d.velop

DSSAGCurrent2023Q4 (d.3 hook & server scripting api (groovy))

Inhaltsverzeichnis

1. d.3 hook & server scripting api (groovy)

1.1. Einleitung

1.1.1. Über diese Dokumentation

In dieser Dokumentation erfahren Sie, wie Sie d.3-Hook-Funktionen und Skripte mit Groovy entwickeln und welche Schnittstellen und Funktionen d.3 server ab Version 8.1.0 dafür bereitstellt.

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 in d.velop documents abgelegten und konfigurierten Daten erhält und Eingriff in die systeminternen Abläufe nimmt. Gehen Sie daher sorgfältig vor und beachten Sie, 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 fällt in den Bereich der Individualentwicklung. Der von Ihnen erstellte Programmcode unterliegt nicht den Pflege- und Supportbedingungen der Produkte der d.velop AG. Der d.velop-Support unterstützt Sie bei Fragen gern. Wenn sich Ihre Anfrage nicht auf einen Fehler in unseren Produkten zurückführen lässt, ist dieser Support kostenpflichtig.

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

1.1.2. Voraussetzungen

Um alle Java- bzw. Groovy-Funktionalitäten verwenden zu können, müssen Sie die Java- bzw. Groovy-Unterstützung in d.3 config aktivieren.

> **Anmerkung**
>
> Die Java- bzw. Groovy-Unterstützung ist in d.3 Server ab Version 8.0.0 standardmäßig aktiviert. d.3 server enthält die bereits aktivierte Java-Laufzeitumgebung von Sun/Oracle.
>
> Weitere Informationen erhalten Sie der Dokumentation zu d.3 admin.

d.3 server enthält die bereits aktivierte Java-Laufzeitumgebung von Sun/Oracle ab Version 8.0.0.

1.1.3. Groovy

Groovy ist eine dynamische Skriptsprache für Java Virtual Machine. Durch die enge Integration mit Java können Sie auf sämtliche Inhalte des Java-Umfelds inklusive vieler umfangreicher Bibliotheken zugreifen.

Groovy umfasst Funktionalitäten, die in Java nicht vorhanden sind. Dazu zählen unter anderem Native Syntax für Maps, Listen und reguläre Ausdrücke sowie ein einfaches Templatesystem, mit dem Sie HTML- und SQL-Code erstellen können. Des Weiteren bietet Groovy eine XQuery-ähnliche Syntax zum Ablaufen von Objektbäumen, eine 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. Groovy ähnelt mehr der Syntax von Ruby und Python als der Syntax von Java oder BeanShell.

Weitere Informationen zu Groovy finden Sie auf der offiziellen Groovy-Website http://www.groovy-lang.org/.

1.2. Entwicklungsumgebung

Bei Groovy-Code handelt es sich um Textdateien, die Sie mit einem einfachen Texteditor bearbeiten können.

Für ein effektives Entwickeln von Hooks empfehlen wir Ihnen jedoch die Benutzung einer Entwicklungsumgebung wie Eclipse, die Anwendende durch Funktionen wie eine automatische Code-Vervollständigung unterstützen.

Die Java-Klassen der d.3-Server-Schnittstelle und Groovy-Unterstützung sowie der Groovy-Interpreter befinden sich im Java-Archiv **groovyhook.jar** im d.3 server-Programmverzeichnis (standardmäßig unter C:\d3\d3server.prg). In Eclipse können Sie das Java-Archiv unter **Project Properties > Java Build Path als External JAR** einbinden.

Installieren Sie in diesem Zusammenhang ein Groovy-Plug-in für Ecplise, damit Sie die bestmögliche Unterstützung für die Syntax-Hervorhebung und Kommandoergänzung erhalten.

> **Anmerkung**
> Aktivieren Sie beim Entwickeln und Testen von Groovy-Programmcode den d.3 config-Parameter **Neuladen von Groovy-Hookdateien bei Änderung aktivieren** (**RELOAD_ON_CHANGE**).
>
> Dies führt beim Speichern von Groovy-Hookdateien dazu, dass die Groovy-Hookdateien automatisch von den Serverprozessen neu geladen werden und die Code-Änderungen sofort aktiv sind.

> **Warnung**
> Aktivieren Sie die Neuladen-Option aus Sicherheitsgründen nicht im Produktivsystem. Andernfalls würden Änderungen an den Groovy-Skripten unmittelbar in Ihrem Produktivsystem aktiv werden.

1.2.1. Verwenden von Eclipse als Entwicklungsumgebung

Wenn Sie Eclipse als Entwicklungsumgebung für die Hook-Entwicklung verwenden möchten, müssen Sie Eclipse zunächst konfigurieren.

So geht's

1. Laden Sie das Java Development Kit (JDK) herunter und führen Sie die Installation durch.

   > **Warnung**
   > Durch die Änderung der Lizenzbedingungen durch Oracle ist die Java-Distribution nicht mehr kostenlos. Dies gilt für JDK/JRE-Updates, die nach Januar 2019 bei Oracle bezogen wurden.
   >
   > Alle betroffenen Produkte werden künftig mit OpenJDK (https://openjdk.java.net/) ausgeliefert, während Hotfixes das Oracle Java 8-Update 201 verwenden.

2. Laden Sie Eclipse IDE for Java Developers unter http://www.eclipse.org/ herunter und entpacken Sie die Datei.
3. Öffnen Sie **eclipse.exe**.

4. Wählen Sie ein geeignetes Verzeichnis für Ihren Workspace aus. Der Workspace ist ein Verzeichnis, in dem Eclipse Ihre Projekte verwaltet. Geben Sie kein Verzeichnis des d.3-Servers an, sondern ein Verzeichnis in Ihren eigenen Dateien.

Nachdem Sie Eclipse gestartet haben, installieren Sie anschließend das Groovy-Plug-in für Eclipse.

So geht's

1. Wählen Sie unter **Help** die Option **Install New Software...** aus.
2. Bestimmen Sie unter https://github.com/groovy/groovy-eclipse/wiki die Ihrer Eclipse-Version entsprechende Update-Site aus. Eine Update-Site ist eine Webadresse, über die Eclipse automatisch Software installieren kann.
3. Tragen Sie unter **Work with** Ihre zuvor ausgewählte Update-Site ein und bestätigen Sie Ihren Eintrag.
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.

> **Anmerkung**
> Nach dem ersten Start von Eclipse in einem neu erstellten Workspace müssen Sie ggf. von der Willkommensseite mit **Workbench** zur Standardansicht wechseln.
>
> **Zielsysteme ohne Internetzugang**
>
> Sie können die Konfiguration der Eclipse-Entwicklungsumgebung auch auf einem separaten Rechner durchführen. Wenn Sie die Konfiguration erfolgreich abgeschlossen haben, können Sie den Eclipse-Ordner ohne zusätzliche Installation in Ihr Zielsystem kopieren.

Erstellen von Hook-Projekten

Um Hooks zu programmieren, müssen Sie ein Hook-Projekt erstellen.

So geht's

1. Navigieren Sie zu **File > New > Projekt**.
2. Wählen Sie **Groovy Projekt** aus und vergeben Sie auf der nächsten Seite einen sprechenden Namen.
3. Fügen Sie ein neues Quelltext-Verzeichnis hinzu, indem Sie unter **Build Settings** nach einem Rechtsklick auf das Projekt die Option **Link Source** auswählen.
4. Geben Sie unter **Linked folder location** das in d.3 admin definierte Verzeichnis für Ihre Groovy-Hooks an (siehe Groovy-Hook-Verzeichnisse für kundenspezifische Programmanpassungen).
5. Vergeben Sie unter **Folder name** einen sprechenden Namen (z.B. **Hooks**) und bestätigen Sie den Namen mit **Finish**.
6. Erstellen Sie das Projekt mit **Finish** .
7. Löschen Sie in Ihrem soeben erstellten Projekt den Ordner **src**.
8. Suchen Sie im Installationsverzeichnis Ihres d.3-Servers (standardmäßig C:\d3\d3server.prg) die Datei **groovyhook.jar** und kopieren Sie die Datei in Ihr Eclipse-Projekt.
9. Fügen Sie **groovyhook.jar** hinzu, indem Sie einen Rechtsklick auf **Build Path** vornehmen und **Add to Build Path** auswählen. Wenn sowohl die Eclipse-Entwicklungsumgebung als auch die d.3 server-Installation auf dem gleichen Rechner installiert sind, können Sie **groovyhook.jar** auch ohne Kopieren direkt als External JAR einbinden.
10. Fügen Sie **groovyhook-javadoc.jar** zum Build Path hinzu, indem Sie für **groovyhook.jar** unter **Libraries** die Option **Javadoc location** bearbeiten und die Datei unter **Javadoc in archive** als **External file** angeben.

> **Warnung**
> Wenn die für das Groovy-Projekt automatisch in den Java-Build Path aufgenommene Version der Groovy-Bibliotheken neuer ist als die in der **groovyhook.jar** enthaltene Bibliothek, meldet Eclipse einen Versionskonflikt. In diesem Fall müssen Sie in der Registerkarte **Libraries** die Einträge **Groovy DSL Support** und **Groovy Libraries** aus dem Java-Build Path entfernen.

11. Erstellen Sie neue Hook-Klassen, indem Sie im Kontextmenü des Ordners **Hooks** unter **New | Other** den Typ **Groovy Class** auswählen.
12. Geben Sie Ihrem Hook einen sprechenden Namen und bestätigen Sie den Namen mit **Finish**.

Sie haben erfolgreich ein Hook-Projekt erstellt, für das Sie nun Ihre Hook-Funktionen implementieren können.

> **Anmerkung**
> Wenn Sie in d.3 admin konfiguriert haben, dass Hooks bei Änderungen automatisch neu geladen werden, müssen Sie in Eclipse nur speichern, um Änderungen zu testen. Wenn Sie das automatische Neuladen deaktiviert haben, müssen Sie Ihre Repositoryprozesse neu starten, um Änderungen zu übernehmen.

1.2.2. Verwenden von IntelliJ IDEA als Entwicklungsumgebung

Wenn Sie IntelliJ IDEA für Hook-Entwicklung als Entwicklungsumgebung verwenden möchten, müssen Sie IntelliJ Idea zuvor konfigurieren.

So geht's

1. Laden Sie das Java Development Kit (JDK) herunter und führen Sie die Installation durch.

> **Warnung**
> Durch die Änderung der Lizenzbedingungen durch Oracle ist die Java-Distribution nicht mehr kostenlos. Dies gilt für JDK/JRE-Updates, die nach Januar 2019 bei Oracle bezogen wurden.
>
> Alle betroffenen Produkte werden künftig mit OpenJDK (https://openjdk.java.net/) ausgeliefert, während Hotfixes das Oracle Java 8-Update 201 verwenden.

2. Laden Sie die kostenfreie Community-Edition von IntelliJ IDEA herunter (https://www.jetbrains.com/idea/download/#section=windows und führen Sie die Installation durch.
3. Laden Sie die Groovy-Bibliothek herunter (http://groovy-lang.org/download.html) und führen Sie die Installation durch.
4. Starten Sie IntelliJ IDEA.
5. Wählen Sie ein geeignetes Verzeichnis für ihr Projekt aus.
6. Klicken Sie auf den Projektordner und wählen Sie unter **Mark Directory as** die Option **Sources Root** aus.

Bekanntgeben der Groovy-API von d.3

Geben Sie als Nächstes die Groovy-API in Ihrem Projekt bekannt.

So geht's

1. Navigieren Sie zu **File > Project Structure**.
2. Fügen Sie unter **Libraries** die Datei **groovyhook.jar** aus dem Programmverzeichnis von d.3 server hinzu.

3. Klicken Sie auf das Plus-Symbol und wählen Sie **Java** aus.
4. Wählen Sie **groovyhook.jar** aus dem Programmverzeichnis von d.3 server aus. Sie erhalten eine Meldung, dass die Bibliothek zu Ihrem Projekt hinzugefügt wird.
5. Bestätigen Sie die Meldung mit **OK**.
6. Speichern Sie Ihre Einstellungen.

Sie haben die Konfiguration von IntelliJ IDEA erfolgreich abgeschlossen und die Groovy-API von d.3 bekanntgegeben. Sie können nun Groovy-Hooks mithilfe der IntelliJ-Entwicklungsumgebung entwickeln.

Weitere Informationen zu IntelliJ IDEA

Die Dokumentation sowie Tutorials zu IntelliJ IDEA finden Sie unter https://www.jetbrains.com/idea/documentation/.

1.3. Grundlagen zu Groovy

In diesem Kapitel erhalten Sie eine Übersicht zu grundlegenden Informationen zum Umgang mit Groovy, die Ihnen den Einstieg erleichtern.

Groovy-Dokumentationen

http://www.groovy-lang.org/

http://www.groovy-lang.org/style-guide.html

http://grails.asia/groovy-list-tutorial-and-examples

Einführung in Groovy

http://www.oio.de/public/java/groovy/groovy-einfuehrung.htm

http://www.oio.de/public/java/groovy-closures-artikel.htm

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/

http://mrhaki.blogspot.de/2009/10/groovy-goodness-finding-data-in.html

Groovy vs. Java

http://www.groovy-lang.org/differences.html

1.3.1. Variablen und Strings

In diesem Kapitel finden Sie Informationen zur Verwendung von Variablen und Strings.

> **Anmerkung**
> Sie können Inhalte mit dem Befehl **println** ausgeben. In d.3 wird dieser Befehl als Infomeldung in der Protokolldatei ausgegeben. Alternativ können Sie auch direkt die Protokollfunktion **d.log.info (..)** verwenden.

Definieren von Variablen

Sie können Variablen mit Groovy mittels dynamischer Typisierung definieren, indem Sie das Schlüsselwort **def** angeben. In Groovy können Sie zudem auch mit den gängigen Java-Typen arbeiten. Achten Sie darauf, gleiche Variablen nur für einen Typ zu verwenden.

```
package com.dvelop.scripts;
import com.dvelop.d3.server.core.D3Interface;


D3Interface d3 = getProperty("d3");

def x = 42;
d3.log.info( "$x --> " + x.getClass() );

x = "Hello World";
d3.log.info( "$x --> " + x.getClass() );
// Output:
// 30.11 09:44:48,258 Master 10080CD8 D3B: 42 --> class java.lang.Integer
// 30.11 09:44:48,258 Master 10080CD8 D3B: Hello World --> class
java.lang.String
```

Was sind GStrings?

Mit Groovy können Sie Variablen innerhalb eines Strings auflösen, indem Sie Variablen mit einem vorangestellten $-Zeichen in den String integrieren. Dieses Konzept wird als GString bezeichnet.

```
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
```

Sie können innerhalb dieser Schreibweise auch auf einzelne Teilstrings zugreifen, indem Sie geschweifte Klammern verwenden. Des Weiteren können Sie in Groovy auf einzelne Buchstaben innerhalb eines Strings zugreifen. In diesem Fall ähnelt die Schreibweise jener zum Zugreifen auf Array-Elemente.

```
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

Sie können Strings in Groovy über mehrere Zeilen definieren. Hierzu benötigen Sie am Anfang und Ende des entsprechenden Strings jeweils drei Anführungszeichen.

```
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
```

1.3.2. Bedingungen

Die Bedingungen in Groovy sind in weiten Teilen identisch mit den Bedingungen in Java. Nachfolgend erhalten Sie Informationen zu den Besonderheiten der Bedingungen in Groovy.

Verwenden eines Operators für eine sichere Navigation

Wenn Sie innerhalb einer Objektstruktur einen Wert überprüften möchten, müssen Sie sicherstellen, dass die einzelnen Elemente der Struktur ungleich des Werts **null** sind. Hierzu haben Sie folgende Möglichkeiten:

Option 1

```
if( company.getContact() != null && company.getContact().getAddress() !=
null && company.getContact.getAddress().getCountry() == Country.NEW_ZEALAND
) { ... }
```

Option 2

```
if( company.getContact()?.getAddress()?.getCountry() == Country.NEW_ZEALAND
) { ... }
```

Wenn das Objekt oder eines der Objektbestandteile nicht vorhanden ist, wird der Wert **null** und keine Fehlermeldung zurückgegeben.

Elvis-Operator

In Groovy gibt es eine komprimierte Schreibweise für if-Statements. Grundsätzlich wird für Wahr-Zweige ein Fragezeichen und für Falsch-Zweige ein Doppelpunkt verwendet.

```
package com.dvelop.scripts;
import com.dvelop.d3.server.core.D3Interface;

D3Interface d3 = getProperty("d3");

def testValue = null;
def name = testValue != null  ? testValue : "default";
d3.log.info( "Normal -->" + Name );
// Output
// 10.12 10:58:23,238 Master   15041A9C D3B: Normal -->default
```

Wenn Sie nur dann einen anderen Wert zuweisen möchten, wenn die überprüfte Variable nicht enthalten ist, können Sie auch mit einer verkürzten Schreibweise arbeiten.

```
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

Groovy ist im Gegensatz zu Java bei Switch-Statements nicht auf numerische Werte begrenzt.

```
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" );
}
```

1.3.3. Schleifen

Der Umgang mit Schleifen in Groovy ist identisch zu Java. Anders als bei Java gibt es bei Groovy jedoch sogenannte Closures, die im nachfolgenden Kapitel beschrieben werden. Da geschweifte Klammern für Closures vorgesehen sind, erfolgt die initiale Befüllung von Listen und Arrays mit eckigen Klammern.

For-Schleife

Im folgenden Beispiel erfahren Sie, wie For-Schleifen aufgebaut sind:

```
package com.dvelop.scripts;
import com.dvelop.d3.server.core.D3Interface;

D3Interface d3 = getProperty("d3");
```

```
def testList = [ "This", "is", "example", "content" ];
int n        = 0;
// A classic for-statement
for( n = 0; n < testList.size(); n++ ){
   d3.log.info( "$n. For --> " + testList[n] );
}
// Output
// 10.12 12:33:11,996 Master   15041A9C D3B: 0. For --> This
// 10.12 12:33:11,997 Master   15041A9C D3B: 1. For --> is
// 10.12 12:33:11,997 Master   15041A9C D3B: 2. For --> example
// 10.12 12:33:11,997 Master   15041A9C D3B: 3. For --> content


// A for-statement with collection
n = 0;
for( def item in testList ){
   d3.log.info( (n++) +  ". For-(Collection) --> " + item );
}

// Output
// 10.12 12:33:11,998 Master   15041A9C D3B: 0. For-(Collection) --> This
// 10.12 12:33:11,999 Master   15041A9C D3B: 1. For-(Collection) --> is
// 10.12 12:33:11,999 Master   15041A9C D3B: 2. For-(Collection) -->
example
// 10.12 12:33:11,999 Master   15041A9C D3B: 3. For-(Collection) -->
content
```

Each-Statements

Im folgenden Beispiel erfahren Sie, wie Each-Statements aufgebaut sind:

```
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
```

Collection-Statements

Im folgenden Beispiel erfahren Sie, wie Collection-Statements aufgebaut sind:

```
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
```

1.3.4. Closures

Bei Closures handelt es sich um kleinere und unbenannte Funktionen, die direkt mit einer Variable verbunden sind.

Variablen mit einer Funktion

Im folgenden Beispiel erfahren Sie, wie Variablen mit einer Funktion aufgebaut sind:

```
package com.dvelop.scripts;
import com.dvelop.d3.server.core.D3Interface;

D3Interface d3 = getProperty("d3");

// Print Hello world ------------------------------------------
def optionOne = {
   d3.log.info( "Option 1: Hello World" );
}
optionOne();

// Output
// 09.12 13:30:35,082 Master   12041F98 D3B: Option 1: Hello World
```

Closures mit Variablen mit festen Typen

Im folgenden Beispiel erfahren Sie, wie Closures aufgebaut sind, die Variablen mit festem Typ beinhalten:

```
// Closures with parameters -----------------------------------
def power = { int x, int y ->
   return Math.pow( x, y ); }
d3.log.info( "Option 2: " + power( 2, 3 ));

// Output
// 09.12 13:30:35,093 Master   12041F98 D3B: Option 2: 8.0
```

Closures mit untypisierten Variablen

Im folgenden Beispiel erfahren Sie, wie Closures mit einer untypisierten Variablen aufgebaut sind:

```
package com.dvelop.scripts;
import com.dvelop.d3.server.core.D3Interface;

D3Interface d3 = getProperty("d3");

// Closure with one dynamic typing variable --------------------
def optionThree = { what ->
   d3.log.info( what ); }
optionThree "Option 3: Hello World"; // same as optionThree( "Option 3:
Hello World" );

// Output
// 09.12 13:30:35,094 Master   12041F98 D3B: Option 3: Hello World
```

Closures mit impliziten it-Variablen

Im folgenden Beispiel erfahren Sie, wie Closures mit impliziten it-Variablen aufgebaut sind:

```
// Closure with an implizit argument --------------------------
def optionFour = { d3.log.info( it ); }
optionFour "Option 4: Hello World"; // same as optionFour("Option 4: Hello
World");

// Output
// 09.12 13:30:35,095 Master   12041F98 D3B: Option 4: Hello World
```

Closures ohne Variablen

Im folgenden Beispiel erfahren Sie, wie Closures aufgebaut sind, die explizit keine Variablen beinhalten:

```
package com.dvelop.scripts;
import com.dvelop.d3.server.core.D3Interface;

D3Interface d3 = getProperty("d3");

// Closure without any argument -------------------------------
def optionFive = { ->
   d3.log.info( "Option 5: This closure does not take any arguments." ); }
optionFive();
// Output
// 09.12 13:30:35,095 Master   12041F98 D3B: Option 5: This closure does
not take any arguments.
```

Wenn Sie möchten, dass ein Wert zurückgegeben wird, können Sie dies auch ohne Return durchführen. Auf diese Weise wird der letzte Wert zurückgegeben:

```
package com.dvelop.scripts;
import com.dvelop.d3.server.core.D3Interface;

D3Interface d3 = getProperty("d3");

// Optional return value --------------------------------------
def square = { it * it };
d3.log.info( "Option 6: " + square(4));

// Output
// 09.12 13:30:35,127 Master   12041F98 D3B: Option 6: 16
```

1.3.5. Datenbankanbindung

Zugriff auf eine d.3-interne Datenbanktabelle (d.3-SQL-Datenbank)

Sie können mittels der d.3-SQL-Schnittstelle auf Datenbanktabellen innerhalb der d.3-Datenbank zugreifen.

```
//(1)
package com.dvelop.scripts;
import com.dvelop.d3.server.core.D3Interface;

D3Interface d3 = getProperty("d3");

//(2)
def resultRows = d3.sql.executeAndGet( "SELECT name FROM CustoemrData" );
//(3)
resultRows.each{ println it.name; }
```

Hinweise zu den einzelnen Abschnitten

- Um die d.3-SQL-Schnittstelle zu verwenden, benötigen Sie die Bibliothek **D3**, die Sie bei Bedarf importieren können. Da über den d3-server-interface-Aufruf implizit die Variable **d3** verfügbar ist, können Sie die Variable mit der Funktion **getProperty("d3")** im Skript bereitstellen. Dadurch können Sie auch während der Programmierung zur Codevervollständigung auf die Variable zugreifen.
- Sie können über diverse Implementierungen bzw. Funktionen ein SQL-Statement an die Datenbanktabelle senden. Die Ergebnisse werden in eine Map-Struktur überführt.
- Sie können die Daten der Ergebnisse mittels eines Each-Statements weiterverarbeiten.

Weitere Hinweise zur Datenbankanbindung

Für Abfragen, bei denen Sie häufig nur einen Treffer erhalten oder explizit nur einen Treffer erhalten möchten, können Sie die Methode **firstRow()** verwenden.

```
def sqlQuery  = "SELECT max(product_count) as value FROM productDB WHERE
product_id = ? ";
def sqlParams = [ 4711 ];
def firstRow  = d3.sql.firstRow (sqlQuery, sqlParams);
int max_no    = firstRow[0] + 1;
```

Alternativ erreichen Sie im SQL-Kommando mit **AS highestNo** eine verbesserte Lesbarkeit und somit einen besseren Code.

```
def sqlQuery  = "SELECT max(product_count) as highestNo FROM productDB
WHERE product_id = ? ";
def sqlParams = [ 4711 ];
def firstRow  = d3.sql.firstRow (sqlQuery, sqlParams)
int max_no    = firstRow.highestNo + 1;
```

Zugriff auf eine externe Datenbank

Nachfolgend erhalten Sie nähere Informationen dazu, wie Sie auf eine externe Datenbank zugreifen.

> **Anmerkung**
> Für den Zugriff auf externe Datenbanken müssen Sie die nötigen JDBC-Treiber der jeweiligen Datenbankhersteller herunterladen und bereitstellen. Anschließend können Sie mittels der Standard-SQL-Schnittstelle eine Verbindung zur Datenbank aufbauen und ein SQL-Statement absetzen.

```
//(1)
import groovy.sql.Sql;
//(2)
def dbConnection = Sql.newInstance( "jdbc:sqlserver://
localhost:1433;databaseName=Name", "User", "Password" );
//(3)
def resultRows = dbConnection.rows( "SELECT name FROM CustomerData " );
//(4)
resultRows.each{ println it.name; }
```

Hinweise zu den einzelnen Abschnitten

- Um die d.3-SQL-Schnittstelle für externe Datenbankzugriffe verwenden zu können, benötigen Sie die Standard-Groovy-Bibliothek für SQL, die Sie bei Bedarf importieren müssen.
- Um auf eine externe Datenbank zuzugreifen, müssen Sie eine Verbindung zur Datenbank herstellen. Verwenden Sie hierzu ebenfalls eine Standard-Groovy-Funktion zum Aufbau einer **newInstance**-Verbindung. genutzt. Weitere Informationen erhalten Sie in der Dokumentation des jeweiligen Datenbankherstellers und der Dokumentation von Groovy.
- Sie können über diverse Implementierungen bzw. Funktionen ein SQL-Statement an die Datenbanktabelle senden. Die Ergebnisse werden in eine Map-Struktur überführt.
- Sie können die Daten der Ergebnisse mittels eines Each-Statements weiterverarbeiten.

1.3.6. d.3-Sonderfälle

Auslesen von d.3-Konfigurationsparametern

Wenn Sie ein Groovy-Skript sowohl in der Test- als auch in der Produktivumgebung verwenden, können Sie über die Abfrage von Konfigurations- und Serverparametern das Skript so konfigurieren, dass das Skript ohne Anpassungen in beiden Umgebungen funktioniert. Eine Portierung bzw. Anpassung ist damit nicht mehr notwendig. Die dabei möglichen Parameter finden Sie in den relevanten d.3-Dokumentationen.

