:

d3server_empfaenger_wv[1]
d3server_sender_wv[1]
d3server_kette_id

Diese Werte können per **d3.hook.property()** Aufruf verändert werden.

Parameter	Beschreibung
doc	das Dokument, welches in die Wiedervorlage gestellt werden soll
recipient	Benutzer- oder Gruppenobjekt des Empfängers
sender	Benutzer- oder Gruppenobjekt des Senders
chainId	Ketten-ID, die für diesen Postkorbeintrag verwendet werden soll
notice	Betrefftext der Postkorbbenachrichtigung
wvTyp	Typ-ID der Postkorbbenachrichtigung Mögliche Werte: "" = normale Postkorbbenachrichtigung "W" = Workflow-Benachrichtigung ... = sonstige (ggf. selbst definierte Werte)

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_holdfile_entry_10" )
  // (4)
  public int doSomething( D3Interface d3, Document doc, UserOrUserGroup recipient, UserOrUserGroup sender,
Integer chainId, String notice, String wvType ){
    // (5)
    d3.log.error("Hello world!");
    // (6)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.14.2 hook_holdfile_entry_20

```
int hook_holdfile_entry_20(D3Interface d3, Document doc, UserOrUserGroup recipient, UserOrUserGroup
sender, Integer chainId, String notice, String wvType)
```

Aufrufzeitpunkt:

Wird aufgerufen, wenn Datum etc. bereits auf Plausibilität geprüft worden sind. Es sind aber noch nicht die Rechte des Empfängers auf das Dokument geprüft worden.

Die Werte sind hier nicht mehr änderbar.

Eingabeparameter siehe hook_holdfile_entry_10 (doc_id, recipient, sender, chain_id).

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_holdfile_entry_20" )
  // (4)
  public int doSomething( D3Interface d3, Document doc, UserOrUserGroup recipient, UserOrUserGroup sender,
Integer chainId, String notice, String wvType ){
    // (5)
    d3.log.error("Hello world!");
    // (6)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.14.3 hook_holdfile_entry_30

```
int hook_holdfile_entry_30(D3Interface d3, Document doc, UserOrUserGroup recipient, UserOrUserGroup
sender, Integer chainId, String notice, String wvType)
```

Aufrufzeitpunkt:

Wird aufgerufen, direkt vor dem Eintrag in die Datenbank, wenn auch schon die Rechte des Empfängers auf das Dokument geprüft wurden. Die Werte sind hier nicht mehr änderbar.

Eingabeparameter:

siehe hook_holdfile_entry_10 (doc_id, recipient, sender, chain_id).

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_holdfile_entry_30" )
  // (4)
  public int doSomething( D3Interface d3, Document doc, UserOrUserGroup recipient, UserOrUserGroup sender,
Integer chainId, String notice, String wvType ){
    // (5)
    d3.log.error("Hello world!");
    // (6)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.14.4 hook_holdfile_exit_10

```
int hook_holdfile_exit_10(D3Interface d3, Document doc, UserOrUserGroup recipient, UserOrUserGroup sender, Integer chainId, Integer errorCode)
```

Aufrufzeitpunkt:

Direkt nach dem Datenbank-Befehl, der die Wiedervorlage aktiviert.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das Dokument, für das eine Postkorbbenachrichtigung eingestellt wurde
recipient	der Empfänger der Benachrichtigung
sender	der Absender der Benachrichtigung
chainId	Ketten-ID, die für diesen Postkorbeintrag verwendet werden soll
errorCode	0: alles OK sonst: Datenbank-Fehlernummer beim Eintrag der Wiedervorlage in die Datenbank

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_holdfile_exit_10" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething( D3Interface d3, Document doc, UserOrUserGroup recipient, UserOrUserGroup sender,
Integer chainId, Integer errorCode ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.15 Senden von E-Mails bei Wiedervorlage

4.1.15.1 hook_send_email_entry_10

```
int hook_send_email_entry_10(D3Interface d3, Document doc, String recipient, String sender, String subject, Integer trigger, String url_link)
```

Aufruf:

Vor dem Versenden einer E-Mail.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das Dokument, für das die E-Mail zur Postkorbnachricht gesendet wird
recipient	Empfänger der E-Mail (d.3-Benutzername oder E-Mail-Adresse)
sender	Absender der E-Mail (d.3-Benutzername)
subject	Betrefftext
trigger	0 = keine Wiedervorlage-E-Mail 1 = E-Mail für Wiedervorlage 2 = E-Mail für Workflow-Wiedervorlage
url_link	HTTP-Link für die Ansicht in d.3one Hinweis: Der Parameter ist nur gefüllt wenn d.3one installiert ist.

In dieser Hook-Funktion können E-Mail-Eigenschaften über die folgenden Hook-Eigenschaftswerte gesetzt werden:

Eigenschaft	Beschreibung
api_email_body_file	E-Mail-Body aus einer Datei laden. Name und Pfad einer Datei, die den Body-Text enthält
api_email_mail_format	"html": HTML-Format sonst: Text-Format (Standard)
api_email_attach	1 = das Dokument als Anhang an die E-Mail hängen 0 = Dokument nicht anhängen (Standard)

```
d3.hook.setProperty("api_email_body_file", "D:/hooks/data/myBody.html")
d3.hook.setProperty("api_email_mail_format", "html")
d3.hook.setProperty("api_email_attach", "1")
```

Die Eigenschaften werden jeweils nach erfolgtem E-Mail-Versand zurückgesetzt. Die E-Mail-Funktion kann durch Return-Wert ungleich 0 abgebrochen werden.

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_send_email_entry_10" )
  // (4)
  public int doSomething( D3Interface d3, Document doc, String recipient, String sender, String subject, Integer
trigger, String url_link ){
    // (5)
    d3.log.error("Hello world!");
    // (6)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.15.2 hook_send_email_entry_20

```
int hook_send_email_entry_20(D3Interface d3, Document doc, String recipient, String sender, String subject, Integer trigger)
```

Aufruf:

Vor dem Versenden einer E-Mail. E-Mailadresse wurde ermittelt, Gruppenauflösung wurde durchgeführt.

> **Hinweis**
>
> Die Hook-Funktion wird nur einmal aufgerufen, also nicht für jede versendete Mail separat.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das Dokument, für das die E-Mail zur Postkorbnachricht gesendet wird
recipient	Empfänger der E-Mail (d.3-Benutzername oder E-Mail-Adresse)
sender	Absender der E-Mail (d.3-Benutzername)
subject	Betrefftext
trigger	0 = keine Wiedervorlage-E-Mail 1 = E-Mail für Wiedervorlage 2 = E-Mail für Workflow-Wiedervorlage
url_link	HTTP-Link für die Ansicht in d.3one Hinweis: Der Parameter ist nur gefüllt wenn d.3one installiert ist.

In dieser Hook-Funktion können E-Mail-Eigenschaften über die folgenden Hook-Eigenschaftswerte gesetzt werden:

Eigenschaft	Beschreibung
api_email_body_file	E-Mail-Body aus einer Datei laden. Name und Pfad einer Datei, die den Body-Text enthält
api_email_mail_format	"html": HTML-Format sonst: Text-Format (Standard)
api_email_attach	1 = das Dokument als Anhang an die E-Mail hängen 0 = Dokument nicht anhängen (Standard)

```
d3.hook.setProperty("api_email_body_file", "D:/hooks/data/myBody.html")
d3.hook.setProperty("api_email_mail_format", "html")
d3.hook.setProperty("api_email_attach", "1")
```

Die Eigenschaften werden jeweils nach erfolgtem E-Mail-Versand zurückgesetzt. Die E-Mail-Funktion kann durch Return-Wert ungleich 0 abgebrochen werden.

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_send_email_entry_20" )
  // (4)
  public int doSomething( D3Interface d3, Document doc, String recipient, String sender, String subject, Integer
trigger ){
    // (5)
    d3.log.error("Hello world!");
    // (6)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.15.3 hook_send_email_exit_10

```
int hook_send_email_exit_10(D3Interface d3, Document doc, String recipient, String sender, String subject,
Integer retCode)
```

Aufruf:

Nach dem Versenden einer E-Mail.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das Dokument für die die E-Mail versendet wurde
recipient	Empfänger der E-Mail (d.3-Benutzername oder E-Mail-Adresse)
sender	Absender der E-Mail (d.3-Benutzername)
subject	Betrefftext
retCode	1 = Nachricht wurde gesendet 0 = Nachricht konnte nicht gesendet werden

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_send_email_exit_10" )
  // (4)
  public int doSomething( D3Interface d3, Document doc, String recipient, String sender, String subject, Integer
retCode ){
    // (5)
    d3.log.error("Hello world!");
    // (6)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.

2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.16 Sperren eines Dokuments

4.1.16.1 hook_block_entry_10

```
int hook_block_entry_10(D3Interface d3, Document doc, User user)
```

Aufrufzeitpunkt:

Vor dem Sperren eines Dokuments im Status "Freigabe".

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das zu sperrende Dokument
user	der ausführende Benutzer

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_block_entry_10" )
  // (4)
  public int doSomething( D3Interface d3, Document doc, User user ){
    // (5)
    d3.log.error("Hello world!");
    // (6)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.16.2 hook_block_exit_10

```
int hook_block_exit_10(D3Interface d3, Document doc, User user)
```

Aufrufzeitpunkt:

Nach dem Sperren eines Dokuments im Status "Freigabe".

Parameter	Beschreibung
d3	die d.3-Schnittstelle

Parameter	Beschreibung
doc	das gerade gesperrte Dokument
user	der ausführende Benutzer

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_block_exit_10" )
  // (4)
  public int doSomething( D3Interface d3, Document doc, User user ){
    // (5)
    d3.log.error("Hello world!");
    // (6)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.17 Stammdaten

4.1.17.1 hook_on_user_change_exit_10

```
int hook_on_user_change_exit_10(D3Interface d3, User actionUser, User newUser, User oldUser)
```

In diesem Einsprungspunkt können Anpassungen an einem gerade geänderten oder neu anzulegenden Benutzerobjekt vorgenommen werden.

Weder das Anlegen des Benutzers, noch ein Ändern des Benutzers kann in diesem Hook verhindert werden. D.h., macht dieser Hook Änderungen am Benutzerobjekt, die nicht in die Datenbank geschrieben werden können (beispielsweise, weil die maximale Spaltenlängen überschritten wurden), wird das Objekt so angelegt, als wäre der Hook nicht ausgeführt worden.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
actionUser	der ausführende Benutzer
newUser	der neue bzw. geänderte Benutzer
oldUser	das ungeänderte Benutzer-Objekt

Über das `newUser` Objekt können mittels der entsprechenden Setter folgende Benutzereigenschaften geändert werden:

Name der Eigenschaft	Beschreibung
email	E-Mail-Adresse des Benutzers
phone	Telefonnummer des Benutzers
plant	Werk des Benutzers
department	Abteilung des Benutzers
optField(int idx)	Optionale Felder zum Benutzer (Array-Indizes 1-10)

Wird der Hook mit einem Returncode ungleich "0" beendet, werden die Änderungen des Hooks am Benutzerobjekt ignoriert.

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_on_user_change_exit_10" )
  // (4)
  public int doSomething( D3Interface d3, User actionUser, User newUser, User oldUser ){
    // (5)
    d3.log.error("Hello world!");
    // (6)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.18 Statustransfer

4.1.18.1 hook_transfer_entry_30

```
int hook_transfer_entry_30(D3Interface d3, User user, Document doc, Integer fileId, String sourceStatus, String destStatus, UserOrUserGroup destEditor)
```

Aufrufzeitpunkt:

Vor dem Statustransfer eines Dokuments in einen anderen Status.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
user	der ausführende Benutzer
doc	das Dokument, das transferiert werden soll
fileId	File-ID der Dokumentversion
sourceStatus	Quellstatus des Dokuments (B, P, F, A)
destStatus	Zielstatus des Dokuments (B, P, A)
destEditor	Zielstatus „Bearbeitung“: Benutzer- oder Gruppenname Zielstatus „Prüfung“: Gruppenname Sonst leer

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.UserOrUserGroup;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_transfer_entry_30" )
  // (4)
  public int doSomething( D3Interface d3, User user, Document doc, Integer fileId, String sourceStatus, String
destStatus, UserOrUserGroup destEditor ){
    // (5)
    d3.log.error("Hello world!");
    // (6)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.

3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.18.2 hook_transfer_exit_30

```
int hook_transfer_exit_30(D3Interface d3, User user, Document doc, Integer fileId, String sourceStatus, String destStatus, UserOrUserGroup destEditor, Integer errorCode)
```

Aufrufzeitpunkt:

Nach dem Statustransfer eines Dokuments in einen anderen Status.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
user	der ausführende Benutzer
doc	das Dokument, das transferiert wurde
fileId	File-ID der Dokumentversion
sourceStatus	Quellstatus des Dokumentes (B, P, F, A)
destStatus	Zielstatus des Dokumentes (B, P, A)
destEditor	Zielstatus „Bearbeitung“: Benutzer- oder Gruppenobjekt Zielstatus „Prüfung“: Benutzer- oder Gruppenobjekt; null falls keine Prüfergruppe angegeben
errorCode	0 = Statustransfer erfolgreich Fehlercode sonst

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.UserOrUserGroup;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_transfer_exit_30" )
  // (4)
  public int doSomething( D3Interface d3, User user, Document doc, Integer fileId, String sourceStatus, String
destStatus, UserOrUserGroup destEditor, Integer errorCode ){
    // (5)
    d3.log.error("Hello world!");
    // (6)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.19 Validieren von Eigenschaftswerten (ValidateAttributes)

4.1.19.1 hook_validate_import_entry_10

```
int hook_validate_import_entry_10(D3Interface d3, User user, DocumentType docType, Document doc, String
nextcall)
```

> **Hinweis**
>
> Falls diese Hook-Funktion einen Wert ungleich 0 liefert, wird die Validierung der Suchbegriffe abgebrochen.
>
> Die API-Funktion ValidateAttributes liefert dann den Wert 9500-(X) zurück (allgemeiner Fehlercode in kundenspezifischer Hook-Funktion). "X" ist hier der Rückgabewert des Hooks.
>
> Es wird empfohlen, im Hook einen negativen Return-Wert zu verwenden, damit die Ausgabe des API-Calls >9500 ist, da dieser Bereich freigehalten wurde.
>
> Es ist dann möglich, auf den Client-Rechnern über die Client-Verteilung eine msglib.usr mit einem beliebigen Text für den Returncode (z.B. 9542) zu hinterlegen.

Aufrufzeitpunkt:

Es wurden lediglich die Eigenschaften des neu zu importierenden Dokuments zugewiesen.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
user	der ausführende Benutzer
docType	Dokumentart des zu validierenden Dokuments
doc	das Dokument, das vor dem Import validiert werden soll
nextcall	der Wert des Parameters "nextcall" der API-Funktion ValidateAttributes

> **Hinweis**
>
> Diese Funktion wird im Kontext der API-Funktion ValidateAttributes ausgeführt. Das bedeutet, dass die Funktion nicht ausgeführt wird, wenn ein Dokument über den Hostimport importiert wird. Sie wird ausgeführt, wenn man einen Import über d.3 import ausführt, da von diesem Programm vor dem Import eines Dokuments diese API-Funktion aufgerufen wird.

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_validate_import_entry_10" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething( D3Interface d3, User user, DocumentType docType, Document doc, String nextcall ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.19.2 hook_validate_search_entry_10

```
int hook_validate_search_entry_10(D3Interface d3, User user, DocumentType docType, Document searchContext, String nextcall)
```

> **Hinweis**
>
> Falls diese Hook-Funktion einen Wert ungleich 0 liefert, wird die Validierung der Suchbegriffe abgebrochen. Die API-Funktion ValidateAttributes liefert dann den Wert 9500-(X) zurück (allgemeiner Fehlercode in kundenspezifischer Hook-Funktion). "X" ist hier der Rückgabewert des Hooks. Es wird empfohlen, im Hook einen negativen Return-Wert zu verwenden, damit die Ausgabe des API-Calls >9500 ist, da dieser Bereich freigehalten wurde. Es ist dann möglich, auf den Client-Rechnern über die Client-Verteilung eine msglib.usr mit einem beliebigen Text für den Returncode (z.B. 9542) zu hinterlegen.

Aufrufzeitpunkt:

Es wurden lediglich die Suchbegriffe in die entsprechenden Kontextfelder transportiert.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
user	der ausführende Benutzer
docType	Dokumentart der gesuchten Dokumente, wenn dokumentart-spezifisch gesucht wird; ansonsten leeres Dokumentart-Objekt
searchContext	Dokument-Objekt über das die Suchbegriffe ausgelesen und geändert werden können
nextcall	der Wert des Parameters "nextcall" der API-Funktion ValidateAttributes

Diese Funktion wird im Kontext der API-Funktion `ValidateAttributes` ausgeführt.

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_validate_search_entry_10" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething( D3Interface d3, User user, DocumentType docType, Document searchContext, String
nextcall ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.19.3 hook_validate_update_entry_10

```
int hook_validate_update_entry_10(D3Interface d3, User user, DocumentType docType, Document doc, String nextcall)
```

Parameter	Beschreibung
d3	die d.3-Schnittstelle
user	der ausführende Benutzer
docType	Dokumentart des zu aktualisierenden Dokuments
doc	das d.3-Dokument, dessen Eigenschaften aktualisiert werden sollen. Diese können hier noch geändert werden. sonst, wenn keine Dokument-ID vorgegeben, wird hier Leerstring übergeben
nextcall	der Wert des Parameters "nextcall" der API-Funktion ValidateAttributes

Diese Hook-Funktion wird im Kontext der API-Funktion ValidateAttributes aufgerufen.

Wird eine Eigenschaft bei mehreren Dokumenten gleichzeitig mit Hilfe des changeatt.dxp geändert, greift der Einsprungpunkt hook_validate_update_entry_10 nicht, da keine Validierung (ValidateAttributes) stattfindet.

Diese Validierung findet statt, wenn ein Attribut bei einem Dokument über die Eigenschaften angepasst wird.

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_validate_update_entry_10" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething( D3Interface d3, User user, DocumentType docType, Document doc, String nextcall ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.20 Verknüpfen von Dokumente bzw. Akten (LinkDocuments)

4.1.20.1 hook_link_entry_30

```
int hook_link_entry_30(D3Interface d3, Document docFather, Document docChild)
```

> **Hinweis**
>
> Diese Hook-Funktion wird nur aktiviert, wenn zuvor kein Fehler aufgetreten ist. Liefert diese Funktion einen Wert ungleich 0, wird die Verknüpfungsaktion mit Fehler abgebrochen.

Aufrufzeitpunkt:

Alle Verknüpfungsdaten sind korrekt. Direkt vor dem Datenbank-Befehl, der die Verknüpfung registriert.

Parameter	Beschreibung
docFather	das übergeordnete Dokument bzw. die Akte
docChild	das untergeordnete Dokument

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_link_entry_30" )
  // (4)
  public int doSomething( D3Interface d3, Document docFather, Document docChild ){
    // (5)
    d3.log.error("Hello world!");
    // (6)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.

2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.20.2 hook_link_exit_10

```
int hook_link_exit_10(D3Interface d3, Document docFather, Document docChild, Integer errorCode)
```

Aufrufzeitpunkt:

Direkt nach Ausführung des Datenbank-Befehls, der die Verknüpfung einträgt.

Parameter	Beschreibung
docFather	das übergeordnete Dokument, bzw. die Akte
docChild	das untergeordnete Dokument
errorCode	0: Verknüpfung war erfolgreich -1: Vater und Kind sind identisch bzw. einer der beiden existiert gar nicht -2: Die beiden Dokumente sind bereits verknüpft -3: Die beiden Dokumente sind bereits in umgekehrter Hierarchie miteinander verknüpft -4: Beim Eintrag der Verknüpfung in die Datenbank trat ein Datenbankfehler auf (s. dazu „error_number“) error_number <> 0: - 91: Die beiden Dokumente sind bereits in umgekehrter Hierarchie miteinander verknüpft sonst Datenbank-Fehlernummer beim Eintrag der Verknüpfung in die Datenbank

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_link_exit_10" )
  // (4)
  public int doSomething( D3Interface d3, Document docFather, Document docChild, Integer errorCode ){
    // (5)
    d3.log.error("Hello world!");
    // (6)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.21 Web-Veröffentlichung

4.1.21.1 hook_webpublish_entry_10

```
int hook_webpublish_entry_10(D3Interface d3, Document doc, User user, Integer publish)
```

Aufrufzeitpunkt:

Vor bzw. nach dem Veröffentlichen eines Dokuments für das Web.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das Dokument, das veröffentlicht/zurückgezogen werden soll
user	der ausführende Benutzer
publish	1: Dokument wird veröffentlicht 0: Veröffentlichung wird zurückgezogen

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_webpublish_entry_10" )
  // (4)
  public int doSomething( D3Interface d3, Document doc, User user, Integer publish ){
    // (5)
    d3.log.error("Hello world!");
    // (6)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.21.2 hook_webpublish_entry_20

```
int hook_webpublish_entry_20(D3Interface d3, Document doc, User user, DocumentType docType, Integer publish)
```

Aufrufzeitpunkt:

Vor bzw. nach dem Veröffentlichen eines Dokuments für das Web.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das Dokument, das veröffentlicht/zurückgezogen werden soll
user	der ausführende Benutzer
docType	die zugehörige Dokumentart
publish	1: Dokument wird veröffentlicht 0: Veröffentlichung wird zurückgezogen

```groovy
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_webpublish_entry_20" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething( D3Interface d3, Document doc, User user, DocumentType docType, Integer publish ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.

2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.21.3 hook_webpublish_entry_30

```
int hook_webpublish_entry_30(D3Interface d3, Document doc, User user, DocumentType docType, Integer publish)
```

Aufrufzeitpunkt:

Vor bzw. nach dem Veröffentlichen eines Dokuments für das Web.

d3	die d.3-Schnittstelle
doc	das Dokument, das veröffentlicht/zurückgezogen werden soll
user	der ausführende Benutzer
docType	die zugehörige Dokumentart
publish	1: Dokument wird veröffentlicht 0: Veröffentlichung wird zurückgezogen

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_webpublish_entry_30" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething( D3Interface d3, Document doc, User user, DocumentType docType, Integer publish ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.21.4 hook_webpublish_exit_10

```
int hook_webpublish_exit_10(D3Interface d3, Document doc, User user, Integer errorCode, DocumentType docType, Integer publish)
```

Aufrufzeitpunkt:

Vor bzw. nach dem Veröffentlichen eines Dokuments für das Web.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc_id	das Dokument, das veröffentlicht/zurückgezogen wurde
user	der ausführende Benutzer
errorCode	0= Aktion erfolgreich Fehlercode sonst
docType	Dokumentartkürzel
publish	1: Dokument wurde veröffentlicht 0: Veröffentlichung wurde zurückgezogen

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_webpublish_exit_10" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething( D3Interface d3, Document doc, User user, Integer errorCode, DocumentType docType,
Integer publish ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.

2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.21.5 hook_webpublish_exit_20

```
int hook_webpublish_exit_20(D3Interface d3, Document doc, User user, Integer errorCode, DocumentType docType, Integer publish)
```

Aufrufzeitpunkt:

Vor bzw. nach dem Veröffentlichen eines Dokuments für das Web.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc_id	das Dokument, das veröffentlicht / zurückgezogen wurde
user	der ausführende Benutzer
errorCode	0= Aktion erfolgreich Fehlercode sonst
docType	Dokumentartkürzel
publish	1: Dokument wurde veröffentlicht 0: Veröffentlichung wurde zurückgezogen

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_webpublish_exit_20" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething( D3Interface d3, Document doc, User user, Integer errorCode, DocumentType docType,
Integer publish ){
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.21.6 hook_webpublish_exit_30

```
int hook_webpublish_exit_30(D3Interface d3, Document doc, User user, Integer errorCode, DocumentType docType, Integer publish)
```

Aufrufzeitpunkt:

Vor bzw. nach dem Veröffentlichen eines Dokuments für das Web.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc_id	das Dokument, das veröffentlicht/zurückgezogen wurde
user	der ausführende Benutzer
errorCode	0= Aktion erfolgreich Fehlercode sonst
docType	Dokumentartkürzel
publish	1: Dokument wurde veröffentlicht 0: Veröffentlichung wurde zurückgezogen

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_webpublish_exit_30" )
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  public int doSomething( 3Interface d3, Document doc, User user, Integer errorCode, DocumentType docType,
Integer publish {
    // (6)
    d3.log.error("Hello world!");
    // (7)
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
6. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
7. Die Funktion wird mit einem Return-Wert "0" beendet.

4.1.22 Workflow

4.1.22.1 hook_workflow_cancel_exit_20

```
int hook_workflow_cancel_exit_20(D3Interface d3, Document doc, String wflId, String stepId, User user)
```

Aufrufzeitpunkt:

Nachdem der Workflow für ein Dokument abgebrochen wurde.

Der Abbruch des Workflows kann hier nicht gestoppt werden.

Diese Hookfunktion wird nur aktiviert, wenn zuvor kein Fehler aufgetreten ist.

Parameter	Beschreibung
doc	das Dokument dessen Workflow-Durchlauf abgebrochen wurde
wflId	die ID des Workflows
stepId	ID des Workflow-Schrittes
user	der ausführende d.3-Benutzer

```
// (1) Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;

// (2)
public class D3Hooks{
  // (3)
  @Entrypoint( entrypoint = "hook_workflow_cancel_exit_20" )
  // (4
  public int doSomething( 3Interface d3, Document doc, String wflId, String stepId, User user {
    // (5
    d3.log.error("Hello world!");
    // (6
    return 0;
  } // end of doSomething
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.