Im folgenden Beispiel erfahren Sie, wie Sie die d.3-Konfigurationsparameter abgefragen:

```
d3.log.error( d3.config.value( "d3fc_server_id" ) );   // Server id
d3.log.error( d3.config.value( "d3fc_server_name" ) ); // Server name
d3.log.error( d3.config.value( "db_server" ) );        // Database type
d3.log.error( d3.config.value( "CUR_60ER_FIELD_NR" ) );// Current max .
sizce of 60-ies feild

d3.log.error( d3.natives.getd3fcLanguage() );        // Get the current
language

// Output
// 08.12 15:25:39,090 Master   130C1308 D3B: B
// 08.12 15:25:39,090 Master   130C1308 D3B: localhost
// 08.12 15:25:39,090 Master   130C1308 D3B: MSQL
// 08.12 15:25:39,091 Master   130C1308 D3B: 100
```

Ab Version 8.1 können Sie Parameter wie die aktuelle API-Sprache oder die App-Version auch in Groovy ermitteln.

```
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 d3RemoteInterfaceExample{

  @Entrypoint( entrypoint = "hook_insert_entry_10" )
//-----------------------
  public int getCustomerDataForInvoice( D3Interface d3, User user,
DocumentType docTypeShort, Document doc ){
    d3.log.error( "App-Version: "  + d3.remote.getVersion());
    d3.log.error( "APP-ID: "       + d3.remote.getVersion()[0..2]);
    d3.log.error( "App-Language: " + d3.remote.getLanguage());
    return 0;
  } // end of getCustomerDataForInvoice

}// end of d3RemoteInterfaceExample
```

d.3-Systemeigenschaften

Die d.3-Schnittstelle definiert folgende Systemeigenschaften als Java-Systemeigenschaften:

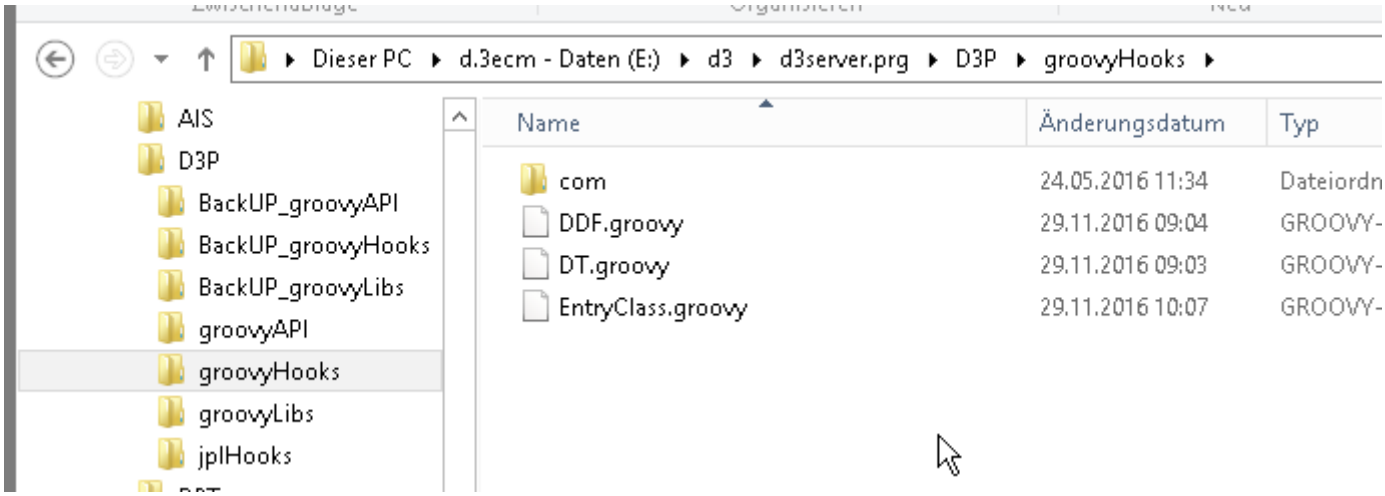
Eigenschaftsname	Beschreibung	Beispielwerte
d3.server.home	Das aktuelle d.3 server-Programmverzeichnis	"D:\d3\d3server.prg"
d3.server.version	Die d.3 server-Versionsnummer	"08.01.00.05"
d3.server.systemUser	Der Systembenutzername des aktuellen d.3 server-Prozesses	"D3Server", "D3Async", "hostimp", "Master"
d3.repository.uuid	Die Archive-UUID des d.3-Repositorys	"8be6de08-d009-4c2b-a33d-38a3a6458617"

Sie können die Systemeigenschaften mit folgendem Aufruf abfragen: **System.getProperty("Property-Name")**

```
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);
}
```

Klasse für globale Konstanten

In JPL wurde mit globalen Konstanten die Referenzierung auf Datenbankpositionen der erweiterten Eigenschaften sprechend gestaltet und zentral gesteuert. Dies erreichen Sie bei Groovy nicht über eine separate Package-Struktur, sondern nur im Root-Verzeichnis mit sämtlichen notwendigen Dateien.



Erstellen Sie hierzu exemplarisch zwei Klassen, die innerhalb der Hook-Funktionen referenziert werden.

Klasse für Dok-Dat-Feld-Positionen

```
class DDF {
   static final int FIRSTNAME = 4; // DB positions
   static final int LASTNAME  = 5;
   static final int STATE_ID  = 6;
}// end of DDF
```

Klasse für Dokument- und Aktenarten

```
class DT {
   static final String INVOCIE      = "DRECH"; // Doc types
   static final String ORDER        = "DBEST";
   static final String EMPLOYEE_FOLDER = "APERS"; // Folder types
}// end of DT
```

Verwenden der globalen Konstanten innerhalb einer Hook-Funktion

Da sich die Klassen im gleichen Verzeichnis befinden, können Sie die Klassen nun in den Hook-Dateien bzw. Hook-Funktionen verwenden.

```
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
```

1.4. Debugging

Aktivieren Sie in d.3 config unter **Java/Groovy** die Einstellung **Java Remote Debugging** (**JAVA_REMOTE_DEBUGGING**), um Fehler im Groovy-Code zu beheben. Wenn Sie die Einstellung aktivieren, starten die d.3-Serverprozesse die Java Virtual Machine im Debug-Modus. Anschließend können Sie sich per Remote Java Debugger mit einem d.3-Serverprozess verbinden, um die Fehler in den Groovy Hooks zu beheben.

Für die Kommunikation wird der Port **43400** verwendet. Da jeder d.3-Prozess eine eigene Java Virtual Machine (JVM) startet, werden die verwendeten Ports hochgezählt, d.h. der erste mittels **JAVA_REMOTE_DEBUGGING** gestartete Prozess öffnet den Port **43400**, der zweite Prozess den Port **43401** usw. Die Ports erscheinen beim Start der JVM mit der Meldung **Java Remote Debugging Port** im d.3-Protokoll.

> **Anmerkung**
> Die JVM wird von d.3 beim ersten Zugriff auf Groovy-Code gestartet. Die JVM steht somit i.d.R. noch nicht direkt nach dem Start des Prozesses zur Verfügung.

Weiterführende Informationen zum Thema Debugging finden Sie in der gängigen Fachliteratur sowie im Internet.

1.4.1. Remote Debugging mit Eclipse

Eine Methode, um Fehler in Hook-Funktionen zu finden und zu beheben, ist Remote Debugging. Beim Remote Debugging öffnet jeder d.3-Serverprozess einen Port, mit dem Sie sich per Eclipse verbinden. Anschließend können Sie den Funktionsablauf kontrollieren.

Die d.3-Serverprozesse starten die Java Virtual Machine (JVM) im Debug-Modus.

> **Anmerkung**
> Die JVM wird von d.3 beim ersten Zugriff auf Groovy-Code gestartet. Die JVM steht somit i.d.R. noch nicht direkt nach dem Start des Prozesses zur Verfügung.

Der zuerst gestartete d.3-Serverprozess verwendet den Port **43400**, der zweite Prozess verwendet den Port **43401** usw.

Beachten Sie, dass auch die d.3-Prozesse d3_server, d3_async und hostimport Remote Debugging unterstützen. Ein wird Archiv wird üblicherweise durch mehrere Serverprozesse unterstützt. Stellen Sie daher stets sicher, dass Sie mit dem korrekten Serverprozess verbunden sind, z.B. indem Sie jeweils nur einen d.3-Serverprozess ausführen.

> **Warnung**
> Während Sie das Remote Debugging durchführen, kann der d.3-Serverprozess keine anderen Aufgaben verarbeiten. Führen Sie das Remote Debugging entsprechend nicht im Produktivbetrieb durch.

Durchführen von Remote Debugging – So geht's

1. Navigieren Sie zu **d.3 admin > d.3 config > Java/Groovy** und aktivieren Sie **Java Remote Debugging**.
2. Navigieren Sie in Ihrem Eclipse-Projekt zu **Run > Debug Configurations**.
3. Erstellen Sie einen neuen Eintrag unter **Remote Java Application**.
4. Tragen Sie unter **Connection Properties** die Verbindungsdaten des Systems ein.

5. Klicken Sie auf **Debug**.
6. Setzen Sie Haltepunkte (Breakpoints) im Quelltext, an denen die Ausführung unterbrochen werden soll. Somit können Sie den Ablauf manuell prüfen und fortführen.

1.4.2. Remote Debugging mit IntelliJ IDEA

Sie können Groovy Hook-Funktionen eines d.3-Serverprozesses mithilfe von Remote Debugging per IntelliJ IDEA beheben. Sie müssen zunächst die Debug-Konfiguration hinzufügen. Anschließend können Sie das Remote Debugging durchführen.

> **Warnung**
> Während Sie das Remote Debugging durchführen, kann der d.3-Serverprozess keine anderen Aufgaben verarbeiten. Führen Sie das Remote Debugging entsprechend nicht im Produktivbetrieb durch.

So geht's

1. Wählen Sie in IntelliJ IDEA im Auswahlmenü für Laufkonfigurationen **Edit Configurations** aus.
2. Klicken Sie im neuen Fenster auf das Plussymbol und wählen Sie **Remote** aus.
3. Tragen Sie folgende Verbindungsdaten zu dem d.3-Applikationsserver ein, auf dem die Hooks und die d.3-Serverprozesse ausgeführt werden:
   - **Host**: Host-Adresse oder FQDN
   - **Port**: Debug-Port des d.3-Serverprozesses
4. Wählen Sie das Projektmodul aus IntelliJ aus, das Ihr Hook-Projekt enthält.
5. Fügen Sie Haltepunkte (Breakpoints) bei allen gewünschten Codezeilen hinzu, indem Sie jeweils neben die Zeilennummern der Codezeilen klicken. Die Ausführung des Codes wird an den Haltepunkten unterbrochen.
6. Stoppen Sie alle d.3-Serverprozesse und starten Sie einen d.3-Serverprozess neu.
7. Starten Sie die Server-Schnittstelle.
8. Prüfen Sie im Protokoll (Log), welchen Port der Serverprozess verwendet. Gehen Sie z.B. wie folgt vor:
   - Suchen Sie im Protokoll nach “Remote Debugging”. Eine gefundene Protokollzeile kann z.B. wie folgt aussehen: **04.04 14:46:47,695 D3SRV_P 68B85950 : Java Remote Debugging Port: 43403**. In diesem Beispiel ist der ermittelte Port somit **43403**.
9. Tragen Sie den ermittelten Port in der Konfiguration ein.
10. Klicken Sie auf das Käfersymbol oder drücken Sie **Umschalt + F9**, um das Remote Debugging zu starten. Bei einem Aufruf der Hooks bzw. Skripte pausieren die Haltepunkte den Prozess. Somit können Sie den Prozess manuell prüfen und fortführen.

Weitere Informationen zum Debugging mit IntelliJ IDEA finden Sie beim Hersteller JetBrains: https://www.jetbrains.com/idea/documentation/.

1.5. Groovy-Hook-Typen

Groovy-Hooks sind definierte Schnittstellen (Einsprungs- bzw. Eintrittspunkte) in unseren Systemen, an denen Sie individuelle Funktionen erstellen können. Für die Erstellung der Funktionen verwenden Sie ein Groovy-Skript.

Es gibt unterschiedliche Typen von Groovy-Hooks:

- Eintrittspunkte sind ereignisgesteuerte Skriptschnittstellen
- Validierungshooks zur erweiterten Validierung von Werten
- Bereitstellung von dynamischen Wertemengen
- Erweiterte Dokumentklassen

1.5.1. Groovy-Schnittstelle in d.3 admin

Nehmen Sie in d.3 admin unter **Systemeinstellungen > d.3 config** Konfigurationen in den Bereichen **Hook-Funktionen** und **Java/Groovy** vor, um die Groovy-Schnittstelle verwenden zu können. Beachten Sie die folgenden Hinweise zu den Konfigurationen.

Hook-Funktionen

- **Groovy-Hooks aktivieren**
  Tragen Sie die Verzeichnisse ein, in denen die per Groovy implementierten Programmanpassungen enthalten sind.
- **Neuladen von Groovy-Hooks**
  Wenn Sie den Schalter aktivieren, werden Groovy-Hook-Dateien neu geladen, sobald Sie Änderungen an den Groovy-Hook-Dateien speichern. Somit müssen Sie keinen d.3-Prozess neu starten und die Änderungen am Code sind sofort aktiv. Verwenden Sie diese Funktion nur während der Entwicklung und nicht in einem Produktivsystem.
- **JPL und Groovy-Hooks für den gleichen Einsprungpunkt aktivieren**
  Wenn Sie diesen Schalter aktivieren, können JPL-Hook-Funktionen und Groovy-Hook-Methoden für den gleichen Hook-Einsprungpunkt registriert werden. Diese Möglichkeit soll die Ersetzung von Bestands-JPL-Code durch moderne Groovy-Hooks erleichtern, da so Logik sukzessive und lösungsorientiert erneuert werden kann.
  Wenn unter **Hook-Funktionen ausführen** in der Spalte **Hook-Funktion** der Name einer JPL-Hook-Funktion steht und gleichzeitig eine Annotation für den Einsprungpunkt in einem Groovy-Hook-Modul existiert, werden die Funktion und die Methode für diesen Einsprungpunkt registriert. Bei der Ausführung wird zuerst die JPL-Hook-Funktion aufgerufen und direkt danach die annotierte(n) Groovy-Hook-Methode(n).

Java/Groovy

- **Java/Groovy Support**
  Aktiviert die Unterstützung für die Ausführung von Java und Groovy-Code in d.3.
- **Java CLASSPATH**
  Dateipfad bzw. Dateipfade getrennt mit Semikola zu den eigenen Java-Klassen. Geben Sie ein oder mehrere Verzeichnisse an, in dem/denen die CLASS-Dateien gespeichert sind. Geben Sie alternativ den Dateinamen einer JAR-Datei an oder verwenden Sie eine CLASSPATH-Datei.
- **Java/Groovy API Funktionen**
  Aktiviert die Plugin-Schnittstelle für API-Funktionen, die in Java bzw. Groovy entwickelt wurden. Groovy-Skripte oder JAR-Dateien, die d.3-API-Funktionen implementieren, werden aus dem Verzeichnis geladen. Wir empfehlen, diese Einstellung nicht zu aktiveren.
- **Java Remote Debugging**
  Startet die Java Virtual Machine (JVM) im Debug-Modus. Sie können sich per Remote Java Debugger mit einem d.3-Server-Prozess verbinden, um Fehler in den darin ausgeführten Groovy-Hooks zu beheben. Für die Kommunikation wird Port **43400** verwendet. Da jeder d.3-Prozess eine eigene Java Virtual Machine (JVM) startet, werden die verwendeten Ports hochgezählt. Der erste mit aktiviertem **JAVA_REMOTE_DEBUGGING** gestartete Prozess öffnet Port **43400**, der Zweite Port **43401** usw. Der ermittelte Port erscheint beim Start der JVM in der Meldung **Java Remote Debugging Port** im d.3-Protokoll.

> **Anmerkung**
> Die JVM wird von d.3 beim ersten Zugriff auf Groovy-Code gestartet. Die JVM steht somit i.d.R. noch nicht direkt nach dem Start des Prozesses zur Verfügung.

1.5.2. Programmierung von Hook-Funktionen

Benötigte Java-Bibliotheken

Im Groovy-Kontext werden Bibliotheken benötigt, die über die Integration der Datei **groovyhook.jar** verfügbar sind. Sie können die Bibliotheken in den Groovy-Dateien referenzieren.

Globale Bibliotheken

Zur Verwendeung des des Objekts **D3Interface** benötigen Sie die Bibliothek **com.dvelop.d3.server.core.D3Interface**.

> **Anmerkung**
> Bei Problemen mit der JavaDoc-Dokumentation und Groovy-Templates bei der Implementierung des Objekts **D3Interface** können Sie zur Programmierung auch die Basisbibliothek **import com.dvelop.d3.server.core.D3** verwenden. Wechseln Sie für die produktive Verwendung jedoch wieder zur Bibliothek **com.dvelop.d3.server.core.D3Interface.**

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;**

> **Anmerkung**
> *Sie können den optionalen Parameter **order** zur Definition einer Reihenfolge der Abarbeitung definieren, wenn auf einem Eintrittspunkt mehrere Groovy-Funktionen definiert sind.

Styleguide-Empfehlung

Zur Bereitstellung der Funktionalität müssen Sie mindestens eine Klasse vom Typ **public** erstellen; wobei **public** im Kontext von Groovy auch weggelassen werden kann. Sie können für die einzelnen Typen von Hook-Funktionen jeweils eine eigene Klasse oder eine eigene Datei bereitstellen. Im Folgenden finden Sie Beispiele für die Klassennamen:

Klassenname	Beschreibung
D3Hooks	Für die Behandlung von Eintrittspunkten
D3DataSets	Für die Implementierung von Wertemengen
D3Validate	Zur Bereitstellung von Validierungsfunktionen auf Eigenschaftsebene
D3DocClasses	Zur Realisierung von spezifischen Dokumentklassen
D3FolderScheme	Falls Sie erweiterte Aktenpläne benötigen, können Sie diese Klasse bereitstellen.

Registrieren einer Groovy-Methode als Hook-Funktion

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

```
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
```

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

> **Anmerkung**
> Eine Groovy-Klasse, die von d.3 geladen und registriert werden soll, muss einen öffentlichen Konstruktor ohne Parameter (**public no-argument constructor**) bereitstellen. Die Voraussetzung ist auch erfüllt, wenn der Konstruktor fehlt, weil in dem Fall der Java Default-Konstruktor implizit existiert.

Rückgabewert von Hook-Methoden

Als Rückgabewert wird eine Zahl (Integer) erwartet. Den zurückgegebenen Wert wertet der Server wie folgt als Fehlercode aus:

- Wert = 0: Erfolg
- Wert <> 0: Fehler in der Hook-Funktion. Je nach Hook-Funktion führt dies zum Abbruch der Aktion, bei der die Hook-Funktion ausgeführt wurde.

> **Anmerkung**
> Bei Groovy ist das Schlüsselwort **return** für das Verlassen einer Methode mit Rückgabewert optional. Sie können das Schlüsselwort somit weglassen.
>
> Wenn Sie das Schlüsselwort nicht angeben, verwendet Groovy den letzten Variablenwert, der vor der schließenden Klammer steht und gibt diesen Wert implizit als Rückgabewert zurück. In diesem Fall können Fehler oder ungewollte Rückgabewerte auftreten. Wir empfehlen daher, eine Hook-Methode explizt mit **return** zu beenden. Wenn der Rückgabewert nicht relevant ist, können Sie **return 0** verwenden.
>
> Groovy-Hook-Funktionen für die d.3-Eintrittspunkte werden automatisch konfiguriert. Beachten Sie folgende Hinweise:
>
> - Sie müssen die Bezeichner für d.3-Eintrittspunke nicht unter **d.3 admin > d.3 config >Hook-Funktionen > Hook-Funktionen ausführen** einzeln eintragen.
> - Wenn beim Laden einer Groovy-Klasse eine Annotation für einen Eintrittspunkt gefunden wird, wird die annotierte Methode registriert.
> - Im Config-Modul ist hinter dem Eintrittspunkt die Registrierungskennzeichnung **<groovy hook>** eingetragen.
> - Die Registrierung mehrerer Methoden pro Eintrittspunkt ist möglich.

> **Warnung**
> Da die Aktualisierung von d.3 admin einige Zeit in Anspruch nehmen kann, können Sie auch die Ausgabe in der Protokolldatei verwenden. In der Protokolldatei wird ebenfalls das erfolgreiche Laden der Groovy-Funktionen zu den Eintrittspunkten dokumentiert.

Verwenden von Java-Bibliotheken

Wenn Sie bei der Umsetzung einer Hook-Funktion Java-Bibliotheken von Drittanbietern oder selbst erstellte Java-Bibliotheken benötigen (z.B. um ihr CRM-System anzusprechen), können Sie diese Bibliotheken für jeden Hook spezifisch angeben.

Erstellen Sie zusätzlich zu ihrer vorhandenen Datei **<Hook-Name>.groovy** eine weitere Datei **<Hook-Name>.classpath**. In der CLASSPATH-Datei definieren Sie zeilenweise Einträge für den Java-Classpath. Sie können absolute Pfade und Pfade relativ zum aktuellen Verzeichnis verwenden, die zu JAR-Dateien führen. Auf diese Weise sind die Java-Bibliotheken voneinander isoliert, sodass Sie in mehreren Hooks unterschiedliche Versionen von Bibliotheken verwenden können.

> **Anmerkung**
> JDBC-Treiber können Sie nicht hookspezifisch einbinden. Die Treiber müssen global geladen werden.

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 können Sie jede Groovy-Klasse mit dem Befehl **import** in einer anderen Groovy-Klasse sichtbar machen. Die Groovy-Klasse können Sie instanziieren oder, im Fall von statischen Elementen, direkt aufrufen.

Wenn es mehrere, thematisch zusammengehörige Hook-Klassen bestehen, können Sie gemeinsam verwendeten Code in eine eigene Klasse und damit ein eigenes Modul auslagern. Beispiel: Wenn mittels Hook beim Import validiert wird, ob eingegebene Kundendaten im CRM-System gespeichert wurden, und gleichzeitig eine Wertemenge möglicher Kunden angeboten werden soll, besteht typischerweise gemeinsam verwendeter Code. Implementieren Sie den Code in einer eigenen Groovy-Klasse. Die Groovy-Klasse kann wiederrum von den anderen Groovy-Hook-Klassen verwendet werden.

Package-Struktur

So wie Java-Klassen werden auch Groovy-Klassen in Packages organisiert. Nennen Sie den Namen des Packages am Anfang der Quelldatei vor den Import-Anweisungen und der ersten Klassendefinition. Sie können die standardmäßige Form der Strukturierung anhand von Domainnamen in umgekehrter Reihenfolge verwenden. Wenn Sie kein Package angeben, wird das Default-Package verwendet.

> **Warnung**
> In einigen Versionen können Sie keine Packages verwenden.

Empfehlungen für Packages

- **com.dvelop.scripts**
  Groovy-Skripte, die im Kontext eines Server-Interfaces bzw. von d.3 process manager aufgerufen und somit im Verzeichnis **ext_groovy** gespeichert werden. Das resultierende Verzeichnis ist z.B. **ext_groovy\com\dvelop\scripts**.
- **com.dvelop.hooks**
  Die Groovy-Hook-Funktionen, die die einzelnen Hook-Eintrittspunkte bedienen, sollten Sie in einem eigenen Package definieren. Das resultierende Verzeichnis ist z.B. **d3server.prg\D3T\groovyHooks\com\dvelop\hooks**.
- **com.dvelop.api**
  Wenn Sie mittels Groovy-Funktionen eigene API-Funktionen bereitstellen, können Sie dieses Package verwenden. Das resultierende Verzeichnis ist z.B. **d3server.prg\D3T\groovyAPI\com\dvelop\api**.

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

Wenn aus einer Serveraktion dynamische Texte auch an die Clientseite übertragen werden sollen (z.B. für dynamische Fehlermeldungen), können Sie die Hook-Eigenschaft **additional_info_text** aus jedem Hook verwenden.

Für folgende Hook-Integrations-Typen stehen die Funktionen **getProperty** und **setProperty** nicht zur Verfügung:

- Validierungs-Hook
- Dokumentklassen-Hook
- Wertemengen-Hook

Die Funktionen stehen nur für die „klassischen“ d.3-Eintrittspunkte zur Verfügung.

Aufruf

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

//Message depending on the language of the client request
if(d3.remote.getLanguage() == "049"){
  d3.hook.setProperty("additional_info_text", "Meine Nachricht für die
Client-Seite.")
} else {
  d3.hook.setProperty("additional_info_text", "My message for the Client
side!")
}
```

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

> **Anmerkung**
> Der Text wird nicht durchgängig bei allen Clients angezeigt.

1.5.4. Nummernkreis für Returnwerte

Die Returnwerte aus den Hook-Funktionen werden intern mit einem Offset-Wert verrechnet. Der Wert, der auf der Clientseite ausgewertet werden kann, ist das Ergebnis aus der Berechnung "Offset-Wert minus Bereichswert".

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

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

Beispiel

```
@Entrypoint(entrypoint = "hook_insert_entry_10")
int myCustomReturnValueImport(D3Interface d3, User user, DocumentType
docType, Document doc) {
        if(docType.id == "DDOKU" && doc.field[10] == null){
        //Client returns error message with id "18000" in msglib.usr
                return -8000
        }
}

@Entrypoint( entrypoint = "hook_new_version_entry_10" )
int myCustomReturnValueNewVersion( D3Interface d3, Document doc, String
fileSource, String fileDestination, User user, DocumentType docType ){
        if(docType.id == "DDOKU" && doc.field[10] == null){
        //Client returns error message with id "28000" in msglib.usr
                return -8000
        }
}

@Entrypoint( entrypoint = "hook_delete_entry_10" )
int myCustomReturnValueDelete( D3Interface d3, Document doc, User user,
DocumentType docType ){
        if(docType.id == "DDOKU" && doc.field[10] == null){
        //Client returns error message with id "5925" in msglib.usr
                return -1925
        }
}

@Entrypoint(entrypoint = "hook_upd_attrib_entry_20")
int myCustomReturnValueUpdAtt(D3Interface d3, Document doc, User user,
DocumentType docType, DocumentType docTypeNew) {
        if(docType.id == "DDOKU" && doc.field[10] == null){
        //Client returns error message with id "9750" in msglib.usr
                return -250
        }
}
```

msglib.usr

```
'Erste Spalte  = Fehlernummer
'Zweite Spalte = Sprache (049=Deutsch, 001=Englisch)
'Dritte Spalte = Typ
'   d.3-Server Fehlermeldungen
'   0 = Warnung
'   1 = Fehler
'   2 = Information
'   3 = Bestätigung
'Vierte Spalte = Nachricht
```

```
18000,049,2,"Bitte Angabe für Feld 10 machen. (Import, RetCode -8000)"
18000,001,2,"Please enter information for field 10. (Import, RetCode
-8000))"

28000,049,2,"Bitte Angabe für Feld 10 machen. (NewVersion, RetCode -8000))"
28000,001,2,"Please enter information for field 10. (NewVersion, RetCode
-8000))"

5925,049,2,"Bitte Angabe für Feld 10 machen. (Delete, RetCode -1925))"
5925,001,2,"Please enter information for field 10. (Delete, RetCode -1925))"

9750,049,2,"Bitte Angabe für Feld 10 machen. (UpdAtt, RetCode -250)"
9750,001,2,"Please enter information for field 10. (UpdAtt, RetCode -250)"
```

Die Datei **msglib.usr** wird in dem Clientverzeichnis, in dem sich die Datei **msglib.dll** befindet, erstellt bzw. bearbeitet. Beispiel für d.velop documents: **D:\d3\d.3one\dms\bin\msglib.usr**

1.5.5. Verwendung des Transportsystems für Groovy-Funktionen

Mit dem Transportsystem können Sie Einstellungen zwischen d.3-Repositorys übertragen. Außerdem können Sie Groovy-Hook-Module übertragen.

Im Bearbeitungsmodus von d.3 admin können Sie Projekte definieren, die die zu transportierenden Einstellungen beinhalten. Um ein Groovy-Hook-Modul einem Tranportprojekt zuzuordnen, tragen Sie den Projektenamen per Annotation **@TransportProject** in das Groovy-Hook-Modul ein.

Die Annotation **@TransportProject** ist auf Klassenebene definiert, weshalb Sie die Annotation einer Java/Groovy-Klassendefinition vorstellen müssen. Als Parameter der Annotation können Sie einen oder mehrere Projektnamen oder die GUIDs der Projekte eintragen.

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

// OPTION 1: Projectname
@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 Project names or numbers
@TransportProject(["ProjectNewInstallation", "931608C4-E075-4E89-
AA35-66E6FD74770B"])
public class MyTestHooks {
   // …
} // end of MyTestHooks
```

Beachten Sie folgende Hinweise zum Transport von Groovy-Hook-Modulen:

- Ändern Sie den Dateinamen der Module darf sich nicht mehr, sobald erstmals ein Meilenstein geschlossen wurde, der zu einem der annotierten Projekte gehört.
- Verwenden Sie pro Modul sollte nur eine Klasse.

- 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. Wenn die Datei gefunden wird, wird die Datei 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 in dem Unterordner gespeichert.
- Wenn die Moduldatei erstmalig im Ziel-Repository gespeichert wird, müssen Sie die d.3-Prozesse neu starten.
- Wenn bereits eine Moduldatei existiert hat und die Moduldatei ausgetauscht wird, bestimmt der Parameter **HOOK_GROOVY_RELOAD_ON_CHANGE**, ob das Modul direkt aktiviert wird.
- Wenn Sie ein Groovy-Hook-Modul im Zielsystem löschen möchten, müssen Sie die Annotation des Moduls im Quellsystem entfernen und die Datei im Zielsystem löschen.

1.5.6. d.3-Eintrittspunkte

Allgemein

Die Eintrittspunkte werden durch Aktionen von Anwendenden angestoßen und anschließend nacheinander abgearbeitet. Wenn eine Hook-Funktion innerhalb der Verarbeitungskette mit einem Wert ungleich 0 verlassen wird, wird die Kette unterbrochen. Beispiel: Während des manuellen Imports einer Rechnung wird festgestellt, dass die Kundennummer einen falschen Wert enthält. In diesem Fall wird die Speicherung der Rechnung im System verhindert.

In den folgenden Unterkapiteln werden die verfügbaren Eintrittspunkte für Hook-Funktionen in d.3 beschrieben. Für die beschriebenen Parameter einer Hook-Funktion werden die im Kontext des Aufrufs verfügbaren d.3-Archivobjekte an die Groovy-Hook-Funktionen übertragen. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer die d.3-Schnittstelle übertragen.

Beispiel

```
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>";
          return 0;
    }// end of checkIncommingDocuments
} // end of MyHooks
```

Anmerkung

Sie können auch mehrere Methoden pro Eintrittspunkt registrieren. Wenn die Reihenfolge des Aufrufs relevant ist, können Sie die Reihenfolge mit dem numerischen Attribut **order** in der Annotation festlegen. Der Standardwert für das Attribut **order** ist "1".

Vor dem Aufruf aller Methoden einer Klasse für denselben Eintrittspunkt werden die Methoden nach dem Attribut **order** aufsteigend sortiert. Wenn für den gleichen Eintrittspunkt eine Methode nach einer anderen Methode aufgerufen werden soll, muss der **order**-Wert der später aufgerufenen Methode größer sein.

Beispiel

```
// ...
    @Entrypoint(entrypoint="hook_insert_entry_10", order = 2)
    def checkIncommingDocuments_2(D3Interface d3, User user, DocumentType
docType, Document doc){
          println "Second method to handle this entrypoint
hook_insert_entry_10";
          doc.field[2] = "<New Value>";
          return 0;
    }// end of checkIncommingDocuments
}/ end of MyHooks
```

Anmerkung

Ein zweiter Annotationstyp ist **@Condition**. Mit diesem Annotationstyp können Sie Bedingungen für den Aufruf einer annotierten Methode definieren.

Mit der Eigenschaft **doctype** können Sie IDs von d.3-Dokumentarten angeben. Wenn der Eintrittspunkt einen Parameter vom Typ **Document** oder ein **DocumentType** besitzt, 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 über konstante Variablen

```
// ...
 static final String Photo                                = "DFOTO";  //
Document type Photo
 static final String curriculumVitae                      = "DLELA";  //
```

```
Document type Curriculum vitae
 static final String personnelMasterData                  = "DPSB";   //
Document type personal master data
 static final String certificates                         = "DZEUG";  //
Document type certificates
 @Condition(doctype=[Photo, curriculumVitae, personnelMasterData,
certificates])
// ...
```

Abhängige Dateien

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.UserOrUserGroup;

// 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
```

Beschreibung der Blöcke

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

4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
5. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
6. Die Funktion wird mit dem Return-Wert **0** beendet.

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 dem Eintrag der abhängigen Datei in die Datenbank.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das Dokument, zu dem die abhängige Datei eingetragen 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.UserOrUserGroup;

// 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
```

1. Import der benötigten Bibliotheken aus der Datei *groovyhook.jar*.
2. Bereitstellung einer eigenen Klasse vom Typ *public*, wobei *public* im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen *doSomething* mit einem sprechenden Namen.
5. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.

6. Die Funktion wird mit dem Return-Wert *0* beendet.

Aktualisieren der Eigenschaftswerte (UpdateAttributes)

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

Aufrufzeitpunkt: Die neuen Attribute wurden empfangen, jedoch noch nicht auf Plausibilität geprüft.

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

> **Anmerkung**
> 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
```

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

2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

hook_upd_attrib_exit_10

> **Anmerkung**
> Diese Hook-Funktion wird nur ausgeführt, wenn zuvor kein Fehler aufgetreten ist. Wenn die Funktion einen Wert ungleich Null liefert, wird die Aktualisierung abgebrochen und die Änderungen werden rückgängig gemacht.

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

Aufrufzeitpunkt: Direkt vor Beendigung der DB-Transaktion.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	Dokumentobjekt mit den aktualisierten Eigenschaftswerten. Die Werte können in dieser Hook-Funktion geändert werden.
errorCode	0: Aktualisierung erfolgreich <> 0: Fehlernummer, vom DB-Server geliefert
user	der ausführende Benutzer
docType	Dokumentart nach der Eigenschaftsaktualisierung
docTypeOld	Dokumentart vor der Eigenschaftsaktualisierung
docOld	Dokumentobjekt vor der Eigenschaftsaktualisierung

> **Anmerkung**
> 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
```

Beschreibung der Blöcke

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

hook_upd_attrib_exit_20

> **Anmerkung**
> Diese Hook-Funktion wird immer ausgeführt, d.h. die Funktion wird auch ausgeführt, wenn zuvor ein Fehler aufgetreten ist. Wir empfehlen daher, den Parameter **errorCode** auszuwerten.

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

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

Aufrufzeitpunkt: Direkt nach Beendigung der DB-Transaktion.

```
// (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
```

Beschreibung der Blöcke

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

Dokumentanlage (Import Document)

hook_hostimp_entry_10

> **Warnung**
> Dieser Einsprungpunkt ist aktuell nicht mit der Groovy-Hook-Schnittstelle kompatibel und noch nicht verwendbar. Verwenden Sie die JPL-Variante des Einsprungpunkts.

```
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. Eine Änderung der übertragenen Eigenschaftswerte ist 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 Dokumentobjekt
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
```

Beschreibung der Blöcke

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

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. Eine Änderung der übertragenen Eigenschaftswerte ist 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 Sie im Hook ändern.

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

> **Anmerkung**
> Szenario: Wenn ein Dokument im d.3-System gespeichert wird, soll in der d.3-Protokolldatei die 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
```

Beschreibung der Blöcke

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

2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

hook_insert_entry_20

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

Aufrufzeitpunkt:

- Vor dem Import. Die Datei wurde bereits in das Zielverzeichnis übertragen.
- Die SQL-Kommandos für die Speicherung der Dokument-Metadaten wurden erstellt.
- Die übertragenen Dokumenteigenschaften können nicht mehr geändert werden.
- Die Eigenschaftswerte sind noch nicht auf Gültigkeit (Wertebereich, reg. Expression, Min.-Max.-Bereich usw.) 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
```

Beschreibung der Blöcke

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

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. Sie können noch eine Zurücknahme erzwingen und den Import rückgängig machen.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das neue Dokument
fileDestination	Pfad und Name der Zieldatei (Angabe, wo die Zieldatei gespeichert wurde)
importOk	1: bisher kein Fehler ausgetreten 0: beim Import des Dokuments ist ein Fehler aufgetreten
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
```

Beschreibung der Blöcke

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

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**). Ein erfolgreicher Import kann nicht mehr rückgängig gemacht werden.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das neue Dokument
fileDestination	Pfad und Name der Zieldatei (Angabe, wo die Zieldatei gespeichert wurde)
importOk	1: bisher ist kein Fehler aufgetreten 0: beim Import des Dokuments ist ein Fehler aufgetreten
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
```

Beschreibung der Blöcke

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

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 Datenbanktransaktion wurde bereits geschlossen.

> **Anmerkung**
> Sie können die Funktion so wie die Funktion hook_insert_exit_20 verwenden.

Dokumente freigeben (Release Document)

hook_release_entry_10

> **Anmerkung**
> Wenn 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 die Berechtigung hat, das Dokument freizugeben.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das freizugebende Dokument

Parameter	Beschreibung
user	der ausführende Benutzer
docType	Dokument des freizugebenden Dokuments
unblock	1: Wenn das Dokument entsperrt wird Wenn das Dokument nicht entsperrt wird, erscheint kein Wert.

```
// (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
```

Beschreibung der Blöcke

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

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 und 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 Wenn die Freigabe nicht erfolgreich war, erscheint ein Fehlercode.
docType	Dokumentart des freigegebenen Dokuments
unblock	1: Wenn das Dokument entsperrt wird Ungleich 1: normale Freigabe

```
// (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
```

Beschreibung der Blöcke

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

Dokument prüfen (Verify Document)

hook_verify_entry_10

> **Anmerkung**
> Wenn 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 die Berechtigung hat, das Dokument zu prüfen.

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
```

Beschreibung der Blöcke

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.

6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

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 Wenn die Prüfung nicht erfolgreich war, erscheint die Datenbank-Fehlernummer.

```
// (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
```

Beschreibung der Blöcke

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

Dokumentsuche (GetDocumentList / SearchDocument)

hook_search_entry_05

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

Aufrufzeitpunkt: Vor der Suche nach Dokumenten. Vor dem Zugriff auf die Volltext-Engine.

In dieser Hook-Funktion können Sie über das Hook-Eigenschaftsfeld **searchtext_expression** den Volltext-Suchbegriff anpassen. Beispiel:

```
// den aktuellen Volltext-Suchbegriff ausgeben
println d3.hook.getProperty("searchtext_expression")
// den Volltext-Suchbegriff verändern
d3.hook.setProperty("searchtext_expression", "mein Hook Volltext-
Suchbegriff")
```

Sie können den Volltext-Suchbegriff auch über **searchContext** anpassen.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
user	der ausführende Benutzer
docType	die Dokumentart der gesuchten Dokumente (falls angegeben)
searchContext	die Suchbegriffe sind über ein Dokumentobjekt 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
```

Beschreibung der Blöcke

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

3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

Wenn Sie mit einem Suchbegriff arbeiten oder einen Suchbegriff ausgeben möchten, können Sie die Methode **getImportParams()** von **D3RemoteInterface** verwenden. Beispiel:

```
@Entrypoint(entrypoint = "hook_search_entry_05")
 int hook_search_entry_05(D3Interface d3, User user, DocumentType docType,
Document searchContext)
 {
     d3.log.info("### hook_search_entry_05 ---- START")

     //(1)
     String searchText

     Map<String, Object> dict;

     dict = d3.remote.getImportParams();

     //(2)
     searchText = dict.get("searchtext_expression").toString()

     d3.log.info("The searchtext is $searchText");
     d3.log.info("### hook_search_entry_05 ---- END")

     return 0;
 }
```

Beschreibung der Blöcke

1. Über **getImportParams()** werden die Importparameter der Suche (**GetDocumentList**) abgefragt und in einer Map gespeichert. Der Importparameter für Volltext heißt **searchtext_expression** und stellt den Schlüssel (Key) dar.
2. Über den Schlüssel (Key) wird der Wert abgefragt und in einem String gespeichert. Der String wird im Protokoll ausgegeben.

Sie können auch alle Schlüssel (Keys) und Werte (Values) der Map ausgeben und somit alle Importparameter mit ihren Werten ermitteln. Beispiel:

```
   @Entrypoint(entrypoint = "hook_search_entry_05")
    int hook_search_entry_05(D3Interface d3, User user, DocumentType
docType, Document searchContext)
    {
        d3.log.info("### hook_search_entry_05 ---- START")

        Map<String, Object> dict;

        dict = d3.remote.getImportParams();
```

```
                //(1)
        for ( String key : dict.keySet() ) {
            d3.log.info("### hook_search_entry_05 Searchitem: Key: " + key);
        }

                //(2)
        for ( Object value : dict.values() ) {
            d3.log.info("### hook_search_entry_05 Searchitem: Value: " +
value);
        }

        d3.log.info("### hook_search_entry_05 ---- END")
        return 0;
    }
```

Beschreibung der Blöcke

1. **KeySet** wird mittels **for**-Schleife ermittelt.
2. Alle Werte werden mittels **for**-Schleife ermittelt.

hook_search_entry_10

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

Aufrufzeitpunkt:

- Vor der Suche nach Dokumenten: Die übertragenen Suchkriterien wurden noch nicht auf Plausibilität geprüft. 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. Die Suchbegriffe können Sie im Hook über **searchContext** ändern.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
user	der ausführende Benutzer
docType	die Dokumentart der gesuchten Dokumente (falls angegeben)
searchContext	die Suchbegriffe sind über ein Dokumentobjekt 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
```

Beschreibung der Blöcke

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

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 bereits zusammengesetzt worden.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
user	der ausführende Benutzer
docType	die Dokumentart der gesuchten Dokumente (falls angegeben)

```
// (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
```

Beschreibung der Blöcke

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

hook_search_exit_30

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

Aufrufzeitpunkt: Am Ende der Suche, bevor die Ergebnisse an den Client geliefert werden.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
user	Benutzer, der die Suche ausführt
errorCode	0: Erfolg Wenn ein Fehler auftritt, erscheint ein Fehlercode.
noResults	Anzahl der Treffer
noRefused	Anzahl verweigerter Treffer
docType	die Dokumentart der gesuchten Dokumente (falls angegeben)

Rückgabewert: wird ignoriert

> **Anmerkung**
> Über die Hook-Eigenschaft **no_results_refused** können Sie die Anzeige der Anzahl der verweigerten Treffer an den Aufrufer deaktivieren. Verwenden Sie diese Eigenschaft z.B., wenn Benutzer nicht sehen sollen, dass es Suchergebnisse 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
```

Beschreibung der Blöcke

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

Einspielen einer neuen Version (ImportNewVersionDocument)

hook_new_version_entry_10

> **Anmerkung**
> Die Funktion wird bei **ValidateAttributes nextcall=ImportNewVersionDocument** aufgerufen.
>
> Ab d.3-Version 8 wurde die Dokumentablage neu strukturiert, weshalb die Parameter **fileSource** und **fileDestination** keinen Inhalt mehr besitzen. Die Parameter sind jedoch weiterhin vorhanden, sodass sich die Hook-Schnittstelle nicht ändert. Die Parameter müssen somit weiterhin entgegengenommen werden. Als Wert wird ein leerer String übertragen.

```
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 in d.3 bereits existiert. Die Dokumenteigenschaften können 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
```

Beschreibung der Blöcke

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

hook_new_version_entry_20

> **Anmerkung**
> Ab d.3-Version 8 wurde die Dokumentablage neu strukturiert, weshalb die Parameter **fileSource** und **fileDestination** keinen Inhalt mehr besitzen. Die Parameter sind jedoch weiterhin vorhanden, sodass sich die Hook-Schnittstelle nicht ändert. Die Parameter müssen somit weiterhin entgegengenommen werden. Als Wert wird ein leerer String übertragen.

```
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. Die Version 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
```

Beschreibung der Blöcke

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

3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

hook_new_version_entry_30

> **Anmerkung**
> Ab d.3-Version 8 wurde die Dokumentablage neu strukturiert, weshalb die Parameter **fileSource** und **fileDestination** keinen Inhalt mehr besitzen. Die Parameter sind jedoch weiterhin vorhanden, sodass sich die Hook-Schnittstelle nicht ändert. Die Parameter müssen somit weiterhin entgegengenommen werden. Als Wert wird ein leerer String übertragen.

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

Aufrufzeitpunkt: Wird nach hook_new_version_entry_20 ausgeführt. Die Angaben verwenden Sie 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
```

Beschreibung der Blöcke

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

2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

hook_new_version_exit_10

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

Aufrufzeitpunkt: Die Datenbanktransaktion wurde gestartet. Alle Eigenschaften inkl. der 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: Das Aktualisieren der Kenndaten war erfolgreich
user	der ausführende Benutzer
docType	Dokumentart der importierten Dateiversion
docOld	Dokumentobjekt vor der Eigenschaftsaktualisierung

```
// (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, Document docOld){
      // (6)
      d3.log.error("Hello world!");
      // (7)
      return 0;
   } // end of doSomething
} // end of D3Hooks
```

Beschreibung der Blöcke

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

hook_new_version_exit_20

> **Anmerkung**
> Ab d.3-Version 8 wurde die Dokumentablage neu strukturiert, weshalb der Parameter **fileDestination** keinen Inhalt mehr besitzt und abgekündigt ist.

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

Aufrufzeitpunkt: Auch die Mehrfacheigenschaften (60er-Felder) wurden aktualisiert. Die neue Dateiversion wurde importiert, ggf. zusammen mit zugehörigen, abhängigen Dateien. 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 war erfolgreich 0: Einspielung der neuen Version wurde mit Fehler abgebrochen
user	der ausführende Benutzer
docType	Dokumentart der importierten Dateiversion
docOld	Dokumentobjekt vor der Eigenschaftsaktualisierung

```
// (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,
Document docOld){
      // (6)
      d3.log.error("Hello world!");
      // (7)
      return 0;
   } // end of doSomething
} // end of D3Hooks
```

Beschreibung der Blöcke

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

hook_new_version_exit_30

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

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 war erfolgreich 0: Einspielung der neuen Version wurde mit Fehler abgebrochen
errorCode	Im Fehlerfall (**importOk=0**): Fehlercode des zuvor aufgetretenen Fehlers
user	der ausführende Benutzer
docType	Dokumentart der importierten Dateiversion
docOld	Dokumentobjekt vor der Eigenschaftsaktualisierung

```
// (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, Document docOld){
      // (6)
      d3.log.error("Hello world!");
      // (7)
      return 0;
   } // end of doSomething
} // end of D3Hooks
```

Beschreibung der Blöcke

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

Erstellen/Bearbeiten von TIFF- oder PDF-Dokumenten

hook_rendition_entry_10

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

Aufrufzeitpunkt: Vor dem Start der Abbildungserstellung, wenn die Erstellung über einen d.3-Benutzer aufgerufen wurde (über d.3-API oder Server-API). Die Funktion wird nicht aufgerufen, wenn ein automatischer Aufruf über gespeicherte Regeln erfolgt.

> **Anmerkung**
> Sie können den Aufruf über einen Return-Wert ungleich 0 abbrechen.

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
```

Beschreibung der Blöcke

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

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 Erstellungsauftrags an **d.ecs rendition service**.

> **Anmerkung**
>
> In dieser Hook-Funktion werden über die Hook-Eigenschaftsfelder **rendition_parameter_name** und **rendition_parameter_value** Render-Optionen an d.ecs rendition service übertragen.
>
> 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

Parameter	Beschreibung
doc	das Dokument, von dem ein TIFF- oder PDF-Abbild erstellt werden soll
docType	Dokumentart dieses Dokuments
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, Document doc, DocumentType
docType, String sourcePath, String sourceFilename, String destFilename ){
      // (6)
      d3.log.error("Hello world!");
      // (7)
      return 0;
   } // end of doSomething
} // end of D3Hooks
```

Beschreibung der Blöcke

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

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 von d.ecs 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 Abbilddatei
errorCode	0: Erfolg -1: Fehler beim Abholen der Datei von d.ecs rendition service (siehe d.3-Protokolldatei)
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
```

Beschreibung der Blöcke

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

Login

hook_val_passwd_entry_10

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

Aufrufzeitpunkt: Vor der Prüfung von Benutzername und Passwort durch die API-Funktion **Validate-PasswordForUser**. 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 übertragen wurde, z.B. **049** für Deutsch oder **001** für Englisch
appVersion	Versionsstring, der von der Anwendung übertragen 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: Protokollierungsstufe (Loglevel), z.B. 9 für DEBUG

Rückgabe:

Ein Wert **!= 0** führt zur Änderung des Rückgabewerts der API-Funktion **ValidatePasswordForUser** und somit zum Abbruch des Anmeldevorgangs. Der Rückgabewert des Hooks wird von 9500 abgezogen (Beispiel für den Wert -1: 9500-(-1) = 9501). Die Zahl wird an den Client zurückgegeben und muss somit in **msglib.usr** gespeichert 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_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
```

Beschreibung der Blöcke

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.

6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

hook_val_passwd_exit_10

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

Aufrufzeitpunkt: Der Test von Benutzername und Passwort in Bezug auf den d.3-Benutzerstamm oder ggf. einen Directory Server (per LDAP/Kerberos) sind gelaufen. Das Ergebnis steht fest und wird als Parameter **error** übertragen.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
errorCode	Fehlercode der Prüfung von Benutzername/Passwort: • 0: Erfolg • 0002: Benutzername/Passwort ungültig
user	anzumeldender d.3-Benutzer
appLanguage	Sprach-ID, die von der Anwendung übertragen wurde, z.B. **049** für Deutsch oder **001** für Englisch
appVersion	Versionsstring, der von der Anwendung übertragen 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
```

Beschreibung der Blöcke

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.

5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

Löschen eines Dokuments (DeleteDocument)

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 durch einen Rückgabewert ungleich 0 noch abgebrochen werden.

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

Privilegiertes Löschen:

Das Löschen von Dokumenten erfolgt standardmäßig durch Verschieben in den internen Papierkorb. Mithilfe des privilegierten Löschens können Sie jedoch eine Dokumentversion sofort vollständig aus dem System (Datenbank, Dokumentenbaum und ggf. auch vom Sekundärspeichersystem) löschen. Sie benötigen eine separate Lizenz von der d.velop AG und ggf. eine zusätzliche Beratung. In diesem Einsprungpunkt können Sie das privilegierte Löschen aktivieren, indem Sie die Hook-Eigenschaft **DELETE_PRIVILEGED** verwenden.

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")
```

```
// (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
```

Beschreibung der Blöcke

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

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. Der Löschvorgang kann in diesem Einsprungpunkt nicht mehr abgebrochen werden. Sie können über den Parameter **errorCode** prüfen, ob der Löschvorgang erfolgreich war.

Parameter	Beschreibung
d3	die d.3-Schnittstelle
doc	das zu löschende Dokument
user	Benutzer, der das Dokument löscht
errorCode	0: Dokument wurde erfolgreich gelöscht Wenn das Löschen fehlschlägt, erscheint ein 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
```

Beschreibung der Blöcke

1. Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen *doSomething* mit einem sprechenden Namen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert *0* beendet.

Löschen von Verknüpfungen (Unlink)

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 Groovy-Kontext auch weggelassen werden kann.
3. Um 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 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 mit einem sprechenden Namen vergeben werden.
5. Mittels einer einfachen Protokollfunktion wird eine Ausgabe in der Protokolldatei erzeugt.
6. Die Funktion wird mit einem Return-Wert **0** beendet.

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: Übergeordnetes und untergeordnetes Element sind identisch bzw. ein Element existiert 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 Groovyx-Kontext 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 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 mit einem sprechenden Namen vergeben werden.
5. Mittels einer einfachen Protokollfunktion wird eine Ausgabe in der Protokolldatei erzeugt.
6. Die Funktion wird mit einem Return-Wert **0** beendet.

Postkorb (SendHoldFile)

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

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

1. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen doSomething mit einem sprechenden Namen.
2. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
3. Die Funktion wird mit dem Return-Wert **0** beendet.

Redlining (WriteRedline)

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

<listitem>
Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
</listitem>
<listitem>
Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
</listitem>
<listitem>
Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
</listitem>
<listitem>
Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
</listitem>
<listitem>
Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
</listitem>
<listitem>
Die Funktion wird mit dem Return-Wert **0** beendet.
</listitem>

- Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.

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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren *Condition*-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

Senden einer Wiedervorlage (Send Holdfile)

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, das 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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-

Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
5. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
6. Die Funktion wird mit dem Return-Wert **0** beendet.

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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
5. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
6. Die Funktion wird mit dem Return-Wert **0** beendet.

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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
5. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
6. Die Funktion wird mit dem Return-Wert **0** beendet.

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 Datenbankbefehl, der die Wiedervorlage aktiviert.

Parameter	Beschreibung
d3	die d.3-Schnittstelle

Parameter	Beschreibung
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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
5. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
6. Die Funktion wird mit dem Return-Wert **0** beendet.

Senden von E-Mails bei Wiedervorlage

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

Anmerkung
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 anhängen 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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
5. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
6. Die Funktion wird mit dem Return-Wert **0** beendet.

hook_send_email_entry_20

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

Aufruf: Vor dem Versenden einer E-Mail. E-Mail-Adresse wurde ermittelt, Gruppenauflösung wurde durchgeführt.

> **Anmerkung**
> 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 anhängen 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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
5. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
6. Die Funktion wird mit dem Return-Wert **0** beendet.

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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
5. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
6. Die Funktion wird mit dem Return-Wert **0** beendet.

Sperren eines Dokuments

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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
5. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
6. Die Funktion wird mit dem Return-Wert **0** beendet.

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
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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
5. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
6. Die Funktion wird mit dem Return-Wert **0** beendet.

Stammdaten

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., wenn dieser Hook Änderungen am Benutzerobjekt vornimmt, 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)
      def retval
      retval = newUser.setOptField(1, "myValue");
      retval = newUser.setPhone("012345-456789");
      retval = newUser.setPlant("myPlant");
      retval = newUser.setDepartment("myDepartment");
      // (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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
5. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
6. Die Funktion wird mit dem Return-Wert **0** beendet.

Statustransfer

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)

Parameter	Beschreibung
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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
5. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
6. Die Funktion wird mit dem Return-Wert **0** beendet.

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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
5. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
6. Die Funktion wird mit dem Return-Wert **0** beendet.

Validieren von Eigenschaftswerten (ValidateAttributes)

hook_validate_import_entry_10

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

> **Anmerkung**
> 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**-Datei mit einem beliebigen Text für den Returncode (z.B. 9542) zu verwenden.
>
> Weitere Informationen finden Sie hier: Nummernkreis für Returnwerte.

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

> **Anmerkung**
> 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. Die Funktion 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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
5. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
6. Die Funktion wird mit dem Return-Wert **0** beendet.

hook_validate_search_entry_10

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

> **Anmerkung**
> 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**-Datei mit einem beliebigen Text für den Returncode (z.B. 9542) zu verwenden.

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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren *Condition*-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

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 Eigenschaften können hier noch geändert werden. Sonst, wenn keine Dokument-ID vorgegeben wird, wird der 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 von **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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren *Condition*-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

Verknüpfen von Dokumente bzw. Akten (LinkDocuments)

hook_link_entry_30

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

> **Anmerkung**
> 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 Datenbankbefehl, 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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
5. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
6. Die Funktion wird mit dem Return-Wert **0** beendet.

hook_link_exit_10

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

Aufrufzeitpunkt: Direkt nach Ausführung des Datenbankbefehls, der die Verknüpfung einträgt.

Parameter	Beschreibung
docFather	das übergeordnete Dokument, bzw. die Akte
docChild	das untergeordnete Dokument
linkCode	0: Verknüpfung war erfolgreich -1: Übergeordnetes und untergeordnetes Element 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 (siehe **errorCode**) sonst: Datenbank-Fehlernummer beim Eintrag der Verknüpfung in die Datenbank
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_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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
5. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
6. Die Funktion wird mit dem Return-Wert **0** beendet.

Web-Veröffentlichung

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 bzw. 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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
5. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
6. Die Funktion wird mit dem Return-Wert **0** beendet.

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 bzw. zurückgezogen werden soll

Parameter	Beschreibung
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_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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren *Condition*-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

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 bzw. 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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren *Condition*-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

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 bzw. 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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren *Condition*-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

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 bzw. zurückgezogen wurde
user	der ausführende Benutzer
errorCode	0= Aktion erfolgreich Sonst: Fehlercode
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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren *Condition*-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen.
6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

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 bzw. zurückgezogen wurde
user	der ausführende Benutzer
errorCode	0= Aktion erfolgreich Sonst: Fehlercode
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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren *Condition*-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen.

6. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
7. Die Funktion wird mit dem Return-Wert **0** beendet.

Workflow

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 Workflowdurchlauf 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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
5. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
6. Die Funktion wird mit dem Return-Wert **0** beendet.

1.5.7. Validierungshooks

Sie können Validierungshookfunktionen zur individuellen Eingabevalidierung verwenden (Plausibilitätshooks).

- Der Funktionsname (z.B. **ITValueValidation)** wird bei der Eigenschaft unter **Plausibilitätskontrolle/Hook Funktion** eingetragen.
- Die Annotation lautet: **@Validation(entrypoint="<Bezeichner in d.3 Adminstration>")**
- Die Funktion wird nur ausgeführt, wenn ein Wert bei der entsprechenden Eigenschaft im Client angegeben wurde.
- Die Funktion wird erst nach **hook_insert_entry_10** ausgeführt.
- Rückgabewert:
  - **0** bei Erfolg
  - <> **0** bei einem 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 -8000;
   }
}// end of validateValue
```

Validierung einer Bestellnummer auf ein gültiges Format

> **Anmerkung**
> Szenario:
>
> Die Bestellnummer soll immer dem Format "Zwei Zahlen-Zwei Buchstaben-Fünf Zahlen" (/[0-9]{2}-[a-zA-Z]{2}-[0-9]{5}/) entsprechen. Dieser Vorgang dient als Beispiel, um die Funktionsweise für die Validierung zu demonstrieren.

Definieren Sie für die Dokumenteigenschaft **Bestellnummer** eine Funktion zur Validierung.