2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

4.2 Validierungshooks

Validierungshook-Funktionen können zur individuellen Eingabe-Validierung genutzt werden (Plausibilitätshooks).

- Die Annotation lautet: @Validation(entrypoint="<Bezeichner in Adminstration>")
- Rückgabewert: 0 bei Erfolg und <> 0 bei Fehler

```
int myValidationHook(D3Interface d3, String value, Document doc)
```

Parameter	Beschreibung
d3	die d.3 Schnittstelle
value	der zu prüfende Eigenschaftswert
doc	das Dokument, zu dem die Eigenschaft gehört

```
@Validation(entrypoint="ITValueValidation")
int validateValue(D3Interface d3, String value, Document doc)
{
  if( value == "Peter" && doc.field["name"] == "Smith"){
    return 0;
  }
  else{
    return -1;
  }
}// end of validateValue
```

Validierung einer Bestellnummer auf ein gültiges Format

> **Hinweis**
>
> Szenario:
>
> Die Bestellnummer soll immer dem Format "Zwei Zahlen-Zwei Buchstaben-Fünf Zahlen" (/[0-9]{2}-[a-zA-Z]{2}-[0-9]{5}/) genügen. Natürlich kann man das direkt in d.3 admin konfigurieren, aber als Beispiel um die Funktionsweise für die Validierung zu demonstrieren, ist es ebenfalls geeignet.

Zur Realisierung wird auf die Dokumenteigenschaft Bestellnummer eine Funktion zur Validierung definiert.

```
//(1)
// Global d.3 libraries
import com.dvelop.d3.server.core.D3;

// Libraries to handle the diferent hook types
import com.dvelop.d3.server.Validation;
//(2)
public class D3Validate{
//(3)
  @Validation( entrypoint = "checkOrderNumber" )
//(4)
  public int checkOrderNumber( D3 d3, def currentValue, Document doc ){
//(5)
    def tmpValue  = currentValue;
    def matchFlag = ( tmpValue ==~ /[0-9]{2}-[a-zA-Z]{2}-[0-9]{5}/ );
//(6)
    return( matchFlag ? 0 : -1 );
  } // end of checkOrderNumber
}// end of D3Validate
```

Das Ganze kann mit Groovy etwas kürzer realisiert werden.

```
//(1)
// Import the required d.3 classes
import com.dvelop.d3.server.core.D3;
import com.dvelop.d3.server.Validation;

 //(2)
 public class D3Validate {
  //(3)
  @Validation( entrypoint = "checkOrderNumber" )
  //(4)
  public int checkOrderNumber( D3 d3, def currentValue ) {
    //(6)
    return( ( currentValue ==~ /[0-9]{2}-[a-zA-Z]{2}-[0-9]{5}/ ) ? 0 : -1 );
  }// end of checkOrderNumber
}// end of D3Validate
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Klassen.
2. Erstellen einer eigenen Klasse.
3. Um nun eine Groovy-Methode für die Validierung einer Dokumenteigenschaft nutzen zu können, erfolgt über die Annotation @Validation eine Registrierung der Methode für eine in d.3 admin konfigurierte, Validierungsfunktion.
4. Die Methode nimmt dann die oben beschriebenen Parameter in der vorgegebenen Reihenfolge entgegen.
5. Innerhalb der Methode kann nun der übergebene Wert überprüft werden, im Beispiel mittels eines regulären Ausdrucks.
6. Entspricht der Wert einem gültigen Wert, kann eine 0 ansonsten eine 1 zurückgegeben werden.

4.3 Wertemengen-Hooks

Wertemengen-Hooks dienen zur Erzeugung dynamischer Wertemengen.

- Die Annotation lautet: @ValueSet(entrypoint="<Bezeichner in Administration>")
- Maximal können mit einen Wertemengen-Hook 10.000 Werte zurückgegeben werden.
- Sortierung: Die durch die Groovy-Hook-Funktion vorgegebene Reihenfolge der Werte wird beibehalten.

```
def myValueSetHook(D3Interface d3, RepositoryField repoField, User user, DocumentType docType, Integer rowNo, Integer validate, Document attribContext)
```

Parameter	Beschreibung
d3	die d.3-Schnittstelle
repoField	das Eigenschaftenfeld für das die Wertemenge definiert ist per Methode provideValuesForValueSet() können die gewünschten Werte übergeben werden
user	der aufrufende Benutzer
docType	Dokumentart, in der die Wertemenge enthalten ist
rowNo	Zeilennummer für Mehrfacheigenschaften
validate	Aufruf zur Wertvalidierung (0/1)
attribContext	Dokument-Objekt mit dem Attributkontext. Dieses enthält bei der Suche: die übrigen Suchkriterien. Beim Import und Update die übrigen bereits gefüllten Attribute.

```
@ValueSet(entrypoint="MyMonths")
def myMonthsList(D3Interface d3, RepositoryField repoField, User user, DocumentType docType, Integer
row_no, Integer validate, Document attribContext) {
  List<String> names = ["01", "02", "03" /*, ...*/];
  repoField.provideValuesForValueSet(names);

  boolean translationGotOutdated = false;
  if(translationGotOutdated){
    d3.getArchive().removeTranslationFromCache("MyMonths", Locale.GERMAN);
    d3.getArchive().removeTranslationFromCache("MyMonths", new Locale("de", "AT"));
  }
}// end of myMonthsList
```

Einfache Wertmengen

Wir starten mit einer statischen einfachen Wertemenge welche über das Skript zur Verfügung gestellt wird.

```
1  // (1) Global d.3 libraries ------------------------------------------------
2  import com.dvelop.d3.server.core.D3Interface;
3  import com.dvelop.d3.server.Document;
4  import com.dvelop.d3.server.User;
5  import com.dvelop.d3.server.DocumentType;
6
7  // Libraries to handle the different hook types ------------------------------
8  import com.dvelop.d3.server.ValueSet;
9
10 // Special libraries ---------------------------------------------------------
11 import com.dvelop.d3.server.RepositoryField;
12
13 // (2)
14 class SimpleValueSet{
15   // (3)
16   @ValueSet( entrypoint = "customerNumbers" )
17   // (4)
18   def getCustomerNumber( D3Interface d3, RepositoryField reposField, User user, DocumentType
     docType, int rowNo, int validate, Document doc ){
19
20     // (5) Define static list of customer numbers ----------------------------
21     def customerList = ["4711", "4712", "4713", "4714" ];
22
23     // (6) Prepare List for interaction ---------------------------------------
24     if( customerList.size() > 0 ){
25       reposField.provideValuesForValueSet( customerList );
26     }
27   } // end of getCustomerNumber
28 }// end of SimpleValueSet
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.

3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen.
5. Ermittlung von Werten welche in der Wertmenge zur Verfügung gestellt werden.
6. Bereitstellung der Werte als Auswahlliste für den User.

Statische Wertmengen interne Datenbank

Damit eine Wertemenge nicht an zwei Stellen gepflegt werden muss, können die Daten aus dem führenden System ermittelt und als Auswahlliste zur Verfügung gestellt werden.

```
1  // (1) Global d.3 libraries ------------------------------------------------
2  import com.dvelop.d3.server.core.D3Interface;
3  import com.dvelop.d3.server.Document;
4  import com.dvelop.d3.server.User;
5  import com.dvelop.d3.server.DocumentType;
6
7  // Libraries to handle the different hook types ------------------------------
8  import com.dvelop.d3.server.ValueSet;
9
10 // Special libraries ---------------------------------------------------------
11 import com.dvelop.d3.server.RepositoryField;
12
13 // (2)
14 class StaticValueSet{
15   // (3)
16   @ValueSet( entrypoint = "customerNumbers" )
17   // (4)
18   def getCustomerNumber( D3Interface d3, RepositoryField reposField, User user, DocumentType
   docType, int rowNo, int validate, Document doc ){
19
20     // (5) Prepare sql statmenet ----------------------------------------------
21     def sqlQuery = "SELECT customerNo FROM CustomerData ORDER BY customerNo DESC"; // !!
   ATTENTION
22
23     // (6) Execute sql statmenet ----------------------------------------------
24     def resultRows = d3.sql.executeAndGet( (String) sqlQuery );
25
26     // (7) Prepare list for user interface ------------------------------------
27     if( resultRows.size() > 0 ){
28       reposField.provideValuesForValueSet( resultRows.collect{ it.customerNo } );
29     }
30   } // end of getCustomerNumber
31 }//end of StaticValueSet
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.

2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen.
5. Bereitstellung eines SQL-Statements zur Ermittlung der notwendigen Werte aus der Datenbank.
6. Ausführung des SQL-Statements gegen die d.3-Datenbanktabelle.
7. Bereitstellung der Werte als Auswahlliste für den User.

Statische Wertmengen externe Datenbank

Damit eine Wertemenge nicht an zwei Stellen gepflegt werden muss, können die Daten aus dem führenden System ermittelt und als Auswahlliste zur Verfügung gestellt werden.

```groovy
// (1) Global d.3 libraries ----------------------------------------------------
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;

// Libraries to handle the different hook types --------------------------------
import com.dvelop.d3.server.ValueSet;

// Special libraries -----------------------------------------------------------
import com.dvelop.d3.server.RepositoryField;

// (2)
class StaticValueSet{
  // (3)
  @ValueSet( entrypoint = "customerNumbers" )
  // (4)
  def getCustomerNumber( D3Interface d3, RepositoryField reposField, User user, DocumentType docType, int rowNo, int validate, Document doc ){

    // (5) Prepare database Connection ---------------------------------
    def dbConnection = Sql.newInstance( "jdbc:sqlserver:<ServerName>\<InstanceName>:0;databaseName=<Databse>", "<User>", „<Password>" );

    // (6) Prepare sql statmenet ---------------------------------------
    def sqlQuery  = "SELECT customerNo FROM CustoemrData ORDER BY customerNo DESC"; // !! ATTENTION

    // (7) Execute sql statmenet ---------------------------------------
    def resultRows = dbConnection.rows( (String) sqlQuery );

    // (8) Prepare list for user interface -----------------------------
    if( resultRows.size() > 0 ){
      reposField.provideValuesForValueSet( resultRows.collect{ it.customerNo } );
    }
  } // end of getCustomerNumber
}// end of StaticValueSet
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen.

5. Aufbau der JDBC-Verbindung zur externen Datenbank; der dazu gehörige JDBC-Treiber muss entsprechend zur Verfügung gestellt werden...
6. Bereitstellung eines SQL-Statements zur Ermittlung der notwendigen Werte aus der Datenbank.
7. Ausführung des SQL-Statements gegen die d.3-Datenbanktabelle.
8. Bereitstellung der Werte als Auswahlliste für den User.

Dynamische Wertemengen

Damit eine Wertemenge nicht an zwei Stellen gepflegt werden muss, können die Daten aus dem führenden System ermittelt und als Auswahlliste zur Verfügung gestellt werden. Dabei ist dann auch eine dynamische Berücksichtigung von Suchkriterien möglich..

```groovy
// (1) Global d.3 libraries ------------------------------------------------
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;

// Libraries to handle the different hook types ----------------------------
import com.dvelop.d3.server.ValueSet;

// Special libraries -------------------------------------------------------
import com.dvelop.d3.server.RepositoryField;

//---------------------------------------------------------------------------
// (2)
public class DynamicValueSet{
  // (3)
  @ValueSet(entrypoint = "customerNumbers" )
  // (4)
  def getCustomerNumber( D3Interface d3, RepositoryField reposField, User user, DocumentType docType, int rowNo, int validate, Document doc ){

    // (5) Get needed values from the document properties ----------------
    def customerNo = doc.field[1];
    def zipCode    = doc.field[6];

    //(6) If needed filter on the customer no ---------------------------
    if( customerNo == null || ( customerNo != null && customerNo.size() < 3  )) {
      reposField.provideValuesForValueSet( "Please enter at least 3 characters!" );
    }

    // (7) Prepare the sql-statement with the needed params ---------------
    def sqlQuery = """SELECT customerNo + ' ' + name AS 'completeName'
                FROM CustomerData WHERE 1 = 1 """;

    def sqlParams = [];

    sqlQuery += " AND customerNo LIKE ? OR name LIKE ?";

    sqlParams.add( customerNo + "%" ); // for the first questionmark after customerNo
    sqlParams.add( customerNo + "%" ); // for the second questionmark after Name

    if( zipCode!= null && zipCode!= "" ){
      sqlQuery += " AND zipCode LIKE ?";
      sqlParams.add( zipCode + "%" );
    }

    // (8) Using external database ---------------------------------------
    def dbConnection = Sql.newInstance( "jdbc:sqlserver:<ServeRName>\<InstanceName>:0;databaseName=<Database>", "<User>", „<Password>" );

    // (9) Using external database ---------------------------------------
    def resultRows = dbConnection.rows( sqlQuery, sqlParams );

    // (10) --------------------------------------------------------------
    if( resultRows.size() > 0 ){
```

```
54          reposField.provideValuesForValueSet( resultRows.collect{ it.completeName } );
55        }
56      } // end of getCustomerNumber
57    } // end of DynamicValueSet
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen.
5. Definition der Filter-Variablen und die Übergabe der erweiterten Eigenschaften.
6. Die Inhalte der erweiterten Eigenschaften werden kontrolliert und bei fehlenden oder zu wenig bereitgestellten Inhalten wird die Generierung abgebrochen.
7. Zusammenstellung des SQL-Statement inkl. der dynamischen Zuordnung der bereitgestellten Filterkriterien.
8. Aufbau der JDBC-Verbindung zur externen Datenbank; der dazu gehörige JDBC-Treiber muss entsprechend zur Verfügung gestellt werden.
9. Ausführung des SQL-Statements gegen die d.3-Datenbanktabelle.
10. Bereitstellung der Werte als Auswahlliste für den User.

Übersetzung von dynamischen Hook-Wertemengen

Es ist möglich, dynamische Hook-Wertemengen zu übersetzen.

- Die Annotation lautet: @ValueSetTranslation(entrypoint="<Bezeichner in Administration>")
- Der Entry-Point-Name entspricht dem Namen des zugehörigen @ValueSet-Hooks.
- Der Übersetzungs-Hook muss immer alle Werte für das jeweilige Eigenschaftsfeld in der angeforderte Sprache liefern, nicht nur die Werte, die für den aktuell aktiven Benutzer oder Kontext relevant sind.
- Die Übersetzungen werden vom d.3-Server gecacht. Die Dauer, für die diese Werte gecacht bleiben, ist implementationsabhängig und kann sich mit zukünftigen Versionen ändern.
  Mit der Funktion d3.getArchive().removeTranslationFromCache() kann ein Neuladen der Übersetzungen zu jeder Zeit erzwungen werden. Diese Funktion kann von beliebiger Stelle aus aufgerufen werden, nicht nur aus dem Wertemengen-Hook.
- Hinweis: diese Schnittstelle ist schon für die Unterstützung von regionalen Dialekten vorbereitet. Bitte beachten Sie aber, dass d.3 zum gegenwärtigen Zeitpunkt noch keine vollständige Unterstützung für dieses Feature bietet.

- An die Volltext-Engine (d.search) wird nur der Speicher-Wert übergeben, nicht die Übersetzungen. Eine Volltextsuche nach den Übersetzungen ist somit nicht möglich.

def myValueSetTranslation(D3Interface d3, Translation transl)

Parameter	Beschreibung
d3	die d.3-Schnittstelle
transl	das Übersetzungs-Objekt. Die angeforderte Zielsprache und der zugehörige Entry-Point sind in diesem Objekt vorgegeben und dürfen nicht geändert werden. Über die *set*-Methode können Werte eingetragen werden.

```
@ValueSetTranslation(entrypoint="MyMonths")
def myMonthsTranslation(D3Interface d3, Translation transl) {
  def lang = transl.locale.language

  if (lang == "de") {
    if (transl.locale.country == "AT")
      transl.set("01", "Jänner");
    else
      transl.set("01", "Januar");
    transl.set("02", "Februar");
    transl.set("03", "März");
    // ...
  } else if (lang == "th") {
    transl.set("01", "มกราคม");
    transl.set("02", "กุมภาพันธ์");
    transl.set("03", "มีนาคม");
    // ...
  } else {
    transl.set("01", "January");
    transl.set("02", "February");
    transl.set("03", "March");
    // ...
  }
}
```

4.4 Dokumentklassen-Hooks

Dokumentklassen-Hooks können zur Bestimmung dynamischer Berechtigungen genutzt werden, die nicht mit d.3 Dokumentklassen und Restriktionsmengen abgebildet werden können.

- In der Administration anzugeben per: `@D3HOOK` ("Bezeichner für den Hook")
- Die Annotation lautet: @DocumentClass(entrypoint="<der per `@D3HOOK` angegebene Bezeichner>")

```
int myDocumentClassHook(D3Interface d3, String value, DocumentType docType, String userId, Document doc)
```

Parameter	Beschreibung
d3	die d.3-Schnittstelle
value	Wert der Dokumenteigenschaft, für das die Hook-Funktion aufgerufen wurde
docType	die Dokumentart des zu prüfenden Dokuments
userId	d.3-Benutzer-ID des ausführenden Benutzers
doc	das zu prüfende Dokument

Rückgabewert:

1: Berechtigt

0: kein Zugriff

```
// (1) Global d.3 libraries ------------------------------------------------------
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Condition;

// Libraries to handle the different hook types ------------------------------
import com.dvelop.d3.server.DocumentClass;
// (2)
public class D3DocumentClassHook{
  // (3)
  @DocumentClass(entrypoint="myDocumentClassHook")
  // (4)
  @Condition( doctype = ["XXXX"] )
  // (5)
  def myDocumentClassHook(D3Interface d3, String value, DocumentType docType, String userId, Document
doc){
    if (value > 10000 && docType.id == "DINV"){
      if (value <= 20000 && doc.owner == "Meyer"){
        return 1;
      }
      else if (value > 20000 && doc.owner == "Chef"){
        return 1;
      }
      else {
        return 0;
      }
    }
    else
      return 1;
    }
  }// end of myDocumentClassHook
}// end of D3DocumentClassHook
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Optional können die Funktionen mit einer weiteren Annotation **Condition** auf bestimmte Dokumentklassen gemapt werden.
5. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen.

> **Hinweis**
>
> Dokumentklassen-Hooks werden bei einer Suche für jedes einzelne Dokument in der Treffermenge aufgerufen!
> Das kann zu Performance-Problemen bei der Rechteprüfung führen und somit die Dokumenten-Suche verlangsamen, insbesondere dann, wenn im Dokumentklassen-Hook SQL-Kommandos abgesetzt werden.
> Noch größere Auswirkungen auf die Performance bestehen, wenn Dokumentklassen-Hooks für Mehrfacheigenschaftsfelder (60er Felder) eingesetzt werden, da diese dann zusätzlich jede mit einem Wert belegte Zeile jeder Mehrfacheigenschaft aufgerufen werden.

> **Wichtig**
>
> Dokumentklassen-Hooks werden bei der Dokumenten-Suche parallel in mehreren Threads ausgeführt. Diese Hook-Funktionen dürfen daher nur threadsicheren Code enthalten und aufrufen.

4.5 Groovy-Schnittstelle in d.3 admin

d.3 admin

Unter **d.3 admin > d.3 config** gibt es die Bereiche **Hook-Funktionen** und **Java/Groovy**, welche beide im Kontext einer Groovy-Nutzung konfiguriert werden müssen.

Hook-Funktionen

In d.3 admin müssen im Bereich **Systemeinstellungen > d.3 config** folgende Konfigurationen vorgenommen werden:

- **Groovy-Hooks aktivieren**
  Geben sie die Verzeichnisse, in denen die per Groovy implementierten, kundenspezifischen Programmanpassungen enthalten sind, hier an.
- **Neuladen von Groovy-Hooks** (NUR WÄHREND DER ENTWICKLUNG UND NUR AUF DEM TEST-SYSTEM)
  Ist dieser Schalter aktiviert, so führt das Speichern von Änderungen in Groovy Hook-Dateien dazu, das diese direkt neu geladen werden. Dadurch muss kein d.3-Prozess neu gestartet werden und die Code-Änderungen sind sofort aktiv. Dieser Schalter kann bei der Entwicklung von Hook-Funktionen hilfreich sein. In Produktivumgebungen sollte er jedoch ausgeschaltet bleiben.

Java/Groovy

In d.3 admin müssen im Bereich **Systemeinstellungen > d.3 config** folgende Konfigurationen vorgenommen werden:

- **Java/Groovy Support**
  Einschalten der Unterstützung für die Ausführung von Java und Groovy Code in d.3.

- **Java CLASSPATH**
  Dateipfad bzw. Dateipfade Semikolon-getrennt zu den eigenen Java Klassen. Hier kann ein Verzeichnis/können Verzeichnisse angegeben werden, in dem/denen die .class-Dateien abgelegt sind oder auch der Dateiname einer JAR-Datei. Hier wäre auch die Option eine Classpath-Datei zu nutzen.
- **Java/Groovy API Funktionen**
  Aktiviert die PlugIn Schnittstelle für API Funktionen entwickelt in Java bzw. Groovy. Groovy-Skripte oder JAR-Dateien, die d.3-API-Funktionen implementieren, werden aus diesem Verzeichnis geladen. Nicht empfohlen.
- **Java Remote Debugging**
  Java Virtual Maschine im Debugmodus starten. Dadurch ist es möglich sich per Remote Java Debugger mit einem d.3-Server-Prozess zu verbinden, um die darin ausgeführten Groovy Hooks zu debuggen. Für die Kommunikation wird Port 43400 benutzt. Da jeder d.3-Prozess eine eigene Java Virtual Machine (JVM) startet, werden die benutzten Ports hochgezählt. Der erste mit aktiviertem JAVA_REMOTE_DEBUGGING gestartete Prozess öffnet Port 43400, der Zweite Port 43401 usw. Der ermittelte Port wird beim Start der JVM per Meldung **Java Remote Debugging Port** in das d.3-Log ausgegeben.
  Hinweis: Die JVM wird von d.3 On-Demand beim ersten Zugriff auf Groovy-Code gestartet und steht damit i.d.R. noch nicht direkt nach Start des Prozesses zur Verfügung.

4.6 Programmierung von Hook-Funktionen

Benötigte Java-Bibliotheken

Im Groovy-Kontext werden Bibliotheken benötigt, welche über die Integration der Datei **groovyhook.jar** zur Verfügung stehen und nur noch in den Groovy-Dateien referenziert werden müssen.

Globale Bibliotheken

Zur Nutzung des d.3-Interface-Objektes wird eigentlich nur die Bibliothek **com.dvelop.d3.server.core.D3Interface** benötigt.

> **Hinweis**
>
> Da es aktuell bei der Nutzung der JavaDoc-Dokumentation und damit der Groovy-Templates mit der Implementierung von "`d3Interface`" noch technische Herausforderungen gibt, kann zur Programmierung auch die Basis-Bibliothek **import com.dvelop.d3.server.core.D3** genutzt werden. Hier sollte aber auf jeden Fall für die produktive Nutzung wieder auf das **D3Interface** gewechselt werden.

Spezielle Bibliotheken für die einzelnen Hook-Typen

Annotation	Einsatz	Syntax	Benötigte Java-Bibliothek
@Entrypoint	d.3 Eintrittspunkte	@Entrypoint(entrypoint="name_in_admin", order* = n)	import com.dvelop.d3.server.Entrypoint;
@ValueSet	Wertemengen-Hooks	@ValueSet(entrypoint="name_in_admin", order* = n)	import com.dvelop.d3.server.ValueSet; import com.dvelop.d3.server.RepositoryField;
@Validation	Validierungs-Hooks	@Validation(entrypoint="name_in_admin", order* = n)	import com.dvelop.d3.server.Validation;
@DocumentClass	Dokumentklassen-Hooks	@DocumentClass(entrypoint="name_in_admin", order* = n)	import com.dvelop.d3.server.DocumentClass;
@Condition	Filter auf bestimmte Dokumentklassen	@Condition(doctype = ["DRECH", "DBEST", "DLIEF"])	import com.dvelop.d3.server.Condition;

> **order-Option**
>
> * Der optionale Parameter **order** kann zur Definition einer Reihenfolge der Abarbeitung definiert werden, wenn auf einem Eintrittspunkt mehrere Groovy-Funktionen definiert sind.