```
//(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
```

Der Vorgang kann mit Groovy etwas kürzer umgesetzt 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. Damit Sie eine Groovy-Methode für die Validierung einer Dokumenteigenschaft verwendenkönnen, erfolgt über die Annotation **@Validation** eine Registrierung der Methode für eine in d.3 admin konfigurierte Validierungsfunktion.
4. Die Methode nimmt die oben beschriebenen Parameter in der vorgegebenen Reihenfolge entgegen.
5. Sie können innerhalb der Methode nun den übertragenen Wert überprüfen. Im Beispiel erfolgt die Prüfung über einen regulären Ausdruck.
6. Wenn der Wert gültig ist, wird eine **0** zurückgegeben. Wenn der Wert ungültig ist, wird eine **1** zurückgegeben.

1.5.8. Wertemengen-Hooks

Mit Wertemengenhooks können Sie dynamische Wertemengen erzeugen.

- Der Funktionsname (z.B. **customerNumbers)** wird bei der Eigenschaft unter **Wertevorrat/Hook Funktion** eingetragen.
- Die Annotation lautet: **@ValueSet(entrypoint="<Bezeichner in d.3 Administration>")**
- Maximal können mit einen Wertemengenhook 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

Parameter	Beschreibung
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	Dokumentobjekt mit dem Attributkontext. Dieses Objekt enthält bei der Suche die übrigen Suchkriterien. Dieses Objekt enthält beim Import und Update die übrigen bereits gefüllten Attribute.

Einfache Wertmengen

Starten Sie mit einer statischen, einfachen Wertemenge, die über das Skript zur Verfügung gestellt wird.

Einfache statische Wertmenge

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

      // (5) Define static list of customer numbers
----------------------------
      def customerList = ["4711", "4712", "4713", "4714" ];

      // (6) Prepare List for interaction
-----------------------------------
      if( customerList.size() > 0 ){
         reposField.provideValuesForValueSet( customerList );
      }
   } // end of getCustomerNumber
}// 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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
5. Mittels Protokollfunktion wird eine Ausgabe in der Protokolldatei erstellt.
6. Die Funktion wird mit dem Return-Wert **0** beendet.

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) 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 sql statmenet
-------------------------------------------------------------
      def sqlQuery = "SELECT customerNo FROM CustomerData ORDER BY
customerNo DESC"; // !! ATTENTION

      // (6) Execute sql statmenet
-------------------------------------------------------------
      def resultRows = d3.sql.executeAndGet( (String) sqlQuery );

      // (7) 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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen.
5. Bereitstellung eines SQL-Statements zur Ermittlung der notwendigen Werte aus der Datenbank.
6. Ausführung des SQL-Statements gegenüber der d.3-Datenbanktabelle.
7. Bereitstellung der Werte als Auswahlliste für Anwendende.

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.

```
// (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=<DateBaseName>", "<DatabaseUser>",
„<Password>" );

      // (6) Prepare sql statmenet ----------------------------------------
      def sqlQuery  = "SELECT customerNo FROM CsutomerData 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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen.
5. Aufbau der JDBC-Verbindung zur externen Datenbank. Sie müssen den dazugehörigen JDBC-Treiber entsprechend zur Verfügung stellen.
6. Bereitstellung eines SQL-Statements zur Ermittlung der notwendigen Werte aus der Datenbank.
7. Ausführung des SQL-Statements gegenüber der d.3-Datenbanktabelle.
8. Bereitstellung der Werte als Auswahlliste für Anwendende.

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.

Wir empfehlen eine Ansicht auf die Tabelle des führenden Systems zu erstellen, damit keine Benutzerdaten in Skripte eingetragen werden müssen.

```
// (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>\
\<InstacneName>:0;databaseName=<DatabaseName>", "<DatabaseUser>",
„<Password>" );

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

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

reposField.provideValuesForValueSet( resultRows.collect{ it.completeName } )
;
      }
   } // end of getCustomerNumber
} // 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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen.
5. Definition der Filtervariablen und die Übergabe der erweiterten Eigenschaften.
6. Die Inhalte der erweiterten Eigenschaften werden kontrolliert. Bei fehlenden oder zu wenig bereitgestellten Inhalten wird die Generierung abgebrochen.
7. Zusammenstellung des SQL-Statements inkl. der dynamischen Zuordnung der bereitgestellten Filterkriterien.
8. Aufbau der JDBC-Verbindung zur externen Datenbank. Sie müssen den dazugehörigen JDBC-Treiber entsprechend zur Verfügung stellen.
9. Ausführung des SQL-Statements gegenüber der d.3-Datenbanktabelle.
10. Bereitstellung der Werte als Auswahlliste für Anwendende.

Übersetzung von dynamischen Hook-Wertemengen

Es ist möglich, dynamische Hook-Wertemengen zu übersetzen.

- Die Annotation lautet: **@ValueSetTranslation(entrypoint="<Bezeichner in d.3 Administration>")**
- Der Entry-Point-Name entspricht dem Namen des zugehörigen @ValueSet-Hooks z. B. **MyMonths.**
- Der Übersetzungshook 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 zwischengespeichert. Die Dauer, für die diese Werte gespeichert bleiben, ist implementationsabhängig und kann sich mit zukünftigen Versionen ändern. Sie können mit der Funktion **d3.getArchive().removeTranslationFromCache()** ein Neuladen der Übersetzungen zu jeder Zeit erzwingen. Sie können diese Funktion von beliebiger Stelle aus aufrufen, 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 Speicherwert übertragen, nicht die Übersetzungen. Eine Volltextsuche nach Übersetzungen ist somit nicht möglich.
- Die Kombination aus Sprache und Anzeigewert darf pro Wertemenge nur einmalig verwendet werden. Wenn dies nicht gewährleistet werden kann, empfehlen wir, den Speicherwert als Präfix oder Suffix der Übersetzung zu nutzen um eine Eindeutigkeit zu erzielen.
- Falls nicht zu jedem Wert der Wertemenge eine Übersetzung vorliegt, müssen Sie sicherstellen, dass diese unübersetzten Werte nicht den Übersetzungen anderer Werte dieser Wertemenge entsprechen.

def myValueSetTranslation(D3Interface d3, Translation transl)

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

Beispiel-Hook

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

```
    import com.dvelop.d3.server.ValueSetTranslation


    @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", "");
            transl.set("02", "");
            transl.set("03", "");
            // ...
        } else {
            transl.set("01", "January");
            transl.set("02", "February");
            transl.set("03", "March");
            // ...
        }
    }
```

Zugehörige dynamische Wertemengenfunktion

Beispiel-Hook

```
@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
```

Weitere Beispiele für Wertemengen-Hooks finden Sie hier.

1.5.9. Dokumentklassenhooks

Dokumentklassenhooks können zur Bestimmung dynamischer Berechtigungen verwendet 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 Hookfunktion 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

Beispielhook

```
// (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 == "Smith"){
            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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen.

Anmerkung

Dokumentklassenhooks werden bei einer Suche für jedes einzelne Dokument in der Ergebnismenge aufgerufen.

Das kann zu Performanceproblemen bei der Rechteprüfung führen und somit die Dokumentensuche verlangsamen, insbesondere dann, wenn im Dokumentklassenhook SQL-Kommandos abgesetzt werden.

Noch größere Auswirkungen auf die Performance bestehen, wenn Dokumentklassenhooks für Mehrfacheigenschaftsfelder (60er-Felder) eingesetzt werden, da diese dann zusätzlich jede mit einem Wert belegte Zeile jeder Mehrfacheigenschaft aufgerufen werden.

Warnung

Dokumentklassenhooks werden bei der Dokumentensuche parallel in mehreren Threads ausgeführt. Diese Hookfunktionen dürfen daher nur threadsicheren Code enthalten und aufrufen.

1.6. Groovy-Hook-Beispiele

1.6.1. Eintrittspunkte

Anmerkung

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

> **Anmerkung**
> Sie können auf jede beliebigen Hookeintrittspunkt beliebig viele Funktionen definieren. Sie können 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 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 übertragen.

> **Anmerkung**
> Für die Hookeintrittspunkte **Wertemengen**, **Validierung** und **Suche** muss immer am Ende noch ein zusätzlicher Parameter vom Typ **Document** angehängt werden.

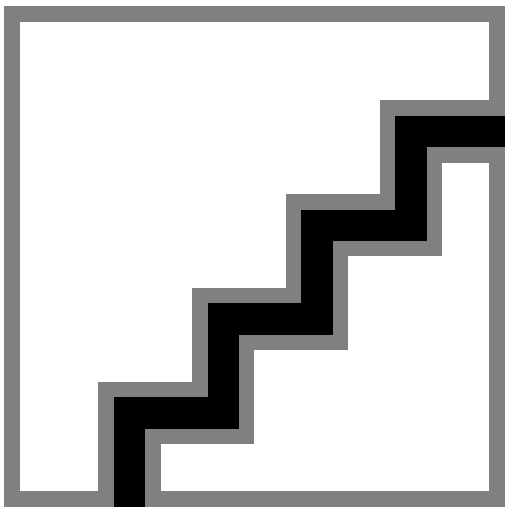
Eine Kette von Eintrittspunkten

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 Serverseite 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 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 Beispiel nicht archiviert.

InsertEntry_10

Hallo Welt!

> **Anmerkung**
> Szenario:
>
> Wenn ein Dokument im d.3-System abgelegt wird, soll in der d.3-Protokolldatei einfach nur eine Fehlermeldung "Hallo Welt" angezeigt werden.

Dazu wird eine Hookfunktion für den Eintrittspunkt "**hook_insert_entry_10**" gespeichert, die die Kundennummer überprüft.

Beispiel

```
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;

//(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)
      d3.log.error("Hello world!");
//(6)
      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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Zusätzlich wird als letzter Parameter noch ein Parameter vom Type **Document** benötigt.
5. Nun wird mittels einfacher Protokollfunktion eine Ausgabe in der Protokolldatei erzeugt.
6. Die Funktion wird mit einem Return-Wert **0** beendet.

Kontrolle und Ergänzung von Dokumenteigenschaften bei Import

Anmerkung

Szenario:

Während der Ablage eines Dokumentes im d.3-System soll die Eigenschaft **Kunden-Nummer** mit einer Datenbank kontrolliert und bei Bedarf die restlichen Kundendaten automatisch ergänzt werden.

Die benötigten Kundendaten in diesem Beispiel liegen dabei in einer Datenbanktabelle, die innerhalb der d.3-Datenbank gespeichert wurden, damit können die Daten über eine einfache SQL-Implementierung abgerufen werden.

Dazu wird eine Hookfunktion für den Eintrittspunkt "**hook_insert_entry_10**" gespeichert, die die Kundennummer überprüft.

Beispiel

```
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 == "DRECH" ){
          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["Strasse"] = resultRows[0].street;
               doc.field["PLZ"]     = 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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. 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 mittels **Condition** die Einschränkung auf einen oder mehrere Dokumenttypen vorgenommen werden.
6. Nachdem die Kundennummer aus den Dokumenteigenschaften ermittelt wurde, kann die Kundennummer mit der Datenbank geprüft werden. Liegt dabei die Datenbanktabelle im d.3-DB-Adressraum, kann mittels der d.3-SQL-Implementierung die native Datenbankschnittstelle des d.3-Servers verwendet 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 Datenbankposition oder direkt und sprechend mit der Eigenschaftenbezeichnung referenziert werden. Einigen Sie sich auf eine Art der Referenzierung.
8. Wenn keine gültige Kundennummer ermittelt werden konnte, erfolgt eine Rückgabe mit einem Returnwert ungleich **0** und damit wird dann an dieser Stelle die Verarbeitungskette unterbrochen.

InsertExit_20

Eintrag in den Dokumentnotizen

> ### Anmerkung
> 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 Hookfunktion für den Eintrittspunkt **hook_insert_exit_20** hinterlegt; der Eintrag wird dann mittels Groovy-Server-API geschrieben..

Beipsiel

```
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_exit_20" )
//(4)
   @Condition( doctype = "DRECH" )
//(5)
   public int insertExit_20( D3Interface d3, Document doc, def fileDest,
def importOK,
                             User user, DocumentType docTypeShort ){
//(6)
       def returnValue = 0;
       returnValue = d3.call.note_add_string( "Hello World!", doc.id,
user.id );
//(7)
      return  0;
   } // end of insertExit_20
} // 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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen.
6. Nun können Sie mit der Groovy-Server-API-Funktion **note_add_string** einen Eintrag zur Dokumentnotiz hinzufügen.
7. Wenn die Aktion erfolgreich war, wird ein Return-Wert gleich **0** zurückgegeben.

Weiterleitung an einen Sachbearbeiter

> **Anmerkung**
> 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.

Beispiel

```
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 = "DRECH" )
//(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("InvoiceNo"),
                                            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 Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen.
6. Nun können Sie mit der Groovy-Server-API-Funktion **note_add_string** einen Eintrag zur Dokumentnotiz hinzufügen.
7. Wenn die Aktion erfolgreich war, wird ein Return-Wert gleich **0** zurückgegeben.

Update von anderen Dokumenten abhängig vom aktuellen

> **Anmerkung**
> 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.

Update von anderen Dokumenten abhängig vom Import eines neuen Dokumentes

```
//(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.EMPLOYEE_NAME];
//(7)
   def sqlQuery   = "SELECT .... ";
   def resultRows = d3.sql.executeAndGet( sqlQuery );
   def childDocObj;
//(8)
   resultRows.each {
      childRef = it.doku_id;
//(9)
      childDocObj = archiveObj.getDocument( childRef, login.id );
//(10)
      docObjRef.field[DDF.persName] = persName;
//(11)
      docObjRef.updateAttributes( login.id, true ); // !!!!
   }

   return  0;
} // 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_exit_20**.
3. Aufruf der separaten Funktion zur Aktualisierung der Personaldaten.
4. Funktionsdefinition der Updatefunktion.
5. Bereitstellung eines **ArchivInterface**-Objekts zwecks Generierung der Document-Objekte.
6. Die benötigten Daten werden aus dem aktuellen Dokument ausgelesen, hierzu wurde hier exemplarisch eine externe 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 wichtig, dieser sorgt mit **true** dafür, dass eine nachgelagerte Hook-Validierung nicht stattfindet, **false** hätte hier den gegenteiligen Effekt.

UpdateAttribEntry_20

Kontrolle und Ergänzung von Dokumenteigenschaften bei der Aktualisierung

> **Anmerkung**
> Szenario:
>
> Wenn die Dokumenteigenschaft **Kunden-Nummer** aktualisiert wird, sollen hier ebenfalls die Daten mithilfe einer Datenbank kontrolliert und bei Bedarf die restlichen Kundendaten automatisch ergänzt werden. Die benötigten Kundendaten in diesem Beispiel liegen dabei in einer externen Datenbank und müssen über eine externe Datenbankverbindung abgerufen werden.

Dazu wird eine Hook-Funktion für den Eintrittspunkt **hook_upd_attrib_entry_20** hinterlegt, welche die Kunden-Nummer überprüft.

Beispiel

```
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://
<ServerName>:<Port>;databaseName=<DataBaseName>",
                                             "<User>", "<Password>" );
         def sqlQuery     = "SELECT name, street, zioCode, 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

<listitem>
Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
</listitem>
<listitem>
Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
</listitem>
<listitem>
Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
</listitem>
<listitem>
Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen.
</listitem>

1. IImport der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
2. Für die Hook-Eintrittspunkte **Wertemengen**, **Validierung** und **Suche** muss immer am Ende noch ein zusätzlicher Parameter vom Typ **Document** angehängt werden.
3. Nun kann innerhalb der Funktion für einen bestimmten Dokumenttyp eine Validierung vorgenommen werden, dabei wird der Dokumenttyp über die Eigenschaft **pDocTypeShort.id** eingeschränkt.
4. Nachdem die Kunden-Nummer aus den Dokumenteigenschaften ermittelt wurde, kann nun die Kunden-Nummer mithilfe der Datenbank geprüft werden. Dabei wird in diesem Beispiel eine Verbindung zu einer externen Microsoft-SQL-Datenbank vorgenommen und die Daten aus dieser Datenbank ermittelt. Details zu der notwendigen Konfiguration können den relevanten Dokumentationen entnommen werden.
5. 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.
6. 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.

Mehrere Hookfunktionen in einer Klasse

Wie können mehrere Hookfunktionen in einer Klasse zusammengeführt werden? Eine Antwort könnte mit dem folgenden Skriptbeispiel 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 == "DRECH" ){
         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:<ServerName>\
\<InstanceName>:0;databaseName=<DataBaseName>", "<DatabaseUser>",
"<Password>" );
         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, "Content from Groovy" );
            return 0;
         }
         else{
            return -1;
         }
      }
      return 0;
   } // end of getCustomerData
} // end of d3Hooks
```

Kommentare zu den einzelnen Blöcken

<listitem>
Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
</listitem>
<listitem>
Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
</listitem>
<listitem>

Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
</listitem>
<listitem>
Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
</listitem>

1. In diesem Beispiel wird der Dokumenttyp innerhalb der Funktion über ein If-Statement kontrolliert und für die Dokumentart **Rechnung** eine entsprechende Aktion ausgeführt.
2. 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 Dokumentnotizen vorgenommen. Zusätzlich wird die Rechnung noch mittels des Befehls **hold_file_send** an einen Sachbearbeiter gesendet.
3. Für einen weiteren Eintrittspunkt **hook_validate_import_entry_10** wird hiereine Funktion registriert, welche aber aktuell noch keine Aufgabe übernimmt.
4. Wenn zum Beispiel in einer Rechnung, die Kundennummer geändert wird, müssen auch die relevanten Kundendaten in der Rechnung ebenfalls angepasst werden, dazu wird der Eintrittspunkt **hook_upd_attrib_entry_20** mit einer Funktion verknüpft. Da die Ermittlung der Kundendaten an mehreren Stellen benötigt wird, wurde die Funktion an dieser Stelle implementiert.

1.6.2. Validierung

> **Warnung**
> 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

> **Anmerkung**
> 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 können Sie 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.

```
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
```

In Groovy können Sie die gleiche Funktion noch kompakter gestalten.

```
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 Datei **groovyhook.jar**.
2. Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Optional können Sie die Funktionen mit einer weiteren **Condition**-Annotation bestimmten Dokumentklassen zuordnen.
5. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.

6. Innerhalb der Funktion kann nun der übergebene Wert überprüft werden, im Beispiel mittels eines regulären Ausdrucks.
7. Entspricht der Wert einem gültigen Wert, kann eine **0**, ansonsten eine **1** zurückgegeben werden.

1.6.3. Wertemengen

In diesem Bereich werden die Möglichkeiten dargestellt, Wertemengen wie folgt bereitzustellen:

- Als statische Liste
- Aus internen d.3-Datenbanktabellen
- Aus externen Datenbanktabellen
- Als dynamische abhängige Datenbanktabelle
- Als übersetzte Wertemenge

Benötigte Bibliotheken für die Wertemengen

> **Warnung**
> 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

> **Warnung**
> Die Sortierung wird hier aus dem Skript übernommen und nicht, wie über eine JPL-Wertemenge, vom Client wieder verfälscht.

Einfache statische Wertemenge

> **Anmerkung**
> Es sit nicht sinnvoll, eine statische Liste über ein Skript bereitzustellen. Der bessere Weg ist die d.3-Administration. Aber für ein einfaches Beispiel kann man hier mit einer statischen Wertemenge starten.

Einfache statische Wertemenge

```
//(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) Define the needed class
-------------------------------------------------
class SimpleValueSet{
   //(5) Combine to 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) Define static list of customer numbers
-----------------------------
      def customerList = ["4711", "4712", "4713", "4714" ];

      //(8) Prepare List for
interaction---------------------------------------
      if( customerList.size() > 0 ){
         reposField.provideValuesForValueSet( customerList );
      }
   } // end of getCustomerNumber
}// 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. externen Datenquellen zu ermitteln, dies wird in diesen Beispielen vorgestellt.

Interne d.3-Datenbanktabelle

> **Anmerkung**
> 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 customerNo 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.kunden_nr } );
      }
   } // 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

> **Anmerkung**
> 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;
import groovy.sql.Sql;

//(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:<Server>\
\<instacneName>:0;databaseName=<DataBaseName>", "<UserName>", „<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.kunden_nr } );
```

```
      }

   } // 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 mithilfe der internen Datenbanktabelle, 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

> ### Anmerkung
> Szenario:
>
> Es soll eine Wertemengen der Kunden-Nummern aus den Kunden-Datenbanktabelle für die Dokumenteigenschaft **KundenNr** bereitgestellt werden. Diese Eigenschaft soll bei Bedarf über den eingegebenen PLZ-Bereich dynamisch begrenzt werden können.

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

Beispiel

```
//(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 D3ValueSets{
   //(3)
   @ValueSet( entrypoint = "dsCustomerNumbers" )
   //(4)
   public int getCustomerNumber( D3Interface d3, RepositoryField
reposField, User user, DocumentType docType, int  rowNo, int  validate,
Document doc )
   {
```

```
      //(5)
      def  zipCode     = doc.field[9];
      def  whereClause = "";
      if( plz != "" ){
         whereClause = "WHERE zipCode LIKE '$zipCode%'";
      }
      //(6)
      def sqlQuery   = "SELECT customerNo FROM CustomerData $whereClause
ORDER BY customerNo DESC"; // !! ATTENTION
      def resultRows = d3.sql.executeAndGet( sqlQuery );
      //(7)
      // Long-Version - Collecting a list and provide list in 2 steps
-----------------------------------------------------------
      def customerList = [];
      resultRows.each{
         customerList.add( it.kunden_nr );
      }
      if( customerList.size() > 0  ){
         reposField.provideValuesForValueSet( customerList );
      }

      // Short-Version Collecting a list and provide list in 2 steps
-----------------------------------------------------------
      if( resultRows.size() > 0 ){
         reposField.provideValuesForValueSet( resultRows.collect{
it.kunden_nr  } );
      }

      return 0
   } // end of getCustomerNumber
} // 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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
5. Um die Aufgabenstellung zu realisieren, wird die Dokumenteigenschaft der PLZ ausgelesen und bei Bedarf ein **Where**-Bestandteil des SQL-Statements zur Verfügung gestellt.
6. Da die verwendete Datenbanktabelle innerhalb der d.3-Datenbank erstellt wurde, kann mittels einer einfachen SQL-Implementierung auf die Daten zugegriffen werden, siehe hierzu auch das Beispiel für **InsertEntry_10**.
7. Wenn Daten zu dem bereitgestellten Postleitzahlenbereich gefunden wurden, können diese Daten mittels der Funktion **provideValuesForValueSet()** der Dokumenteigenschaft bereitgestellt werden. Hier wurden zwei unterschiedliche Wege der Implementierung aufgenommen. Verwenden Sie nur einen der beiden Wege.

Bereitstellung übersetzte Wertemenge

Szenario

Anmerkung

Es soll eine Wertemenge aus der Tabelle **custom_articles** zur Verfügung gestellt werden. Dabei soll in der Datenbank von d.velop documents die Artikelnummer gespeichert werden. Abhängig von der eingestellten Sprache des Browsers sollen die Artikel auf Englisch oder Deutsch übersetzt werden.

Achtung

Bei übersetzten Wertemengen muss darauf geachtet werden, dass die Übersetzungen jeweils eindeutig sind.

In der Volltextdatenbank werden die Werte gespeichert, die in der Datenbank gespeichert sind. In diesem Beispiel sind das die Artikelnummern. Das bedeutet, es kann nicht nach den Artikelnamen (Deutsch/Englisch) per Volltext gesucht werden.

Beispiel

```
//(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 D3ValueSets{
    def gArticleListTranslated = false
    //(3)
    @ValueSet(entrypoint="articleList")
    //(4)
    def getArticleList(D3Interface d3, RepositoryField repoField, User
user, DocumentType docType, Integer row_no, Integer validate, Document doc)
{
        //(6)
        String sqlQuery = "select articleNumber as id, articleName as value
from custom_articles order by value"
        def resultRows = d3.sql.executeAndGet( sqlQuery );
        //(7)
        repoField.provideValuesForValueSet( resultRows.collect{
it.id.toString() } );
        //translate value list if row count has changed
        if (gArticleListTranslated != resultRows.size())
        {
            d3.log.info("groovyHook - articleList -> translate")
            //(8)

d3.getArchive().removeTranslationFromCache("gesellschaftenList", new
Locale("de"));
            //(9)
```

```
d3.getArchive().removeTranslationFromCache("gesellschaftenList", new
Locale("en"));
            gArticleListTranslated = resultRows.size();
        } else {
            d3.log.info("groovyHook - articleList -> from cache")
        }
    }
    //10
    @ValueSetTranslation(entrypoint="articleList")
    def translateArticleList(D3Interface d3, Translation transl) {
       def sqlQuery
       //11
       if(transl.locale.language == "en"){
          sqlQuery = "select articleNumber as id, articleNameDE as value
from custom_articles"
       } else if(transl.locale.language == "en"){
          sqlQuery = "select articleNumber as id, articleNameEN as value
from custom_articles"
       } else {
          d3.log.info("no translation for language " +
transl.locale.language + " -> english translation")
          sqlQuery = "select articleNumber as id, articleNameEN as value
from custom_articles"
       }
       def resultRows = d3.sql.executeAndGet( sqlQuery );
       resultRows.each {
          //12
          transl.set(it.id.toString(), it.value);
       }
    }

} // 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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
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 verwendete Datenbanktabelle innerhalb der d.3-Datenbank erstellt wurde, kann mittels einer einfachen SQL-Implementierung auf die Daten zugegriffen werden, siehe hierzu auch das Beispiel für **InsertEntry_10**.
7. Über die Funktion **provideValuesForValueSet()** werden der Dokumenteigenschaft die Werte bereitgestellt.
8. Die Übersetzung für **de** für Deutsch wird ausgelöst.
9. Die Übersetzung für **en** für Englisch wird ausgelöst.

10. Um nun eine Groovy-Funktion für Wertemengenübersetzung einer Wertemenge nutzen zu können erfolgt, über eine Groovy-Annotation mit den vorgegebenen Werten, eine Registrierung der Funktion zu einer Wertemengenfunktion, die in d.3 admin konfiguriert ist.
11. Ein SQL-Statement zur Ermittlung der Werte für die Übersetzung wird in Abhängigkeit der transferierten Sprache erstellt.
12. Über die Funktion **transl.set()** wird pro Wert (id) die Übersetzung (value) geliefert.

1.6.4. Dokumentklassen

> **Warnung**
> 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

> **Anmerkung**
> 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 erstellt, die benötigte Implementierung ist im folgenden Beispiel dokumentiert.

Beispiel

```
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 = "CustomerInvoice" )
//(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 == "stark" );
            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 von Groovy auch weggelassen werden kann.
3. Um eine Groovy-Funktion einem Eintrittspunkt zuzuweisen, erfolgt über eine Groovy-Annotation mit den vorgegebenen Werten eine Registrierung der Funktion zu dem Eintrittspunkt.
4. Die unterschiedlichen Eintrittstypen benötigen die in der Groovy-Hook-Dokumentation beschriebenen Parameter in der vorgegebenen Reihenfolge. Zunächst wird bei dem Aufruf einer Groovy-Hook-Funktion immer das d.3-Objekt übertragen. Der Funktionsname muss nicht dem Eintrittspunkt entsprechen. Versehen Sie den Funktionsnamen **doSomething** mit einem sprechenden Namen.
5. In diesem Beispiel wird nun über eine Switch-Abfrage abhängig von dem dritten Wert der Kundennummer ein statischer User gesetzt. Sie können auch mit Restriction-Sets arbeiten.

> **Warnung**
> Bitte beachten Sie hier, dass Dokumentklassen-Hooks mit viel Vorsicht genutzt werden sollten, da sich die Hooks, je nach implementierter Funktion, stark auf die Performance des Gesamtsystems auswirken.

6. Je nachdem, wie diese Prüfung abgelaufen ist, wird nun eine **1** (erlaubt) oder eine **0** (verweigert) zurückgegeben.

1.7. Groovy API-Funktionen

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

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

Diese können wie andere d.3-API-Funktionen über das d.3-Kommunikations-Protokoll d3fc aufgerufen werden.