Styleguide-Empfehlung

Zur Bereitstellung der Funktionalität muss mind. eine Klasse vom Typ **public** angelegt werden; wobei **public** im Kontext Groovy auch weggelassen werden kann. Hier könnte man für die einzelnen Typen von Hook-Funktionen jeweils eine eigene Klasse, vielleicht sogar eine eigene Datei, bereitstellen. Die einzelnen Klassennamen könnten wie folgt aussehen:

Klassenname	Beschreibung
D3Hooks	Für die Behandlung von Eintrittspunkten
D3DataSets	Für die Implementierung von Wertemengen
D3Validate	Zur Bereitstellung von Validierungs-Funktionen auf Eigenschaftenebene
D3DocClasses	Zur Realisierung von spezifischen Dokumentklassen

Klassenname	Beschreibung
D3FolderScheme	Falls erweiterte Aktenpläne benötigt werden könnte diese Klasse bereitgestellt werden.

Natürlich könnte es auch sinnvoll sein, abhängig von Projekten oder Lösungen unterschiedliche Dateien, Klassen und Funktionen bereitzustellen. Hier ist die Namenskonvention nur ein Vorschlag.

Registrieren einer Groovy-Methode als Hook-Funktion

Für die Registrierung der verschiedenen Hook-Typen stehen die folgenden Annotationen zur Verfügung:

Die Registrierung besteht darin, dass die Annotation im Quellcode direkt der Java/Groovy-Methode vorangestellt wird, die für den per **entrypoint** angegebenen Hook-Eintrittspunkt aufgerufen werden soll.

```
import com.dvelop.d3.server.Entrypoint;

public class MyHooks{
  @Entrypoint(entrypoint="hook_insert_entry_10", order = 1 )
  @Condition( doctype= ["DRECH", "DBEST", "DLIEF" ] )
  int checkIncommingDocs(D3Interface d3, User user, DocumentType docType, Document doc){
    println "The function checkIncommingDocs was called inside the hook function entry Point
hook_insert_entry_10";
    return 0;
  } // end of checkIncommingDocs
}// end of MyHooks
```

> **Hinweis**
>
> Eine Groovy-Klasse die von d.3 geladen und registriert werden soll, muss einen öffentlichen Konstruktor ohne Parameter (public no-argument constructor) bereitstellen. Das ist auch erfüllt, wenn der Konstruktor weggelassen wird, weil dann der Java Default-Konstruktor implizit existiert.

Rückgabewert von Hook-Methoden

Als Rückgabewert wird eine Zahl (Integer) erwartet. Dieser zurückgegebene Wert wird vom Server als Fehlercode ausgewertet.
Wert = 0 ==> Erfolg!
Wert <> 0 ==> Fehler in Hook-Funktion. Je nach Hook-Funktion führt dies zum Abbruch der Aktion in deren Kontext die Hook-Funktion ausgeführt wurde.

Hinweis

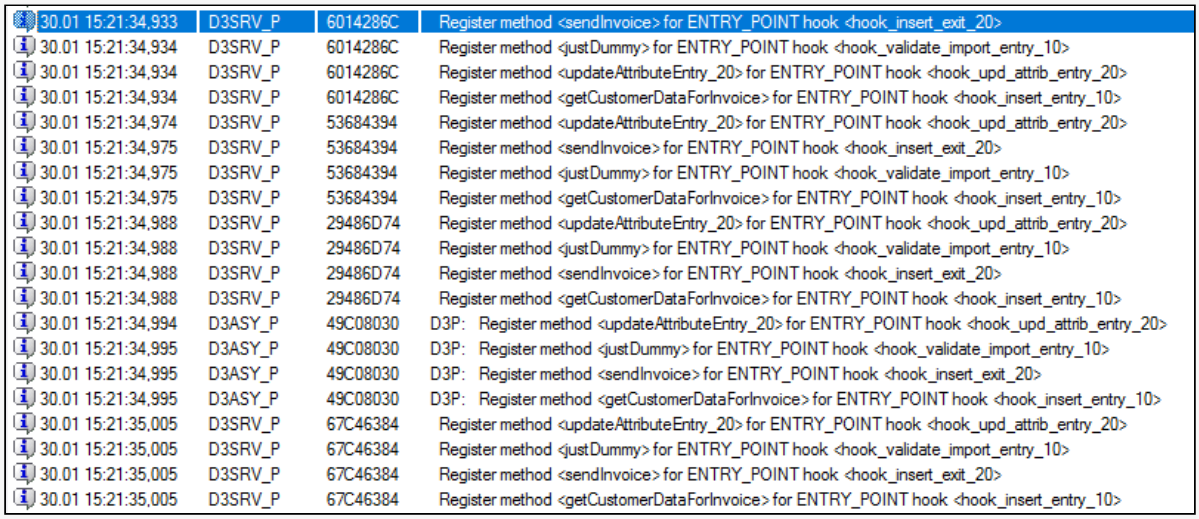
Bei Groovy ist das Schlüsselwort **return**“ für das Verlassen einer Methode mit Rückgabewert optional, kann also weggelassen werden.
Wenn es nicht angegeben ist, dann nimmt Groovy kurzer Hand den letzten Variablenwert, der vor der schließenden Klammer benutzt wurde und gibt diesen implizit als Returnwert zurück.
Dies kann zu Fehlern oder ungewollten Rückgabewerten führen. Deshalb sollte eine Hook-Methode explizt mit return beendet werden. Wenn der Rückgabewert nicht relevant ist, dann mit **return 0**.

Groovy Hook-Funktionen für die d.3-Eintrittspunkte werden automatisch konfiguriert.

- Die Bezeichner für d.3-Eintrittspunke müssen nicht mehr unter **d.3 admin > d.3 config >Hook-Funktionen > Hook-Funktionen ausführen** einzeln eingetragen werden.
- Wird beim Laden einer Groovy-Klasse eine Annotation für einen Eintrittspunkt gefunden, so wird die annotierte Methode dafür registriert.
- Im Config-Modul ist hinter dem Eintrittspunkt dann **<groovy hook>** als Kennzeichnung für die Registrierung eingetragen.
- Es können auch mehrere Methoden pro Eintrittspunkt registriert werden.

Wichtig

Da die Aktualisierung von d.3 admin manchmal etwas dauern kann, kann auch die Ausgabe im Log-File genutzt werden, dort wird das erfolgreiche Laden der Groovy-Funktionen zu den Eintrittspunkten ebenfalls dokumentiert.



Verwenden von Java-Bibliotheken

Sollte es bei der Umsetzung einer Hook-Funktion notwendig werden, von Drittanbietern oder selbst erstellte Java-Bibliotheken zu verwenden (zum Beispiel um ihr CRM-System anzusprechen), können Sie diese Bibliotheken für jeden Hook spezifisch angeben.

Legen sie dazu neben ihrer vorhandene **<Hookname>.groovy** eine weitere Datei **<Hookname>.classpath** an. In dieser Datei können zeilenweise Einträge für den Java-Classpath definiert werden. Es können absoluter Pfade, sowie Pfade relativ zum aktuellen Verzeichnis verwendet werden, die auf JAR-Dateien zeigen.

Auf diese Weise sind die Java-Bibliotheken voneinander isoliert, sodass Sie in mehreren Hooks unterschiedliche Versionen von Bibliotheken verwenden können.

> **Hinweis**
>
> JDBC-Treiber können nicht hookspezifisch eingehängt werden, sondern müssen global geladen werden. Details dazu unter Access to other databases.

Isolation von Hook-Klassen

Jede Hook-Klasse wird von einer eigenen Groovy-Classloader-Instanz geladen. Dadurch kann ein Hook-Objekt nicht auf die Eigenschaften anderer Hook-Objekte zugreifen.
Allerdings kann jede Groovy-Klasse per "import" Kommando in einer anderen sichtbar gemacht werden. Die Klasse kann dann instanziiert werden oder, im Fall von statischen Elementen, können diese direkt aufgerufen werden.

Gibt es mehrere, thematisch zusammengehörige Hook-Klassen, so sollte gemeinsam genutzter Code in eine eigene Klasse und damit ein eigenes Modul ausgelagert werden.

Soll mittels Hook beim Import validiert werden, ob die Kundendaten so wie eingegeben auch im CRM-System hinterlegt wurden, und gleichzeitig eine Wertemenge möglicher Kunden angeboten werden, dann findet man hier typischerweise gemeinsam genutzten Code. Dieser sollte in einer eigenen Groovy-Klasse implementiert werden, die wiederrum von den anderen Groovy-Hook-Klassen genutzt werden kann.

Package-Struktur

Genau wie Java-Klassen werden auch Groovy-Klassen in Packages organisiert. Der Name des Packages muss am Anfang der Quelldatei vor den Import-Anweisungen und der ersten Klassendefinition genannt sein. Dabei können Sie die gewohnte Form der Strukturierung anhand von Domain-Namen in umgekehrter Reihenfolge verwenden. Wird kein "package" angegeben wird das "Default"-Package vorgegeben.

> **Package-Struktur: Aktuell leider nicht nutzbar!**
>
> In der aktuellen Version können leider keine Packages genutzt werden!

Anbei ein paar Empfehlungen für Package-Namen:

com.dvelop.scripts

Groovy-Skripte welche im Kontext eines Server-Interfaces bzw. des Prozessmanagers aufgerufen und damit im Verzeichnis "ext_groovy" abgelegt werden. Die resultierende Verzeichnisstruktur wäre dann "ext_groovy\com\dvelop\scripts".

com.dvelop.hooks

Die Groovy-Hook-Funktionen, welche die einzelnen Hook-Eintrittspunkte bedienen, sollten hier ebenfalls in einem eigenen Package definiert werden. Das resultierende Verzeichnis wäre dann zum Beispiel "d3server.prg\D3T\groovyHooks\com\dvelop\hooks".

com.dvelop.api

Werden mittels Groovy-Funktionen eigene API-Funktionen bereitgestellt, könnte dieses Package genutzt werden. Eine resultierende Verzeichnisstruktur könnte dann wie folgt aussehen "d3server.prg\D3T\groovyAPI\com\dvelop\api".

4.7 d.3-dynamische Rückmeldungen aus den Hook-Funktionen

Sollten aus einer Serveraktion dynamische Texte auch an die Clientseite übergebenen werden, zum Beispiel für dynamischer Fehlermeldungen, kann die Hook-Eigenschaft "additional_info_text" aus jedem Hook gesetzt werden.

```
d3.hook.setProperty("additional_info_text", "My message text for the Client!")
```

Wird der Hook von einem d.3 server-Prozess (nicht hostimp oder async) aufgerufen, wird dieser Text als zusätzlicher Exportparameter bei dem aktuellen API-Call an an den Client übergeben.

> **Hinweis**
>
> Dieser Text wird dem Benutzer nicht durchgängig bei allen Clients angezeigt.

4.8 Nummernkreis für Returnwerte

Die Returnwerte aus den Hook-Funktionen werden intern mit einem Offset-Wert verrechnet.

Der Wert, welcher auf der Client-Seite ausgewertet werden kann, ist dabei das Ergebnis aus der Berechnung "Offset-Wert – Bereichswert "!

Eintrittspunkt	Bereich	Offset	Ergebnis*	Beispiel (Offset - Eigener-Wert)
ImportDocument	-8000 -> -9999	10000	18000 ->19999	10000 - (-8000) = 18000
ImportNewVersionDocument	-8000 -> -9999	20000	28000 ->29999	20000 - (-8500) = 28500

DeleteDocument	-1900 -> -1999	4000	5900 -> 5999	4000 - (-1925) = 5925
[Alle anderen]	-1 -> -499	9500	9501 -> 9999	9500 - (-250) = 9750

* Offset-Wert – Bereichswert = Ergebnis

4.9 Nutzung des Transportsystems für Groovy-Funktionen

Mit dem Transportsystem können Einstellungen zwischen d.3-Repositorys übertragen werden. Auch Groovy-Hook-Module können damit transportiert werden.
Hierzu werden im Bearbeitungsmodus von d.3 admin Projekte definiert, die alle zu transportierenden Einstellungen klammern.

Um ein Hook-Modul einem Tranportprojekt zuzuordnen, wird der Projektename per Annotation @TransportProject in das Groovy-Hook-Modul eingetragen.
Die Annotation @TransportProject ist auf Klassenebene definiert und muss deshalb einer Java/Groovy Klassendefinition vorangestellt werden.

Als Parameter der Annotation können ein oder mehrere Projektnamen oder auch die GUID's der Projekte angegeben werden.

```
import com.dvelop.d3.server.TransportProject;

// OPTION 1: Project name
@TransportProject("myProject")
public class MyTestHooks {

} // end of MyTestHooks

// OPTION 2: Project-GUID
@TransportProject("6ACDA408-3638-4B70-8E0D-036CC9559F7E")
public class MyTestHooks {

} // end of MyTestHooks

// OPTION 3: or a combination of Project name and prodjct GUID
@TransportProject(["New installation", "931608C4-E075-4E89-AA35-66E6FD74770B"])
public class MyTestHooks {
  // ...
} // end of MyTestHooks
```

Folgende Regeln sind zu beachten, um Groovy-Hook-Module transportieren zu können:

- Der Dateiname der Module darf sich nicht mehr ändern, sobald erstmals ein Meilenstein geschlossen wurde, der zu einem der annotierten Projekte gehört
- Pro Modul sollte nur eine Klasse verwendet werden.
- Die Module werden in jeden Meilenstein der annotierten Projekte aufgenommen.

- Beim Import eines Meilensteines wird jedes Modul ausgetauscht, also überschrieben.
- Beim Import wird in den per Parameter HOOK_GROOVY_DIRS_CUSTOMER eingestellten Verzeichnissen nach dem Dateinamen der Module gesucht. Wird die Datei gefunden, wird diese überschrieben.
- Wenn keine Datei mit dem Namen gefunden wurde, wird die Datei in das erste Verzeichnis aus HOOK_GROOVY_DIRS_CUSTOMER kopiert.
- Wenn kein Verzeichnis über HOOK_GROOVY_DIRS_CUSTOMER konfiguriert ist, wird in dem d.3-Konfigurationsverzeichnis (Speicherort der **d3config.ini**) ein Unterordner **groovy_hooks** erstellt und die Datei dort abgelegt.
- Wird die Modul-Datei erstmalig im Ziel-Repository abgelegt, müssen die d.3-Prozesse neu gestartet werden.
- Wird die Modul-Datei ausgetauscht, existiert vorher also schon, hängt es vom Parameter HOOK_GROOVY_RELOAD_ON_CHANGE ab, ob das Modul direkt aktiviert wird.
- Wenn ein Groovy-Modul im Zielsystem gelöscht werden soll, muss die Annotation des Moduls im Quellsystem entfernt werden und die Datei im Zielsystem gelöscht werden.

5 Groovy API-Funktionen

d.3 server verfügt ab Version 8 über eine PlugIn-Schnittstelle für API-Funktionen.

Damit können eigene, in Java/Groovy entwickelte API-Funktionen registriert werden.

Diese können dann genauso wie jede andere d.3-API-Funktion über das d.3-Kommunikations-Protokoll d3fc aufgerufen werden.

> **Wichtig**
>
> Groovy API-Funktionen stehen über die d.3 web webservice-API-Schnittstelle **nicht** zur Verfügung Sie können von d.ecs forms über das dortige Skripting genutzt werden, sowie für d.velop-eigene Projekte.

Aktivieren der PlugIn-Schnittstelle

1. Öffnen Sie **d.3 admin > Systemeinstellungen > d.3 config**.
2. Geben Sie darin im Abschnitt **Java/Groovy** für den Eintrag **Java/Groovy API Funktionen** ein Verzeichnis an.

Groovy-Skripte, die d.3-API-Funktionen implementieren, werden dann in diesem Verzeichnis gesucht und daraus geladen.
Dadurch wird die PlugIn-Schnittstelle für API-Funktionen aktiviert.

Damit eine Java-Klasse als d.3 API-Funktion geladen und registriert wird, muss diese von der Klasse D3ApiCall abgeleitet sein und eine Methode public int execute(D3Interface d3) implementieren.

Darüber steht dann das D3Interface zur Verfügung.

```
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.D3ApiCall;
import com.dvelop.d3.server.User;

public class GetMyDBData extends D3ApiCall
{
   public int execute(D3Interface d3)
   {
      def import_param = d3.remote.getImportParams()
      def id_value  = import_param.get("id")
      def resultset = d3.sql.executeAndGet("SELECT column1, column2 FROM mytable WHERE id like ?",
[id_value])

      d3.remote.setExportTable( resultset )
      d3.remote.setExportParams(["number" : resultset.size()])

      return 0
   }
}
```

5.1 Groovy-API und Nutzung in JPL

Mit der Integration von Groovy als Server-Skriptsprache steht nun auch die Möglichkeit zur Verfügung eigene API-Funktionen zu erstellen und zu nutzen.

Ein erstes Beispiel mit einen Aufruf aus JPL wird im Anschluss vorgestellt.

> **Hinweis**
>
> Szenario:
>
> Beispielhaft wird hier eine Funktion dargestellt, die mittels SQL Werte aus einer Tabelle in der d.3-Datenbank liest und diese zurückliefert.

Groovy-Beispiel SQL-Abfrage als Serverfunktion

```groovy
package com.dvelop.api;
// (1)
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.D3ApiCall;
import com.dvelop.d3.server.User;

// (2)
public class GetMyCustomerData extends D3ApiCall
{
// (3)
  public int execute( D3Interface d3 ){
    def importParams = d3.remote.getImportParams();
    def idValue     = importParams.get("id");
    def resultSet    = d3.sql.executeAndGet( """SELECT name, customerNo, zipCode, city, street
                                  FROM CustomerData
                                 WHERE customerNo = ?""", [idValue] );
// (4)
    d3.remote.setExportTable( resultset );
// (5)
    d3.remote.setExportParams(["number" : resultSet.size() ]);
    return 0;
  }
} // end of GetMyCustomerData
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann. Hier ist aber wichtig dass die Erweiterung **extendsD3ApiCall** hinzugefügt wird.
3. Die Funktion, welche nun als API-Call bereitgestellt werden soll, wird nun definiert. Dabei können über den Befehl **getImportParams** auch Parameter, welche über den Aufruf der Funktion bereitgestellt werden, ausgelesen werden.
4. Mittels der Funktion **setExportTable** stehen dann auch die Ergebnisse aus der Funktion dem Aufrufenden Programm zur Verfügung.

Ein d3FC-Aufruf mittels JPL implementiert

```
// (1)
vars lReturnValue

// (2)
call api_function("d3fc_user_set",        "dvelop")
call api_function("d3fc_password_set",    "dvelop")
call api_function("d3fc_remote_server_set", "127.0.0.1")
call api_function("d3fc_port_set",        "3400")
call api_function("d3fc_timeout_set",     "60")
call api_function("d3fc_server_set",      "B")

// (3)
// ACHTUNG ERST FUNKTION DANN PARAMETER
call api_function("d3fc_function_name_set",  "GetMyCustomerData")
call api_function("d3fc_importing_set",     "id" , "<CustomerNo>")
call api_function("d3fc_exporting_set",     "number")
call api_function("d3fc_table_set_headline", "name customerNo zipCode city street .","O")

// (4)
lReturnValue = api_function("d3fc_execute")

// (5)
call api_function("d3fc_exporting_get" ,"number")
vars lTableRowCount = api_single_info


// (6)
call api_log_error("SIZE :lTableRowCount  ")
call api_log_error(" :lReturnValue ")


// (7)
vars lRet, lName, lKdnr
lRet = api_function( "d3fc_first" )

// (8)
while( lRet != EOT)
{
 call api_function("d3fc_field_get","name")
 lName = api_single_info

 call api_function("d3fc_field_get","customerNo")
 lKdnr = api_single_info

 call api_log_error(":lName :lKdnr")
 lRet = api_function("d3fc_next")
}
```

Kommentare zu den einzelnen Blöcken

1. Zur Aufnahme des Rückgabewertes wird eine lokale Variable angelegt.
2. Im nächsten Schritt müssen die Login-Parameter für den d3FC-Aufruf vorgegebenen werden.
3. Um nun die Funktion gemäß der eigenen Definition aufrufen zu können, muss der Funktionsname festgelegt und die Parameter übergeben werden.

4. Sind alle notwendigen Einstellungen vorgenommen kann der eigene API-Befehl nun mittels "d3fc_execute" ausgeführt werden!
5. Die bereitgestellten Rückgabewerte können nun ausgelesen und in lokale Variablen übernommen werden.
6. Zur Dokumentation der Funktion bzw. der Ergebnisse werden diese hier als Error-Message im Log-File ausgegeben.
7. Im nächsten Schritt werden nun die einzelnen Kundendaten ausgelesen und ebenfalls im Log-File ausgegeben.

6 Groovy-Skripte

Über d.3 server interface können neben den externen JPL-Skripten nun auch Groovy-Skripte ausgeführt werden.

In einer Groovy-Skriptdatei steht die d.3-Schnittstelle als Field-Variable **d3** zur Verfügung und kann direkt genutzt werden.

```
d3.log.info("Groovy-Script started!")
```

Um in einer Entwicklungsumgebung den Typ des vordefinierten Fields **d3** bekannt zu geben, damit Typprüfungen, Kommandovervollständigung etc. funktionieren können, sollten die folgenden beiden Zeilen am Anfang eines Skripts eingefügt werden.

```
import com.dvelop.d3.server.core.D3Interface
D3Interface d3 = getProperty("d3")

d3.log.info("Groovy-Script started !")
```

Verwenden von Groovy-Klassen und Java-Bibliotheken in Skripten

Das Groovy-Skript Verzeichnis "ext_groovy" sowie die definierten Groovy-Hook-Verzeichnisse werden bei der Ausführung eines Skripts dem CLASSPATH hinzugefügt, sodass Klassen aus anderen Groovy-Skripten im auszuführenden Skript genutzt werden können. Außerdem werden auch für Skripte Classpath-Dateien (Dateiname: "<skriptname>.classpath") unterstützt. Auch die darin enthaltenden Pfade (zB. absoluter Pfad einer JAR-Datei) werden dem Classpath hinzugefügt, um diese Resourcen in dem Skript nutzen zu können.

Beispiel: In einem Skript ../ext_groovy/myScript.groovy soll die JavaMail API (**javax.mail.jar**) benutzt werden. Dazu wird der absolute Pfad der JAR-Datei in eine gleichnamige Classpath-Datei ../ext_groovy/myScript.classpath eingetragen:

Beispielhafter Dateinhalt: D:\downloads\java\ext_jars\javax.mail-1.5.6.jar

Skripte zeitgesteuert starten

Soll ein Skript nicht interaktiv gestartet werden, sondern automatisch und zeitgesteuert, kann dieses auch als sechster Kommandozeilenparameter eines Server-Prozesses in d.3 process manager angegeben werden. Der Target-Eintrag in d.3 process manager sieht dann ähnlich aus wie folgender:

..\d3odbc32.exe haupt "" Master password D3P ext_groovy/myScript.groovy

7 d.3-Schnittstelle (D3Interface)

```
package com.dvelop.d3.server.core;

public interface D3Interface
{
   public interface ArchiveInterface       // d.3 Archiv
   public interface SqlD3Interface         // d.3 SQL Datenbank
   public interface D3RemoteInterface      // Client API
   public interface ScriptCallInterface    // Server API
   public interface ConfigInterface        // Config-Parameter
   public interface LogInterface           // Logging
   public interface HookInterface          // Hook-Eigenschaften
   public interface StorageManagerInterface  // Storagemanager

   public ArchiveInterface       getArchive();
   public SqlD3Interface         getSql();
   public D3RemoteInterface      getRemote();
   public ScriptCallInterface    getCall();
   public ConfigInterface        getconfig();
   public LogInterface           getlog();
   public HookInterface          getHook();
   public StorageManagerInterface  getStorageManager();
 }
```

Hinweis

Das D3Interface wird vom Server bei allen Aufrufen registrierter Groovy-Funktionen als erster Parameter übergeben.
Dadurch steht die d.3-Schnittstelle in allen Hook-, API- und Skript-Funktionen zur Verfügung.

7.1 d.3 Archiv (ArchiveInterface)

ArchiveInterface

```
public interface ArchiveInterface {
    public Document          getDocument(String id, String contextUser);
    public Document          getDocument(String id);
    public DocumentType      getDocumentType(String id);
    public PredefinedValueSet  getPredefinedValueSet(String id);
    public RepositoryField    getRepositoryField(String id);
    public User              getUser(String id);
    public UserGroup         getUserGroup(String id);
    public UserOrUserGroup    getUserOrUserGroup(String id);
    public AuthorizationProfile getAuthorizationProfile(String id);
    public void              removeTranslationFromCache(String entryPoint, Locale lang);
    public Document          newDocument();
    public Document          importDocument(Document doc, Path importFilePath);
    public Document          importDocument(Document doc);
}
```

Das ArchiveInterface liefert verschiedene Java-Objekte, über die direkt auf die entsprechenden d.3-Archiv-Objekte zugegriffen werden kann.

```
import com.dvelop.d3.server.core.D3Interface
import com.dvelop.d3.server.Document
import com.dvelop.d3.server.exceptions.D3Exception
import java.nio.file.Path
import java.nio.file.Paths

D3Interface d3 = getProperty("d3");

// Create an new empty document
Document newDoc = d3.archive.newDocument();

// Add system properties
newDoc.type = "DA1";                          // ID of target document
newDoc.status = Document.DocStatus.DOC_STAT_RELEASE;     // Target state of document
newDoc.editor = "dvelop";                      // Handler
newDoc.setText(1, "Import per Groovy Skript");          // Comment
// erweiterte Eigenschaften zuweisen
newDoc.field[1] = "Attribute value 1 for new document";
newDoc.field[2] = "Attribute value 2 for new document";
newDoc.field[60][1] = "Multi value field 60-1 for new document";
newDoc.field[60][2] = "Multi value field 60-2 for new document";

// Define file for new document
Path importFile = Paths.get("D:\\temp\\my_file.txt");
try {
  // Import the document
  newDoc = d3.archive.importDocument(newDoc, importFile);
}
catch (D3Exception e) {
  println e.message;
  return;
}
println "Doc-ID of newly created document: " + newDoc.id;
```

7.1.1 Archivobjekte (ArchiveObject)

ArchiveInterface

```
public interface ArchiveObject
{
   public String getId();
}
```

Alle d.3-Archivobjekte sind von dieser Klasse abgeleitet. Damit kann in allen Archivobjekten die d.3-ID des Objekts ermittelt werden.

```
Document doc;
doc.id      // Document ID

DocumentType docType;
docType.id    // Document type

User user;
user.id      // User ID
```

7.1.2 Dokument (Document)

Document

```
class Document extends ArchiveObject
{
  public Field         getField()     // Groovy Collection Support for reading and writing the property values
  public DocumentType  getType()
  public void          setType(DocumentType type)
  public void          setType(String typeId)
  public String        getNumber()
  public void          setNumber(String docNumber)
  public DocStatus     getStatus()
  public void          setStatus(DocStatus docStatus)
  public void          setStatus(String docStatus)
  public int           getVarnumber()
  public void          setVarnumber(int varNumber)
  public UserOrUserGroup getEditor()
  public void          setEditor(UserOrUserGroup editor)
  public void          setEditor(String editor)
  public String        getOwner()
  public String        getFilename()
  public String        getFileExtension()
  public Long          getDocSize()
  public String        getText(int lineIdx)
  public void          setText(int lineIdx, String text)
  public Timestamp     getCreated()
  public Timestamp     getLastAccess()
  public Timestamp     getLastUpdateFile()
  public Timestamp     getLastUpdateAttribute()
  public Timestamp     getLastUpdate()
  public Integer       getSignaturesRequired()
  public Integer       getColorCode()
  public void          setColorCode(Integer colorCode)
  public String        getReleaseVersionStatus()
  public boolean       getIsVerified()
  public boolean       getIsWebPublished()
  public boolean       getIsInWorkflow()
  public int           getNumericId()
  public boolean       getIsArchived()
  public Integer       getLastAlterationNumber()
  public Integer       getAlterationNumberReleased()
  public Integer       getCodepage()
  public Integer       getMaxArchiveIndex()
  public String        getCaption()
  public boolean       getHasMultData()

  public void          updateAttributes(String userId)
  public void          updateAttributes(String userId, boolean noHooks)
  public int           changeType (String docTypeId, String userId)
  public String        getPermission (String userId)
  public int           block (boolean block, String userId)
  public int           verify (String userId)
```

```
  public int          transfer (String destination, String newEditor, String changeRemark, boolean asynchronous,
int archivIndex, String userId)
  public int          publishForWeb (boolean publish, String userId)
  public int          addDependent (String filename, String docStatus, String docExt, String userId)
  public int          addDependent (String filename, DocStatus docStatus, String docExt, String userId)
  public int          deleteDependent (char docStatus, String docExt, String userId)
  public int          deleteDependent (String docStatus, String docExt, String userId)
  public int          deleteDependent (String docStatus, String docExt, String userId)
  public int          startLifetime (boolean overwriteOldDate, int lifeTimeDays)
  public int          setCacheDays (char docStatus, int archiveIndex, int daysInCache)
  public int          checkFolderScheme (String userId);

  public String       getFileFormat()
  public String       getFileFormatPublic()
  public void         setFileFormat(String fileFormat)

  public DocumentVersion[] getVersions()
  public PhysicalVersion[] getPhysicalVersions()
  public Integer        getFileIdCurrentVersion()
  public Integer        getFileIdRelease()
  public Timestamp      getEndOfRetentionDate()
  public DocumentNote[]   getNotes()

  public SysFields          getSysField()    // Groovy Collection Support für das Lesen und Schreiben von
Systemeigenschaften
  public void               addSysField(String fieldName)
  public List<DocumentSysValue> getSysValues()
  public Set<String>        getSysFieldNames()
 }
```

*) Der Collection Support für Fields ermöglicht die Nutzung des Subscript Operators [], für den Zugriff auf die erweiterten Eigenschaften eines Dokumentes.

```
Document doc
doc.field[1] = "Value of property "   // Writing value
println doc.field[1]              // Reading value

// Accessing with property name
doc.field["Name"] = "Meier"          // Writing value
println "Name = " + doc.field["Name"]  // Reading value
```

Hinweis

Wenn ein Feld nicht existiert oder ein Feld keinen Wert besitzt, so wird NULL zurückgegeben.

Nutzung der Schnittstelle für die Systemeigenschaften:

```
println doc.sysField["InvoiceNo"][1]                    // Reading value
doc.sysField["InvoiceNo"][1] = "Value of system property"      // Add or change value / add new system property
with value
doc.sysField["InvoiceNo"].add("Wert")                    // Another way to do this
doc.sysField["InvoiceNo"].remove(2)                     // Delete value

doc.addSysField("New system field")                     // Add new system property without value
doc.sysField["InvoiceNo"].clear()                     // Delete all values for this property

println doc.sysFieldNames                          // Go through all system properties
println doc.sysField["InvoiceNo"].sysValues                 // Go though all values of one system property
println doc.sysValues                             // Go through all values of all system properties
```

7.1.2.1 Dokumentversionen (DocumentVersion)

DocumentVersion

```
public class DocumentVersion extends ArchiveObject
{
   public enum Category {
       DOC_VERS_REGULAR,
       DOC_VERS_OVERWRITTEN,
       DOC_VERS_BLOCKED,
       DOC_VERS_REPLACED,
       DOC_VERS_DELETED,
       DOC_VERS_ERASED,
       DOC_VERS_MIGRATED,
       DOC_VERS_BOOKED,
       INVALID_CATEGORY
     };

   public String         getDocId()
   public boolean        isCurrentVersion()
   public boolean        hasStatus()
   public DocStatus      getStatus()
   public boolean        hasHistStatus()
   public DocStatus      getHistStatus()
   public boolean        hasFileId()
   public Integer        getFileId()
   public boolean        hasHistFileId()
   public Integer        getHistFileId()
   public Category       getCategory()
   public String         getChangeReason()
   public String         getDeleteReason()
   public PhysicalVersion  getPhysicalVersion()
   public String         getCreator()
   public Timestamp      getCreateDate()
   public String         getReleaseUser()
   public Timestamp      getReleaseDate()
   public String         getBlockUser()
   public Timestamp      getBlockDate()
   public String         getArchiveUser()
   public Timestamp      getArchiveDate()
   public String         getVerifier()
   public Timestamp      getVerifyDate()
   public String         getDeleter()
   public Timestamp      getDeleteDate()
   public Double         getExternalVersionId()
}
```

Beispiel für den Zugriff auf die Versionen eines Dokuments.

```
import com.dvelop.d3.server.Document
import com.dvelop.d3.server.DocumentVersion
import com.dvelop.d3.server.PhysicalVersion
import com.dvelop.d3.server.DependentFile

 Document doc
 // Go / interate through all versions
 for (version in doc.versions)
 {
   // Get the properties of the current Version
   println version.archiveDate
   println version.creator
   println version.status
   // ..

   // When the current Version has a physical file attached, get access
   if (version.physicalVersion)
   {
     // Get the properties of the file
     println version.physicalVersion.fileSize
     println version.physicalVersion.fileLocalisation
     println version.physicalVersion.fileFormat
     // ..

     // Get all properties of depending files
     for (dependentFile in version.physicalVersion.dependentFiles)
     {
       // Get the properties of each depending file
       println dependentFile.fileFormat
       println dependentFile.fileSize
       println dependentFile.fileId
       // ..
     }
   }
 }
```

7.1.2.2 Dateiversionen (PhysicalVersion)

PhysicalVersion

```
public final class PhysicalVersion extends ArchiveObject
{
   enum FileLocalisation
        FILE_LOC_NO_FILE,
        FILE_LOC_DISK_ONLY,
        FILE_LOC_STORAGE_ONLY,
        FILE_LOC_DISK_AND_STORAGE,
        FILE_LOC_PROXY_PLACEHOLDER,
        FILE_LOC_PROXY_PENDING
      };
   public Integer          getFileId()
   public String           getDocId()
   public Document.DocStatus getStatus()
   public String           getFileExtension()
   public String           getFileFormat()
   public Long             getFileSize()
   public FileLocalisation   getFileLocalisation()
   public String           getD3Hash()
   public String           getFileHash()
   public SignatureInfo[]    getSignatureInfos()
   public DependentFile[]    getDependentFiles()
}
```

7.1.2.2.1 abhängige Dateien (DependentFile)

DependentFile

```
public class DependentFile extends ArchiveObject
{
   public String         getExtension()
   public String         getDocId()
   public Integer         getFileId()
   public String         getFileFormat()
   public FileLocalisation getFileLocalisation()
   public Long           getFileSize()
   public Integer         getFileSizeInKb()
   public String         getD3Hash()
   public Timestamp       getProcDate()
   public String         getExternalMedium()
   public String         getExpired()
   public String         getFlagStorageExport()
   public Timestamp       getStorageExportDate()
   public String         getD3FileHash()
 }
```

7.1.2.2.2 Signaturen (SignatureInfo)

SignatureInfo

```
public class SignatureInfo  extends ArchiveObject
{
   enum SigContentType
       SIG_DETACHED, SIG_EMBEDDED, SIG_ATTACHED, SIG_EMBEDDED_AND_DETACHED,
       SIG_INVALID_CONTENT_TYPE;
     };
   public String       getExtension()
   public SigContentType getContent()
   public Integer       getReachedNumber()
   public Integer       getRequestedNumber()
   public Integer       getReachedLevel()
   public Integer       getRequestedLevel()
   public String       getTodo()
   public String       getDone()
   public String       getRemark()
}
```

7.1.2.3 Systemeigenschaften (DocumentSysValue)

DocumentSysValue

```
public class DocumentSysValue extends ArchiveObject
{
   public String       getDocId()
   public Integer      getFieldId()
   public String       getFieldName()
   public Integer      getFieldIndex()
   public void         setFieldIndex(int fieldIndex)
   public String       getFieldValue()
   public void         setFieldValue(String value)
   public String       toString()
}
```

7.1.2.4 Notizen (DocumentNote)

DocumentVersion

```
public class DocumentNote extends ArchiveObject
{
   public String    getMessage()
   public User      getUser()
   public Timestamp  getDateCreated()
}
```

Skript-Beispiel für die Ausgabe der Notizen eines Dokuments.

```
import com.dvelop.d3.server.core.D3Interface
import com.dvelop.d3.server.Document
import com.dvelop.d3.server.User
import com.dvelop.d3.server.DocumentNote
import java.text.SimpleDateFormat

D3Interface d3 = getProperty("d3")

Document doc = d3.archive.getDocument("P000000123")

println "Notes form the document <" + doc.id + "> :"

doc.notes.each  { note ->
  String noteCreated = new SimpleDateFormat("dd.MM.yyyy HH:mm:ss").format(note.dateCreated);
  println "  User:" + note.user.realName + " Created: " + noteCreated + " Note: " + note.message
}
```

7.1.3 Dokumentart (DocumentType)

DocumentType

```
class DocumentType extends ArchiveObject
{
   public DocumentTypeAttribute getfield()   // Groovy Collection Support für den Zugriff auf die Attribute der
Dokumentart
   public String   getDefaultName()
   public String   getName()
   public String   getType()
   public boolean  getIsFolder()
   public boolean  getIsExportedToStorage()
   public boolean  getIsSystemDocType()
   public boolean  getIsDummyType()
   public boolean  getIsMultiType()
   public boolean  getIsTemplateType()
   public String   getArchiveId()
   public String   getDsearchCorpus()
   public int      getFieldNoByName(String attribName)
}
```

7.1.3.1 Eigenschaften einer Dokumentart (DocumentTypeAttribute)

DocumentTypeAttribute

```
public final class DocumentTypeAttribute extends ArchiveObject
{
  public RepositoryField getRepositoryField()
  public RepositoryField getRepoField()
  public boolean      getIsMandatory()
  public boolean      getIsModifiable()
  public boolean      getIsHidden()
  public boolean      getViewInSearchMask()
  public boolean      getViewInImportMask()
  public boolean      getViewInResultSet()
}
```

7.1.4 Benutzer (User)

User

```
public final class User extends ArchiveObject
{
  public enum MemberType {DIRECT, RECURSIVE};

  public String           getLongName()
  public String           getRealName()
  public String           getEmail()
  public String           getPhone()
  public String           getPlant()
  public String           getDepartment()
  public boolean            getIsCheckedOut()
  public boolean            getIsSysUser()
  public String           getCheckoutText()
  public String           getOptField(int idx)
  public String           getLdapDN()
  public UserGroup[]        getGroups()
  public boolean            isMemberOfGroup(String groupId)    // check of direct member (MemberType.DIRECT)
  public boolean            isMemberOfGroup(String groupId, MemberType memberType)
  public AuthorizationProfile[] getAuthorizationProfiles()
  public boolean            hasAuthorizationProfile(String profileId)
  public DocumentType[]       getDocTypes()
 }
```

Beispiel für die Nutzung in einem Groovy-Skript

```groovy
import com.dvelop.d3.server.core.D3Interface
import com.dvelop.d3.server.User
import com.dvelop.d3.server.UserGroup
import com.dvelop.d3.server.AuthorizationProfile

// Get a user object
def user = d3.archive.getUser("dvelop")

// Reading some user properties
println "Die EMail-Adresse des Benutzers " + user.realName + " (" + user.id + ") lautet: " + user.email

// Get all groups, with the current user as a member
user.getGroups().each { group ->
  println group.id + " - " + group.name
}

// Get all right profiles, with the current user as a member
user.getAuthorizationProfiles().each { profile ->
  println profile.id + " - " + profile.name
}

// Get all document types, to which the user has access to
user.getDocTypes().each { docType ->
  println docType.id + " - " + docType.name
}
```

7.1.5 Benutzergruppen (UserGroup/UserOrUserGroup)

User / UserOrUserGroup

```
public final class UserOrUserGroup extends ArchiveObject
{
   public String     getDefaultName()
   public String     getLongName()
   public String     getDisplayName()
   public User       getUser()
   public UserGroup   getUserGroup()
}


public final class UserGroup extends ArchiveObject
{
   public enum MemberType {DIRECT, RECURSIVE};

   public String              getDefaultName()
   public String              getName()
   public UserOrUserGroup[]     getMembers()
   public UserOrUserGroup[]     getMembers(MemberType memberType)
   public boolean             isMemberOfGroup (String parentGroupId)
   public boolean             isMemberOfGroup (String parentGroupId, MemberType memberType)
   public AuthorizationProfile[] getAuthorizationProfiles()
   public boolean             hasAuthorizationProfile(String profileId)
}
```

```
import com.dvelop.d3.server.core.D3Interface
import com.dvelop.d3.server.UserGroup
import com.dvelop.d3.server.UserOrUserGroup
import com.dvelop.d3.server.AuthorizationProfile

// Get user object
def userGroup = d3.archive.getUserGroup("GroupId");

println "Show group information for <" + userGroup.name + "> ";

// Get all users and groupy, which are members of the current Group
userGroup.getMembers(UserGroup.MemberType.RECURSIVE).each { userOrGroup ->
  println userOrGroup.id + " - " + userOrGroup.defaultName
}

// Get all right profiles with the current group
userGroup.getAuthorizationProfiles().each { profile ->
  println profile.id + " - " + profile.name
}
```

7.1.6 Wertemengen (PredefinedValueSet)

```
public final class PredefinedValueSet extends ArchiveObject
{
   public String  getName()
   public String  getDataType()
   public String  getSortFlag()
   public String  getValues(int valIdx)
}
```

7.1.7 Eigenschaftsfelder (RepositoryField)

RepositoryField

```
public final class RepositoryField extends ArchiveObject
{
   public String          getName()
   public String          getText()
   public String          getDataType()
   public boolean           getHasPredefinedValues()
   public Integer         getPreferedFieldNumber()
   public PredefinedValueSet getPredefinedValueSet()
   public boolean           getHasPlausibilityHookFunction()
   public boolean           getHasValuesProvidingHookFunction()
   public void            provideValuesForValueSet(List<Object> values)
}
```

7.1.8 Berechtigungsprofil (AuthorizationProfile)

AuthorizationProfile

```
public final class AuthorizationProfile extends ArchiveObject
{
   public String getName()
}
```

7.2 d.3 SQL Datenbank (SqlD3Interface)

SqlD3Interface

```
   public interface SqlD3Interface {
      public int execute(String query, List<Object> params) throws Exception;
      public int execute(String query) throws Exception;
      public GroovyRowResult firstRow(String sqlquery, List<Object> params) throws Exception;
      public GroovyRowResult firstRow(String query) throws Exception;
      public List<GroovyRowResult> executeAndGet(String query, List<Object> params) throws Exception;
      public List<GroovyRowResult> executeAndGet(String query, List<Object> params, int maxRows) throws
 Exception;
      public List<GroovyRowResult> executeAndGet(String query, List<Object> params, int offset, int maxRows)
throws Exception;
      public List<GroovyRowResult> executeAndGet(String query) throws Exception;
      public List<GroovyRowResult> executeAndGet(String query, int maxRows) throws Exception;
      public List<GroovyRowResult> executeAndGet(String query, int offset, int maxRows) throws Exception;
   }
```

> **Hinweis**
>
> Die d.3-SQL-Schnittstelle bietet einen einfachen, groovy-konformen Zugriff auf die native Datenbank-Schnittstelle des d.3-Servers.
>
> Es ist **kein** JDBC Treiber erforderlich.

Modifizierer und Typ	Methode und Beschreibung
int	execute(String query, List<Object> params) Ausführen eines SQL-Kommandos mit Bind-Variablen (z.B. ein `insert` oder `update` Kommando)
int	execute(String query) Ausführen eines SQL-Kommandos ohne Bind-Variablen (z.B. ein `insert` oder `update` Kommando)
GroovyRowResult	firstRow(String sqlquery, List<Object> params) Ausführen eines SQL-SELECT-Kommandos mit Bind-Variablen. Nur die erste Zeile der Ergebnismenge wird abgerufen und zurückgeliefert.
GroovyRowResult	firstRow(String query) Ausführen eines SQL-SELECT-Kommandos ohne Bind-Variablen. Nur die erste Zeile der Ergebnismenge wird abgerufen und zurückgeliefert.
List<GroovyRowResult>	executeAndGet(String query, List<Object> params) Ausführen eines SQL-SELECT-Kommandos mit Bind-Variablen. Alle Zeilen der Ergebnismenge werden abgerufen und zurückgeliefert.
List<GroovyRowResult>	executeAndGet(String query, List<Object> params, int maxRows) Ausführen eines SQL-SELECT-Kommandos mit Bind-Variablen. Die ersten `maxRows` Zeilen der Ergebnismenge werden abgerufen und zurückgeliefert.

Modifizierer und Typ	Methode und Beschreibung
List<GroovyRowResult>	executeAndGet(String query, List<Object> params, int offset, int maxRows) Ausführen eines SQL-SELECT-Kommandos mit Bind-Variablen. Beginnend mit `offset` werden `maxRows` Zeilen aus der Ergebnismenge abgerufen und zurückgeliefert.
List<GroovyRowResult>	executeAndGet(String query) Ausführen eines SQL-SELECT-Kommandos ohne Bind-Variablen. Alle Zeilen der Ergebnismenge werden abgerufen und zurückgeliefert.
List<GroovyRowResult>	executeAndGet(String query, int maxRows) Ausführen eines SQL-SELECT-Kommandos ohne Bind-Variablen. Die ersten `maxRows` Zeilen der Ergebnismenge werden abgerufen und zurückgeliefert.
List<GroovyRowResult>	executeAndGet(String query, int offset, int maxRows) Ausführen eines SQL-SELECT-Kommandos ohne Bind-Variablen. Beginnend mit `offset` werden `maxRows` Zeilen aus der Ergebnismenge abgerufen und zurückgeliefert.

7.3 Client API (D3RemoteInterface)

D3RemoteInterface

```
public interface D3RemoteInterface {
  // Parameter
  public Map<String, Object>       getImportParams();
  public void                      setExportParams(Map<String, Object> exportParams);

  // Table
  public Iterable<Map<String,Object>> getImportTable(String[] columnNames);
  public void                      setExportTable(Iterable<Map<String,Object>> exportTable);
  // File (table binary)
  public ByteBuffer                getImportBytes();
  public void                      setExportBytes(ByteBuffer exportBytes);

  public void                      setReturnCode(int retCode);

  // d3fc header information
  public String                    getLanguage();
  public String                    getFunctionName();
  public String                    getUserName();
  public String                    getVersion();
  public String                    getServerId();
  public String                    getSourceIpAddress();
}
```

Das Interface „D3RemoteInterface“ kann für die Implementierung eigener d3fc API-Funktionen per Groovy genutzt werden (siehe Kapitel PlugIn Schnittstelle für API Funktionen).

Außerdem kann auf die Kontext-Informationen eines d3fc-Funktionsaufrufs (d3fc Header) zugegriffen werden. Das heißt wenn eine Hook-Funktion im Kontext einer d.3 API-Funktion ausgeführt wird, dann kann auf Informationen des API-Aufrufs zugegriffen werden.

Modifizierer und Typ	Methode und Beschreibung
Map<String, Object>	getImportParams() Entgegennehmen der Liste der Importparameter des Aufrufers. Key der Map = Name des Parameters Value der Map = Wert des Parameters
void	setExportParams(Map<String, Object> exportParams) Die Liste mit den Rückgabewerten füllen. Schlüssel in der Map = Name des Parameters Wert in der Map = Wert des Parameters

Modifizierer und Typ	Methode und Beschreibung
Iterable<Map<String,Object>>	getImportTable(String[] columnNames) Entgegennehmen aller Werte aus der d3fc-Importtabelle der Spaltennamen, die per columnNames Liste angegeben wurden. Der Spaltenname ist jeweils der Schlüssel in der zurückgelieferten Map.
void	setExportTable(Iterable<Map<String,Object>> exportTable) Senden der übergebenen Liste von Maps als d3fc-Exporttabelle. Der Spaltenname ist jeweils der Schlüssel in der Map.
ByteBuffer	getImportBytes() Entgegennehmen eines binären Objekts vom Aufrufer. (zB. einer Datei)
void	setExportBytes(ByteBuffer exportBytes) Zurückgeben eines binären Objekts
void	setReturnCode(int retCode) Den Returnwert der API-Funktion festlegen. Wird die Methode nicht aufgerufen, so ist der Defaultwert "0" (Erfolg) zurückgeliefert.
`String`	`getLanguage()` Sprachkürzel, mit dem die aktuelle API-Funktion aufgerufen wurde
`String`	`getFunctionName()` der Funktionsname der aktuellen API-Funktion
`String`	`getUserName()` d.3-Benutzer, der die aktuelle API-Funktion aufgerufen hat
`String`	`getVersion()` Client-Versionsinformation des aktuellen API-Aufrufs
`String`	`getServerId()` Kürzel des d.3ecm Repositorys des aktuellen API-Aufrufs (z.B. 'P')

Modifizierer und Typ	Methode und Beschreibung
`String`	`getSourceIpAddress()` Client-IP-Adresse des Aufrufers der aktuellen API-Funktion

```
import com.dvelop.d3.server.Document
import com.dvelop.d3.server.DocumentType
import com.dvelop.d3.server.Entrypoint
import com.dvelop.d3.server.User
import com.dvelop.d3.server.core.D3Interface

public class Test{
  @Entrypoint( entrypoint = "hook_insert_entry_10" ) //----------------------
  public int showAppId( D3Interface d3, User user, DocumentType docTypeShort, Document doc ){

    d3.log.error( "---------->>>>>>>" + d3.remote.getVersion());
    def appID = d3.remote.getVersion()[0..2];
    d3.log.error( "APP-ID---------->>>>>>>" + appID);
    return 0;
  } // end of showAppId
}// end of Test
```

Hinweis

Das Aufrufen von d3fc Calls aus Groovy-Hook-Funktion ist aktuell nicht vorgesehen. Hier sollten lokale Methoden Aufrufe über das d.3-Interface vorgezogen werden.

7.4 Server API Funktionen (ScriptCallInterface)

ScriptCallInterface