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

Aktivieren der Plug-in-Schnittstelle

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

Groovy-Skripte, die d.3-API-Funktionen implementieren, werden in diesem Verzeichnis gesucht und daraus geladen.

Dadurch wird die Plug-in-Schnittstelle für API-Funktionen aktiviert.

Damit eine Java-Klasse als d.3 API-Funktion geladen und registriert wird, muss die Java-Klasse 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
    }
}
```

1.7.1. Groovy-API und Nutzung in JPL

Mit der Integration von Groovy als Server-Skriptsprache können Sie eigene API-Funktionen erstellen und nutzen.

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

> **Anmerkung**
> 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

```
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

<listitem>
Import der benötigten Bibliotheken aus der Datei **groovyhook.jar**.
</listitem>
<listitem>
Bereitstellung einer eigenen Klasse vom Typ **public**, wobei **public** im Kontext von Groovy auch weggelassen werden kann. Es ist wichtig, dass die Erweiterung **extendsD3ApiCall** hinzugefügt wird.
</listitem>

1. Die Funktion, die nun als API-Call bereitgestellt werden soll, wird definiert. Dabei können über den Befehl **getImportParams** auch Parameter, die über den Aufruf der Funktion bereitgestellt werden, ausgelesen werden.
2. Mittels der Funktion **setExportTable** stehen dann auch die Ergebnisse aus der Funktion dem aufrufenden Programm zur Verfügung.

Ein d3FC-Aufruf mittels JPL implementiert

Beispielaufruf aus JPL

```
// (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 erstellt.
2. Im nächsten Schritt müssen die Login-Parameter für den d3FC-Aufruf vorgegebenen werden.
3. Um die Funktion gemäß der eigenen Definition aufrufen zu können, muss der Funktionsname festgelegt und die Parameter übergeben werden.
4. Wenn alle notwendigen Einstellungen vorgenommen sind, kann der eigene API-Befehl mittels **d3fc_execute** ausgeführt werden.

5. Die bereitgestellten Rückgabewerte können ausgelesen und in lokale Variablen übernommen werden.
6. Zur Dokumentation der Funktion bzw. der Ergebnisse werden diese hier als Error-Message in der Protokolldatei ausgegeben.
7. Im nächsten Schritt werden nun die einzelnen Kundendaten ausgelesen und ebenfalls in der Protokolldatei ausgegeben.

> **Warnung**
> Damit die definierten API-Funktionen vom d.3-Server gefunden und registriert werden, muss in der d.3-Konfiguration beim Parameter **JAVA_API_FUNCTIONS_DIR** der Pfad zu den Groovy-Dateien mit den erstellten Funktionen angegeben werden.

1.8. 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.Document
import com.dvelop.d3.server.core.D3Interface
import com.dvelop.d3.server.exceptions.D3Exception
D3Interface d3 = getProperty("d3");
String scriptName = getClass().getName()

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

// d.3 User which executes the update of documents
String scriptUser = "d3_groovy"
final def maxRows = 3

//Einzelne Dokumente/Akten anhand der ID
def resultSet = d3.sql.executeAndGet("select doku_id from firmen_spezifisch
where kue_dokuart = 'DTEST' and dok_dat_feld_1 = 'oldValue'", maxRows)
resultSet.each{
    try {
        d3.log.info (scriptName + "> Updating document with ID " +
it.doku_id)
                //Get document object of doc id
        Document myDoc = d3.archive.getDocument(it.doku_id, scriptUser)
                //Set new value
                myDoc.field[1] = "newValue"
                //Update metadata without triggering hook entrypoints
        myDoc.updateAttributes(scriptUser, true)
    } catch (D3Exception e) {
        d3.log.error(scriptName + "> Error updating meta data " +
e.getMessage())
    }
}
```

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 "" d3_server d3_server D3P ext_groovy/myScript.groovy

1.9. 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();
  }
```

> **Anmerkung**
> 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.

> **Anmerkung**
> Die **Javadoc** Dokumentation für die d.3 Schnittstelle ist in der Datei **groovyhook-java-doc.jar** enthalten.
>
> Fügen Sie diese Datei in ihrer Entwicklungsumgebung ihrem Groovy-Projekt hinzu, um eine Beschreibung für die Klassen und Methoden der d.3 Schnittstelle angezeigt zu bekommen.
>
> Für Eclipse wird das genauer beschrieben auf der Seite Erstellen von Hook-Projekten .

1.9.1. d.3 Archiv (ArchiveInterface)

ArchiveInterface

```
    public interface ArchiveInterface {
        public Document             getDocument(String id, String
contextUser); //Laden Objekt vom Typ "Dokument"
        public Document             getDocument(String id); //Laden Objekt
vom Typ "Dokument"
        public DocumentType         getDocumentType(String id); //Laden
Objekt vom Typ "Dokumenttyp"
        public PredefinedValueSet   getPredefinedValueSet(String id); //
Laden Objekt vom Typ "Wertemenge"
        public RepositoryField      getRepositoryField(String id); //
Laden Objekt vom Typ "erweiterte Eigenschaft"
        public User                 getUser(String id); //Laden Objekt vom
Typ "Benutzer"
        public UserGroup            getUserGroup(String id); //Laden Objekt
vom Typ "Gruppe" geladen
        public UserOrUserGroup      getUserOrUseGroup(String id); //
Ermittlung, ob es sich beei dem Objekt um eine Gruppe oder einen Benutzer
handelt
        public AuthorizationProfile getAuthorizationProfile(String id);
//Laden Objekt vom Typ "Berechtigungsprofil"
        public void                 removeTranslationFromCache(String
entryPoint, Locale lang);    //Übersetzung einer Groovy-Hookwertemenge
        public Document             newDocument(); //Erzeugen eine neuen
Objekts vom Typ "Dokument" oder "Akte"
        public Document             importDocument(Document doc, Path
importFilePath); //Erstellen eines Dokuments
        public Document             importDocument(Document doc); //
Erstellen einer Akte
 }
```

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

Document getDocument (String id, String contextUser)

Beschreibung: Mit dieser Funktion wird ein Objekt vom Typ "Dokument" oder "Akte" geladen.

Parameter:

- id: Id der Akte oder des Dokuments, dessen Objekt geladen werden soll
- contextUser: BenutzerId des Benutzers in dessen Kontext das Dokument oder Akte importiert werden soll

Rückgabewerte:

- Dokumentobjekt

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", Const.userHook)
```

Document getDocument(String id)

Beschreibung: Mit dieser Funktion wird ein Objekt vom Typ "Dokument" oder "Akte" geladen.

Parameter:

- id: Id der Akte oder des Dokuments, dessen Objekt geladen werden soll

Rückgabewerte:

- Dokumentobjekt

```
Document myDoc = d3.archive.getDocument("P000000001")
```

DocumentType getDocumentType(String id)

Beschreibung: Mit dieser Funktion wird ein Objekt vom Typ "Dokumenttyp" geladen.

Parameter:

- id: Id des Akten- oder des Dokumenttyp, dessen Objekt geladen werden soll

Rückgabewerte:

- Dokumenttypobjekt

```
DocumentType myDocumentType = d3.archive.getDocumentType("DDOKU");
d3.log.info("default name of document type is
"+myDocumentType.getDefaultName());
d3.log.info("name of document type is "+myDocumentType.getName());
d3.log.info("type of document type is "+myDocumentType.getType());
if(myDocumentType.getType() == "d"){
    d3.log.info("type of this document type is 'document'");
} else if(myDocumentType.getType() == "a"){
    d3.log.info("type of document type is 'folder'");
}
```

PredefinedValueSet getPredefinedValueSet(String id)

Beschreibung: Mit dieser Funktion wird ein Objekt vom Typ "Wertemenge" geladen.

Parameter:

- id: Id der Wertemenge, dessen Objekt geladen werden soll

Rückgabewerte:

- Wertemengenobjekt

```
PredefinedValueSet myPredefinedValueSet =
d3.archive.getPredefinedValueSet("16");
d3.log.info("name of PredefinedValueSet is
"+myPredefinedValueSet.getName());
myPredefinedValueSet.values.each{
    d3.log.info("value of PredefinedValueSet -> " + it)
}
```

RepositoryField getRepositoryField(String id)

Beschreibung: Mit dieser Funktion wird ein Objekt vom Typ "erweiterte Eigenschaft" geladen.

Parameter:

- id: Id der erw. Eigenschaft, dessen Objekt geladen werden soll

Rückgabewerte:

- Eigenschaftsobjekt

```
RepositoryField repoField = d3.archive.getRepositoryField("196")
d3.log.info("repository field text "+ repoField.text)
d3.log.info("repository field name "+ repoField.name)
d3.log.info("repository field prefered field number "+
repoField.preferedFieldNumber)
d3.log.info("repository field data type "+ repoField.dataType)
```

User getUser(String id)

Beschreibung: Mit dieser Funktion wird ein Objekt vom Typ "Benutzer" geladen.

Parameter:

- id: Id des Benutzers, dessen Objekt geladen werden soll

Rückgabewerte:

- Wertemengenobjekt

```
User myUser = d3.archive.getUser("d3_groovy");
d3.log.info("User name of " + myUser.id + " is " + myUser.getRealName())
```

UserGroup getUserGroup(String id)

Beschreibung: Mit dieser Funktion wird ein Objekt vom Typ "Gruppe" geladen.

Parameter:

- id: Id der Gruppe, dessen Objekt geladen werden soll

Rückgabewerte:

- Gruppenobjekt

```
UserGroup myGroup = d3.archive.getUserGroup("MYTESTGROUP");
d3.log.info("Group name of " + myGroup.id + " is " + myGroup.getName())
```

UserOrUserGroup getUserOrUserGroup(String id)

Beschreibung: Mit dieser Funktion kann ermittelt werden, ob es sich bei dem zu der ID gehörenden Objekt um eine Gruppe oder einen Benutzer handelt.

Parameter:

- id: Id der Gruppe oder Benutzer, dessen Objekt geladen werden soll

Rückgabewerte:

- Gruppen- oder Benutzerobjekt

```
UserOrUserGroup myUserOrGroup = d3.archive.getUserOrUserGroup("d3_groovy")
if(myUserOrGroup.isUserGroup == true){
```

```
    d3.log.info(myUserOrGroup.id + " is a group")
}
else if(myUserOrGroup.isUser == true){
    d3.log.info(myUserOrGroup.id + " is a user")
}
```

AuthorizationProfile getAuthorizationProfile(String id)

Beschreibung: Mit dieser Funktion wird ein Objekt vom Typ "Berechtigungsprofil" geladen.

Parameter:

- id: Id des Berechtigungsprofil, dessen Objekt geladen werden soll

Rückgabewerte:

- Berechtigungsprofilobjekt

```
AuthorizationProfile myAuthorizationProfile =
d3.archive.getAuthorizationProfile("1");
d3.log.info("name of " + myAuthorizationProfile.id + " is " +
myAuthorizationProfile.name)
```

void removeTranslationFromCache(String entryPoint, Locale lang)

Beschreibung: Mit dieser Funktion wird die Übersetzung einer Groovy-Hookwertemenge (neu) geladen.

Parameter:

- entryPoint: Eintrittspunkt unter dem die Wertemenge registriert wurde (nicht Funktionsname der Wertemenge)
- Locale: Sprache, für die die Übersetzung (neu) geladen werden soll

Rückgabewerte:

- -

```
//Lade Übersetzung für Sprache "Deutsch"
d3.getArchive().removeTranslationFromCache("meineGroovyWertemenge", new
Locale("de"));
//Lade Übersetzung für Sprache "Englisch"
d3.getArchive().removeTranslationFromCache("meineGroovyWertemenge", new
Locale("en"));

//Komplettes Beispiel für eine übersetzte Hook-Werktemenge finden Sie hier
```

Document newDocument()

Beschreibung: Mit dieser Funktion wird ein neues Dokument- oder Aktenobjekt erzeugt.

Parameter:

- -

Rückgabewerte:

- Leeres Dokument- oder Aktenobjekt

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

```
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";
```

Document importDocument(Document doc, Path importFilePath)

Beschreibung: Mit dieser Funktion wird ein Dokument mit Datei in d.velop documents angelegt.

Parameter:

- doc: Dokumentobjekt, welches im d.velop documents angelegt werden soll
- Path: Pfad der Datei die zu dem Dokument abgelegt werden. soll

Rückgabewerte:

- Dokumentobjekt, welches angelegt wurde

```
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) {
   d3.log.error(e.message);
   return;
}
d3.log.info("Doc-ID of newly created document: " + newDoc.id);
```

Document importDocument(Document doc)

Beschreibung: Mit dieser Funktion wird eine Akte ohne Datei in d.velop documents angelegt.

Parameter:

- doc: Aktenobjekt, welches im d.velop documents angelegt werden soll

Rückgabewerte:

- Dokumentobjekt, welches angelegt wurde

```
import com.dvelop.d3.server.core.D3Interface
import com.dvelop.d3.server.Document
import com.dvelop.d3.server.exceptions.D3Exception

D3Interface d3 = getProperty("d3");

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

// Add system properties
newDoc.type = "BAKT";                                        // ID of target
document
newDoc.status = Document.DocStatus.DOC_STAT_RELEASE;       // Target state
of document
newDoc.editor = "wfl_admin";                                    // Handler
newDoc.setText(1, "Folder via importDocument");// Comment

// Assign extended properties
newDoc.field[1] = "Attribute value 1 for new document";
newDoc.field[2] = "Attribute value 2 for new document";

try {
    // Import the document / create the folder
    newDoc = d3.archive.importDocument(newDoc);
}
catch (D3Exception e) {
    d3.log.error(e.message);
    return;
}

d3.log.info("Doc-ID of newly created folder: " + newDoc.id);
```

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
```

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         getIsInWorkflow()
    public int             getNumericId()
    public Integer         getLastAlterationNumber()
    public Integer         getAlterationNumberReleased()
    public Integer         getCodepage()
    public String          getCaption()
```

```
    public boolean          getHasMultData()
    public Integer          getFileIdCurrentVersion()
    public Integer          getFileIdRelease()
    public Timestamp        getEndOfRetentionDate()

    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              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(DocStatus 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 DocumentVersion[] getVersions()
    public PhysicalVersion[] getPhysicalVersions()
    public DocumentNote[]    getNotes()

    public SysFields        getSysField()     // Groovy Collection Support
for reading and writing system properties
    public void             addSysField(String fieldName)
    public List             getSysValues()
    public Set              getSysFieldNames()
}
//Siehe auch verschiedene Beispiele
```

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

Field getField()

Beschreibung: Liefert die Werte zu erweiterten Eigenschaften von einem Dokument.

Parameter:

- -

Rückgabewerte:

- Erweiterte Eigenschaften zu einem Dokument

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
def myField = myDoc.getField();d3.log.info("Value of docField 1 is
```

```
"+myField[1]);
d3.log.info("Value of docField 60 Row 1 is "+myField[60][1]);

#Direkter lesender Zugriff auf "Field";
d3.log.info("Value of docField 1 is "+myDoc.field[1]);
d3.log.info("Value of docField 60 Row 1 is "+myDoc.field[60][1]);

#Direkter schreibender Zugriff auf "Field";
myDoc.field[1] = "Value for docField 1";
myDoc.field[60][1] = "Value for docField 1 Row 1";
```

DocumentType getType()

Beschreibung: Liefert das Objekt “Dokumenttyp” zu einem Dokument.

Parameter:

- -

Rückgabewerte:

- Dokumenttyp des Dokuments

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
DocumentType myType = myDoc.getType();
d3.log.info("docTypeShort of document is "+myType.id);
d3.log.info("docTypeLong of document is "+myType.getName());

#Direkter Zugriff auf "DocumentType";
d3.log.info("docTypeShort of document is "+myDoc.type.id);
d3.log.info("docTypeLong of document is "+myType.type.name);
```

void setType(DocumentType type)

Beschreibung: Setzt den Dokumenttyp eines neuen Objekts (Akte oder Dokument). Zum Wechseln der Dokument-/Aktenart wird die Server API Funktion “document_change_type” empfohlen.

Parameter:

- Zu setzender Dokumenttyp

Rückgabewerte:

- -

Beispiel:

```
Document myDoc;
DocumentType myDocumentType = d3.archive.getDocumentType("DDOKU");
myDoc.setType(myDocumentType;)
```

void setType(String typeId)

Beschreibung: Setzt den Dokumenttyp eines neuen Objekts (Akte oder Dokument). Zum Wechseln der Dokument-/Aktenart wird die Server API Funktion “document_change_type” empfohlen.

Parameter:

- Id des Dokumenttyps, der gesetzt werden soll

Rückgabewerte:

- -

Beispiel:

```
Document myDoc;
myDoc.setType("APROJ");
```

String getNumber()

Beschreibung: Liefert die Dokumentnummer (zeich_nr) eines Dokuments oder einer Akte.

Parameter:

- -

Rückgabewerte:

- Dokumentnummer (zeich_nr) eines Dokuments oder einer Akte

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
d3.log.info("document number of document is "+myDoc.getNumber());
```

void setNumber(String docNumber)

Beschreibung: Setzt die Dokumentnummer (zeich_nr) eines neuen Objekts (Akte oder Dokument) oder bei Import eines neuen Objekts im Eintrittspunkt “hook_insert_entry_10”.

Parameter:

- Max 30-stellige Zeichenkette (Buchstaben + Zahlen)

Rückgabewerte:

- -

Beispiel:

```
def myDoc = d3.archive.newDocument();
myDoc.setNumber("EXTERNALNUMBER10000");
```

DocStatus getStatus()

Beschreibung: Liefert den Dokumentstatus eines Douments

Parameter:

- -

Rückgabewerte:

- DOC_STAT_PROCESSING = Bearbeitung;
- DOC_STAT_VERIFICATION = Prüfung;
- DOC_STAT_RELEASE = Freigabe ;
- DOC_STAT_ARCHIVE = Archiv;

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
d3.log.info("document status of document is "+myDoc.getStatus());
d3.log.info("document status of document is "+myDoc.status);
```

void setStatus(DocStatus docStatus)

Beschreibung: Setzt den Dokumentstatus eines neuen Objekts (Akte oder Dokument) oder bei Import/ NewVersion eines neuen Objekts im Eintrittspunkt “hook_insert_entry_10” / “hook_new_version_entry”. Zum Wechseln des Dokumentstatus außerhalb der o. g. Beispiele wird die Server API Funktion “document_transfer” empfohlen.

Parameter:

- Dokumentstatus
  - Bearbeitung = Document.DocStatus.DOC_STAT_PROCESSING;
  - Prüfung = Document.DocStatus.DOC_STAT_VERIFICATION;
  - Freigabe = Document.DocStatus.DOC_STAT_RELEASE;
  - Archiv = Document.DocStatus.DOC_STAT_ARCHIVE;

Rückgabewerte:

- -

Beispiel:

```
def myDoc = d3.archive.newDocument();
myDoc.setStatus(Document.DocStatus.DOC_STAT_RELEASE);
myDoc.status = Document.DocStatus.DOC_STAT_RELEASE;
```

void setStatus(String docStatus)

Beschreibung: Setzt den Dokumentstatus eines neuen Objekts (Akte oder Dokument) oder bei Import/ NewVersion eines neuen Objekts im Eintrittspunkt “hook_insert_entry_10” / “hook_new_version_entry”. Zum Wechseln des Dokumentstatus außerhalb der o. g. Beispiele wird die Server API Funktion “document_transfer” empfohlen.

Parameter:

- Bearbeitung = Bearbeitung
- Prüfung = Prüfung;
- Freigabe = Freigabe;
- Archiv = Archiv;

Rückgabewerte:

- -

Beispiel:

```
def myDoc = d3.archive.newDocument();
myDoc.setStatus("Freigabe");
myDoc.status = "Freigabe";
```

int getVarnumber()

Beschreibung: Liefert die Variantennummer (var_nr) eines Dokuments oder einer Akte

Parameter:

- -

Rückgabewerte:

- Variantennummer (var_nr) eines Dokuments oder einer Akte

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
d3.log.info("document variant number of document is "+myDoc.getVarnumber());
```

void setVarnumber(int varNumber)

Beschreibung: Setzt die Variantennummer (var_nr) eines neuen Objekts (Akte oder Dokument) oder bei Import eines neuen Objekts im Eintrittspunkt “hook_insert_entry_10”.

Parameter:

- Variantennummer die gesetzt werden soll

Rückgabewerte:

-

Beispiel:

```
def myDoc = d3.archive.newDocument();
myDoc.setVarnumber(1);
```

UserOrUserGroup getEditor()

Beschreibung: Liefert den Editor (Benutzer oder Gruppe) eines Dokuments oder einer Akte, sofern sich das Dokument im Status Bearbeitung befindet.

Parameter:

- -

Rückgabewerte:

- Editor des Dokuments (Benutzer oder Gruppe)

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
UserOrGroup myUserOrGroup = myDoc.getEditor();
if(myUserOrGroup == null){
    d3.log.info("document has no editor")
} else if (myUserOrGroup.getIsUser()){
    d3.log.info("editor is a user with id " + myUserOrGroup.id)
} else if (myUserOrGroup.getIsUserGroup()){
    d3.log.info("editor is a group with id " + myUserOrGroup.id)
}
```

void setEditor(UserOrUserGroup editor)

Beschreibung: Setzt den Bearbeiter eines neuen Objekts (Akte oder Dokument) oder bei Import/New-Version eines Objekts im Eintrittspunkt “hook_insert_entry_10” / “hook_new_version_entry”, dies ist nur bei Status Bearbeitung notwendig. Zum Wechseln des Bearbeiters außerhalb der o. g. Beispiele wird die Server API Funktion “document_transfer” empfohlen.

Parameter:

- Benutzer- oder Gruppenobjekt

Rückgabewerte:

•

Beispiel:

```
def myDoc = d3.archive.newDocument();
UserOrUserGroup myUserOrGroup = d3.archive.getUserOrUserGroup("d3_groovy");
myDoc.setEditor(myUserOrGroup);
```

void setEditor(String editor)

Beschreibung: Setzt den Bearbeiter eines neuen Objekts (Akte oder Dokument) oder bei Import/New-Version eines neuen Objekts im Eintrittspunkt “hook_insert_entry_10” / “hook_new_version_entry”. Zum Wechseln des Bearbeiters außerhalb der o. g. Beispiele wird die Server API Funktion “document_transfer” empfohlen.

Parameter:

- Benutzer- oder GruppenId

Rückgabewerte:

•

Beispiel:

```
def myDoc = d3.archive.newDocument();
myDoc.setEditor("d3_groovy");
```

String getOwner()

Beschreibung: Liefert den Besitzer eines Dokuments oder einer Akte. Der Besitzer ist der Benutzer, welcher zuletzt ein Dokument freigegeben hat.

Parameter:

•

Rückgabewerte:

- Besitzer eines Dokuments oder einer Akte

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
d3.log.info("owner of document is "+myDoc.getOwner());
```

String getFilename()

Beschreibung: Liefert den Dateinamen eines Dokuments. Der Dateiname ist in der Regel der ursprüngliche Dateiname, von der Datei die beim Dokumentimport ausgewählt wurde oder per API gesetzt wurde.

Parameter:

- -

Rückgabewerte:

- Dateiname des Dokuments oder Akte

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
d3.log.info("file name of document is "+myDoc.getFilename());
```

String getFileExtension()
Beschreibung: Liefert die Dateierweiterung eines Dokuments von der aktuellen Version.

Parameter:

- -

Rückgabewerte:

- Dateiname es Dokuments

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
d3.log.info("file extension of document is "+myDoc.getFileExtension());
```

Long getDocSize()
Beschreibung: Liefert die Dateigröße in Byte eines Dokuments von der aktuellen Version.

Parameter:

- -

Rückgabewerte:

- Dateigröße in Byte eines Dokuments von der aktuellen Version.

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
d3.log.info("file size in byte of document is "+myDoc.getDocSize());
```

String getText(int lineIdx)
Beschreibung: Liefert den Bemerkungstext eines Dokuments oder einer Akte zu dem angegebenen Index (1-4).

Parameter:

- -

Rückgabewerte:

- Bemerkungstext eines Dokuments oder einer Akte zu dem angegebenen Inedx (1-4).

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
d3.log.info("text 1 of document is "+myDoc.getText(1));
```

void setText(int lineIdx, String text)
Beschreibung: Setzt den Bemerkungstext eines Dokuments oder einer Akte zu dem angegebenen Index (1-4).

Parameter:

- lineIdx = Index zwischen 1 und 4
- text = Zu setzender Text

Rückgabewerte:

- -

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
myDoc.setText(1, "my comment for text 1");
myDoc.updateAttributes("d3_groovy");
```

Timestamp getCreated()
Beschreibung: Liefert das Erstelldatum eines Dokuments oder einer Akte.

Parameter:

-

Rückgabewerte:

- Erstelldatum des Dokuments oder der Akte

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
d3.log.info("create date of document is "+myDoc.getCreated());
```

Timestamp getLastAccess()
Beschreibung: Liefert das Datum des letzten Zugriffs (Vorschau oder Export) auf eine Datei eines Dokuments.

Parameter:

- -

Rückgabewerte:

- Datum des letzten Zugriffs (Vorschau oder Export) auf eine Datei eines Dokuments

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
d3.log.info("last access date of document is "+myDoc.getLastAccess());
```

Timestamp getLastUpdateFile()
Beschreibung: Liefert das Datum des letzten Hochladens einer Datei eines Dokuments, unabhängig vom Status bzw. bzw. Statustransfer.

Parameter:

- -

Rückgabewerte:

- Datum des letzten Hochladens einer Datei eines Dokuments

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
d3.log.info("last file update of document was "+myDoc.getLastUpdateFile());
```

Timestamp getLastUpdateAttribute()

Beschreibung: Liefert das Datum der letzten Aktualisierung der Eigenschaften eines Dokuments oder einer Akte.

Parameter:

- -

Rückgabewerte:

- Datum der letzten Aktualisierung der Eigenschaften eines Dokuments oder einer Akte.

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
d3.log.info("last update of metadata of document was
"+myDoc.getLastUpdateAttribute());
```

Timestamp getLastUpdate()

Beschreibung: Liefert das Datum der letzten Änderung eines Dokuments oder einer Akte (Eigenschaften aktualisieren, Hochladen von Dateien, Statustransfer).

Parameter:

- -

Rückgabewerte:

- Datum der letzten Änderung eines Dokuments oder einer Akte

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
d3.log.info("last action date of document is "+myDoc.getLastUpdate());
```

Integer getColorCode()

Beschreibung: Liefert den Code der Farbe eines Dokuments oder einer Akte

Parameter:

- -

Rückgabewerte:

- 0 = Nicht gesetzt
- 1 - 24 = Farbcode

Farben von links oben = 1, nach rechts unten = 24

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
d3.log.info("the color code of document is "+myDoc.getColorCode());
```

void setColorCode(Integer colorCode)

Beschreibung: Setzt den Code der Farbe eines Dokuments oder einer Akte

Parameter:

- 0 = Keine Farbe
- 1 - 24 = Farbcode

Farben von links oben = 1, nach rechts unten = 24

Rückgabewerte:

- -

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
myDoc.setColorCode(2);
myDoc.updateAttributes("d3_groovy");
```

String getReleaseVersionStatus()

Beschreibung: Liefert zu einem Dokument oder einer Akte, ob der Status “Freigabe Gesperrt” ist.

Parameter:

- -

Rückgabewerte:

- f = Freigabe
- g = Gesperrt

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
d3.log.info("release version status of document is
"+myDoc.getReleaseVersionStatus());
```

boolean getIsVerified()

Beschreibung: Liefert zu einem Dokument oder einer Akte, ob der Status “Prüfung Geprüft” ist.

Parameter:

- -

Rückgabewerte:

- False = Ungeprüft
- True= Geprüft

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
if(myDoc.getIsVerified() == true){
   d3.log.info("document is verified");
} else {
   d3.log.info("document is not verified");
}
```

boolean getIsInWorkflow()

Beschreibung: Liefert zu einem Dokument oder einer Akte, ob sich das Objekt in einem aktiven Workflow befindet.

Parameter:

- -

Rückgabewerte:

- False = Befindet sich momentan nicht in einem Workflow
- True= Befindet sich momentan in einem Workflow

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
if(myDoc.getIsInWorkflow() == true){
   d3.log.info("document is in a workflow");
} else {
   d3.log.info("document is not in a workflow");
}
```

int getNumericId()

Beschreibung: Liefert zu einem Dokument oder einer Akte den Zähler zu der DokumentId.