```
public interface ScriptCallInterface {
    // Documents
    public int document_change_type ( String doc_type_short, String doc_id, String user_name);
    public int document_delete (String reason, boolean del_from_each_status, boolean del_file_always, String
doc_id, String user_name, boolean del_privileged);
    public String document_get_permission (String doc_id, String user_name);
    public int document_block (boolean block, String user_name, String doc_id);
    public int document_verify (String user_name, String doc_id);
    public int document_transfer (String destination, String new_editor, String change_remark, boolean
 asynchronous, int archiv_index, String user_name, String doc_id);
    public int document_publish_for_web (boolean publish, String doc_id, String user_name);
    // -- Notes
    public int note_add_file(String file_name, String doc_id, String user_id);
    public int note_add_string(String line, String doc_id, String user_id);
    //  Links
    public String[] link_get_parents (String doc_id, String user_name);
    public String[] link_get_children (String doc_id, String user_name);
    public int     link_documents (String doc_id_parent, String doc_id_child, String user_name, boolean
 folder_definition, boolean use_folder_plan);
    public boolean  link_exists (String doc_id_parent, String doc_id_child, boolean test_vice_versa);
    public int     link_delete (String doc_id_parent, String doc_id_child, String user_name);
    //  dependent files
    public int document_dependent_add (String filename, char doc_status, String doc_ext, String doc_id, String
user_name);
    public int document_dependent_delete (char doc_status, String doc_ext, String doc_id, String user_name);
    public int document_start_lifetime (String doc_id, boolean overwrite_old_date, int life_time_days);
    public int document_set_cache_days (String doc_id, char doc_status, int archive_index, int days_in_cache);
    public int folder_create (Document doc);
    public int document_register_dependent (String doc_id, int archive_index, char doc_status, String
user_group);
    public String document_get_file_path ( String doc_id, char doc_status, int archive_index, String user_group,
String dependent_ext);
    public int document_send_to_dsearch (String doc_id, String ocr_file, int version, String dsearch_corpus,
boolean use_existent_ocr_file,
            char doc_status, int archive_index, String user_name);
    public int restore_from_jukebox (char doc_status, int archiv_index, String doc_id, String user_name);
    public int restore_from_history (int aktion_id, String doc_id, String user_name);
    //  Right inheritance:
    public int add_inherit_doc_rights (String doc_id, String granter, String grantee, String right_flags, Timestamp
tstamp_expire);
    public int remove_inherit_doc_rights (String doc_id, String user_name, String grantee);
    // Inbox / resubmission
    public int hold_file_send ( String recipient, String notice, String doc_id, Timestamp tstamp_acknowledge,
Timestamp tstamp_remember,
            boolean expand_groups, boolean ignore_checkout, Timestamp date_activate, char type, String sender,
int chain_id, boolean remove_immediately,
            boolean inherit_class_rule, Timestamp inherit_class_tstamp, byte inherit_class_right, boolean
 check_write_access);
```

```
    public String[] hold_file_find ( String recipient, String doc_id, String user_name);
    public int     hold_file_delete ( long chain_id, Byte sent_received, boolean workflow_only, String recipient,
String doc_id, String user_name);
    // Workflow:
    public int workpath_end_document ( String doc_id, String user_name, boolean delete_jobs);
    public int workpath_start_document ( String wfl_id, String doc_id, String user_name);
    public int workpath_go_to_next_step (byte exit_value, String next_step_id, String doc_id, String
user_name);
    // Authorization profiles:
    public Long[]   roll_get ();
    public String[] roll_get_names ();
    public String[] roll_get_users (long roll_id, String roll_name);
    // TIFF / PDF functions
    public int document_render (String source, String destination, byte render_option, boolean ocr, boolean
 asynchronous,
            boolean replace_doc, boolean overwrite, String doc_id, String user_name, char doc_status, int
 archiv_index, String prio);
    public int tiff_concat (String source, String destination, String source_rdl, String destination_rdl);
    public int document_render_wfl_prot (String doc_id, String user_name);
    // Object properties
    public int object_property_set (String property_name, String object_id, byte object_class_id, String
property_value);
    // Lock token
    public int lock_token_acquire (String object_id, String object_name, String token, String object_info, int ttl,
String user_name);
    public int lock_token_release (String object_id, String object_name, String token);
    // Restriction sets:
    public int d3set_add_filter (String user_name, String doc_id, String set_name, String object_id, String filter,
boolean overwrite);
    public int d3set_remove_filter (String user_name, String doc_id, String set_name, String object_id, String
filter);
    public int d3set_remove_set (String user_name, String doc_id, String set_name, String object_id);
    // new from version 8.0:
    // Async jobs
    public int d3async_job_open(String doc_id_ref, String job_type, String user_name);
    public int d3async_job_set_attribute(String attr_name, String attr_value, int attr_type);
    public int d3async_job_set_lin002_attribute(String attr_type, String attr_name, String attr_value);
    public int d3async_job_close();
    // Various / Miscellaneous
    public int regular_expression_test (String regular_expression, String test_value, byte syntax_id, boolean
 case_sensitivity);
    public int send_email (String recipient, String notice, Timestamp date, String doc_id, String user_name,
String body_file, String mail_format, boolean attach,
            char doc_status, int archiv_index,    String attach_abh, String attach_file, boolean use_recip_array,
boolean use_cc_array, boolean use_bcc_array);
  }
```

Hinweis

Die Methoden des ScriptCallInterface entsprechen den Funktionen der d.3 server scripting API JPL (alte Bezeichnung vor Version 8: d.3 Server API).
In der Dokumentation sind alle diese Funktionen mit Parametern und Rückgabewerten beschrieben. Die dort beschriebenen globalen Variablen werden hier nicht unterstützt.

```
def callScriptFunction(D3Interface d3, Document doc, reposField, def user)
{
  def sqlQuery  = "SELECT note FROM notes_table WHERE doc_id = ?";
  def resultRows = d3.sql.executeAndGet(sqlQuery, [doc.id]);

  resultRows.each
  {
    d3.log.info("Add note $it.note to document $doc.caption ");
    d3.call.note_add_string(it.note, doc.id, user.id);
  }
}
```

Wichtig

Für den Aufruf von "folder_create" wird ein Dokument-Objekt benötigt. Wenn keines als Parameter zur Verfügung steht, so kann über die "Archive" Schnittstelle ein Dokument-Objekt für ein existierendes Dokument erzeugt werden.
Dieses wird als Vorlage benutzt und dessen Attribute werden wie gewünscht angepasst.

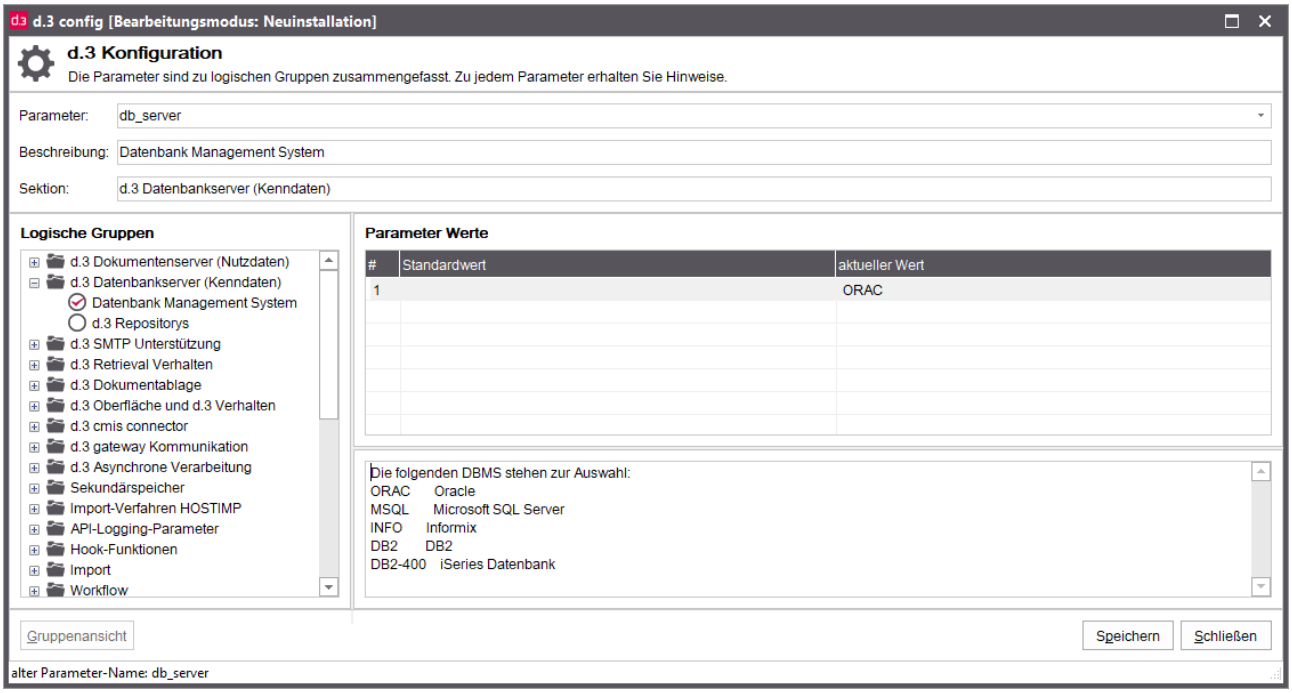
```
import com.dvelop.d3.server.core.D3Interface
import com.dvelop.d3.server.Document

D3Interface d3 = getProperty("d3")

def doc = d3.archive.getDocument("P000000001", "d3user") // Please change the values

doc.type = "APERS"
doc.status = Document.DocStatus.DOC_STAT_RELEASE
doc.editor = "d3user"
doc.setText(1, "Comment text row 1")

doc.field[1] = "folder_create - Attrib 1"
doc.field[2] = "folder_create - Attrib 2"
// ...
doc.field[60][1] = "folder_create - Attrib 60-1"
// ...

def error = d3.call.folder_create(doc)
if (error)
  println "$error within folder creation!"
else
  println "Folder creation successfull!"
```

7.5 Config-Parameter (ConfigInterface)

ConfigInterface

```
public interface ConfigInterface {
  public String value(String paramName);
  public String value(String paramName, Integer paramIndex);
}
```

Über das ConfigInterface können alle d.3 config Parameter abgefragt werden. Mögliche Parameternamen sind alle Parameter, die in d.3 config angezeigt werden.




```
D3Interface d3
// Getting the ID of the DBMS and writing it to the log file
def dbServer = d3.config.value("db_server");
d3.log.info( "Database: ${dbServer}");

// Get the first hostimport directory
def hostimpDir1 = d3.config.value("HOSTIMP_IMPORT_DIR", 1)
```

7.6 Logging (LogInterface)

LogInterface

```
public interface LogInterface extends GroovyLogInterface{
   public void critical(Object msg);
   public void error(Object msg);
   public void warn(Object msg);
   public void info(Object msg);
   public void debug(Object msg);
   public boolean isDebugEnabled();
   public void message(Object msg, int logLevel);
}
```

> **Hinweis**
>
> Des Weiteren ist die Groovy-Methode **println()** gemappt auf **LogInterface.info()**, so dass an jeder Stelle im Groovy-Code einfach per **println()** in das d.3-Log geschrieben werden kann.

7.7 Hook-Eigenschaften (HookInterface)

Einige d.3-Eintrittspunkte besitzen spezifische Eigenschaften, die in der entsprechenden Hook-Funktion geändert werden können. Diese Hook-Eigenschaften-Schnittstelle dient dazu, diese Eigenschaften auslesen und ändern zu können.

Wenn solche Eigenschaften existieren, so sind diese in der Beschreibung des d.3-Eintrittspunktes angegeben. Beispiele dafür sind die Render-Optionen von "hook_rendition_entry_20" oder die E-Mail-Eigenschaften von "hook_send_email_entry_20".

Aufruf der Methoden in einer Hook-Funktion:

- Auslesen eines Eigenschaftswertes: d3.hook.getProperty("Eigenschaftsname")
- Ändern eines Eigenschaftswertes: d3.hook.setProperty("Eigenschaftsname", "Eigenschaftswert")

Bei mehrzeilige Eigenschaften entsprechend:

- Auslesen des ersten Wertes einer Mehrfacheigenschaft: d3.hook.getProperty("Eigenschaftsname", 1)
- Ändern des zweiten Wertes einer Mehrfacheigenschaft: d3.hook.setProperty("Eigenschaftsname", 2, "Eigenschaftswert")

HookInterface

```
public interface HookInterface {
  public String getProperty (String propName);
  public String getProperty (String propName, int propIndex);
  public void   setProperty (String propName, String propValue);
  public void   setProperty (String propName, int propIndex, String propValue);
}
```

7.8 Fehlerbehandlung (D3Exception)

D3Exception

```
package com.dvelop.d3.server.exceptions;

public class D3Exception    extends RuntimeException

  public class AmbitiousResultException       extends D3Exception
  public class ConnectionError                extends D3Exception
  public class GroovyAPIFunctionException     extends D3Exception
  public class GroovyAPIFunctionRuntimeException extends D3Exception
  public class GroovyHookException            extends D3Exception
  public class GroovyHookRuntimeException     extends D3Exception
  public class InvalidDateFormatException     extends D3Exception
  public class InvalidFormatException         extends D3Exception
  public class InvalidInputException          extends D3Exception
  public class InvalidParameterException      extends D3Exception
  public class ObjectNotFoundByIdException    extends D3Exception
  public class ReferenceToUnknownObjectException extends D3Exception
  public class SQLException                   extends D3Exception
  public class TimeoutException               extends D3Exception
  public class UnconvertableException         extends D3Exception
  public class NullValueException             extends D3Exception
```

7.9 Storagemanager

StorageManagerInterface

```
public interface StorageManagerInterface {
  public void addFileToReload(String docId, int fileId);
  public void addFileToReload(String docId, int fileId, String dependentExtension);
  public void addFileToReload(String fileName);     // Not document related file

  public void addFileToReload(Document doc);       // All files for the document
  public void addFileToReload(PhysicalVersion physVers);
  public void addFileToReload(DependentFile depFile);

  public void setNumberOfFilesPerJob(int value);
  public void setSMThreadsToUse(int value);
  public void setReloadToCachedDocs(boolean value);
  public void setMoveFilesToDocsDir(boolean value);

  public String getReloadPrefix();
  public int reloadFiles();
}
```

7.10 d.3-Systemeigenschaften

Die d.3-Schnittstelle definiert folgende Systemeigenschaften als Java System Properties:

Property-Name	Beschreibung	Beispielwerte
"d3.server.home"	Das aktuelle d.3 server-Programmverzeichnis	"D:\d3\d3server.prg"
"d3.server.version"	Die d.3 server-Versionsnummer	"08.01.00.05"
"d3.server.systemUser"	Der Systembenutzername des aktuellen d.3 server-Prozesses	"D3Server", "D3Async", "hostimp", "Master"
"d3.repository.uuid"	Die Archive-UUID des d.3-Repositorys	"8be6de08-d009-4c2b-a33d-38a3a6458617"

Diese können abgefragt werden per Aufruf: **System.getProperty("Property-Name")**

```groovy
import javax.swing.JOptionPane
import javax.swing.UIManager

def scriptRequireD3Version = "08.01.00.19"

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

if (System.getProperty("d3.server.version") < scriptRequireD3Version)
{
   showMessageDialog("This script requires d.3 server version <i>${scriptRequireD3Version}</i> or later!", "Check d.3 server version")
   return
}

def showMessageDialog(String messageHtml, String title)
{
   JOptionPane.showMessageDialog(null, "<html>" + messageHtml + "</html>", title, JOptionPane.ERROR_MESSAGE);
}
```

8 Debugging

Um Groovy-Code debuggen zu können, muss der Java Remote Debugging Support in d.3 config aktiviert werden (Einstellung **Java Remote Debugging** (**JAVA_REMOTE_DEBUGGING**) unter **Java/Groovy**).

Dadurch wird die Java Virtual Machine beim Laden durch die d.3-Server-Prozesse im Debugmodus gestartet.

Anschließend ist es möglich, sich per Remote Java Debugger mit einem d.3-Server-Prozess zu verbinden, um die darin ausgeführten Groovy Hooks zu debuggen.

Für die Kommunikation wird Port 43400 benutzt. Da jeder d.3-Prozess eine eigene Java Virtual Machine (JVM) startet, werden die benutzten Ports hochgezählt.

Der erste mit aktiviertem **JAVA_REMOTE_DEBUGGING** gestartete Prozess öffnet Port 43400, der zweite Port 43401 und so weiter.
Der ermittelte Port wird beim Start der JVM per Meldung **Java Remote Debugging Port** in das d.3-Log ausgegeben.

> **Hinweis**
>
> Die JVM wird von d.3 On-Demand beim ersten Zugriff auf Groovy-Code gestartet und steht damit i.d.R. noch nicht direkt nach Prozess-Start zur Verfügung.

Wie Remote Debugging aus Eclipse heraus verwendet werden kann, wird auf der Seite Remote-Debugging mit Eclipse beschrieben.

8.1 Remote Debugging mit Eclipse

Häufig müssen Sie während der Entwicklung von Hook-Funktionen Fehler finden und beheben. Neben der klassischen Variante, alle relevanten Variablen in das d.3 Log zu schreiben, können Sie auch "Remote Debugging" verwenden. Dabei öffnet jeder d.3-Server-Prozess einen Port, zu dem Sie sich mit Eclipse verbinden und ganz genau den Funktionsablauf kontrollieren können.

> **Hinweis**
>
> Dadurch wird die Java Virtual Machine beim Laden durch die d.3-Server-Prozesse im Debugmodus gestartet.
> Die JVM wird von d.3 On-Demand beim ersten Zugriff auf Groovy-Code gestartet und steht damit i.d.R. noch nicht direkt nach Prozess-Start zur Verfügung.

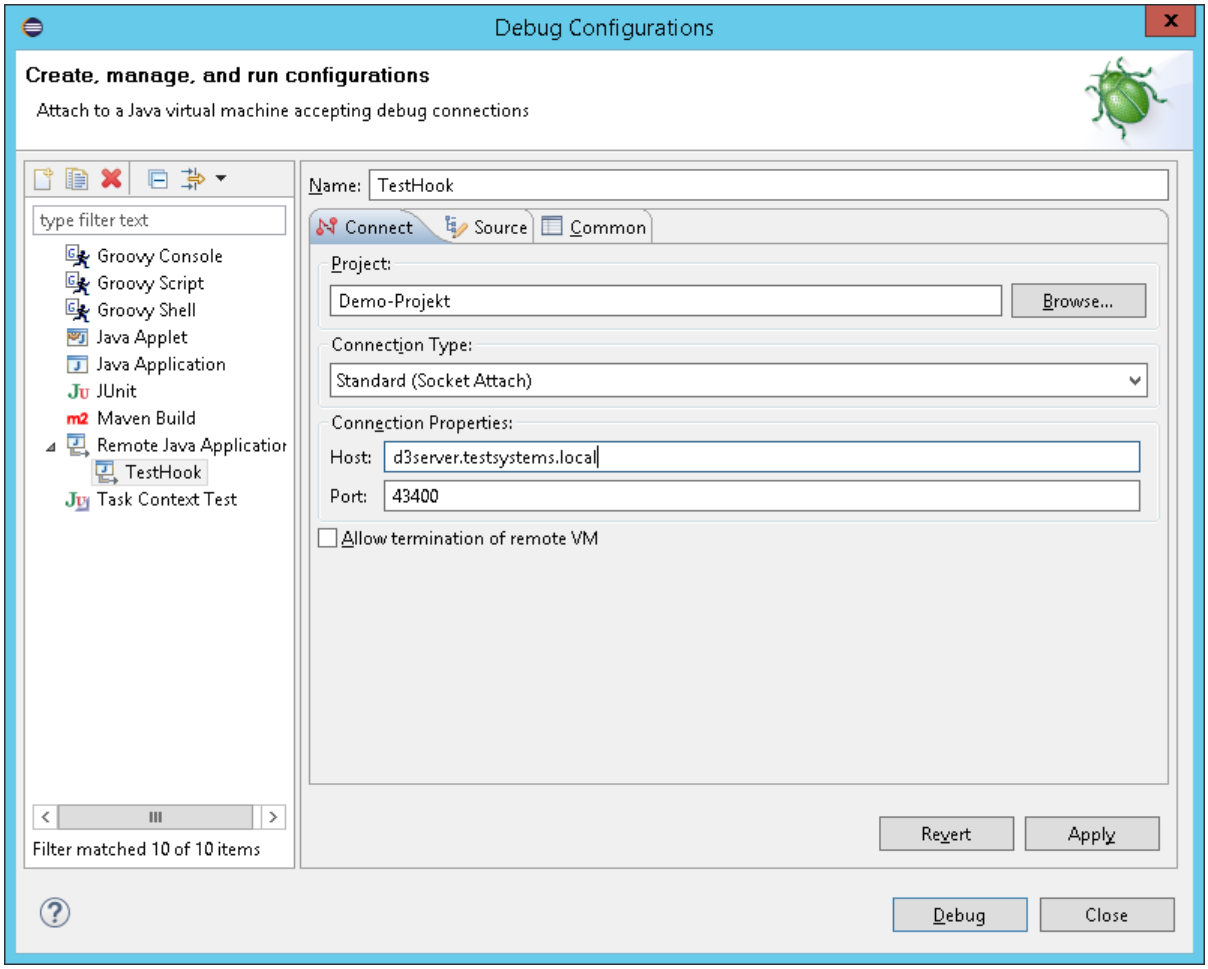
Der erste d.3-Server-Prozess, der startet, verwendet den Port 43400, der zweite Prozess verwendet 43401 und so weiter.

Beachten Sie, dass auch d.3 async- und d.3 Hostimport-Prozesse Remote Debugging unterstützen. Beachten Sie außerdem, dass ein Archiv üblicherweise durch mehrere Server-Prozesse bedient wird. Das hat zur Folge, dass Sie sicherstellen müssen, zum korrekten Serverprozess verbunden zu sein. Der einfachste Weg, dies zu gewährleisten, ist, nur einen d.3 Server-Prozess auszuführen.

> **Wichtig**
>
> Während des Remote Debugging kann der d.3-Serverprozess keine anderen Aufgaben verarbeiten. Es sollte daher niemals im Produktivbetrieb Remote Debugging durchgeführt werden.

Um Remote Debugging zu verwenden, gehen Sie die folgenden Schritte durch:

1. Aktivieren Sie in **d.3 admin > d.3 config > Java/Groovy > Java Remote Debugging**.
2. In Ihrem Eclipse-Projekt wählen Sie **Run > Debug Configurations**.
3. Erstellen Sie eine neue **Remote Java Application**.
4. Geben Sie bei den **Connection Properties** die Adresse des zu debuggenden Systems an.



5. Wählen Sie **Debug**, um das Remote Debugging zu starten.
6. Sie können jetzt in Ihrem Quelltext **Breakpoints** setzen, an denen die Ausführung unterbrochen wird, sodass Sie den Ablauf manuell prüfen und fortführen können.

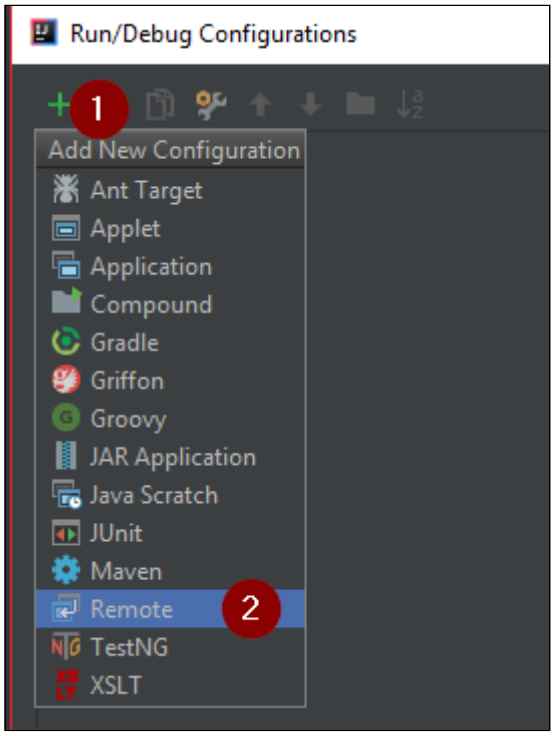
> **Hinweis**
>
> Das Thema "Debugging" im Detail zu beschreiben überschreitet den Umfang dieser Dokumentation. Sie können aber in Fachliteratur sowie im Internet umfangreiche Informationen zu dem Thema finden.

8.2 Remote Debugging mit IntelliJ IDE

Auf dieser Seite werden die notwendigen Schritte beschrieben, um Groovy Hooks eines d.3 Server Prozesses mit Hilfe von IntelliJ IDEA zu debuggen.

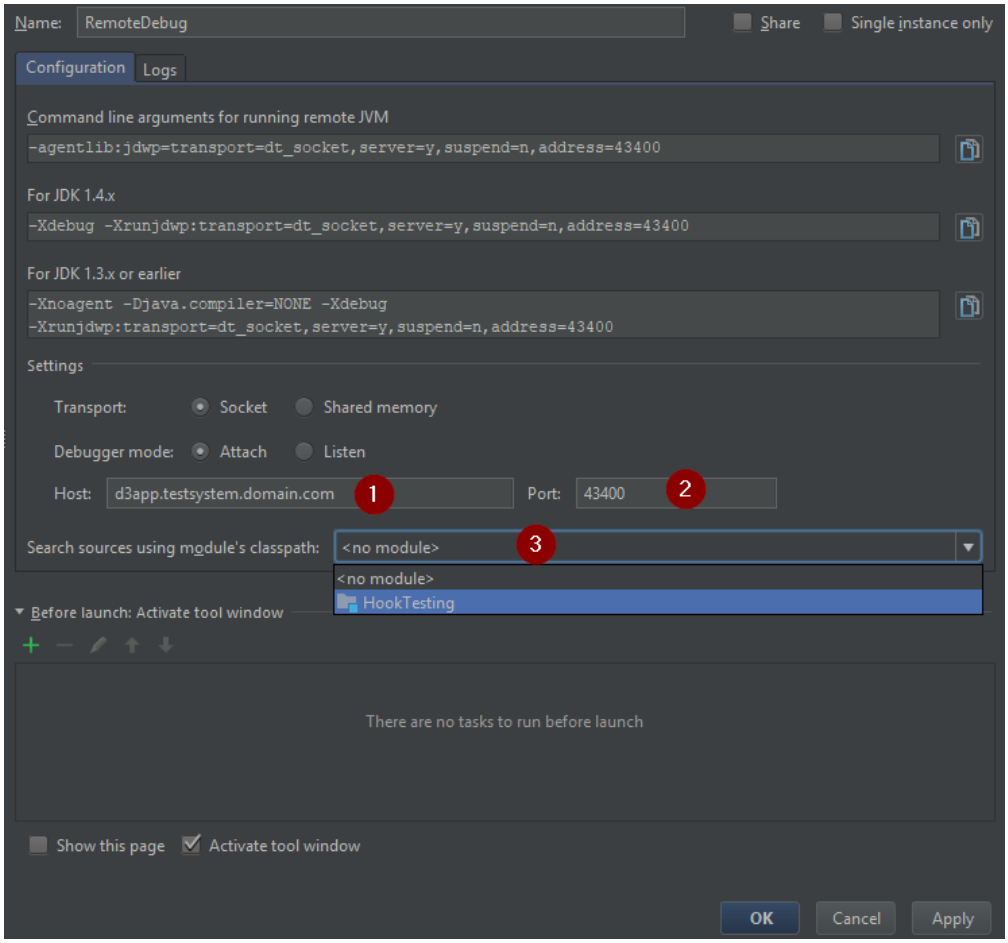
Hinzufügen der Debugkonfiguration

1. Wählen Sie über das Dropdown-Menü für Laufkonfiguration **Edit Configurations...** aus.
2. Wählen Sie in dem neu geöffneten Fenster über **+** (1) eine neue **Remote**-Konfiguration (2) aus.



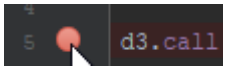
3. Tragen Sie nun die Verbindungsdaten zu dem d.3-Applikationsserver ein, auf dem die Hooks und die d.3-Server-Prozesse ausgeführt werden.
4. Dazu werden Adresse (1) oder FQDN benötigt, sowie der Debug-Port des d.3-Server-Prozesses (2).

5. Abschließend wählen Sie das Projekt-Modul aus IntelliJ aus (3), welches Ihr Hook-Projekt enthält.

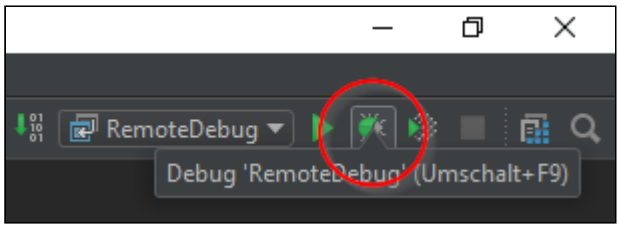


Durchführen des Debugging

1. Setzen Sie an gewünschten Codezeilen **Breakpoints** durch einen Linksklick der Maus neben der Zeilennummer.



2. Stoppen Sie alle d.3-Server-Prozesse
3. Starten Sie einen einzigen d.3-Server-Prozess neu.
4. Aktivieren Sie das Remote Debugging in IntelliJ IDEA.



5. Bei einem Aufruf der Hooks/Skripte werden nun die Breakpoints erreicht und pausieren den Prozess zur Analyse.

> **Hinweis**
>
> Starten Sie erst das Server Interface. Im Log können Sie nachschauen, welcher Port vom Serverprozess gerade belegt wurde. Suchen Sie einfach nach "Remote Debugging". Eine Logzeile könnte bspw. folgendermaßen aussehen:
>
> 04.04 14:46:47,695 D3SRV_P 68B85950 : Java Remote Debugging Port: 43403
>
> Diesen Port müssen Sie in der Konfiguration hinterlegen. Starten Sie danach das Debugging und führen den Hook oder das Skript aus.

Hinweise zum Debugging

- Hinweise zur Debug-Option finden sich auf der Seite Debugging.
- Während des Debuggings ist das d.3-System nicht in der Lage andere Jobs zu verarbeiten.
- Zum Thema Debugging verweisen wir auf gängige Fachliteratur.
- Anleitungen zum Debugging unter IntelliJ IDEA finden sich beim Hersteller JetBrains: https://www.jetbrains.com/idea/documentation/

9 Groovy-Grundlagen

> **Hinweis**
>
> Dies ist kein Groovy-Tutorial oder eine Dokumentation der Sprache, sondern eine Aufstellung von ein paar Grundlagen, welche vielleicht interessant sind und den Einstieg erleichtern.

Interessante Links

Groovy-Dokumentation

http://www.groovy-lang.org/

http://www.groovy-lang.org/style-guide.html

http://grails.asia/groovy-list-tutorial-and-examples

Einführung in die Sprache Groovy

http://www.javabeat.net/introduction-to-groovy-scripting-language/

Tutorials

https://www.timroes.de/2015/06/27/groovy-tutorial-for-java-developers/

http://mrhaki.blogspot.de/

z.B. verschiedene Möglichkeiten um in Listen oder Maps nach Einträgen zu suchen:

http://mrhaki.blogspot.de/2009/10/groovy-goodness-finding-data-in.html

Coole deutsche Einführung in Groovy

http://www.oio.de/public/java/groovy/groovy-einfuehrung.htm

http://www.oio.de/public/java/groovy-closures-artikel.htm

Groovy vs. Java

http://www.groovy-lang.org/differences.html

9.1 Variablen und Strings

> **Hinweis**
>
> Die Ausgabe von Inhalten kann mittels des Befehls **println** erfolgen. Im Kontext d.3 wird dieser Befehl als Infomeldung im Logfile ausgegeben. Es kann aber auch direkt mit der Log-Funktion "d.log.info(..)" gearbeitet werden.

Variablen definieren

Variablen können mit Groovy mittels dynamic typing über das Schlüsselwort **def** definiert werden. Natürlich kann in Groovy auch mit den gängigen Java-Typen gearbeitet werden.

> **Hinweis**
>
> Hierbei sollten Sie nie ein und die selbe Variable für unterschiedliche Typen nutzen!

```
1  package com.dvelop.scripts;
2  import com.dvelop.d3.server.core.D3Interface;
3
4
5  D3Interface d3 = getProperty("d3");
6
7  def x = 42;
8  d3.log.info( "$x --> " + x.getClass() );
9
10 x = "Hello World";
11 d3.log.info( "$x --> " + x.getClass() );
12 // Output:
13 // 30.11 09:44:48,258 Master 10080CD8 D3B: 42 --> class java.lang.Integer
14 // 30.11 09:44:48,258 Master 10080CD8 D3B: Hello World --> class java.lang.String
```

Was hat ein GString mit Strings zu tun?

Mittels Groovy können Variablen innerhalb eines Strings aufgelöst werden. Dazu werden die Variablen mit einem $-Zeichen vorangestellt in den String integriert. Dieses Konzept nennt man dann GString.

```
package com.dvelop.scripts;
import com.dvelop.d3.server.core.D3Interface;

D3Interface d3 = getProperty("d3");

def x = "World";
d3.log.info( "Hello, $x" );

// Output:
// 30.11 09:44:48,258 Master   10080CD8 D3B: Hello, World
```

Man kann auch auf einzelne Teilstrings innerhalb dieser Schreibweise zugreifen, dazu muss aber dann mit geschweiften Klammern gearbeitet werden. Über eine ähnliche Schreibweise, wie man zum Zugriff auf Array-Elemente nutzt, kann in Groovy auch auf einzelne Buchstaben innerhalb eines Strings zugegriffen werden.

```
package com.dvelop.scripts;
import com.dvelop.d3.server.core.D3Interface;

D3Interface d3 = getProperty("d3");

def firstName = "Douglas";
def name      = "Adams";
d3.log.info( "Hello, ${firstName[0]}. $name" );

// Output:
// 30.11 09:44:48,285 Master   10080CD8 D3B: Hello, D. Adams
```

Mehrzeilige Strings

In Groovy können Strings über mehrere Zeilen definiert werden, dazu werden dann am Anfang und Ende drei Anführungszeichen benötigt.

```
package com.dvelop.scripts;
import com.dvelop.d3.server.core.D3Interface;

D3Interface d3 = getProperty("d3");

def s = """This is
a multiline
string""";

d3.log.info( s );
// Output:
// 30.11 09:50:34,925 Master   10080CD8 D3B: This is
// 30.11 09:50:34,925 Master   10080CD8 D3B: a multiline
// 30.11 09:50:34,925 Master   10080CD8 D3B: string
```

9.2 Bedingungen

Im Vergleich mit Java gibt es hier nicht viele Unterschiede, aus diesem Grund werden an dieser Stelle nur die Besonderheiten erwähnt.

Save Navigation Operator

Soll innerhalb einer Objektstruktur ein Wert überprüft werden, muss auch geprüft werden ob die einzelnen Elemente der Struktur ungleich **null** sind. Dies kann auf zwei Wegen erfolgen:

1.

```
if( company.getContact() != null && company.getContact().getAddress() != null &&
company.getContact.getAddress().getCountry() == Country.NEW_ZEALAND ) { ... }
```

2.

```
if( company.getContact()?.getAddress()?.getCountry() == Country.NEW_ZEALAND ) { ... }
```

Wenn also das Objekt selbst oder eine der Bestandteile nicht vorhanden ist, wird einfach **null** zurückgegeben und keine Fehlermeldung.

Elvis-Operator

In Groovy gibt es auch eine komprimierte Schreibweise für ein If-Statement welches wie folgt genutzt werden kann; dabei kommt nach dem "?" der Wahr-Zweig und nach dem ":" der Falsch-Zweig.

```
1  package com.dvelop.scripts;
2  import com.dvelop.d3.server.core.D3Interface;
3
4  D3Interface d3 = getProperty("d3");
5
6  def testValue = null;
7  def name = testValue != null  ? testValue : "default";
8  d3.log.info( "Normal -->" + Name );
9  // Output
10 // 10.12 10:58:23,238 Master   15041A9C D3B: Normal -->default
```

Möchten Sie nun nur dann einen anderen Wert zuweisen, wenn die überprüfte Variable nicht gesetzt ist, können Sie auch mit einer verkürzten Schreibweise arbeiten.

```groovy
package com.dvelop.scripts;
import com.dvelop.d3.server.core.D3Interface;

D3Interface d3 = getProperty("d3");

def testValue = null;
def name = testValue ?: "default";
d3.log.info( "Elvis -->" + Name );
// Output
// 10.12 10:58:23,239 Master   15041A9C D3B: Elvis -->default
```

Switch-Statement

Im Gegensatz zu Java ist Groovy im Switch-Statement nicht auf numerische Werte begrenzt. Folgendes Beispiel verdeutlicht den Sachverhalt.

```groovy
package com.dvelop.scripts;
import com.dvelop.d3.server.core.D3Interface;

D3Interface d3 = getProperty("d3");

def testValue = "ABC";
switch( testValue ){
  case 100: // Integer
    d3.log.info( "The number 100" );
    break;
  case "ABC": // String
    d3.log.info( "The string ABC" );
    break;
  case Long: // Class
    d3.log.info( "A Long value" );
    break ;
  case ['alpha','beta','gamma']: // List
    d3.log .info( "alpha, beta or gamma" );
    break;
  case {it > -0.1 && it < 0.1}: // Closure
    d3.log.info( "A number near zero" );
    break;
  case null: // null
    d3.log.info( "An empty value " );
    break;
  case ~/Groov.*/ : // Regulare Expression
    d3.log.info( "Begins with Groov" );
    break;
  default:
    d3.log.info( "Something completely different" );
}
```

9.3 Schleifen

Schleifen sind hier ebenfalls mit den Java-Basics identisch. Hier gibt es zusätzlich noch das Konzept der "Closures", welches im nachfolgenden Kapitel kurz beschrieben wird. Da geschweifte Klammern für "Closures" reserviert sind erfolgt die initiale Befüllung von Listen/Arrays mit eckigen Klammern.

For-Schleife

```
1  package com.dvelop.scripts;
2  import com.dvelop.d3.server.core.D3Interface;
3
4  D3Interface d3 = getProperty("d3");
5
6  def testList = [ "This", "is", "example", "content" ];
7  int n      = 0;
8  // A classic for-statement
9  for( n = 0; n < testList.size(); n++ ){
10   d3.log.info( "$n. For --> " + testList[n] );
11 }
12 // Output
13 // 10.12 12:33:11,996 Master   15041A9C D3B: 0. For --> This
14 // 10.12 12:33:11,997 Master   15041A9C D3B: 1. For --> is
15 // 10.12 12:33:11,997 Master   15041A9C D3B: 2. For --> example
16 // 10.12 12:33:11,997 Master   15041A9C D3B: 3. For --> content
17
18
19 // A for-statement with collection
20 n = 0;
21 for( def item in testList ){
22   d3.log.info( (n++) +  ". For-(Collection) --> " + item );
23 }
24
25 // Output
26 // 10.12 12:33:11,998 Master   15041A9C D3B: 0. For-(Collection) --> This
27 // 10.12 12:33:11,999 Master   15041A9C D3B: 1. For-(Collection) --> is
28 // 10.12 12:33:11,999 Master   15041A9C D3B: 2. For-(Collection) --> example
29 // 10.12 12:33:11,999 Master   15041A9C D3B: 3. For-(Collection) --> content
```

Each-Statements

```groovy
package com.dvelop.scripts;
import com.dvelop.d3.server.core.D3Interface;

D3Interface d3 = getProperty("d3");

// Each-Statement
n = 0;
testList.each{
  d3.log.info( (n++) +  ". Each --> " + it );
}
// Output
// 10.12 12:33:12,008 Master   15041A9C D3B: 0. Each --> This
// 10.12 12:33:12,008 Master   15041A9C D3B: 1. Each --> is
// 10.12 12:33:12,008 Master   15041A9C D3B: 2. Each --> example
// 10.12 12:33:12,009 Master   15041A9C D3B: 3. Each --> content
// Each-Statement with index
testList.eachWithIndex { val, idx ->
  d3.log.info( idx +  ". Each with index --> " + val );
}
// Output
// 10.12 12:33:12,010 Master   15041A9C D3B: 0. Each with index --> This
// 10.12 12:33:12,010 Master   15041A9C D3B: 1. Each with index --> is
// 10.12 12:33:12,010 Master   15041A9C D3B: 2. Each with index --> example
// 10.12 12:33:12,010 Master   15041A9C D3B: 3. Each with index --> content
```

Ein Collection-Statement

```groovy
package com.dvelop.scripts;
import com.dvelop.d3.server.core.D3Interface;

D3Interface d3 = getProperty("d3");

n = 0;
def newList = testList.collect { it; }
newList.each{
  d3.log.info( (n++) + ". Collect --> " + it );
}
// Output
// 10.12 12:33:12,012 Master   15041A9C D3B: 0. Collect --> This
// 10.12 12:33:12,012 Master   15041A9C D3B: 1. Collect --> is
// 10.12 12:33:12,012 Master   15041A9C D3B: 2. Collect --> example
// 10.12 12:33:12,012 Master   15041A9C D3B: 3. Collect --> Content
```

9.4 Closures

Closures sind kleine, unbenannte Funktionen welche direkt an eine Variable gebunden werden.

Eine Variable mit einer Funktion

```
1  package com.dvelop.scripts;
2  import com.dvelop.d3.server.core.D3Interface;
3
4  D3Interface d3 = getProperty("d3");
5
6  // Print Hello world ----------------------------------------
7  def optionOne = {
8    d3.log.info( "Option 1: Hello World" );
9  }
10 optionOne();
11
12 // Output
13 // 09.12 13:30:35,082 Master   12041F98 D3B: Option 1: Hello World
```

Closures mit Variablen mit festen Typen

```
1  // Closures with parameters ---------------------------------
2  def power = { int x, int y ->
3    return Math.pow( x, y ); }
4  d3.log.info( "Option 2: " + power( 2, 3 ));
5
6  // Output
7  // 09.12 13:30:35,093 Master   12041F98 D3B: Option 2: 8.0
```

Closure mit einer untypisierten Variable

```
1  package com.dvelop.scripts;
2  import com.dvelop.d3.server.core.D3Interface;
3
4  D3Interface d3 = getProperty("d3");
5
6  // Closure with one dynamic typing variable ----------------
7  def optionThree = { what ->
8    d3.log.info( what ); }
9  optionThree "Option 3: Hello World"; // same as optionThree( "Option 3: Hello World" );
10
11 // Output
12 // 09.12 13:30:35,094 Master   12041F98 D3B: Option 3: Hello World
```

Closure, bei einer Variable kann auch die implizite Variable "it" genutzt werden

```
1  // Closure with an implizit argument -------------------------
2  def optionFour = { d3.log.info( it ); }
3  optionFour "Option 4: Hello World"; // same as optionFour("Option 4: Hello World");
4
5  // Output
6  // 09.12 13:30:35,095 Master   12041F98 D3B: Option 4: Hello World
```

Closure mit explizit KEINER Variablen

```
1
2  package com.dvelop.scripts;
3  import com.dvelop.d3.server.core.D3Interface;
4
5  D3Interface d3 = getProperty("d3");
6
7  // Closure without any argument -------------------------------
8  def optionFive = { ->
9    d3.log.info( "Option 5: This closure does not take any arguments." ); }
10 optionFive();
11 // Output
12 // 09.12 13:30:35,095 Master   12041F98 D3B: Option 5: This closure does not take any arguments.
```

Soll ein Wert zurückgegeben werden, kann dies auch ohne Return erfolgen, dann wird der letzte Wert zurückgegeben.

```
1
2  package com.dvelop.scripts;
3  import com.dvelop.d3.server.core.D3Interface;
4
5  D3Interface d3 = getProperty("d3");
6
7  // Optional return value ------------------------------------
8  def square = { it * it };
9  d3.log.info( "Option 6: " + square(4));
10
11 // Output
12 // 09.12 13:30:35,127 Master   12041F98 D3B: Option 6: 16
```

9.5 Datenbankanbindung

Zugriff auf eine d.3-interne Datenbanktabelle (d.3-SQL-Datenbank)

Der Zugriff auf Datenbanktabellen welche sich innerhalb der d.3-Datenbank befinden erfolgt über d.3-SQL-Schnittstelle, welche einen einfachen Zugriff auf die Daten zur Verfügung stellt.

Beispiel - Zugriff auf die d.3 interne Datenbank-Schnittstelle:

```
1  //(1)
2  package com.dvelop.scripts;
3  import com.dvelop.d3.server.core.D3Interface;
4
5  D3Interface d3 = getProperty("d3");
6
7  //(2)
8  def resultRows = d3.sql.executeAndGet( "SELECT name FROM CustomerData" );
9  //(3)
10 resultRows.each{ println it.name; }
```

Kommentare zu den einzelnen Blöcken

1. Zur Nutzung der d.3-SQL-Schnittstelle wird die Bibliothek D3 benötigt und muss bei Bedarf importiert werden. Da über den d3-server-interface-Aufruf implizit die Variable **d3** zur Verfügung steht, kann diese über die Funktion **getProperty("d3")** im Skript zur Verfügung stellt werden und steht damit auch währender Programmierung zur Codevervollständigung bereit.
2. Über diverse Implementierungen/Funktionen kann nun ein SQL-Statement gegen die Datenbanktabelle abgeschickt werden. Die Ergebnisse werden dabei in einer Map-Struktur übergeben.
3. Mittels zum Beispiel eines Each-Statements können die Daten weiterverarbeitet werden.

Specials

Und für solche häufigen Abfragen, bei denen man nur einen Treffer hat oder nur einen haben will, existiert die Methode **firstRow()**.

```
1  def sqlQuery  = "SELECT max(product_count) as value FROM productDB WHERE product_id = ? ";
2  def sqlParams = [ 4711 ];
3  def firstRow  = d3.sql.firstRow (sqlQuery, sqlParams);
4  int max_no    = firstRow[0] + 1;
```

Allerdings noch besser lesbar und damit besserer Code ist – mit **AS highestNo** im SQL-Kommando:

```
1  def sqlQuery  = "SELECT max(product_count) as value FROM productDB WHERE product_id = ? ";
2  def sqlParams = [ 4711 ];
3  def firstRow  = d3.sql.firstRow (sqlQuery, sqlParams)
4  int max_no    = firstRow.highestNo + 1;
```

Zugriff auf eine externe Datenbank

> **Hinweis**
>
> Für den Zugriff auf externe Datenbanken müssen die benötigten JDBC-Treiber bei den Datenbankherstellern heruntergeladen werden und bereitgestellt werden. Dann kann über die Standard-SQL-Schnittstelle eine Verbindung zur Datenbank aufgebaut werden und ein SQL-Statement abgesetzt werden.

Beispiel - Zugriff auf eine externe Datenbank:

```
1  //(1)
2  import groovy.sql.Sql;
3  //(2)
4  def dbConnection = Sql.newInstance( "jdbc:sqlserver://localhost:1433;databaseName=Name",
   "User", "Password" );
5  //(3)
6  def resultRows = dbConnection.rows( "SELECT name FROM CustomerData" );
7  //(4)
8  resultRows.each{ println it.name; }
```

Kommentare zu den einzelnen Blöcken

1. Zur Nutzung der d.3-SQL-Schnittstelle für externe Datenbankzugriffe, wird die Standard-Groovy-Bibliothek SQL benötigt und muss bei Bedarf importiert werden.
2. Um nun auf eine externe Datenbank zugreifen zu können, muss eine Verbindung zur Datenbank hergestellt werden. Dazu wird hier ebenfalls eine Standard-Groovy-Funktion zum Aufbau einer Verbindung **newInstance** genutzt. Weitere Details zu den Parametern können den gängigen Dokumentationen zu den Datenbanken bzw. zu Groovy nachgeschlagen werden.
3. Über diverse Implementierungen/Funktionen kann nun ein SQL-Statement gegen die Datenbanktabelle abgeschickt werden. Die Ergebnisse werden dabei in einer Map-Struktur übergeben.
4. Mittels zum Beispiel eines Each-Statements können die Daten weiterverarbeitet werden.

9.6 d.3-Specials

9.6.1 d.3-Konfigurationsparameter auslesen

Möchten Sie zum Beispiel ein Groovy-Skript sowohl in der Test-Umgebung als auch in der Produktivumgebung nutzen, können Sie über die Abfrage von Konfigurations- und Serverparametern das Skript so programmieren, dass ein Skript ohne Anpassungen in beiden Welten funktioniert. Eine Portierung bzw. Anpassung ist damit dann nicht mehr notwendig. Eine Dokumentation der möglichen Parameter ist in den relevanten d.3-Dokumentationen zu finden. Anbei ein kleines Beispiel.