Parameter:

- -

Rückgabewerte:

- Zähler zu der DokumentId eines Dokuments oder einer Akte

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
d3.log.info("numeric id of document is "+myDoc.getNumericId());
```

Integer getLastAlterationNumber()

Beschreibung: Liefert die letzte Änderungsnummer von Versionen zu einem Dokument. Wird z. B. ein Dokument in den Status Bearbeitung übernommen und ohne Änderung in den Status Freigegeben überführt, wird diese Zahl hochgezählt auch wenn keine tatsächliche Änderung der Datei vorgenommen wurde.

Parameter:

- -

Rückgabewerte:

- Letzte Änderungsnummer

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
d3.log.info("last alteration number of document is
"+myDoc.getLastAlterationNumber());
```

Integer getAlterationNumberReleased()

Beschreibung: Liefert die letzte Änderungsnummer von Versionen zu einem Dokument vom Status Freigabe.

Parameter:

- -

Rückgabewerte:

- Letzte Änderungsnummer im Status Freigabe

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
d3.log.info("last alteration number released of document is
"+myDoc.getAlterationNumberReleased());
```

Integer getCodepage()

Beschreibung: Liefert die in der Datenbank zu dem Dokument oder der Akte gespeicherten Codepage der Eigenschaften. Hinweis: Dieser Wert ist nur befüllt, wenn nicht die Standardcodepage verwendet wurde

Parameter:

- -

Rückgabewerte:

- Codepage der Eigenschaften zu einer Akte oder Dokument

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
if(myDoc.getCodepage() == 0){
    d3.log.info("codepage of document is the default codepage")
} else {
    d3.log.info("codepage of document is "+myDoc.getCodepage())
}
```

String getCaption()

Beschreibung: Liefert zu einem Dokument oder einer Akte den Titel bzw. die Caption.

Parameter:

- -

Rückgabewerte:

- Titel bzw. Caption zu einem Dokument oder zu einer Akte

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
d3.log.info("document xxx of document is "+myDoc.getCaption());
```

boolean getHasMultData()

Beschreibung: Liefert, ob zu einem Dokument oder einer Akte mindestens eine Zeile eines Mehrfachfelds befüllt ist.

Parameter:

- -

Rückgabewerte:

- true = Mindestens eine Zeile eines Mehrfachfelds ist befüllt
- false = Keine Zeile eines Mehrfachfelds ist befüllt

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
if(myDoc.getHasMultData() == true){
    d3.log.info("document has no multi metadata given")
} else {
    d3.log.info("document has multi metadata given")
}
```

Integer getFileIdCurrentVersion()

Beschreibung: Liefert die Id der Datei in der aktuellen Version.

Parameter:

- -

Rückgabewerte:

- Id der Datei in der aktuellen Version.

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
d3.log.info("file id of document is "+myDoc.getFileIdCurrentVersion());
```

Integer getFileIdRelease()

Beschreibung: Liefert die Id der Datei in der Freigabe Version.

Parameter:

- -

Rückgabewerte:

- Id der Datei in der Freigabe Version.

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
d3.log.info("file id in state release of document is
"+myDoc.getFileIdRelease());
```

Timestamp getEndOfRetentionDate()

Beschreibung: Liefert das Datum zum Ende der Aufbewahrungsdauert des Dokuments oder der Akte.

Parameter:

- -

Rückgabewerte:

- Datum zum Ende der Aufbewahrungsdauert des Dokuments oder der Akte.

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
d3.log.info("retention date of document is "+myDoc.getEndOfRetentionDate());
```

void updateAttributes(String userId)

Beschreibung: Aktualisiert Eigenschaften oder Systemeigenschaften einges Dokuments oder einer Akte

Parameter:

- userId = Id des Benutzer, welcher die Eigenschaften aktualisieren soll

Rückgabewerte:

- -

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
myDoc.field[1] = "Value for field 1"
myDoc.docMultField(62, 1, "")
try{
   def retVal = myDoc.updateAttributes("d3_groovy")
   writeLog("Info", "Update successful for " + myDoc.id)
} catch (D3Exception e){
   d3.log.error("Update failed for " + myDoc.id + " with error message: " +
e.getMessage())
}
```

void updateAttributes(String userId, boolean noHooks)

Beschreibung: Aktualisiert Eigenschaften oder Systemeigenschaften eines Dokuments oder einer Akte

Parameter:

- userId = Id des ändernden Benutzers
- noHooks = true → Eintrittspunkte werden nicht ausgeführt, false → Eintrittspunkte werden ausgeführt (default). Diese Parameter hat keinen Einfluss auf die Bildung des Aktenplans

Rückgabewerte:

- -

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
myDoc.field[1] = "Value for field 1"
myDoc.docMultField(62, 1, "")
try{
   def retVal = myDoc.updateAttributes("d3_groovy", true)
   writeLog("Info", "Update successful for " + myDoc.id)
} catch (D3Exception e){
   d3.log.error("Update failed for " + myDoc.id + " with error message: " +
e.getMessage())
}
```

id changeType(String docTypeId, String userId)

Beschreibung: Ändert die Kategorie einer Akte oder eines Dokuments

Parameter:

- docTypeId = ID des Dokuments, dessen Dokumenttyp geändert werden soll
- userId = Id des ändernden Benutzers

Rückgabewerte:

- -

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
myDoc.changeType("DTEST", "d3_groovy")
```

String getPermission(userId)

Beschreibung: Ermittelt die Berechtigungen eines Benutzers auf ein Dokument

Parameter:

- Id des Benutzers dessen Berechtigungen abgegfragt werden sollen

Rückgabewerte:

- a1a2a3a4a5a6a7a8-b1b2b3b4b5b6b7b8-c1c2-d1d2

Parameter	Beschreibung
a1	Zugriff auf Dokumenttyp erlaubt
a2	Notizdatei vorhanden
a3	Dokumentenverknüpfungen vorhanden
a4	Benutzerdatei vorhanden
a5	Benutzerdatei verschlüsselt
a6	Freigegebene Version vorhanden (1: freigegeben, 2: gesperrt)
a7	Letzte Version in Prüfung (1: geprüft, 2: ungeprüft)
a8	Mindestens eine Version im Status Archiv
b1	Berechtigung zur Darstellung der Nutzdaten
b2	Berechtigung zum Ändern der Identifikationsdaten
b3	Berechtigung zum Ändern der Benutzerdaten
b4	Berechtigung zum Löschen des Dokuments
b5	Berechtigung zum Status Transfer
b6	Berechtigung zum Prüfen des Dokuments
b7	Berechtigung zum Freigeben/Sperren des Dokuments
b8	Berechtigung zum Durchführen beider Prüfschritte (Prüfung und Freigabe) in einem Schritt

c1	Berechtigung zur Darstellung einer älteren Version in Freigabe
c2	Berechtigung zum Freigeben/Sperren einer älteren Version in Freigabe
d1	Berechtigung zum Einsehen von Versionen im Status Archiv
d2	Dokument ist im Workflow

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
```

int block()

Beschreibung: Sperren oder entsperren eines Dokuments

Parameter:

- block: true = sperren, false = entsperren
- userId: Id des ausführenden Benutzers

Rückgabewerte:

- -

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
int returnValue = myDoc.block(false, "d3_groovy")
if(returnValue == 0){
   d3.log.info("success")
} else {
   //exception handling
}
```

int verify()

Beschreibung: Setzen von Status Prüfung geprüft

Parameter:

- userId: Id des ausführenden Benutzers

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen
- **-3**: unbekannter Fehler, kontaktieren Sie den d.velop-Support
- **-201**: ungültiger Benutzername angegeben
- **-301**: Benutzer hat keine Berechtigungen
- **-302**: Fehler bei der Ermittlung der Benutzerberechtigungen
- **-303**: Benutzer hat keine Zugriffsberechtigungen für das Dokument
- **-307**: Benutzer hat keine Berechtigungen zum Verifizieren dieses Dokuments
- **-509**: Version im Status **Verifizierung** ist bereits verifiziert
- **-510**: Es gibt keine Version im Status **Verifizierung**.
- **< -9500**: Fehler in kundenspezifischer Hook-Funktion, für Details siehe d.velop logviewer

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
int returnValue = myDoc.verify("d3_groovy")
if(returnValue == 0){
   d3.log.info("success")
} else {
   d3.log.error("error")
   //exception handling
}
```

int transfer(String destination, String newEditor, String changeRemark, boolean asynchronous, int archivIndex, String userId)

Beschreibung:

Parameter:

- destination : Zielstatus Bearbeitung, Prüfung, Freigabe, Archiv
- newEditor: Benutzer-/Gruppenname erforderlich für Ziel = Bearbeitung (Benutzer oder Gruppe); Ziel = Prüfung (Gruppe). Pflichtfeld, wenn beim Parameter destination der Zielstatus Bearbeitung angegeben wurde.
- changeRemark: Muss angegeben werden, wenn ein Dokument aus der Bearbeitung in die Prüfung übernommen wird und das Dokument gegenüber einer früheren Version im Status Bearbeitung oder Archiv geändert wurde.
- asynchronous: Gibt an, ob der Transfer sofort oder asynchron über einen asynchronen Job für d.3 async ausgeführt werden soll (false: sofort (Standard); true: asynchron durch d.3 async).
- archivIndex : Index für die wiederherzustellende Version im Status Archiv
- userId: Benutzer, der die Übertragung durchführen soll. Wenn nichts angegeben wird, wird der Benutzer aus dem Kontext verwendet.

Rückgabewerte:

- -

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
int returnValue = myDoc.transfer("Bearbeitung", "d3_groovy", "", false, 0,
"d3_groovy")
if(returnValue == 0){
   d3.log.info("success")
} else {
   d3.log.error("error")
   //exception handling
}
```

int addDependent(String filename, String docStatus, String docExt, String userId)

Beschreibung:

Parameter:

- filename
- docStatus
- docExt
- userId

Rückgabewerte:

- -

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
int returnValue = myDoc.addDependent("C:/Temp/myFileOnServer.pdf",
"Freigabe", "p1", "d3_groovy")
if(returnValue == 0){
   d3.log.info("success")
} else {
   d3.log.error("error")
   //exception handling
}
```

int addDependent(String filename, DocStatus docStatus, String docExt, String userId)

Beschreibung:

Parameter:

- filename
- docStatus
- docExt
- userId

Rückgabewerte:

- -

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
int returnValue = myDoc.addDependent("C:/Temp/myFileOnServer.pdf",
Document.DocStatus.DOC_STAT_RELEASE, "p1", "d3_groovy")
if(returnValue == 0){
   d3.log.info("success")
} else {
   d3.log.error("error")
   //exception handling
}
```

int deleteDependent (char docStatus, String docExt, String userId)

Beschreibung: Löscht die abhängige Datei im angegebenen Status

Parameter:

- docStatus: Dokumentstatus der zu löschenden abhängigen Datei
  - **B** für **Bearbeitung**
  - **P** für **Prüfung**
  - **F** für **Freigabe**
- docExt: Endung der zu löschenden abhängigen Datei
- userId: Id des ausführenden Benutzers

Rückgabewerte:

- -

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
int returnValue = myDoc.deleteDependent("F", "p1", "d3_groovy")
```

```
if(returnValue == 0){
   d3.log.info("success")
} else {
   d3.log.error("error")
   //exception handling
}
```

int deleteDependent (DocStatus docStatus, String docExt, String userId)

Beschreibung: Löscht die abhängige Datei im angegebenen Status

Parameter:

- docStatus: Dokumentstatus der zu löschenden abhängigen Datei
- docExt: Endung der zu löschnenden abhängigen Datei
- userId: Id des ausführenden Benutzers

Rückgabewerte:

- -

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
int returnValue =
myDoc.deleteDependent(Document.DocStatus.DOC_STAT_RELEASE, "p1",
"d3_groovy")
if(returnValue == 0){
   d3.log.info("success")
} else {
   d3.log.error("error")
   //exception handling
}
```

int startLifetime (boolean overwriteOldDate, int lifeTimeDays)

Beschreibung: Legt die Lebensdauer eines Dokuments oder einer Akte fest. Hinweis: Bei entfernen oder verkürzen der Lebensdauer hat dies ggf. keinen Einfluss auf den Sekundärspeicher.

Parameter:

- overwriteOldDate: true = vorhandener Wert soll überschrieben werden, false = vorhandener Wert soll nicht überschrieben werden
- lifeTimeDays: Wenn die für den Dokumententyp konfigurierte Lebensdauer nicht verwendet werden soll, können Sie die Anzahl der Tage angeben (Standard:**0**- konfigurierte Lebensdauer). Wird **-1** angegeben, wird die Lebensdauer auf “unbegrenzt” gesetzt.

Rückgabewerte:

- -

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
int returnValue = myDoc.startLifetime(true, 365)
if(returnValue == 0){
   d3.log.info("success")
} else {
   d3.log.error("error")
```

```
   //exception handling
}
```

int setCacheDays (char docStatus, int archiveIndex, int daysInCache)

Beschreibung: Die Funktion legt die Anzahl der Tage für eine Dokumentversion fest, die die Dokumentversion im d.3-Dokumentenbaum "ab heute" verbleiben soll. Die Anzahl von Tagen ist nur relevant, wenn das Dokument auf einen Sekundärspeicher gespeichert wurde oder wird.

Parameter:

- docStatus: Dokumentstatus des Dokuments
- archiveIndex: Index der Version, nur relevant für Status Archiv
- daysInCache: Anzahl der Tage, die die Dokumentversion im Cache verbleiben soll

Rückgabewerte:

- -

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
int returnValue = myDoc.setCacheDays("F", 0, 30)
if(returnValue == 0){
   d3.log.info("success")
} else {
   d3.log.error("error")
   //exception handling
}
```

int checkFolderScheme (String userId);

Beschreibung: Stößt die Aktenplanregeln für die Akte oder das Dokument an

Parameter:

- userid: Id des ausführenden Benutzers

Rückgabewerte:

- -

Beispiel:

```
Document myDoc = d3.archive.getDocument("P000000001", "d3_groovy");
int returnValue = myDoc.checkFolderScheme("d3_groovy")
if(returnValue == 0){
   d3.log.info("success")
} else {
   d3.log.error("error")
   //exception handling
}
```

Beispiele

Beispiel - Setzen von erweiterten Eigenschaften

```
Document doc
doc.field[1] = "Value for field 1"                    // Writing value
d3.log.info(doc.field[1])                             // Reading value
```

```
// Referencing via the title of the property
 doc.field["Name"] = "Meier"                          // Writing value
println "Name = " + doc.field["Name"]                 // Reading value

doc.docMultField(62, 1, "value for multifield in first line") // Writing
value to a multifield

println(doc.docMultFieldAsString(62, 1);); // Reading the first line
println(doc.field[62]) // Reading the value, only the first line is returned
```

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

Wenn Sie eine Eigenschaft (Attribut) ändern möchten, ist dies über die Methode updateAttributes möglich. Diese Methode wird benötigt, wenn der Wert nicht direkt zugewiesen werden. In "entry"-Hooks, in denen die Werte noch nicht an die Datenbank übergeben wurden, können die Werte direkt zugewiesen werden. Dies erkennen Sie im obigen Beispiel. Wenn Sie in einem Exithook (bspw. in hook_insert_exit_30 nach dem Import) eine Eigenschaftsänderung durchführen möchten, müssen Sie neben der Zuweisung danach die Methode updateAttributes ausführen. Die Methode gibt es in zwei Ausprägungen. Sie können den ausführenden Benutzer übergeben, um zu protokollieren, durch wen die Attributänderung erfolgt ist. Eine Rechteüberprüfung findet jedoch nicht statt. Diese Prüfung können Sie über die Methode getPermission(String userId) realisieren. Zusätzlich kann ein Boolean übergeben werden, um anzugeben, ob ein Updatehook ausgeführt werden soll. Diese Angabe ist bspw. nützlich, wenn Sie im hook_attrib_upd_exit_20-Hook eine Eigenschaftsänderung durchführen und nicht möchten, dass der Updatehook nochmals ausgeführt wird.

Beispiel - Aktualisieren von Eigenschaften

```
doc.field[1] = "Value for field 1"
doc.docMultField(62, 1, "")

doc.updateAttributes("User", true)
```

Der Wert "true" bewirkt, dass kein Update-Hook nach der Ausführung gestartet wird.

Systemeigenschaften

Nutzung der Schnittstelle für die Systemeigenschaften. Bei Systemeigenschaften handelt es sich nicht um einfache Eigenschaftsfelder, die einer Dokumentart oder einer Akte zugeordnet sind. Sie wird also nicht in den Standardclientanwendungen von d.velop angezeigt. Es handelt sich um ein selbst definiertes, technisches Feld, welches man zum Speichern von Informationen nutzen kann. Die Felder werden in der Tabelle doc_sys_fields und die zugehörigen in der Tabelle doc_sys_values verwaltet und gespeichert.

Beispiel - Setzen von Systemeigenschaften

```
println doc.sysField["InvoiceNo"][1]                              // Reading
value
doc.sysField["InvoiceNo"][1] = "Wert für Systemeigenschaft"       // 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
```

Damit die Werte auch tatsächlich gespeichert werden, muss noch die Methode updateAttributes ausgeführt werden.

Beispiel - Aktualisieren von Systemeigenschaften

```
D3Interface d3 = getProperty("d3");

String sDocIDToManip = "P000000001";

try {
   d3.log.info("hinzufuegen SysField: " + sDocIDToManip)

   Document doc = null;
   doc = d3.archive.getDocument(sDocIDToManip, "Administr2");

   doc.addSysField("myField")

   doc.sysField["myField"][1] = "Wert fuer Systemeigenschaft"
   doc.sysField["myField"].add("Ein Wert für myField")
   doc.updateAttributes("Administr2")

   d3.log.info(doc.getSysField())
   d3.log.error("doc.sysField[myField].sysValues =" +
doc.sysField["myField"].sysValues)

   d3.log.info("Sysfield erfolgreich gesetzt")

} catch (Exception ex){
          d3.log.error("Fehler beim setzen des Sysfields: " +
ex.getMessage());
}
```

Beispiel - Importieren von Dokumenten

```
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) {
   d3.log.error(e.message);
   return;
}
d3.log.info("Doc-ID of newly created document: " + newDoc.id);
```

Beispiel - Importieren von Akten

```
import com.dvelop.d3.server.core.D3Interface
import com.dvelop.d3.server.Document
import com.dvelop.d3.server.exceptions.D3Exception

D3Interface d3 = getProperty("d3");

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

// Add system properties
newDoc.type = "BAKT";                                       // ID of target
document
newDoc.status = Document.DocStatus.DOC_STAT_RELEASE;       // Target state
of document
newDoc.editor = "wfl_admin";                                  // Handler
newDoc.setText(1, "Folder via importDocument");// Comment

// Assign extended properties
newDoc.field[1] = "Attribute value 1 for new document";
newDoc.field[2] = "Attribute value 2 for new document";

try {
    // Import the document / create the folder
    newDoc = d3.archive.importDocument(newDoc);
}
catch (D3Exception e) {
    d3.log.error(e.message);
    return;
}

d3.log.info("Doc-ID of newly created folder: " + newDoc.id);
```

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

// Get document object
Document myDoc = d3.archive.getDocument("LV00004228", Const.userHook)
// Go / interate through all versions
```

```
for (version in myDoc.versions)
{
    // Get the properties of the current Version
    d3.log.info("- version id " + version.id)
    d3.log.info("- category of version is " +
version.getCategory().toString())

    if(version.getCategory() == DocumentVersion.Category.DOC_VERS_REGULAR){
        if(version.isCurrentVersion() == true){
            d3.log.info("- this version is the current version")
        }
        if(version.hasStatus() == true){
            d3.log.info("- status of version is " + version.getStatus())
        }
        if(version.hasHistStatus() == true){
            d3.log.info("- history status of version is " +
version.getHistStatus())
        }
        if(version.hasFileId() == true){
            d3.log.info("- file id of version is " + version.getFileId())
        }
        if(version.hasHistFileId() == true){
            d3.log.info("- history file id of version is " +
version.getHistFileId())
        }

        d3.log.info("-- change reason of this version is " +
version.getChangeReason())
        d3.log.info("-- delete reason of this version is " +
version.getDeleteReason())
        d3.log.info("-- creator of this version is " + version.getCreator())
        d3.log.info("-- create date of this version is " +
version.getCreateDate())
        d3.log.info("-- release user of this version is " +
version.getReleaseUser())
        d3.log.info("-- release date of this version is " +
version.getReleaseDate())
        d3.log.info("-- archive user of this version is " +
version.getArchiveUser())
        d3.log.info("-- archive date of this version is " +
version.getArchiveDate())
        d3.log.info("-- verify user of this version is " +
version.getVerifier())
        d3.log.info("-- verify date of this version is " +
version.getVerifyDate())
        d3.log.info("-- delete user of this version is " +
version.getDeleter())
        d3.log.info("-- delete date of this version is " +
version.getDeleteDate())
        d3.log.info("-- external version id of this version is " +
version.getExternalVersionId())

        // When the current Version has a physical file attached, get access
        if (version.physicalVersion)
        {
```

```
            // Get the properties of the file
            d3.log.info("--- file id of this version is " +
version.physicalVersion.getFileId())
            d3.log.info("--- doc id of this version is " +
version.physicalVersion.getDocId())
            d3.log.info("--- status of this version is " +
version.physicalVersion.getStatus())
            d3.log.info("--- file extension of this version is " +
version.physicalVersion.getFileExtension())
            d3.log.info("--- file format of this version is " +
version.physicalVersion.getFileFormat())
            d3.log.info("--- file size of this version is " +
version.physicalVersion.getFileSize())
            d3.log.info("--- file localisation of this version is " +
version.physicalVersion.getFileLocalisation())
            d3.log.info("--- d3 hash of this version is " +
version.physicalVersion.getD3Hash())
            d3.log.info("--- file hash of this version is " +
version.physicalVersion.getFileHash())

            // Get all properties of depending files
            for (dependentFile in version.physicalVersion.dependentFiles)
            {
                // Get the properties of each depending file
                d3.log.info("---- file extesion of dependend file of this
version is " + dependentFile.getExtension())
                d3.log.info("---- doc id of dependend file of this version
is " + dependentFile.getDocId())
                d3.log.info("---- file id of dependend file of this version
is " + dependentFile.getFileId())
                d3.log.info("---- file format of dependend file of this
version is " + dependentFile.getFileFormat())
                d3.log.info("---- file localisation of dependend file of
this version is " + dependentFile.getFileLocalisation())
                d3.log.info("---- file size in byte of dependend file of
this version is " + dependentFile.getFileSize())
                d3.log.info("---- file size in kilobyte of dependend file
of this version is " + dependentFile.getFileSizeInKb())
                d3.log.info("---- d3 hash of dependend file of this version
is " + dependentFile.getD3Hash())
                d3.log.info("---- create date of dependend file of this
version is " + dependentFile.getProcDate())
                d3.log.info("---- export flag of dependend file of this
version is " + dependentFile.getFlagStorageExport())
                d3.log.info("---- export date of dependend file of this
version is " + dependentFile.getStorageExportDate())
                d3.log.info("---- file hash of dependend file of this
version is " + dependentFile.getD3FileHash())
                // ..
            }
        }
    } else if(version.getCategory() ==
DocumentVersion.Category.DOC_VERS_BLOCKED){
        d3.log.info("block user of this version is " +
version.getBlockUser())
```

```
        d3.log.info("block date of this version is " +
version.getBlockDate())
    } else if(version.getCategory() ==
DocumentVersion.Category.DOC_VERS_DELETED){
        d3.log.info("block user of this version is " +
version.getBlockUser())
        d3.log.info("block date of this version is " +
version.getBlockDate())
    } else {
        d3.log.info(version.getCategory())
    }
}
```

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()
}
```

Beispiel siehe Dokumentversionen (DocumentVersion)

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()
 }
```

Beispiel siehe Dokumentversionen (DocumentVersion)

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()
}
```

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()
}
```

Beispiel

```
D3Interface d3 = getProperty("d3");

try {
    //Retrieve document object
    Document doc = d3.archive.getDocument("P000000001", "d3_groovy");
    //adding sysField "myField" to document
    doc.addSysField("myField")
    //adding / changing value for sysField "myField" at index position 1
    doc.sysField["myField"][1] = "value for sysField"
    //adding value for sysField "myField" at new free index position
```

```
    doc.sysField["myField"].add("another value for myField")
    //update document object
    doc.updateAttributes("d3_groovy")

    //Listing all SysValues asssined to document object
    doc.sysValues.eachWithIndex(){ it, row ->
        d3.log.info("doc id of sysField = " + it.getDocId() + " in row " +
row)
        d3.log.info("id of sysField = " + it.getFieldId() + " in row " +
row)
        d3.log.info("name of sysField = " + it.getFieldName() + " in row "
+ row)
        d3.log.info("idx of sysField = " + it.getFieldIndex() + " in row "
+ row)
        d3.log.info("value of sysField = " + it.getFieldValue() + " in row
" + row)
        d3.log.info("just print all values for sysField = " + it.toString()
+ " in row " + row)

    }
    d3.log.info("just print all values for document = " +
doc.sysValues.toString())

} catch (Exception ex){
    d3.log.error("error updating document object: " + ex.getMessage());
}
```

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
}
```

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)
}
```

Beispiel für den Zugriff auf die Eigenschaften einer Dokumentart.

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

DocumentType myDocumentType = d3.archive.getDocumentType("DDOKU");
d3.log.info("default name of document type is
"+myDocumentType.getDefaultName());
d3.log.info("name of document type is "+myDocumentType.getName());
d3.log.info("type of document type is "+myDocumentType.getType());
if(myDocumentType.getType() == "d"){
    d3.log.info("type of this document type is 'document'");
} else if(myDocumentType.getType() == "a"){
    d3.log.info("type of document type is 'folder'");
}
if(myDocumentType.getIsFolder() == true){
    d3.log.info("document type is a folder");
} else {
    d3.log.info("document type is a document");
}
if(myDocumentType.getIsExportedToStorage() == true){
    d3.log.info("document type is configured to be exported to a sec.
storage");
}
if(myDocumentType.getIsSystemDocType() == true){
    d3.log.info("document type is a system document type");
}
if(myDocumentType.getIsDummyType() == true){
    d3.log.info("document type is a dummy document type");
}
d3.log.info("archive id document type is "+myDocumentType.getArchiveId());
d3.log.info("d.3 search corpus of document type is
"+myDocumentType.getDsearchCorpus());
```

```
d3.log.info("field number of the property 'Dokumentname' of document type
is "+myDocumentType.getFieldNoByName("Dokumentname"));

// Go / interate through all fields
for (int i = 1; i <=89; i++) {
    try {
        RepositoryField repoField =
myDocumentType.field[i].getRepositoryField()

        d3.log.info("field infos for number "+ i)
        d3.log.info("repository text "+ repoField.text)
        d3.log.info("repository name "+ repoField.name)
        d3.log.info("repository prefered field number "+
repoField.preferedFieldNumber)
        d3.log.info("repository data type "+ repoField.dataType)
        d3.log.info("repository has a plausibility hook hunction assigned
"+ repoField.hasPlausibilityHookFunction())
        d3.log.info("repository has a values providing hook function
assigned "+ repoField.hasValuesProvidingHookFunction())
        d3.log.info("repository has a predifined value list assigned "+
repoField.hasPredefinedValues())

        if(myDocumentType.field[i].getIsMandatory() == true){
            d3.log.info("field is mandatory")
        } else {
            d3.log.info("field is not mandatory")
        }

        if(myDocumentType.field[i].getIsModifiable() == true){
            d3.log.info("field is modifiable")
        } else {
            d3.log.info("field is not modifiable")
        }

        if(myDocumentType.field[i].getIsHidden() == true){
            d3.log.info("field is hidden")
        } else {
            d3.log.info("field is not hidden")
        }

        if(myDocumentType.field[i].getViewInSearchMask() == true){
            d3.log.info("field is visibile in search mask")
        } else {
            d3.log.info("field is not visibile in search mask")
        }

        if(myDocumentType.field[i].getViewInImportMask() == true){
            d3.log.info("field is visibile in import mask")
        } else {
            d3.log.info("field is not visibile in import mask")
        }

        if(myDocumentType.field[i].getViewInResultSet() == true){
            d3.log.info("field is visibile in result set")
        } else {
```

```
            d3.log.info("field is not visibile in result set")
        }
    } catch (Exception ex){
        d3.log.info("field with number ${i} does not exist")
    }
}
```

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()
}
```

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

```
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
}

// Check if user is member of a specific group
if(user.isMemberOfGroup("myGroup") == true){
   d3.log.info(user.id+" is a member of myGroup");
}

// 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
}
```

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
}
```

Wertemengen (PredefinedValueSet)

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

Beispiel zur Erstellung einer Wertemenge per Hookfunktion Wertemengen

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)
}
```

Beispiel zur Erstellung einer Wertemenge per Hookfunktion Wertemengen

Berechtigungsprofil (AuthorizationProfile)

AuthorizationProfile

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

1.9.2. d.3 SQL Datenbank (SqlD3Interface)

Die d.3-SQL-Schnittstelle bietet einen einfachen, groovy-konformen Zugriff auf die native Datenbankschnittstelle des d.3-Servers.

Es ist kein JDBC Treiber erforderlich.

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;
    }
```

Modifizierer und Typ	Methode und Beschreibung	Anwendungsfall

int	**execute(String query, List<Object> params)**
	
	Ausführen eines SQL-Kommandos mit Bind-Variablen (z.B. ein **insert** oder **update** Kommando)
	
	Die **execute()**-Methode liefert nach erfolgreicher Ausführung die Anzahl der betroffenen Zeilen zurück.
		List<String> queryParam = []
		
		//SQL-Statement
		
		String **sqlStamm = "update custom_customer set customerType = ? where customerType = ?"**
		
		//Setze Wert für 1. Bind-Variable (**customerType** = Neuer Wert)
		
		queryParam.add("A")
		
		//Setze Wert für 2. Bind-Variable **(customerType** = Alter Wert)
		
		queryParam.add("C")
		
		//Führe SQL-Statement aus
		
		int sqlResult = d3.sql.execute(sqlStamm, queryParam)
		
		//Die Anzahl der betroffenen Zeilen steht im Rückgabewert
		
		d3.log.info**(sqlResult + " betroffene Zeilen")**
		
		Hinweis: Nur für Insert und Update-Statements sinnvoll.