```
1  d3.log.error( d3.conf.value( "d3fc_server_id" ) );   // Server id
2  d3.log.error( d3.conf.value( "d3fc_server_name" ) ); // Server name
3  d3.log.error( d3.conf.value( "db_server" ) );       // Database type
4  d3.log.error( d3.conf.value( "CUR_60ER_FIELD_NR" ) );// Current max . sizce of 60-ies feild
5
6  d3.log.error( d3.natives.getd3fcLanguage() );       // Get the current language
7
8  // Output
9  // 08.12 15:25:39,090 Master   130C1308 D3B: B
10 // 08.12 15:25:39,090 Master   130C1308 D3B: localhost
11 // 08.12 15:25:39,090 Master   130C1308 D3B: MSQL
12 // 08.12 15:25:39,091 Master   130C1308 D3B: 100
13
```

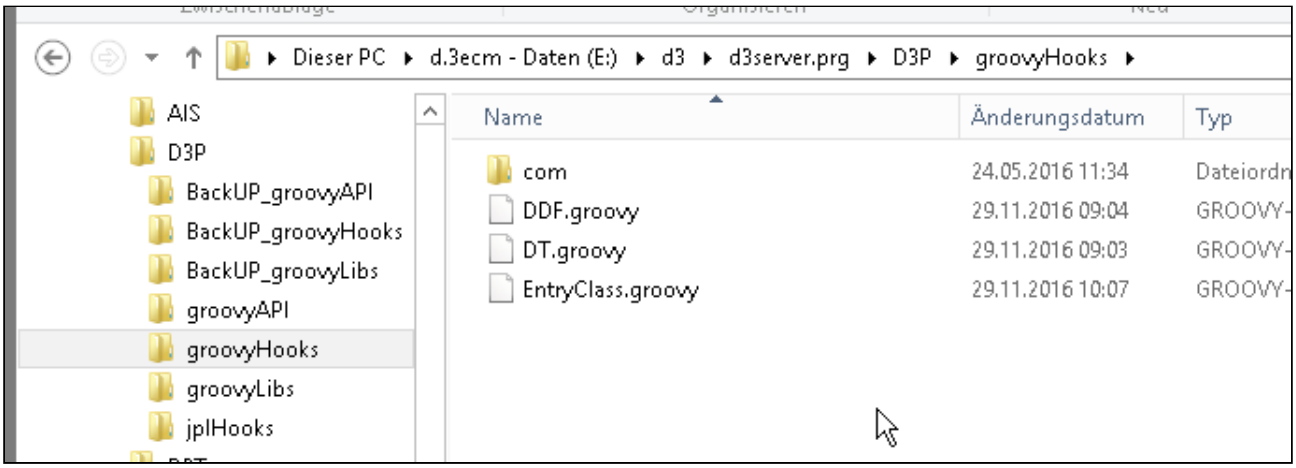
Ab Version 8.1 können Parameter wie die aktuelle API-Sprache oder die APP-Version auch in Groovy ermittelt werden.

```
1  import com.dvelop.d3.server.Document
2  import com.dvelop.d3.server.DocumentType
3  import com.dvelop.d3.server.Entrypoint
4  import com.dvelop.d3.server.User
5  import com.dvelop.d3.server.core.D3Interface
6  public class d3RemoteInterfaceExample{
7
8    @Entrypoint( entrypoint = "hook_insert_entry_10" ) //----------------------
9    public int getCustomerDataForInvoice( D3Interface d3, User user, DocumentType docTypeShort,
   Document doc ){
10     d3.log.error( "App-Version: "  + d3.remote.getVersion());
11     d3.log.error( "APP-ID: "       + d3.remote.getVersion()[0..2]);
12     d3.log.error( "App-Language: " + d3.remote.getLanguage());
13     return 0;
14   } // end of getCustomerDataForInvoice
15
16 }// end of d3RemoteInterfaceExample
```

9.6.2 Klasse für globale Konstanten

Im JPL wurde mit globalen Konstanten die Referenzierung auf Datenbank-Positionen der erweiterten Eigenschaften sprechend gestaltet und zentral gesteuert.

Dies funktioniert aktuell nicht über eine separate Package-Struktur sondern kann nur im Root-Verzeichnis mit allen notwendigen Dateien genutzt werden.



Dazu werden hier exemplarisch zwei Klassen angelegt, welche dann innerhalb der Hook-Funktionen referenziert werden.

Klasse für die Dok-Dat-Feld-Positionen

```
1  class DDF {
2    static final int FIRSTNAME = 4; // DB positions
3    static final int LASTNAME  = 5;
4    static final int STATE_ID  = 6;
5  }// end of DDF
```

Klasse für die Dokument- und Aktenarten

```
1  class DT {
2    static final String INVOCIE     = "DRECH"; // Doc types
3    static final String ORDER       = "DBEST";
4    static final String EMPLOYEE_FOLDER = "APERS"; // Folder types
5  }// end of DT
```

Da die Klassen, im gleichen Verzeichnis abgelegt wurden, können diese nun in den Hook-Dateien / -Funktionen genutzt werden.

Nutzung der globalen Konstanten innerhalb einer Hook-Funktion

```groovy
import com.dvelop.d3.server.Document
import com.dvelop.d3.server.DocumentType
import com.dvelop.d3.server.Entrypoint
import com.dvelop.d3.server.User
import com.dvelop.d3.server.core.D3Interface

class D3Hooks {

  @Entrypoint( entrypoint = "hook_insert_entry_10" )
  public int myEntryPoint( D3Interface d3, User d3User, DocumentType docTypeShort, Document doc ){
    d3.log.error( "+++ MyGlobal-Test+++++ " + DDF.FIRSTNAME);
    d3.log.error( "+++ MyGlobal-Test+++++ ${DDF.LASTNAME}" );
    d3.log.error( "+++ MyGlobal-Test+++++ $DDF.STATE_ID" );

    d3.log.error( "+++ MyGlobal-Test+++++ " + DT.INVOICE);
    d3.log.error( "+++ MyGlobal-Test+++++ " + DT.ORDER );
    d3.log.error( "+++ MyGlobal-Test+++++ " + DT.EMPLOYEE_FOLDER);
    return 0;
  }// end of myEntryPoint
}// end of D3Hooks
```

10 Groovy-Hook-Beispiele

10.1 Eintrittspunkte

> **Hinweis**
>
> Für die Nutzung von Eintrittspunkten müssen diese Bibliotheken importiert werden:
>
> **Globale d.3-Bibliotheken**
>
> - import com.dvelop.d3.server.core.D3Interface
> - import com.dvelop.d3.server.Document
> - importcom.dvelop.d3.server.User
> - import com.dvelop.d3.server.DocumentType
>
> **Spezifische Bibliotheken für die Eintrittspunkte**
>
> - import com.dvelop.d3.server.Entrypoint
> - import com.dvelop.d3.server.Condition

> **Hinweis**
>
> Man kann auf jede beliebigen Hook-Eintrittspunkt beliebig viele Funktionen definieren! Man könnte also Lösungsbezogen, den Eintrittspunkt "hook_insert_entry_10" für die Überprüfung der Bestellnummer einer Rechnung nutzen, die Personalnummer eines Urlaubsantrag überprüfen oder die Daten eines Auftrags vervollständigen.

Die unterschiedlichen Eintrittstypen, benötigen dann eine definierte Anzahl von Parametern in einer vorgegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3 Objekt übergeben.

> **Hinweis**
>
> Für die Hook-Eintrittspunkte Wertemengen, Validierung und Suche muss immer am Ende noch ein zusätzlicher Parameter vom Typ "Document" angehängt werden.

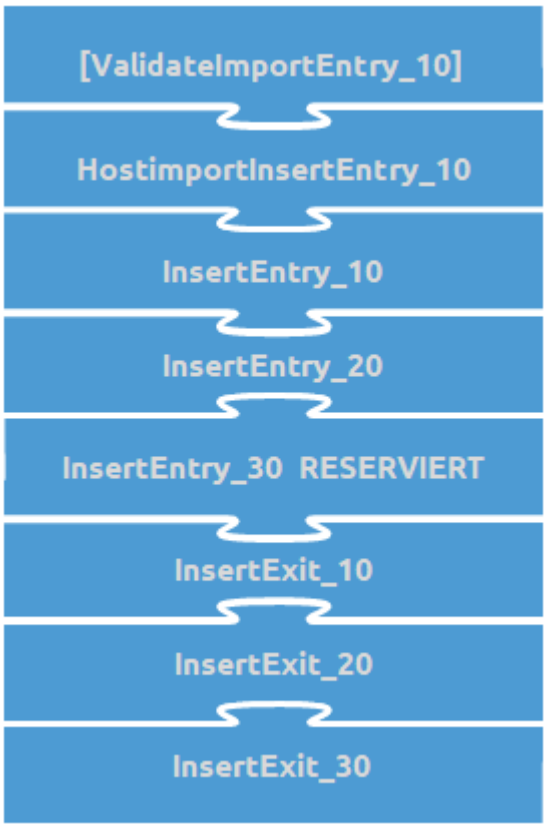
Eine Kette von Eintrittspunkten

Wird zum Beispiel ein Dokument über die Verzeichnisüberwachung (Hostimport) abgelegt, wird auf der Server-Seite folgende Kette von Eintrittspunkten abgearbeitet:

- hook_hostimp_entry_10 →hook_insert_entry_10 → hook_insert_entry_20 → hook_insert_exit_10 → hook_insert_exit_20 → hook_insert_exit_30

Wird zum Beispiel ein Dokument per manuellen Import abgelegt, wird auf der Server-Seite folgende Kette von Eintrittspunkten abgearbeitet:

- hook_validate_import_entry_10 → hook_insert_entry_10 → hook_insert_entry_20 → hook_insert_exit_10 → hook_insert_exit_20 → hook_insert_exit_30



Wird nun eine Funktion zu einem Eintrittspunkt mit einem Fehler bzw. einem Wert ungleich 0 beendet bzw. liefert einen Wert ungleich 0 zurück; wird die komplette Kette unterbrochen und das Dokument wird zum Beipsiel nicht archiviert.

10.1.1 InsertEntry_10

Hallo Welt!

> **Hinweis**
>
> Szenario:
>
> Wird ein Dokument im d.3-System abgelegt, soll im d.3 Log-File einfach nur eine Fehlermeldung "Hallo Welt" angezeigt werde.

Dazu wird eine Hook-Funktion für den Eintrittspunkt "**hook_insert_entry_10**" hinterlegt, welche die Kunden-Nummer überprüft.

```
1  package com.dvelop.hooks;
2
3  //(1)
4  // Global d.3 libraries
5  import com.dvelop.d3.server.core.D3Interface;
6  import com.dvelop.d3.server.Document;
7  import com.dvelop.d3.server.User;
8  import com.dvelop.d3.server.DocumentType;
9
10 // Libraries to handle the different hook types
11 import com.dvelop.d3.server.Entrypoint;
12
13 //(2)
14 public class D3Hooks{
15 //(3)
16   @Entrypoint( entrypoint = "hook_insert_entry_10" ) //----------------------
17 //(4)
18   public int insertEntry_10( D3Interface d3, User user, DocumentType docTypeShort, Document
   doc ){
19 //(5)
20     d3.log.error("Hello world!");
21 //(6)
22     return 0;
23   } // end of insertEntry_10
24 } // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der **groovyhook.jar**-Datei.
2. Bereitstellung einer eigenen Klasse vom Typ "public", wobei "public" im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen erfolgt, über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen, benötigen dann eine definierte Anzahl von Parametern in einer vorgegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-

Funktion immer das d.3-Objekt übergeben. Zusätzlich wird als letzter Parameter noch ein Parameter vom Type "Document" benötigt.

5. Nun wird mittels einfacher Log-Funktion eine Ausgabe im Log-File erzeugt.
6. Die Funktion wird mit einem Return-Wert "0" beendet.

Kontrolle und Ergänzung von Dokumenteigenschaften bei Import

> **Hinweis**
>
> Szenario:
>
> Während der Ablage eines Dokumentes im d.3-System soll die Eigenschaft Kunden-Nummer gegen eine Datenbank kontrolliert und bei Bedarf die restlichen Kundendaten automatisch ergänzt werden.
> Die benötigten Kundendaten in diesem Beispiel liegen dabei in einer Datenbank-Tabelle welche innerhalb der d.3-Datenbank hinterlegt wurde damit können die Daten über eine einfache SQL-Implementierung abgerufen werden.

Dazu wird eine Hook-Funktion für den Eintrittspunkt "**hook_insert_entry_10**" hinterlegt, welche die Kunden-Nummer überprüft.

```groovy
package com.dvelop.hooks;

//(1)
// Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;

// Libraries to handle the diferent hook types
import com.dvelop.d3.server.Entrypoint;

//(2)
public class D3Hooks{
//(3)
@Entrypoint( entrypoint = "hook_insert_entry_10" )
//(4)
public int insertEntry_10( D3Interface d3, User user, DocumentType docTypeShort, Document doc ){
//(5)
    if( docTypeShort.id == "DINV" ){
      def customerID = doc.field[14];
//(6)
     if( customerID != "" ){
       def sqlQuery   = "SELECT name, street, zipCode, city FROM CustomerData WHERE customerNo = ? ";
       def sqlParams  = [ customerID ];
       def resultRows = d3.sql.executeAndGet( sqlQuery, sqlParams );
//(7)
      if( resultRows.size() == 1 ){
        doc.field["Street"]  = resultRows[0].street;
        doc.field["ZipCode"] = resultRows[0].zipCode.toString(); // ATTENTION!
        doc.field[10]        = resultRows[0].city;
        return  0;
      }
(8)
      else {
        return  -1;
      }
    }
  }
   return  0;
  } // end of insertEntry_10
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ "public", wobei "public" im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen erfolgt, über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.

4. Die unterschiedlichen Eintrittstypen, benötigen dann eine definierte Anzahl von Parametern in einer vorgegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Zusätzlich wird als letzter Parameter noch ein Parameter vom Type "Document" benötigt.
5. Nun kann innerhalb der Funktion für einen bestimmten Dokumenttyp eine Validierung vorgenommen werden, dabei wird der Dokumenttyp über die Eigenschaft **pDocTypeShort.id** eingeschränkt. In diesem Beispiel wird die Einschränkung mittels IF-Statement vorgenommen, hier kann aber auch über eine "**Condition**" die Einschränkung auf einen oder mehrere Dokumenttypen vorgenommen werden.
6. Nachdem die Kunden-Nummer aus den Dokumenteigenschaften ermittelt wurde, kann nun die Kunden-Nummer gegen die Datenbank geprüft werden. Liegt dabei die Datenbanktabelle im d.3-DB-Adressraum kann mittels der d.3-SQL-Implementierung die native Datenbank-Schnittstelle des d.3-Servers genutzt werden.
7. Da im Kontext des Eintrittspunktes **hook_insert_entry_10** alle Dokumenteigenschaften sowohl lesend als auch schreibend zur Verfügung stehen, können nun die zusätzlichen Kundendaten ergänzt werden. Hierbei können die Eigenschaften über die Datenbank-Position oder direkt und sprechend mit der Eigenschaftenbezeichnung referenziert werden. Hier sollte man sich natürlich auf eine Art der Referenzierung einigen.
8. Konnte keine gültige Kunden-Nummer ermittelt werden, erfolgt eine Rückgabe mit einem Returnwert ungleich "0" und damit wird dann an dieser Stelle die Verarbeitungskette unterbrochen.

10.1.2 InsertExit_20

Eintrag in den Dokumentnotizen

> **Hinweis**
>
> Szenario:
>
> Wurde eine Rechnung erfolgreich importiert, kann automatisch ein Eintrag in die Dokument-Notizen geschrieben werden.
>
> Details zu den hier verwendeten Groovy-Server-API-Funktionen können den spezifischen Dokumentationen entnommen werden.

Dazu wird eine Hook-Funktion für den Eintrittspunkt **hook_insert_exit_20** hinterlegt; der Eintrag wird dann mittels Groovy-Server-API geschrieben..

```
1  package com.dvelop.hooks;
2
3  //(1)
4  // Global d.3 libraries
5  import com.dvelop.d3.server.core.D3Interface;
6  import com.dvelop.d3.server.Document;
7  import com.dvelop.d3.server.User;
8  import com.dvelop.d3.server.DocumentType;
9
10 // Libraries to handle the diferent hook types
11 import com.dvelop.d3.server.Entrypoint;
12 //(2)
13 public class D3Hooks{
14 //(3)
15   @Entrypoint( entrypoint = "hook_insert_exit_20" )
16 //(4)
17   @Condition( doctype = [ "DINV" ] )
18 //(5)
19   public int insertExit_20( D3Interface d3, Document doc, def fileDest, def importOK,
20                             User user, DocumentType docTypeShort ){
21 //(6)
22     def returnValue = 0;
23     returnValue = d3.call.note_add_string( "Hello World!", doc.id, user.id );
24 //(7)
25     return  0;
26   } // end of insertExit_20
27 } // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der **groovyhook.jar**-Datei.
2. Bereitstellung einer eigenen Klasse vom Typ "public", wobei "public" im Kontext Groovy auch weggelassen werden kann.

3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen erfolgt, über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. In diesem Beispiel wird nun die Einschränkung mittels **Condition** auf einen oder mehrere Dokumenttypen vorgenommen.
5. Die unterschiedlichen Eintrittstypen, benötigen dann eine definierte Anzahl von Parametern in einer vorgegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben.
6. Nun kann mittels der Groovy-Server-API-Funktion "note_add_string" ein Eintrag zur Dokumentnotiz hinzugefügt werden.
7. War die Aktion erfolgreich wird ein Return-Wert gleich "0" zurückgegeben.

Weiterleitung an einen Sachbearbeiter

> **Hinweis**
>
> Szenario:
>
> Wurde eine Rechnung erfolgreich importiert, kann diese direkt einer Gruppe oder einem User in den Postkorb gelegt werden.

Dazu wird eine Hook-Funktion für den Eintrittspunkt "**hook_insert_exit_20**" hinterlegt und über einer Groovy-Server-Funktion das Dokument einem User in den Postkorb gelegt.

```groovy
package com.dvelop.hooks;

//(1)
// Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;
import com.dvelop.d3.server.Condition;

// Special libraries
import groovy.sql.Sql;
import java.sql.Timestamp;

//(2)
public class D3Hooks{
//(3)
  @Entrypoint( entrypoint = "hook_insert_exit_20" )
//(4)
  @Condition( doctype = [ "DINV" ] )
//(5)
  public int insertExit_20( D3Interface d3, Document doc, def  fileDest, def  importOK,
                    User user, DocumentType docTypeShort ){
    // Define variables ---------------------------------------------
    def      returnValue = 0;
    Timestamp currentDate = new Date().toTimestamp();
//(6)
    returnValue = d3.call.hold_file_send( user.id, "Invoice: " + doc.getField("RechnungsNr"),
                                doc.id, currentDate, currentDate, false , true ,
                                currentDate, "", "dvelop", 0, false , false , currentDate, 0, false );
//(7)
    return 0;
  } // end of insertExit_20
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der **groovyhook.jar**-Datei.
2. Bereitstellung einer eigenen Klasse vom Typ "`public`", wobei "`public`" im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen erfolgt, über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. In diesem Beispiel wird nun die Einschränkung mittels "**Condition**" auf einen oder mehrere Dokumenttypen vorgenommen.
5. Die unterschiedlichen Eintrittstypen, benötigen dann eine definierte Anzahl von Parametern in einer vorgegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben.

6. Nun kann mittels der Groovy-Server-API-Funktion "note_file_send" das aktuelle Dokument in die Wiedervorlage eines d.3-Users, in diesem Beispiel desjenigen welcher das Dokument archiviert hat, gesendet werden.
7. War die Aktion erfolgreich wird ein Return-Wert gleich "0" zurückgegeben.

Update von anderen Dokumenten abhängig vom aktuellen

> **Hinweis**
>
> Szenario:
>
> Eine Akte wird im d.3-System angelegt und es sollen Daten aus der Akte in andere Dokumente übernommen werden.

Dazu wird eine Hook-Funktion für den Eintrittspunkt "**hook_insert_exit_20**" hinterlegt und über einer Groovy-Server-Funktion das Dokument einem User in den Postkorb gelegt.

```groovy
//(1)
import com.dvelop.d3.server.core.D3Interface
import com.dvelop.d3.server.core.D3Interface.ArchiveInterface

import com.dvelop.d3.server.Document
import com.dvelop.d3.server.DocumentType
import com.dvelop.d3.server.User

import com.dvelop.d3.server.Condition
import com.dvelop.d3.server.Entrypoint

//-----------------------------------------------------------------------------------------
/**
*  Add function to entry point hook_insert_exit_30 for handling change in personal data
*
*  @param   d3, doc, fileDest, importOK, user, docType --> Server definied
*  @return  integer as result value
*/
//(2)
@Entrypoint(entrypoint="hook_insert_exit_30")
@Condition( doctype =  [ "PERSA" ] )
public int entryUpdatePersonalData( D3Interface d3, Document docObj, def fileDest, def
 importOK, User user, DocumentType docType ) {
  int retValue;
//(3)
  retValue =  updatePersData( d3, doc, user );
  return retValue;
} // end of entryUpdatePersonalData

//-----------------------------------------------------------------------------------------
/**
*  Function for updating function
*
*  @param   d3     D3Interface object
*  @param   doc    Document object
*  @param   login   current user object
*  @return  integer as result value
*/
//(4)
public int updatePersData( D3Interface d3, Document doc, User login ) {

  Document docObjRef;
//(5)
  ArchiveInterface archiveObj = d3.getArchive();

//(6)
  def currentDocId = doc.id;
  def client      = doc.field[DDF.MANDANT];   // DDF... = Database position in ext. class
  def persNumber   = doc.field[DDF.EMPLOYEE_NO];
  def persName     = doc.field[DDF.EMPLYOEE_NAME];
//(7)
  def sqlQuery   = "SELECT .... ";
  def resultRows = d3.sql.executeAndGet( sqlQuery );
  def childDocObj;
//(8)
```

```
55      resultRows.each {
56        childRef = it.doku_id;
57    //(9)
58        childDocObj = archiveObj.getDocument( childRef, login.id );
59    //(10)
60        docObjRef.field[DDF.EMPLOYEE_NAME] = persName;
61    //(11)
62        docObjRef.updateAttributes( login.id, true ); // !!!!
63      }
64
65      return 0;
66    } // end of updatePersDataddd
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der **groovyhook.jar**-Datei.
2. Bereitstellung einer Funktion auf dem Einspungpunkt "hook_insert_exti_20".
3. Aufruf der separaten Funktion zur Aktualisierung der Personaldaten.
4. Funktionsdefinition der Update-Funktion.
5. Bereitstellung eines ArchivInteface-Objects zwecks Generierung der Document-Objekte.
6. Die benötigten Daten werden aud dem aktuellen Dokument ausgelesen, hierzu wurde hier exemplarisch eine extn. Klasse zur Definition von globalen Konstanten genutzt.
7. An dieser Stelle könnte nun über die bereitgestellten Eigenschaften eine Suche weiterer Dokument-IDs zur aktuellen Personal-Nr. ermittelt werden; dies kann mittels SQL-Statement erfolgen.
8. Über eine Schleife werden dann alle erkannten Dokumente bearbeitet.
9. Für jede Dokument-ID wird ein Dokument-Objekt erzeugt.
10. Die neue Eigenschaft wird zugewiesen.
11. Danach erfolgt ein Update der Daten, hier ist der zweite Parameter sehr wichtig, dieser sorgt mit "true" dafür das eine nachgelagerte Hook-Validierung nicht stattfindet, "false" hätte hier den gegenteiligen Effekt.

10.1.3 UpdateAttribEntry_20

Kontrolle und Ergänzung von Dokument-Eigenschaften bei der Aktualisierung

> **Hinweis**
>
> Szenario:
>
> Wird die Dokumenteigenschaft "Kunden-Nummer" aktualisiert sollen hier ebenfalls die Daten gegen eine Datenbank kontrolliert und bei Bedarf die restlichen Kundendaten automatisch ergänzt werden. Die benötigten Kundendaten in diesem Beispiel liegen dabei in einer ext. Datenbank und müssen über eine ext. Datenbankverbindung abgerufen werden.

Dazu wird eine Hook-Funktion für den Eintrittspunkt "**hook_upd_attrib_entry_20**" hinterlegt, welche die Kunden-Nummer überprüft.

```groovy
package com.dvelop.hooks;

//(1)
// Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;

// Libraries to handle the different hook types
import com.dvelop.d3.server.Entrypoint;
import com.dvelop.d3.server.Condition;

// Special libraries
import groovy.sql.Sql;

//(2)
public class D3Hooks{
//(3)
  @Entrypoint( entrypoint = "hook_upd_attrib_entry_20" )
  @Condition( doctype = "DRECH" )
//(4)
  public int updateAttributeEntry_20( D3Interface d3, Document doc, User user, DocumentType docTypeShort,
                                      DocumentType docTypeShortNew ){
//(5)
    return getCustomerData( d3, doc );
  } // end of updateAttributeEntry_20
  public int getCustomerData( D3 d3, Document doc ){
//(6)
    def customerID = doc.field[14];
    if( customerID != "" ){
      def dbConnection = Sql.newInstance( "jdbc:sqlserver://localhost:1433;databaseName=Name",
                                          "User", "Password" );
      def sqlQuery    = "SELECT name, street, zipCode, City FROM CustomerData WHERE customerNo = ? ";
      def sqlParams   = [ customerID ];
      def resultRows  = dbConnection.rows( sqlQuery, sqlParams );
      if( resultRows.size() == 1 ){
//(7)
        doc.field[8]  = resultRows[0].street;
        doc.field[9]  = resultRows[0].zipCode.toString(); // ATTENTION!
        doc.field[10] = resultRows[0].city;
        return 0;
      }
      else {
//(8)
        return -1;
      }
    }
    return  0;
  } // end of getCustomerData
} // end of D3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der **groovyhook.jar**-Datei.
2. Bereitstellung einer eigenen Klasse vom Typ "public", wobei "public" im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen erfolgt, über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt. Des weiteren können an dieser Stelle auch über eine sog. **Condition** auch Einschränkungen auf die Dokumenttypen vorgenommen werden.
4. Die unterschiedlichen Eintrittstypen, benötigen dann eine definierte Anzahl von Parametern in einer vorgegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben.
   Für die Hook-Eintrittspunkte Wertemengen, Validierung und Suche muss immer am Ende noch ein zusätzlicher Parameter vom Typ "Document" angehängt werden.
5. Nun kann innerhalb der Funktion für einen bestimmten Dokumenttyp eine Validierung vorgenommen werden, dabei wird der Dokumenttyp über die Eigenschaft **pDocTypeShort.id** eingeschränkt.
6. Nachdem die Kunden-Nummer aus den Dokumenteigenschaften ermittelt wurde, kann nun die Kunden-Nummer gegen die Datenbank geprüft werden. Dabei wird in diesem Beispiel eine Verbindung zu einer externen MS-SQL-Datenbank vorgenommen und die Daten aus dieser Datenbank ermittelt. Details zu der notwendigen Konfiguration können den relevanten Dokumentationen entnommen werden.
7. Da im Kontext des Eintrittspunktes **hook_upd_attrib_entry_20** alle Dokumenteigenschaften sowohl lesend als auch schreibend zur Verfügung stehen, können nun die zusätzlichen Kundendaten ergänzt werden.
8. Konnte keine gültige Kunden-Nummer ermittelt werden, erfolgt eine Rückgabe mit einem Returnwert ungleich "0" und damit wird dann an dieser Stelle die Verarbeitungskette unterbrochen.

10.1.4 Eine Klasse, mehrere Hook-Funktionen

Wie können mehrere Hook-Funktionen in einer Klasse zusammengeführt werden? Eine Antwort könnte mit dem folgenden Skript-Beispiel gegeben werden.

```
// (1) Global d.3 libraries ----------------------------------------------------
import java.sql.Timestamp;
import com.dvelop.d3.server.Condition;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.DocumentType;
import com.dvelop.d3.server.Entrypoint;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.core.D3Interface;
import groovy.sql.Sql;

//------------------------------------------------------------------------
// (2)
public class D3EntryPoints{
  // (3)
  @Entrypoint( entrypoint = "hook_insert_entry_10" ) //----------------------
  // (4)
  public int getCustomerDataForInvoice( D3Interface d3, User user, DocumentType docTypeShort, Document
doc ){
  // (5)
    if( docTypeShort.id == "DINV" ){
      d3.log.error("Hello world!");
      //return getCustomerData( d3, doc );
    }
    return 0;
  } // end of getCustomerDataForInvoice
  // (6)
  @Entrypoint( entrypoint = "hook_insert_exit_20" ) //----------------------
  @Condition( doctype = [ "DINV" ] )
  public int sendInvoice( D3Interface d3, Document doc, def fileDest, def importOK, User user, DocumentType
docTypeShort ){
    // Define variables ------------------------------------------
    def returnValue = 0;
    Timestamp currentDate = new Date().toTimestamp();
    // Add entry to document note --------------------------------------
    returnValue = d3.call.note_add_string( "Hello World!", doc.id, user.id );
    // SEND to user ----------------------------------------------------
    returnValue = d3.call.hold_file_send( user.id, "Invoice: " + doc.field[14], doc.id, currentDate, currentDate,
false, true, currentDate, "", "dvelop", 0, false, false, currentDate, 0, false);
  } // end of  sendInvoice

  // (7)
  @Entrypoint( entrypoint = "hook_validate_import_entry_10" ) //-------------
  @Condition( doctype = [ "DINV" ] )
  public int justDummy( D3Interface d3, User user, DocumentType docTypeShort, Document doc ){
    return 0;
  } // end of  justDummy

  // (8)
  @Entrypoint( entrypoint = "hook_upd_attrib_entry_20" ) //------------------
  public int updateCustomerDataForInvoice( D3Interface d3, Document doc, User user, DocumentType
docTypeShort, DocumentType docTypeShortNew  ){
    if( docTypeShort.id == ["DINV" ] ){
      def oldDocOnj = d3.archive.getDocument( doc.id ); // TODO alte inhalte
      return getCustomerData( d3, doc );
```

```
    }
    return 0;
  } // end of updateCustomerDataForInvoice

  // (9) ------------------------------------------------------------------
  public int getCustomerData( D3Interface d3, Document doc ){

    def customerID = doc.field[1];
    if( customerID != null && customerID != "" ){
      def dbConnection = Sql.newInstance( "jdbc:sqlserver://Teilnehmer-VM.training.d-velop.de\
\SQLEXPRESS:0;databaseName=SolutionsDB", "dEcsFormsDBUser", "Academy1!" );
      def sqlQuery   = "SELECT name, street, zipCode, city FROM CustomerData WHERE customerNo = ? ";
      def sqlParams  = [ customerID ];
      def resultRows = dbConnection.rows( sqlQuery, sqlParams );
      //def resultRows = d3.sql.executeAndGet( sqlQuery, sqlParams );

      if( resultRows.size() == 1 ){

        doc.field["Straße"]      = resultRows[0].street.trim();
        doc.field["Postleitzahl"] = resultRows[0].zipCode.toString(); // ATTENTION!
        doc.field[5]             = resultRows[0].city.trim();
        doc.field[1]             = resultRows[0].name.trim();

        doc.setText( 1, "Inhalt von Groovy" );
        return 0;
      }
      else{
        return -1;
      }
    }
    return 0;
  } // end of getCustomerData
} // end of d3Hooks
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ public, wobei public im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen dann, die in der Groovy-Hook-Dokumentation dafür beschriebenen Parameter in der dort angegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben. Der Funktionsname muss nicht, wie im JPL dem Eintrittspunkt entsprechen. Der Funktionsname **doSomething** muss hier natürlich mit einem "sprechenden" Namen vergeben werden.
5. In diesem Beispiel wird der Dokumenttyp innerhalb der Funktion über ein If-Statement kontrolliert und für die Dokumentart Rechnung eine entsprechende Aktion ausgeführt.
6. Ein weiteres Beispiel wird auf den Eintrittspunkt "**InsertExit_20**" und die Dokumentart Rechnung "DRECH" realisiert. In diesem Beispiel wird über die Server-API-Calls "note_add_string" bei Eingang

einer Rechnung ein Eintrag in den Dokument-Notizen vorgenommen. Zusätzlich wird die Rechnung noch mittels des Befehls "hold_file_send" an einen Sachbearbeiter gesendet.

7. Für einen weiteren Eintrittspunkt **hook_validate_import_entry_10** wird hier einfach mal eine Funktion registriert, welche aber aktuell noch keine Aufgabe übernimmt.
8. Wird, zum Beispiel in einer Rechnung, die Kundennummer geändert, müssen auch die relevanten Kundendaten in der Rechnung ebenfalls angepasst werden, dazu wird der Eintrittspunkt **hook_upd_attrib_entry_20** mit einer Funktion verlingt.
9. Da die Ermittlung der Kundendaten an mehreren Stellen benötigt wird, wurde die Funktion an dieser Stelle implementiert.

10.2 Validierung

> **Wichtig**
>
> Für die Nutzung einer Validierung müssen diese Bibliotheken importiert werden:
>
> **Globale d.3-Bibliotheken**
>
> - import com.dvelop.d3.server.core.D3Interface
>
> **Spezifische Bibliotheken für die Validierung**
>
> - import com.dvelop.d3.server.Validate

Nützliche Links

- Regular-Expression Online Tool: https://regex101.com/

Validierung einer Bestellnummer auf ein gültiges Format

> **Hinweis**
>
> Szenario:
>
> Die Bestellnummer soll immer dem Format "Zwei Zahlen-Zwei Buchstaben-Fünf Zahlen" (/[0-9]{2}-[a-zA-Z]{2}-[0-9]{5}/) genügen. Natürlich kann man das direkt in der d.3 Administration konfigurieren, aber als Beispiel um die Funktionsweise für die Validierung zu demonstrieren, ist es ebenfalls geeignet.

Zur Realisierung wird auf die Dokumenteigenschaft Bestellnummer eine Funktion zur Validierung definiert.

```groovy
package com.dvelop.hooks;

//(1)
//Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;

// Libraries to handle the diferent hook types
import com.dvelop.d3.server.Validation;
//(2)
public class D3Validate{
//(3)
  @Validation( entrypoint = "checkOrderNumber" )
//(4)
  public int checkOrderNumber( D3Interface d3, def currentValue, Document doc ){
//(5)
    def tmpValue  = currentValue;
    def matchFlag = ( tmpValue ==~ /[0-9]{2}-[a-zA-Z]{2}-[0-9]{5}/ );
//(6)
    return( matchFlag ? 0 : -1 );
  } // end of checkOrderNumber
}// end of D3Validate
```

Das Ganze, kann man auch, dank Groovy, etwas kürzer realisieren.

```groovy
package com.dvelop.hooks;

//(1)
// Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;

// Libraries to handle the diferent hook types
import com.dvelop.d3.server.Validation;
//(2)
 public class D3Validate{
//(3)
   @Validation( entrypoint = "checkOrderNumber" )
//(4)
  public int checkOrderNumber( D3Interface d3, def currentValue ){
//(6)
    return( ( currentValue ==~ /[0-9]{2}-[a-zA-Z]{2}-[0-9]{5}/ ) ? 0 : -1 );
  } // end of checkOderNumber
}// end of D3Validate
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der **groovyhook.jar**-Datei.
2. Bereitstellung einer eigenen Klasse vom Typ "public", wobei "public" im Kontext Groovy auch weggelassen werden kann.
3. Um nun eine Groovy-Funktion für eine Validierung einer Dokumenteigenschaft nutzen zu können erfolgt, über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einer, in d.3 admin konfigurierten, Validierungsfunktion.

4. Die Validierungs-Funktion benötigt dann eine definierte Anzahl von Parametern in einer vorgegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben.
5. Innerhalb der Funktion kann nun der übergebene Wert überprüft werden, im Beispiel mittels eines regulären Ausdrucks.
6. Entspricht der Wert einem gültigen Wert, kann eine 0 ansonsten eine 1 zurückgegeben werden.

10.3 Wertemengen

In diesem Bereich werden die Möglichkeiten dargestellt, Wertemengen wie folgt bereitzustellen:

- als statische Liste
- aus internen d.3-Datenbanktabellen
- als dynamische abhängige Datenbanktabelle

Benötigte Bibliotheken für die Wertemengen

> **Wichtig**
>
> Für die Implementierung von Wertemengen müssen diese Bibliotheken importiert werden:
>
> **Globale d.3-Bibliotheken**
>
> - import com.dvelop.d3.server.core.D3Interface
> - import com.dvelop.d3.server.Document
> - Import com.dvelop.d3.server.User
> - import com.dvelop.d3.server.DocumentType
>
> **Spezifische Bibliotheken für die Wertemengen**
>
> - import com.dvelop.d3.server.ValueSet
> - import com.dvelop.d3.server.RepositoryField

> **Wichtig**
>
> Die Sortierung wird hier aus dem Script übernommen und nicht, wie über eine JPL-Wertemenge, vom Client wieder verfälscht.

Einfache statische Wertemenge

> **Hinweis**
>
> Natürlich macht es wenig Sinn eine statische Liste über ein Skript bereitzustellen, dies kann man viel besser über die d.3-Administration! Aber für ein einfaches Beispiel kann man hier mit einer statischen Wertemenge starten.

```
1  //(1) Global d.3 libraries ------------------------------------------------------
2  import com.dvelop.d3.server.core.D3Interface;
3  import com.dvelop.d3.server.Document;
4  import com.dvelop.d3.server.User;
5  import com.dvelop.d3.server.DocumentType;
6
7  //(2) Libraries to handle the different hook types ----------------------------
8  import com.dvelop.d3.server.ValueSet;
9
10 //(3) Special libraries ---------------------------------------------------------
11 import com.dvelop.d3.server.RepositoryField;
12
13 //(4) Define the needed class ---------------------------------------------------
14 class SimpleValueSet{
15   //(5) Combine to the value set entry point ---------------------------------
16   @ValueSet( entrypoint = "customerNumbers" )
17   //(6) Define the function ------------------------------------------------
18   def getCustomerNumber( D3Interface d3, RepositoryField reposField, User user, DocumentType
docType,
19                          int rowNo, int validate, Document doc ){
20
21     //(7) Define static list of customer numbers ------------------------------
22     def customerList = ["4711", "4712", "4713", "4714" ];
23
24     //(8) Prepare List for interaction-----------------------------------------
25     if( customerList.size() > 0 ){
26       reposField.provideValuesForValueSet( customerList );
27     }
28   } // end of getCustomerNumber
29 }// end of SimpleValueSet
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der **groovyhook.jar**-Datei.
2. Import der speziellen Bibliothek zur Unterstützung der Wertemengen.
3. Import einer speziellen Bibliothek zur Rücklieferung der Wertemenge an den Benutzer.
4. Definition einer öffentlichen Klasse.
5. Registrierung des Eintrittspunktes für die Wertemenge.
6. Definition der speziellen Funktion zur Ermittlung der Wertemenge.
7. Definition der statischen Wertemenge.
8. Bereitstellung der Wertemenge für den User.

Einfache statische Wertemenge aus einer Datenbanktabelle

Sinnvoller ist es, die Wertemengen aus internen bzw. ext, Datenquellen zu ermittelt; dies wird in diesen Beispielen vorgestellt.

Interne d.3-Datenbanktabelle

> **Hinweis**
>
> In diesem Beispiel werden die Daten aus einer internen d.3-Datenbanktabelle ermittelt, dabei werden eventuelle Eingaben des Users nicht berücksichtigt.

```
//(1) Global d.3 libraries ------------------------------------------------
Import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;

//(2) Libraries to handle the different hook types ------------------------
import com.dvelop.d3.server.ValueSet;

//(3) Special libraries ---------------------------------------------------
Import com.dvelop.d3.server.RepositoryField;

//(4) Example for an internal d.3 database table --------------------------
class StaticValueSet{
  //(5) Define the value set entry point ----------------------------------
  @ValueSet( entrypoint = "customerNumbers" )
  //(6) Define function for the value set ---------------------------------
  def getCustomerNumber( D3Interface d3, RepositoryField reposField, User user, DocumentType docType,
                int rowNo, int validate, Document doc ){

    //(7) Prepare sql statmenet -------------------------------------
    def sqlQuery = "SELECT custoemrNo FROM CustomerData ORDER BY customerNo DESC"; // !! ATTENTION

    //(8) Execute sql statmenet -------------------------------------
    def resultRows = d3.sql.executeAndGet( (String) sqlQuery );

    //(9) Prepare list for user interface ---------------------------
    if( resultRows.size() > 0 ){
      reposField.provideValuesForValueSet( resultRows.collect{ it.customerNo } );
    }
  } // end of getCustomerNumber
}// end of StaticValueSet
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Import der speziellen Bibliothek zur Unterstützung der Wertemengen.
3. Import einer speziellen Bibliothek zur Rücklieferung der Wertemenge an den Benutzer.
4. Definition einer öffentlichen Klasse.
5. Registrierung des Eintrittspunktes für die Wertemenge.
6. Definition der speziellen Funktion zur Ermittlung der Wertemenge.
7. Definition des SQL-Statements zur Ermittlung der Wertemenge.

8. Ausführung des SQL-Statements gegen die interne Datenbank-Tabelle, wird hier über das d.3-Objekt realisiert.
9. Darstellung der Wertemenge für den User.

Externe Datenbanktabelle

> **Hinweis**
>
> Wertemengen können auch aus externen Datenbanken ermittelt werden; dieses Beispiel zeigt einen Lösungsansatz.

```
//(1) Global d.3 libraries ------------------------------------------------------
Import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;

//(2) Libraries to handle the different hook types ------------------------------
import com.dvelop.d3.server.ValueSet;

//(3) Special libraries ---------------------------------------------------------
Import com.dvelop.d3.server.RepositoryField;

//(4) Example for an external database table ------------------------------------
class StaticValueSet{
  //(5) Define the value set entry point ----------------------------------------
  @ValueSet( entrypoint = "customerNumbers" )
  //(6) Define the function -----------------------------------------------------
  def getCustomerNumber( D3Interface d3, RepositoryField reposField, User user, DocumentType docType,
                int rowNo, int validate, Document doc ){

    //(7) Prepare database Connection ---------------------------------------
    def dbConnection = Sql.newInstance( "jdbc:sqlserver:<ServerName>\
\<InstanceName>:0;databaseName=<DatabaseName>", "<User>", „<Password>" );

    //(8) Prepare sql statmenet----------------------------------------------
    def sqlQuery  = "SELECT customerNo FROM CustomerData ORDER BY customerNo DESC"; // !! ATTENTION

    //(9) Execute sql statmenet ---------------------------------------------
    def resultRows = dbConnection.rows( (String) sqlQuery );

    //(10) Prepare lit for user interface  ----------------------------------
    if( resultRows.size() > 0 ){
      reposField.provideValuesForValueSet( resultRows.collect{ it.customerNo } );
    }

  } // end of getCustomerNumber
}// end of StaticValueSet
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Import der speziellen Bibliothek zur Unterstützung der Wertemengen.
3. Import einer speziellen Bibliothek zur Rücklieferung der Wertemenge an den Benutzer.
4. Definition einer öffentlichen Klasse.
5. Registrierung des Eintrittspunktes für die Wertemenge.
6. Definition der speziellen Funktion zur Ermittlung der Wertemenge.
7. Initialisierung einer externen JDBC-Datenbankverbindung.
8. Definition des SQL-Statements zur Ermittlung der Wertemenge.
9. Ausführung des SQL-Statements gegen die interne Datenbank-Tabelle, wird hier über das d.3-Objekt realisiert.
10. Darstellung der Wertemenge für den User.

Abhängige dynamische Wertemenge aus einer Datenbanktabelle

Bereitstellung einer Kunden-Nummern-Liste

> **Hinweis**
>
> Szenario:
>
> Es soll eine Wertemengen der Kunden-Nummern aus den Kunden-Datenbanktabelle für die Dokumenteigenschaft "KundenNr" bereitgestellt werden. Diese soll bei Bedarf über den eingegebenen PLZ-Bereich dynamisch begrenzt werden können.

Zur Realisierung wird auf die Dokumenteigenschaft Bestellnummer eine Funktion zur Validierung definiert.

```
1  //(1) Global d.3 libraries ----------------------------------------------------
2  import com.dvelop.d3.server.core.D3Interface;
3  import com.dvelop.d3.server.Document;
4  import com.dvelop.d3.server.User;
5  import com.dvelop.d3.server.DocumentType;
6
7  // Libraries to handle the different hook types
8  Import com.dvelop.d3.server.ValueSet;
9
10 // Special libraries
11 Import com.dvelop.d3.server.RepositoryField;
12 //(2)
13 public class D3ValueSets{
14 //(3)
15   @ValueSet( entrypoint = "dsCustomerNumbers" )
16 //(4)
17   public int getCustomerNumber( D3Interface d3, RepositoryField reposField, User user,
   DocumentType docType, int  rowNo, int  validate, Document doc )
18   {
19 //(5)
20     def  zipCode   = doc.field[9];
21     def  whereClause = "";
22     if( zipCode != "" ){
23       whereClause = "WHERE zipCode LIKE '$zipCode%'";
24     }
25 //(6)
26     def sqlQuery   = "SELECT customerNo FROM CustomerNo $whereClause ORDER BY
   customerNo DESC"; // !! ATTENTION
27     def resultRows = d3.sql.executeAndGet( sqlQuery );
28 //(7)
29     // Long-Version ----------------------------------------------------
30     def customerList = [];
31     resultRows.each{
32       customerList.add( it.customerNo );
33     }
34     if( customerList.size() > 0  ){
35       reposField.provideValuesForValueSet( customerList );
36     }
37     // Short-Version ----------------------------------------------------
38     if( resultRows.size() > 0 ){
39       reposField.provideValuesForValueSet( resultRows.collect{ it.customerNo } );
40     }
41
42     return 0
43   } // end of getCustomerNumber
44 } // end of D3ValueSets
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ "public", wobei "public" im Kontext Groovy auch weggelassen werden kann.

3. Um nun eine Groovy-Funktion für Wertemengen-Generierung einer Dokumenteigenschaft nutzen zu können erfolgt, über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einer, in d.3 admin konfigurierten, Wertemengenfunktion.
4. Die Wertemengen-Funktion benötigt dann eine definierte Anzahl von Parametern in einer vorgegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben.
5. Um die Aufgabenstellung zu realisieren, wird nun die Dokumenteigenschaft der PLZ ausgelesen und bei Bedarf ein "Where"-Bestandteil des SQL-Statements zur Verfügung gestellt.
6. Da die genutzte Datenbanktabelle innerhalb der d.3-Datenbank angelegt wurde, kann mittels einer einfachen SQL-Implementierung, siehe hierzu auch das Beispiel für **InsertEntry_10**, auf die Daten zugegriffen werden.
7. Wurden nun Daten zu dem bereitgestellten Postleitzahlen-Bereich gefunden, können diese mittels der Funktion **provideValuesForValueSet()** der Dokumenteigenschaft bereitgestellt werden. Hier wurden zwei unterschiedliche Wege der Implementierung aufgenommen; natürlich sollte nur ein Weg verwendet werden.

10.4 Dokumentklassen

> **Wichtig**
>
> Für die Implementierung von Wertemengen müssen diese Bibliotheken importiert werden:
>
> **Globale d.3-Bibliotheken**
>
> - import com.dvelop.d3.server.core.D3Interface
> - import com.dvelop.d3.server.Document
> - Import com.dvelop.d3.server.User
> - import com.dvelop.d3.server.DocumentType
>
> **Spezifische Bibliotheken für den Dokumentklassen-Hook**
>
> - import com.dvelop.d3.server.DocumentClass

Rechnungen nur von bestimmten Usern bearbeitbar

> **Hinweis**
>
> Szenario:
>
> Rechnungen sollen nun, abhängig von der 3. Stelle der Kundennummern, nur von bestimmten Benutzern bearbeitet werden können. Da in der Demo-Umgebung keine entsprechende Datenverlinkung zwischen User und Kundennummer vorhanden ist, wird die Implementierung hier exemplarisch mittels einer statischen Zuordnung über ein "Switch"-Statement realisiert.

Zur Realisierung wird eine neue Dokumentklasse "KundenRechnungen" in der d.3-Administration angelegt, die benötigte Implementierung ist im folgenden Beispiel dokumentiert.

```groovy
package com.dvelop.hooks;

//(1)
// Global d.3 libraries
import com.dvelop.d3.server.core.D3Interface;
import com.dvelop.d3.server.Document;
import com.dvelop.d3.server.User;
import com.dvelop.d3.server.DocumentType;

// Libraries to handle the different hook types
import com.dvelop.d3.server.DocumentClass;
//(2)
public class d3DocumentClass{
//(3)
  @DocumentClass( entrypoint = "KundenRechnung" )
//(4)
  pudblic int customerDocClass( D3Interface d3, String value, DocumentType docType, String userId, Document doc ){
    // STEP 1: Get customer number
    def customerNo  = value;
    // STEP 2: Get current user
    def currentUser = userId;
    // STEP 3: Set user depending on the third place in the customer number
    def returnFlag = false;
    def tmpValue = customerNo[2];
//(5)
    switch( tmpValue ){
      case "0":
        returnFlag = ( currentUser == "chef" );
        break;
      case "1":
        returnFlag = ( currentUser == "smith" );
        break;
      case "2":
        returnFlag = ( currentUser == "larson" );
        break;
      case "3":
        returnFlag = ( currentUser == "parker" );
        break;
      case "4":
        returnFlag = ( currentUser == "funny" );
        break;
      default:
        break;
    }
    return( returnFlag ? 1 : 0 );
  } // end of customerDocClass
} // end of d3DocumentClasss
```

Kommentare zu den einzelnen Blöcken

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ "public", wobei "public" im Kontext Groovy auch weggelassen werden kann.

3. Um nun eine Groovy-Funktion für Dokumentklassen-Hooks nutzen zu können wird über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu einer, in d.3 admin konfigurierten, Dokumentklasse vorgenommen..
4. Die Dokumenklassen-Funktion benötigt dann eine definierte Anzahl von Parametern in einer vorgegebenen Reihenfolge. Als erster Parameter wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übergeben.
5. In diesem Beispiel wird nun über eine Switch-Abfrage abhängig von dem dritten Wert der Kundennummer ein statischer User gesetzt. Hier könnte man natürlich auch mit "Restriction-Sets" arbeiten.

> **Wichtig**
> Bitte beachten Sie hier das Dokumentklassen-Hooks mit viel Vorsicht genutzt werden sollten, da diese je nach implementierter Funktion, sehr auf die Performance des Gesamtsystems gehen!

6. Je nach dem wie diese Prüfung abgelaufen ist, wird nun eine "1" (erlaubt) oder eine "0" (verweigert) zurückgegeben.

Index

T

U

V

W