Modifizierer und Typ	Methode und Beschreibung	Anwendungsfall

int	**execute(String query)**
	
	Ausführen eines SQL-Kommandos ohne Bind-Variablen (z.B. ein **insert** oder **update** Kommando)
	
	Die **execute()**-Methode liefert nach erfolgreicher Ausführung die Anzahl der betroffenen Zeilen zurück.
		//SQL-Statement
		
		String **sqlQuery = "update custom_customer set customerType = 'A' where customerType = 'C'"**
		
		//Führe SQL-Statement aus
		
		int sqlResult = d3.sql.execute(sqlQuery)
		
		//Die Anzahl der betroffenen Zeilen steht im Rückgabewert
		
		d3.log.info(sqlResult + " betroffene Zeilen")
		
		Hinweis: Nur für Insert und Update-Statements sinnvoll.
		
		Hinweis: Aus Performance- und Sicherheitsgründen wird die Verwendung von Bind-Variablen empfohlen.

Groovy-RowResult	**firstRow(String sqlquery, List<Object> params)**
	
	Ausführen eines SQL-SELECT-Kommandos mit Bind-Variablen.
	
	Nur die erste Zeile der Ergebnismenge wird abgerufen und zurückgeliefert.
		List<String> bindParam = []
		
		//SQL-Statement
		
		String **sqlQuery = "select customerName from custom_customer where customerNo = ?"**
		
		//Setze Wert für 1. Bind-Variable **(customerNo** = Wert)
		
		bindParam.add(10000)
		
		//Führe SQL-Statement aus
		
		GroovyRowResult resultFirstRow = d3.sql.firstRow(sqlQuery, bindParam)
		
		if(resultFirstRow.size() == 1){
		
		//Es wurde (mindestens) ein Eintrag gefunden
		
		d3.log.info("Name = "+resultFirstRow.customerName)
		
		} else {
		
		//Es wurde kein Eintrag gefunden
		
		d3.log.info("Zu customerNo = 10000 gibt es keinen Eintrag")
		
		}
		
		Hinweis: Nur sinnvoll wenn sichergestellt ist, dass nur ein Ergebnis zurückgeliefert/benötigt wird. Auch wenn das SQL-Statement mehrere Ergebnisse zurückliefert, ist die Anzahl der Zeilen (**resultFirstRow.size() = 1**).

Modifizierer und Typ	Methode und Beschreibung	Anwendungsfall

Groovy-RowResult	**firstRow(String query)**
	
	Ausführen eines SQL-SELECT-Kommandos ohne Bind-Variablen.
	
	Nur die erste Zeile der Ergebnismenge wird abgerufen und zurückgeliefert.
		//SQL-Statement
		
		String **sqlQuery = "select customerName from custom_customer where customerNo = 10000"**
		
		//Führe SQL-Statement aus
		
		GroovyRowResult resultFirstRow = d3.sql.firstRow(sqlQuery)
		
		if(resultFirstRow.size() == 1){
		
		//Es wurde (mindestens) ein Eintrag gefunden
		
		d3.log.info**("Name = "+resultFirstRow.customerName)**
		
		} else {
		
		//Es wurde kein Eintrag gefunden
		
		d3.log.info**("Zu customerNo = 10000 gibt es keinen Eintrag")**
		
		}
		
		Hinweis: Nur sinnvoll wenn sichergestellt ist, dass nur ein Ergebnis zurückgeliefert oder benötigt wird. Auch wenn das SQL-Statement mehrere Ergebnisse zurückliefert, ist die Anzahl der Zeilen **(resultFirstRow.size() = 1**).
		
		Hinweis: Aus Performance- und Sicherheitsgründen wird die Verwendung von Bind-Variablen empfohlen.

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<String> bindParam = []
		
		//SQL-Statement
		
		String **sqlQuery = "SELECT * FROM firmen_spezifisch where kue_dokuart = ? and dok_dat_feld_1 = ?"**
		
		//Setze Wert für 1. Bind-Variable (**kue_dokuart** = Wert)
		
		bindParam.add("DDOKU")
		
		//Setze Wert für 2. Bind-Variable **(dok_dat_feld_40** = Wert)
		
		bindParam.add("Gutschrift")
		
		//Führe SQL-Statement aus
		
		List<GroovyRowResult> sqlResult = d3.sql.executeAndGet(sqlQuery, bindParam)
		
		//Die Anzahl der betroffenen Zeilen steht im Objekt **sqlResult**
		
		d3.log.info**(sqlResult.size() + " betroffene Zeilen")**
		
		//Überprüfe Rückgabe
		
		if(sqlResult != null) {
		
		if(sqlResult.size() > 0) {
		
		//Durchlaufe Liste, wenn mehr als eine betroffene Zeile vorhanden ist
		
		sqlResult.each{
		
		d3.log.info**("DokuID: "+it.doku_id+"/dok_dat_feld_2: "+it.dok_dat_feld_2)**
		
		}
		
		} else {
		
		//Es wurde keine Zeile gefunden
		
		d3.log.info**("Es wurden keine Ergebnisse gefunden")**
		
		}
		
		}

Modifizierer und Typ	Methode und Beschreibung	Anwendungsfall

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.
		List<String> bindParam = []
		
		//SQL-Statement
		
		String **sqlQuery = "SELECT * FROM firmen_spezifisch where kue_dokuart = ? and dok_dat_feld_1 = ?"**
		
		//Setze Wert für 1. Bind-Variable (**kue_dokuart** = Wert)
		
		bindParam.add("DDOKU")
		
		//Setze Wert für 2. Bind-Variable (**dok_dat_feld_40** = Wert)
		
		bindParam.add("Gutschrift")
		
		//Führe SQL-Statement aus, es werden maximal 100 zurückgegeben
		
		List<GroovyRowResult> sqlResult = d3.sql.executeAndGet(sqlQuery, bindParam, 100)
		
		//Die Anzahl der betroffenen Zeilen steht im Objekt **sqlResult**
		
		d3.log.info(sqlResult.size() + " betroffene Zeilen")
		
		//Überprüfe Rückgabe
		
		if(sqlResult != null) {
		
		if(sqlResult.size() > 0) {
		
		//Durchlaufe Liste, wenn mehr als eine betroffene Zeile vorhanden ist
		
		sqlResult.each{
		
		d3.log.info("DokuID: "+it.doku_id+"/dok_dat_feld_2: "+it.dok_dat_feld_2)
		
		}
		
		} else {
		
		//Es wurde keine Zeile gefunden
		
		d3.log.info("Es wurden keine Ergebnisse gefunden")
		
		}
		
		}

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.	Offset bzw. Paging wird momentan nicht unterstützt

Modifizierer und Typ	Methode und Beschreibung	Anwendungsfall

List<GroovyRowResult>	**executeAndGet(String query)**
	
	Ausführen eines SQL-SELECT-Kommandos ohne Bind-Variablen.
	
	Alle Zeilen der Ergebnismenge werden abgerufen und zurückgeliefert.
		//SQL-Statement
		
		String **sqlQuery = "SELECT * FROM firmen_spezifisch where kue_dokuart = 'DDOKU' and dok_dat_feld_1 = 'Audi'"**
		
		//Führe SQL-Statement aus
		
		List<GroovyRowResult> sqlResult = d3.sql.executeAndGet(sqlQuery)
		
		//Die Anzahl der betroffenen Zeilen steht im Objekt **sqlResult**
		
		d3.log.info(sqlResult.size() + " betroffene Zeilen")
		
		//Überprüfe Rückgabe
		
		if(sqlResult != null) {
		
		if(sqlResult.size() > 0) {
		
		//Durchlaufe Liste, wenn mehr als eine betroffene Zeile vorhanden ist
		
		sqlResult.each{
		
		d3.log.info("DokuID: "+it.doku_id+"/dok_dat_feld_2: "+it.dok_dat_feld_2)
		
		}
		
		} else {
		
		//Es wurde keine Zeile gefunden
		
		d3.log.info("Es wurden keine Ergebnisse gefunden")
		
		}
		
		}
		
		Hinweis: Aus Performance- und Sicherheitsgründen wird die Verwendung von Bind-Variablen empfohlen.

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.
		//SQL-Statement
		
		String **sqlQuery = "SELECT * FROM firmen_spezifisch where kue_dokuart = 'DDOKU' and dok_dat_feld_1 = 'Audi'"**
		
		//Führe SQL-Statement aus, es werden maximal 100 zurückgegeben
		
		List<GroovyRowResult> sqlResult = d3.sql.executeAndGet(sqlQuery, 10)
		
		//Die Anzahl der betroffenen Zeilen steht im Objekt **sqlResult**
		
		d3.log.info(sqlResult.size() + " betroffene Zeilen")
		
		//Überprüfe Rückgabe
		
		if(sqlResult != null) {
		
		if(sqlResult.size() > 0) {
		
		//Durchlaufe Liste, wenn mehr als eine betroffene Zeile vorhanden ist
		
		sqlResult.each{
		
		d3.log.info("DokuID: "+it.doku_id+"/dok_dat_feld_2: "+it.dok_dat_feld_2)
		
		}
		
		} else {
		
		//Es wurde keine Zeile gefunden
		
		d3.log.info("Es wurden keine Ergebnisse gefunden")
		
		}
		
		}
		
		Hinweis: Aus Performance- und Sicherheitsgründen wird die Verwendung von Bind-Variablen empfohlen.

Modifizierer und Typ	Methode und Beschreibung	Anwendungsfall
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.	Offset bzw. Paging wird momentan nicht unterstützt

1.9.3. 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

Modifizierer und Typ	Methode und Beschreibung
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
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')
`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
```

> **Anmerkung**
> 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.

1.9.4. Server-API-Funktionen (ScriptCallInterface)

ScriptCallInterface-Funktionen

Die Methoden von ScriptCallInterface entsprechen den Funktionen der JPL-Datei von d.3 server scripting API (alte Bezeichnung vor Version 8: d.3 Server API). In der Dokumentation finden Sie alle Funktionen mit Parametern und Rückgabewerten. Die beschriebenen globalen Variablen werden nicht unterstützt.

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, String
doc_status, String doc_ext, String doc_id, String user_name);
        public int document_dependent_delete (String 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, String
doc_status, int archive_index, int days_in_cache);
        public int folder_create (Document doc);
        public String document_get_file_path (String doc_id, String
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,
                String doc_status, int archive_index, String user_name);
        public int restore_from_jukebox (String doc_status, int
archiv_index, String doc_id, String user_name);
        public int restore_from_jukebox (String doc_status, int
archiv_index, String doc_id, String user_id, String category, String
no_validation, boolean to_cache_only);
        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, String type, String sender, int chain_id, boolean
remove_immediately,
                boolean inherit_class_rule, Timestamp inherit_class_tstamp,
int 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 (int exit_value, String
next_step_id, String doc_id, String user_name);
        // Authorization profiles:
        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, int
render_option, boolean ocr, boolean asynchronous,
                boolean replace_doc, boolean overwrite, String doc_id,
String user_name, String 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, int 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);
        // 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 send_email (String recipient, String notice,
Timestamp date, String doc_id, String user_name, String body_file, String
mail_format, boolean attach,
                                String doc_status, int archiv_index, String
attach_abh, String attach_file);
     }
```

```
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);
    }
}
```

Im Folgenden finden Sie Beschreibungen, Parameter und Rückgabewerte der Funktionen:

document_change_type

Beschreibung: Die Funktion ändert den Dokumenttyp.

Parameter:

- **doc_type_short** (Pflichtfeld): Kurzname des neuen Dokumenttyps
- **doc_id** (Pflichtfeld): ID des Dokuments, dessen Dokumenttyp geändert werden soll
- **user_name**: Name des ändernden Benutzers

Rückgabewerte:

- **0**: alles in Ordnung
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer

- **-2**: Funktion wurde falsch aufgerufen
- **-3**: unbekannter Fehler, kontaktieren Sie den d.velop-Support
- **-101**: weder Dokument-ID übergeben noch Kontext gesetzt
- **-104**: Dokumenttyp-Kurzname nicht angegeben
- **-201**: ungültiger Benutzername angegeben
- **-205**: ungültiger Dokumenttyp-Kurzname angegeben
- **-301**: ungültige Dokument-ID angegeben oder Benutzer hat keine Berechtigungen
- **-302**: Fehler bei der Ermittlung der Benutzerberechtigungen
- **-303**: Benutzer hat keine Zugriffsberechtigungen für das Dokument

Beispiel:

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

D3Interface d3 = getProperty("d3")

String doc_type_short = "DRECH";
String doc_id = "P000000001"
String user_name = "d3_groovy";

def error = d3.call.document_change_type(doc_type_short, doc_id, user_name)
if (error){
  d3.log.info("$error while changing document type")
}
else {
  d3.log.info("Changing document type successful!")
}
```

document_delete

Beschreibung: Die Funktion löscht Dokumente aus d.3.

Parameter:

- **reason**: Grund für die Löschung, notwendig für Freigabe-/Archiv-Version. Pflichtfeld für den Status **Freigabe**/**Archiv**, sonst optional.
- **del_from_each_status** (optional): Löschen aus allen Status: **false** = nur die aktuelle Version, **true**= alle Versionen, Standard: **false**
- **del_file_always** (optional): Dokumentendatei unabhängig vom Status löschen: **false** = nur Dokumentendateien in Bearbeitung/Überprüfung löschen, **true** = Dokumentendatei in jedem Fall löschen, Standard: **false**
- **doc_id** (Pflichtfeld): ID des zu löschenden Dokuments: Wenn nichts übergeben wird, wird die ID des Kontexts verwendet.
- **user_name** (optional): Benutzer, für den das Dokument gelöscht werden soll: Wird nichts angegeben, wird der ausführende Benutzer verwendet.
- **del_privileged** (optional): Wenn auf **true** gesetzt, wird die entsprechende Lizenz bzw. Berechtigung des ausführenden Benutzers zum privilegierten Löschen geprüft. Wenn die Berechtigung nicht vorhanden ist, treten die Fehler **319** und **320** auf.

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen

- **-3**: unbekannter Fehler, kontaktieren Sie den d.velop-Support
- **-101**: weder Dokument-ID übergeben noch Kontext gesetzt
- **-120**: Grund für Löschung nicht angegeben
- **-201**: ungültiger Benutzername angegeben
- **-301**: ungültige Dokument-ID angegeben oder Benutzer hat keine Berechtigungen
- **-302**: Fehler bei der Ermittlung der Benutzerberechtigungen
- **-303**: Benutzer hat keine Zugriffsberechtigungen für das Dokument
- **-311**: Benutzer hat keine Berechtigung zum Löschen des Dokuments
- **-319**: keine Lizenz zum Löschen von Privilegien (neu in Version 7.0)
- **-320**: Benutzer hat keine Berechtigung zum Löschen von Privilegien (neu in Version 7.0)
- **< -9500**: Fehler in kundenspezifischer Hook-Funktion, für Details siehe d.3 logviewer

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

D3Interface d3 = getProperty("d3")

String reason = "Wrong file uploaded";
boolean del_from_each_status = true;
boolean del_file_always = false;
String doc_id = "P000000001";
String user_name = "d3_groovy";
boolean del_privileged = false;

def error = d3.call.document_delete(reason, del_from_each_status,
del_file_always, doc_id, user_name, del_privileged)
if (error){
  d3.log.info("$error while deleting document")
}
else {
  d3.log.info("Deleting document successful!")
}
```

document_get_permission

Beschreibung: Die Funktion legt die Berechtigungen eines Benutzers zur Abfrage eines Dokuments fest. Die Angabe wird in die **api_single_info** in der Form geschrieben, die im API-Aufruf **SearchDocument** (**doc_permission**) beschrieben ist: **1**: Ja, **0**: Nein.

Parameter:

- **doc_id** (Pflichtfeld): ID des Dokuments, dessen Berechtigungen geändert werden sollen. Wird nichts angegeben, wird die ID des Kontexts verwendet. Entweder als Parameter oder Kontext **api_doc_id**.
- **user_name** (optional): Benutzer, dessen Berechtigungen abgefragt werden sollen. Wird nichts angegeben, wird der ausführende Benutzer verwendet.

Rückgabewerte: Die Funktion gibt die Abfrageberechtigungen eines Benutzers für ein Dokument an. Die Berechtigungen werden als String zurückgegeben: **a1a2a3a4a5a6a7a8-b1b2b3b4b5b6b7b8-c1c2-d1d2**.

Parameter	Beschreibung
a1	Zugriff auf Dokumenttyp erlaubt
a2	Notizdatei vorhanden
a3	Dokumentenverknüpfungen vorhanden
a4	Benutzerdatei vorhanden
a5	Benutzerdatei verschlüsselt
a6	Freigegebene Version vorhanden (**1**: freigegeben, **2**: gesperrt)

Parameter	Beschreibung
a7	Letzte Version in Prüfung (**1**: geprüft, **2**: ungeprüft)
a8	Mindestens eine Version im Status **Archiv**
b1	Berechtigung zur Darstellung der Nutzdaten
b2	Berechtigung zum Ändern der Identifikationsdaten
b3	Berechtigung zum Ändern der Benutzerdaten
b4	Berechtigung zum Löschen des Dokuments
b5	Berechtigung zum Status **Transfer**
b6	Berechtigung zum Prüfen des Dokuments
b7	Berechtigung zum Freigeben/Sperren des Dokuments
b8	Berechtigung zum Durchführen beider Prüfschritte (**Prüfung** und **Freigabe**) in einem Schritt
c1	Berechtigung zur Darstellung einer älteren Version in Freigabe
c2	Berechtigung zum Freigeben/Sperren einer älteren Version in Freigabe
d1	Berechtigung zum Einsehen von Versionen im Status **Archiv**
d2	Dokument ist im Workflow

- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-3**: Benutzer existiert nicht

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

D3Interface d3 = getProperty("d3")

String doc_id = "P000000001";
String user_name = "d3_groovy";

String returnValue = d3.call.document_get_permission(doc_id, user_name)
if (returnValue.length() >= 45){
  d3.log.info("Current permissions for user $user_name are $returnValue")
  if(returnValue.substring(0,1) == "1"){
        d3.log.info("Access to document type allowed")
  } else {
    d3.log.info("Access to document type not allowed")
  }
}
else {
  d3.log.info("$error while getting document permission")
}
```

document_block

Beschreibung: Mit dieser Funktion wird ein Dokument gesperrt oder entsperrt.

Parameter:

- **block** (optional):
  - **true**: Dokument entsperren
  - **false**: Dokument sperren (Standard)
- **user_name** (optional): Benutzer, der sperrt bzw. entsperrt. Wird nichts angegeben, wird der Benutzer aus dem Kontext verwendet.
- **doc_id** (Pflichtfeld): Zu übertragendes Dokument. Wenn nichts angegeben wird, wird die ID aus dem Kontext verwendet.

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen
- **-3**: unbekannter Fehler, kontaktieren Sie den d.velop-Support
- **-101**: weder Dokument-ID übergeben noch Kontext gesetzt
- **-201**: ungültiger Benutzername angegeben
- **-301**: ungültige Dokument-ID angegeben oder Benutzer hat keine Berechtigungen
- **-302**: Fehler bei der Ermittlung der Benutzerberechtigungen
- **-303**: Benutzer hat keine Zugriffsberechtigungen für das Dokument
- **-306**: Der Benutzer hat keine Berechtigung zum Sperren oder Entsperren dieses Dokuments.
- **-506**: Es gibt keine freigegebene Version.
- **-507**: Version in Freigabe ist bereits gesperrt
- **-508**: Version in Freigabe ist nicht gesperrt
- **< -9500**: Fehler in kundenspezifischer Hook-Funktion, für Details siehe d.3 logviewer

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

D3Interface d3 = getProperty("d3")

boolean block = true;
String user_name = "d3_groovy";
String doc_id = "P000000001";

def error = d3.call.document_block(block, user_name, doc_id)
if (error){
  d3.log.info("$error while blocking document ")
}
else {
  d3.log.info("Blocking document successful!")
}
```

document_verify

Beschreibung: Diese Funktion versetzt ein Dokument vom Status **Verifizierung, ungeprüft** in den Status **Verifizierung, geprüft**.

Parameter:

- **user_name** (optional): Benutzer, der die Verifizierung durchführen soll. Wird nichts angegeben, wird der Benutzer aus dem Kontext verwendet.
- **doc_id** (Pflichtfeld): Zu übertragendes Dokument. Wird nichts angegeben, wird die ID aus dem Kontext verwendet. Entweder als Parameter oder Kontext **api_doc_id**.

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen
- **-3**: unbekannter Fehler, kontaktieren Sie den d.velop-Support
- **-101**: weder Dokument-ID übergeben noch Kontext gesetzt
- **-201**: ungültiger Benutzername angegeben
- **-301**: ungültige Dokument-ID angegeben oder Benutzer hat keine Berechtigungen

- **-302**: Fehler bei der Ermittlung der Benutzerberechtigungen
- **-303**: Benutzer hat keine Zugriffsberechtigungen für das Dokument
- **-307**: Benutzer hat keine Berechtigungen zum Verifizieren dieses Dokuments
- **-509**: Version im Status **Verifizierung** ist bereits verifiziert
- **-510**: Es gibt keine Version im Status **Verifizierung**.
- **< -9500**: Fehler in kundenspezifischer Hook-Funktion, für Details siehe d.3 logviewer

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

D3Interface d3 = getProperty("d3")

String user_name = "d3_groovy";
String doc_id = "P000000001";

def error = d3.call.document_verify(user_name, doc_id)
if (error){
  d3.log.info("$error while verifying document ")
}
else {
  d3.log.info("Verifying document successful!")
}
```

document_transfer

Beschreibung: Diese Funktion führt eine Statusübertragung durch. Es wird die aktuelle Version verwendet, es sei denn, eine archivierte Version ist vorhanden und ein **archiv_index** ist angegeben.

Parameter:

- **destination** (Pflichtfeld): Zielstatus **Bearbeitung**, **Prüfung**, **Freigabe**, **Archiv**
- **new_editor**: Benutzer-/Gruppenname erforderlich für Ziel = **Bearbeitung** (Benutzer oder Gruppe); Ziel = **Prüfung** (Gruppe). Pflichtfeld, wenn beim Parameter **destination** der Zielstatus **Bearbeitung** angegeben wurde.
- **change_remark**: Muss angegeben werden, wenn ein Dokument aus der Bearbeitung in die Prüfung übernommen wird und das Dokument gegenüber einer früheren Version im Status **Bearbeitung** oder **Archiv** geändert wurde.
- **asynchronous** (optional): Gibt an, ob der Transfer sofort oder asynchron über einen asynchronen Job für d.3 async ausgeführt werden soll (**false**: sofort (Standard); **true**: asynchron durch d.3 async).
- **archiv_index** (optional): Index für die wiederherzustellende Version im Status **Archiv**
- **user_name** (optional): Benutzer, der die Übertragung durchführen soll. Wenn nichts angegeben wird, wird der Benutzer aus dem Kontext verwendet.
- **doc_id** (Pflichtfeld): Zu übertragendes Dokument

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen
- **-3**: unbekannter Fehler, kontaktieren Sie den d.velop-Support
- **-101**: weder Dokument-ID übergeben noch Kontext gesetzt
- **-113**: Gruppenname nicht angegeben
- **-113**: tritt bei der Statusverifizierung nicht mehr auf
- **-114**: Zielstatus nicht angegeben
- **-115**: Archivindex nicht angegeben

- **-116**: Bearbeitungskommentar nicht angegeben
- **-201**: ungültiger Benutzername angegeben
- **-213**: ungültiger Gruppenname angegeben
- **-214**: ungültiger Zielstatus angegeben
- **-301**: ungültige Dokument-ID angegeben oder Benutzer hat keine Berechtigungen
- **-302**: Fehler bei der Ermittlung der Benutzerberechtigungen
- **-303**: Benutzer hat keine Zugriffsberechtigungen für das Dokument
- **-308**: Benutzer hat keine Berechtigung zur Freigabe dieses Dokuments
- **-309**: Benutzer hat keine Berechtigung zur Übertragung des Dokuments
- **-310**: Benutzer hat keine Berechtigung zur Ansicht alter Versionen im Status **Archiv**
- **-511**: Diese Übertragung ist nicht erlaubt
- **-512**: Es gibt eine neuere Version. Um die Version zu überschreiben, muss sie in den Status **Bearbeitung** versetzt werden.
- **-513**: Fehler bei der Erstellung des Jobs
- **-514**: Fehler beim Vergleich der Dokumentendatei von alter und neuer Version
- **< -9500**: Fehler in kundenspezifischer Hook-Funktion **hook_transfer_<Name der Hook-Funktion>**

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

D3Interface d3 = getProperty("d3")

String destination = "Bearbeitung"
String new_editor = "myUser"
String change_remark = ""
boolean asynchronous = false;
int archiv_index = 0;
String user_name = "d3_groovy";
String doc_id  = "P000000001";

def error = d3.call.document_transfer(destination, new_editor,
change_remark, asynchronous, archiv_index, user_name, doc_id)
if (error){
  d3.log.info("$error while transferring document ")
}
else {
  d3.log.info("Transferring document successful!")
}
```

document_publish_for_web

Beschreibung: Kennzeichnet ein Dokument als im Web veröffentlicht.

Parameter:

- **publish**:
  - **0**: Dokument als nicht im Web veröffentlicht kennzeichnen
  - **1**: Dokument als im Web veröffentlicht kennzeichnen (Standard)
- **doc_id**: Zu veröffentlichendes Dokument
- **user_name**: Veröffentlichender Benutzer

Rückgabewerte:

- **0**: alles in Ordnung
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-614**: Dokumenttyp für Webveröffentlichung nicht definiert

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

D3Interface d3 = getProperty("d3")

boolean publish = true;
String doc_id  = "P000000001";
String user_name = "d3_groovy";

def error = d3.call.document_publish_for_web(publish, doc_id, user_name)
if (error){
  d3.log.info("$error while publishing document for web")
}
else {
  d3.log.info("Publishing document for web successful!")
}
```

note_add_file

Parameter:

- **file_name** (Pflichtfeld): Name der Datei, deren Inhalt an die Notiz angehängt werden soll
- **doc_id** (Pflichtfeld): ID des Dokuments, dessen Notizdatei geändert werden soll. Wird nichts angegeben, wird die ID des Kontexts verwendet. Entweder als Parameter oder Kontext **api_doc_id**.
- **user_id** (optional): Benutzer, für den die Notiz geschrieben werden soll. Wird nichts angegeben, wird der ausführende Benutzer verwendet.

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen
- **-3**: unbekannter Fehler, kontaktieren Sie den d.velop-Support
- **-101**: weder Dokument-ID übergeben noch Kontext gesetzt
- **-150**: Dateiname nicht angegeben
- **-201**: ungültiger Benutzername angegeben
- **-250**: Datei nicht gefunden
- **-301**: ungültige Dokument-ID angegeben oder Benutzer hat keine Berechtigungen
- **-302**: Fehler bei der Ermittlung der Benutzerberechtigungen
- **-303**: Benutzer hat keine Zugriffsberechtigungen für das Dokument
- **-401**: Die Datei konnte nicht verschlüsselt/entschlüsselt werden.
- **-402**: Verschlüsselte/entschlüsselte Datei konnte nicht richtig umbenannt werden
- **-403**: Verschlüsselte/entschlüsselte temporäre Datei konnte nicht gelöscht werden
- **< -9500**: Fehler in kundenspezifischer Hook-Funktion

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

D3Interface d3 = getProperty("d3")

String file_name = "C:\\temp\\myNote.txt";
String user_name = "d3_groovy";
String doc_id  = "P000000001";

def error = d3.call.note_add_file(file_name, doc_id, user_name)
```

```
if (error){
  d3.log.info("$error while adding note")
}
else {
  d3.log.info("Adding note successful!")
}
```

note_add_string

Beschreibung: Diese Funktion fügt den Inhalt einer Datei an die Notizdatei des Dokuments an.

Parameter:

- **line** (Pflichtfeld): Zeile, die an die Notiz angehängt werden soll
- **doc_id** (Pflichtfeld): ID des Dokuments, dessen Notizdatei geändert werden soll. Wird nichts angegeben, wird die ID des Kontexts verwendet. Entweder als Parameter oder Kontext **api_doc_id**.
- **user_id** (optional): Benutzer, für den die Notiz geschrieben werden soll. Wird nichts angegeben, wird der ausführende Benutzer verwendet.

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen
- **-3**: unbekannter Fehler, kontaktieren Sie den d.velop-Support
- **-101**: weder Dokument-ID übergeben noch Kontext gesetzt
- **-111**: keine Zeile zum Einfügen angegeben
- **-201**: ungültiger Benutzername angegeben
- **-301**: ungültige Dokument-ID angegeben oder Benutzer hat keine Berechtigungen
- **-302**: Fehler bei der Ermittlung der Benutzerberechtigungen
- **-303**: Benutzer hat keine Zugriffsberechtigungen für das Dokument
- **-401**: Die Datei konnte nicht verschlüsselt/entschlüsselt werden.
- **-402**: Verschlüsselte/entschlüsselte Datei konnte nicht richtig umbenannt werden
- **-403**: Verschlüsselte/entschlüsselte temporäre Datei konnte nicht gelöscht werden
- **< -9500**: Fehler in kundenspezifischer Hook-Funktion **hook_transfer_<Name der Hook-Funktion>**

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

D3Interface d3 = getProperty("d3")

String line = "myNote";
String user_name = "d3_groovy";
String doc_id = "P000000001";

def error = d3.call.note_add_string(line, doc_id, user_name)
if (error){
  d3.log.info("$error while adding note")
}
else {
  d3.log.info("Adding notesuccessful!")
}
```

link_get_parents

Beschreibung: Diese Funktion ermittelt die übergeordneten Dokumente und gibt die entsprechenden Dokument-IDs zurück.

Parameter:

- **doc_id**: ID des Dokuments, dessen Berechtigungen abgefragt werden sollen
- **user_name**: Benutzer, für den die Verknüpfungen ermittelt werden sollen

Rückgabewerte:

Array der Dokument-IDs

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

String doc_id = "P000000001";
String user_name = "d3_groovy";

String[] parents = d3.call.link_get_parents(doc_id, user_name);
for(String parent : parents){
  d3.log.info("link " + parent + "/" + doc_id)
  def error = d3.call.link_delete(parent, doc_id, user_name);
  if (error){
    d3.log.info("$error while deleting link")
  }
  else {
    d3.log.info("Deleting link successful!")
  }
}
```

link_get_children

Beschreibung: Diese Funktion ermittelt die untergeordneten Dokumente und gibt die entsprechenden Dokument-IDs zurück.

Parameter:

- **doc_id**: ID des Dokuments, dessen Unterdokumente ermittelt werden sollen
- **user_name**: Benutzer, für den die Verknüpfungen ermittelt werden sollen

Rückgabewerte:

Array der Dokument-IDs

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

D3Interface d3 = getProperty("d3")

String doc_id = "P000000001";
String user_name = "d3_groovy";

String[] children = d3.call.link_get_children(doc_id, user_name);
for(String child : children){
  d3.log.info("link " + child + "/" + doc_id)
  def error = d3.call.link_delete(doc_id, child, user_name);
  if (error){
    d3.log.info("$error while deleting link")
  }
  else {
    d3.log.info("Deleting link successful!")
```

```
  }
}
```

link_documents

Beschreibung: Diese Funktion verknüpft zwei Dokumente.

Parameter:

- **doc_id_parent**: ID des übergeordneten Dokuments
- **doc_id_child**: ID des untergeordneten Dokuments
- **user_name**: Benutzer, für den die Dokumente verknüpft werden
- **folder_definition**: **true**: Akte erstellen, wenn diese nicht existiert
- **use_folder_plan**: Wenn **doc_id_parent** nicht angegeben ist; **true**: Aktenerstellung für das Dokument verwenden

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen
- **-3**: unbekannter Fehler, kontaktieren Sie den d.velop-Support
- **-101**: weder Dokument-ID übergeben noch Kontext gesetzt
- **-201**: ungültiger Benutzername angegeben
- **-219**: übergeordnete und untergeordnete Dokument-ID sind identisch
- **-301**: ungültige Dokument-ID angegeben oder Benutzer hat keine Berechtigungen
- **-302**: Fehler bei der Ermittlung der Benutzerberechtigungen
- **-303**: Benutzer hat keine Zugriffsberechtigungen für das Dokument
- **-312**: Benutzer hat keine Berechtigungen zum Verknüpfen/Trennen der Dokumente
- **-520**: Die Dokumente sind bereits verknüpft.
- **-521**: Die Dokumente sind bereits entgegengesetzt verknüpft.
- **< -9500**: Fehler in der kundenspezifischen Hook-Funktion **hook_link**

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

D3Interface d3 = getProperty("d3")

String doc_id_parent = "P000000001"
String doc_id_child = "P000000002"
String user_name = "d3_groovy"
Boolean folder_definition = false
Boolean use_folder_plan = false

def error = d3.call.link_documents(doc_id_parent, doc_id_child, user_name,
folder_definition, use_folder_plan)
if (error){
  d3.log.info("$error while linking documents")
}
else {
  d3.log.info("Linking documents successful!")
}
```

link_exists

Beschreibung: Diese Funktion prüft, ob zwei Dokumente miteinander verknüpft sind.

Parameter:

- **doc_id_parent**: ID des übergeordneten Dokuments
- **doc_id_child**: ID des untergeordneten Dokuments
- **test_vice_versa**: Zurzeit wird nur **false** unterstützt

Rückgabewerte:

- **False**: Dokumente sind nicht verknüpft
- **True**: Dokumente sind verknüpft
- **>2**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen
- **-101**: weder Dokument-ID übergeben noch im Kontext für über- oder untergeordnetes Dokument gesetzt
- **-219**: über- und untergeordnete Dokument-ID sind identisch

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

D3Interface d3 = getProperty("d3")

String doc_id_parent = "P000000001"
String doc_id_child = "P000000002"
Boolean test_vice_versa = true

def error = d3.call.link_exists(doc_id_parent, doc_id_child,
test_vice_versa)
if (error == true){
  d3.log.info("The documents are linked")
} else if (error == false){
  d3.log.info("The documents are not linked")
} else {
  d3.log.info("$error while checking link existence")
}
```

link_delete

Beschreibung: Diese Funktion löscht eine Verknüpfung zwischen einem übergeordneten und einem untergeordneten Dokument.

Parameter:

- **doc_id_parent**: ID des übergeordneten Dokuments
- **doc_id_child**: ID des untergeordneten Dokuments
- **user_name**: Benutzer, für den die Verknüpfung ermittelt werden soll

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen
- **-3**: unbekannter Fehler, kontaktieren Sie den d.velop-Support
- **-101**: weder Dokument-ID übergeben noch im Kontext für über- oder untergeordnetes Dokument gesetzt
- **-201**: ungültiger Benutzername angegeben

- **-219**: über- und untergeordnete Dokument-ID sind identisch
- **-301**: ungültige Dokument-ID angegeben oder Benutzer hat keine Berechtigungen
- **-302**: Fehler bei der Ermittlung der Benutzerberechtigungen
- **-303**: Benutzer hat keine Zugriffsberechtigungen für das Dokument
- **-312**: Benutzer hat keine Berechtigungen zum Verknüpfen/Trennen der Dokumente
- **-522**: Die Dokumente sind nicht auf diese Weise verknüpft.
- **< -9500**: Fehler in kundenspezifischer Hook-Funktion **hook_unlink**

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

D3Interface d3 = getProperty("d3")

String doc_id = "P000000001";
String user_name = "d3_groovy";

String[] children = d3.call.link_get_children(doc_id, user_name);
for(String child : children){
  d3.log.info("link " + child + "/" + doc_id)
  def error = d3.call.link_delete(doc_id, child, user_name);
  if (error){
    d3.log.info("$error while deleting link")
  }
  else {
    d3.log.info("Deleting link successful!")
  }
}
```

document_dependent_add

Beschreibung: Speichert ein abhängiges Dokument oder überschreibt ein bestehendes abhängiges Dokument.

Parameter:

- **filename**: Dateipfad der neuen abhängigen Datei
- **doc_status**: Status des zu speichernden abhängigen Dokument (**B** für **Bearbeitung**, **P** für **Prüfung**, **F** für **Freigabe**); Standard: Neueste Version
- **doc_ext**: Erweiterung der abhängigen Datei (Standard: Erweiterung des Dateinamens)
- **doc_id**: ID des Stammdokuments
- **user_name**: Name des ausführenden Benutzers

Rückgabewerte:

- **0**: alles in Ordnung
- **-1**: Interpreter-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen
- **-101**: keine Dokument-ID angegeben
- **-150**: Dateiname nicht angegeben
- **-214**: ungültiger Status angegeben (muss **B**, **P** oder **F** sein)
- **-250**: Datei nicht gefunden
- **-313**: Benutzer darf Dokumentdateien nicht ändern
- **-350**: Zugriff auf Dateien widerrufen
- **-555**: Fehler beim Umbenennen der Datei
- **-556**: Fehler beim Kopieren der Datei
- **-617**: Fehler bei der Abfrage des aktuellen Status in der Datenbank

- **-670**: Die Funktion kann eine Signaturdatei (**.s1**) für ein Dokument nicht mehr speichern.

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

D3Interface d3 = getProperty("d3")

String filename = "C:/temp/depDoc.pdf"
String doc_status = "F"
String doc_ext = "P1"
String doc_id = "P000000001"
String user_name = "d3_groovy"

def error = d3.call.document_dependent_add(filename, doc_status, doc_ext,
doc_id, user_name)
if (error){
  d3.log.info("$error while adding dependent document")
}
else {
  d3.log.info("Adding dependent document successful!")
}
```

document_dependent_delete

Beschreibung: Die Funktion löscht ein abhängiges Dokument.

Parameter:

- **doc_status**: Status des abhängigen Dokuments, das gelöscht werden soll
  - Mögliche Status:
    - **B** für **Bearbeitung**
    - **P** für **Prüfung**
    - **F** für **Freigabe**
  - Standard: aktuelle Version
- **doc_ext**: Dateierweiterung der abhängigen Datei
- **doc_id**: Dokument-ID des abhängigen Dokuments
- **user_name**: Name des ausführenden Benutzers

Rückgabewerte:

- **0**: alles in Ordnung
- **< 0**: Datenbankfehler
- **-1**: Syntaxfehler im Interpreter, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen
- **-101**: keine Dokument-ID angegeben
- **-139**: Keine Erweiterung der abhängigen Datei (**doc_ext**) angegeben
- **-214**: ungültiger Status angegeben (muss **B**, **P** oder **F** sein)
- **-313**: Benutzer darf Dokumentdateien nicht ändern
- **-350**: Zugriff auf Dateien widerrufen
- **-617**: Fehler bei der Abfrage des aktuellen Status in der Datenbank

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

D3Interface d3 = getProperty("d3")

String doc_status = "F"
```

```
String doc_ext = "P1"
String doc_id = "P000000001"
String user_name = "d3_groovy"

def error = d3.call.document_dependent_delete(doc_status, doc_ext, doc_id,
user_name)
if (error){
  d3.log.info("$error while deleting dependent document")
}
else {
  d3.log.info("Deleting dependent document successful!")
}
```

document_start_lifetime

Beschreibung: Die Funktion legt die Lebensdauer für ein Dokument fest.

Hinweis: Wenn Sie eine neue Dokumentversion importieren, dann wird die vorher festgelegte Dauer zurückgesetzt und muss neu gesetzt werden.

Hinweis: Wenn ein Sekundärspeicher eingesetzt wird, dann kann für die Dateien zu dem Dokument auf dem Sekundärspeicher nicht reduziert oder entfernt werden.

Parameter:

- **doc_id**: Dokument-ID
- **overwrite_old_date**: überschreibt den aktuellen Wert (Standard: **true**)
- **life_time_days**: Wenn die für den Dokumententyp konfigurierte Lebensdauer nicht verwendet werden soll, können Sie die Anzahl der Tage angeben (Standard: **0** - konfigurierte Lebensdauer, **-1** - Unbegrenzte Aufbewahrungsdauer).

Rückgabewerte:

- **0**: alles in Ordnung
- **-1**: Syntaxfehler im Interpreter, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen
- **-621**: Es ist nicht vorgesehen, die Lebensdauer des Dokumenttyps zu ändern und für den Parameter **overwrite_old_date** den Wert **0** anzugeben.
- **-622**: Datenbankfehler, für Details siehe d.3 logviewer

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

D3Interface d3 = getProperty("d3")

String doc_id = "P000000001"
boolean overwrite_old_date = true
int life_time_days = 3650

def error = d3.call.document_start_lifetime(doc_id, overwrite_old_date,
life_time_days)
if (error){
  d3.log.info("$error while starting lifetime")
}
else {
  d3.log.info("Starting lifetime successful!")
}
```

document_set_cache_days

Beschreibung: Die Funktion legt die Anzahl der Tage für eine Dokumentversion fest, die die Dokumentversion im d.3-Dokumentenbaum "ab heute" verbleiben soll. Die Anzahl von Tagen ist nur relevant, wenn das Dokument auf einen Sekundärspeicher gespeichert wurde oder wird.

Parameter:

- **doc_id**: Dokument-ID
- **doc_status**: Status des Dokument
  - Mögliche Status:
    - **Freigabe** (**Fr**)
    - **Archiv**
  - Standard: **Freigabe** (**Fr**)
- **archive_index** (Pflichtfeld): Index der Archivversion (verpflichtend für den Status **Archiv**); wird bei Freigabe nicht berücksichtigt
- **days_in_cache** (Pflichtfeld): Anzahl der Tage, die die Dokumentversion im Cache verbleiben soll

Rückgabewerte:

- **0**: alles in Ordnung
- **-1**: Syntaxfehler im Interpreter, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen
- **-115**: kein Index oder keine Archivversion für den Status **Archiv** angegeben
- **-223**: ungültiger Status angegeben (muss **F** oder **A** sein)
- **-623**: Anzahl der Tage fehlt, die die Dokumentversion im d.3-Dokumentenbaum verbleiben soll
- **-624**: Dokumentversion existiert nicht oder ist bereits im d.3-Dokumentenbaum gelöscht

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

D3Interface d3 = getProperty("d3")

String doc_id = "P000000001"
String doc_status = "Fr"
int archive_index = 0
int days_in_cache = 10

def error = d3.call.document_set_cache_days(doc_id, doc_status,
archive_index, days_in_cache)
if (error){
  d3.log.info("$error setting cache days")
}
else {
  d3.log.info("Setting cache days successful!")
}
```

folder_create

Beschreibung: Mit der Funktion werden neue Akten erstellt.

Parameter:

doc: Element des Dokuments mit allen Eigenschaften für die neue Akte

Rückgabewerte:

- **0**: alles in Ordnung

- **-548**: Fehler beim Erstellen des Ordners, für Details siehe d.3 logviewer

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

D3Interface d3 = getProperty("d3")

def doc = d3.archive.newDocument();

doc.type = "APERS"
doc.status = Document.DocStatus.DOC_STAT_RELEASE
doc.editor = "d3_groovy"
doc.setText(1, "Comment text row 1")
doc.setNumber("myNumber2")

doc.field[1] = "folder_create - Attrib 1"
doc.field[2] = "folder_create - Attrib 2"
doc.field[4] = "folder_create - Attrib 4"
...
doc.field[60][1] = "folder_create - Attrib 60-1"
...

def error = d3.call.folder_create(doc)
if (error) {
  d3.log.info("$error while folder creation!")
}
else {
  d3.log.info("Folder creation successful!")
}
```

document_register_dependent (Deprecated)

Beschreibung: Die Funktion registriert die abhängigen Dateien eines Dokuments.

Parameter:

- **doc_id**: Originaldokument, zu dem die abhängigen Dokumenente registriert werden sollen
- **archive_index**: Index der Archivversion ist erforderlich für den Status **Archiv**
- **status**: Status des Dokuments (**Bearbeitung**, **Freigabe**, **Prüfung** oder **Archiv**)
- **user_group**: Wenn dieser Parameter übergeben wird, steht der Parameter vor dem eigentlichen Bearbeiter des Dokuments, das unter **doc_id** angegeben ist. Der Parameter ist nur für den Status **Bearbeitung** und **Prüfung**.

Rückgabewerte:

- **0**: alles in Ordnung
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen

document_get_file_path

Beschreibung: Die Funktion gibt den vollständigen Pfad eines Dokuments im d.3-Dokumentenbaum zurück.

Parameter:

- **doc_id**: Originaldokument, für das der Pfad ermittelt werden soll
- **doc_status**: Status des Dokuments (**B** für **Bearbeitung**, **P** für **Prüfung**, **F** für **Freigabe**)

- **archive_index**: Index der Archivversion (verpflichtend für den Status **Archiv**)
- **user_group**: Wenn dieser Parameter übergeben wird, steht der Parameter vor dem eigentlichen Bearbeiter des Dokuments, das unter **doc_id** angegeben ist. Der Parameter ist nur für den Status **Bearbeitung** und **Prüfung**.
- **dependent_ext**: Wenn es sich um eine abhängige Datei handelt, geben Sie die Erweiterung der Datei an.

Rückgabewerte:

- **0**: alles in Ordnung
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen
- **-115**: kein Index oder keine Archivversion für den Status **Archiv** angegeben
- **-126**: weder Dokument-ID übergeben noch Kontext festgelegt

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

String doc_id = "P000000001"
String doc_status = "F"
int archive_index = 0
String user_group = null
String dependent_ext = ""

def pathOfDocument = d3.call.document_get_file_path(doc_id, doc_status,
archive_index, user_group, dependent_ext)
if (pathOfDocument){
  d3.log.info("Getting path successful! -> " + pathOfDocument)
}
```

document_send_to_dsearch

Beschreibung: Die Funktion sendet OCR-Informationen zu einer Dokumentversion an d.3 search.

Parameter:

- **doc_id**: Dokument, das in d.3 search aktualisiert werden soll
- **ocr_file**: OCR-Datei mit den OCR-Informationen. Wenn die Eigenschaften übertragen werden sollen oder **use_existent_orc_file** aktiviert ist, kann der Parameter leer bleiben.
- **version**: Version des Dokuments, das geändert werden soll. Wenn keine Version angegeben ist, wird die aktuelle Version verwendet. Für Notizdateien muss die Konstante nicht übergeben werden.
- **dsearch_corpus**: Korpus, an den das Dokument übergeben werden soll. Sie müssen den Parameter festlegen, wenn ein bestehendes Dokument an ein anderes oder neues Korpus übergeben werden soll.
- **use_existent_ocr_file**: Legen Sie diesen Parameter fest, um bestehende OCR-Dateien zu verwenden. Sie können mit dem Parameter bestehende Dokumente erneut an die d.3-Suche übergeben, z.B. für Wiederherstellen, neue Volltextmaschine (Standard: **false**).
- **doc_status**: Status der Dokumentversion, die aktualisiert werden soll. Wenn keine Version angegeben ist, wird die aktuelle Version des Dokuments verwendet (**B** für **Bearbeitung**, **P** für **Prüfung**, **F** für **Freigabe**).
- **archive_index**: Archivindex einer Archivversion. Der Parameter muss nur übergeben werden, wenn für den Parameter **doc_status** der Wert **Archiv** festgelegt ist.
- **user_name**: Name des ausführenden Benutzers

Rückgabewerte:

- **0**: alles in Ordnung
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer

- **-2**: Funktion wurde falsch aufgerufen
- **-101**: keine Dokument-ID angegeben
- **-313**: Benutzer darf Dokumentdateien nicht ändern
- **-660**: kein Wert für **DSEARCH_SUPPORT**
- **-661**: Fehler beim Erstellen des OCR-Unterverzeichnisses
- **-662**: Fehler beim Kopieren der OCR-Datei
- **-663**: d.3 search meldet einen Fehler, siehe d.3 logviewer für die Fehlernummer

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

D3Interface d3 = getProperty("d3")

String doc_id = "P000000001"
String ocr_file = null
int version = 0
String dsearch_corpus = ""
boolean use_existent_ocr_file = false
String doc_status = "B"
int archive_index = 0
String user_name = "d3_groovy"

def error = d3.call.document_send_to_dsearch(doc_id, ocr_file, version,
dsearch_corpus, use_existent_ocr_file, doc_status, archive_index, user_name)
if (error){
  d3.log.info("$error while sending to d.3 search")
}
else {
  d3.log.info("Sending to d.3 search successful!")
}
```

restore_from_jukebox

Beschreibung: Die Funktion stellt eine Datei aus dem Sekundärspeicher wieder her.

Parameter:

- **doc_status**: Status der Dokumentversion, die wiederhergestellt werden soll (**F** für **Freigabe**, **Archiv** für **Archiv**)
- **archive_index**: Archivindex einer Archivversion. Der Parameter muss nur übergeben werden, wenn für den Parameter **doc_status** der Wert **Archiv** festgelegt ist.
- **doc_id**: Dokument, das aus dem Sekundärspeicher wiederhergestellt werden soll
- **user_id**: ID des ausführenden Benutzers

Rückgabewerte:

- **0**: alles in Ordnung
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen
- **-3**: unbekannter Fehler
- **-101**: keine Dokument-ID angegeben
- **-126**: weder Dokument-ID übergeben noch Kontext festgelegt
- **-201**: ungültige Benutzer-ID
- **-223**: ungültiger Dokumentstatus
- **-301**: ungültige Dokument-ID oder fehlende Berechtigungen
- **-302**: Fehler beim Abrufen der Benutzerberechtigung

- **-303**: Benutzer hat keine Berechtigungen
- **-310**: Benutzer hat keine Berechtigungen für Archivversion
- **-506**: keine öffentliche Version vorhanden
- **-531**: Keine Archivversion vorhanden
- **-532**: Pfad des Dokuments kann nicht ermittelt werden
- **-533**: Datei ist bereits im Dokumentbaum, Wiederherstellen nicht notwendig
- **-534**: d.ecs storage manager hat die Datei nicht geliefert
- **-556**: Fehler beim Kopieren der Datei

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

D3Interface d3 = getProperty("d3")

String doc_status = "Freigabe"
int archiv_index = 0
String doc_id = "P000000001"
String user_name = "d3_groovy"

def error = d3.call.restore_from_jukebox(doc_status,archiv_index, doc_id,
user_name)
if (error){
  d3.log.info("$error while restoring document")
}
else {
  d3.log.info("Restoring document successful!")
}
```

restore_from_history

Beschreibung: Die Funktion stellt eine Datei aus der Historie wieder her.

Parameter:

- **aktion_id**: ID der Aktion, die das Dokument gelöscht hat. Wenn nichts angegeben ist, wird die höchste Aktionsnummer verwendet, die für dieses Dokument verwendet wird.
- **doc_id**: Dokument, das aus dem Sekundärspeicher wiederhergestellt werden soll
- **user_id**: ID des ausführenden Benutzers

Rückgabewerte:

- **0**: alles in Ordnung
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen

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

D3Interface d3 = getProperty("d3")

int aktion_id = 0;
String doc_id = "P000000001"
String user_name = "d3_groovy"

def error = d3.call.restore_from_history(aktion_id, doc_id, user_name)

if (error){
```

```
  d3.log.info("$error while restoring document")
} else {
  d3.log.info("Restoring document successful!")
  d3.log.info("Now trigger folder plan for document")
  Document myDoc = d3.archive.getDocument(doc_id, user_name)
  myDoc.checkFolderScheme(user_name)
}
```

add_inherit_doc_rights

Beschreibung: Die Funktion vererbt Berechtigungen für ein Dokument.

Parameter:

- **doc_id**: Dokument, für das die Berechtigungen erteilt werden sollen
- **granter**: Benutzer, der die Berechtigungen vererbt
- **grantee**: Benutzer, der die Berechtigungen erbt
- **right_flags**:
  - Position 1:
    - **1**: Leserechte
    - **2**: Schreibrechte (beinhaltet auch Leserechte)
  - Position 2:
    - **1**: Berechtigungen laufen nach dem Zeitstempel ab
    - **2**: Berechtigungen laufen nach Bestätigung ab
  - Position 3: **1**: Rekursive Berechtigungen (betrifft alle Dokumente, die mit dem Dokument als Kind verknüpft sind)
- **tstamp_expire**: Zeitstempel für den Ablauf, nur bei korrekter Kennzeichnung

Rückgabewerte:

- **0**: alles in Ordnung
- **3**: Parameter fehlt
- **120**: Vererbender Benutzer erbender Benutzer existiert nicht
- **450**: Vererbender Benutzer darf keine Berechtigungen vererben
- **451**: Vererbender Benutzer hat kein Dokument
- **452**: Vererbender Benutzer hat bereits Berechtigungen für das Dokument
- **453**: Berechtigungen für das Dokument können nicht vererbt werden, da der erbende Benutzer keine Berechtigungen für die Dokumentart hat

```
import com.dvelop.d3.server.core.D3Interface
import com.dvelop.d3.server.Document
import java.sql.Timestamp;

D3Interface d3 = getProperty("d3")

String doc_id = "P000000001"
String granter = "user1"
String grantee = "user2"
String right_flags = 211 //Inherit write rights for a given period of time.
For given document and all linked children
Timestamp tstamp_expire = new Date().plus(30).toTimestamp() // 30 days

def error = d3.call.add_inherit_doc_rights(doc_id, granter, grantee,
right_flags, tstamp_expire)
if (error){
  d3.log.info("$error while inheriting doc rights")
```

```
}
else {
  d3.log.info("Inheriting doc rights successful!")
}
```

remove_inherit_doc_rights

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

D3Interface d3 = getProperty("d3")

String doc_id = "P000000001"
String granter = "user01"
String grantee = "user02"

def error = d3.call.remove_inherit_doc_rights(doc_id, granter, grantee)
if (error){
  d3.log.info("$error while removing inherited doc rights")
}
else {
  d3.log.info("Removing inherited doc rights successful!")
}
```

hold_file_send

Beschreibung: Die Funktion sendet ein Dokument an das Postfach eines d.3-Benutzers.

Parameter:

- **recipient** (Pflichtfeld): Empfänger der Wiedervorlage (d.3-Benutzer oder d.3-Gruppe). Trennen Sie mehrere Empfänger mit einem Semikolon.
- **notice** (optional): Text für den Betreff
- **doc_id**: ID des Dokuments, das versendet wird
- **tstamp_acknowledge** (optional): Zeitstempel, bis wann die Einträge bestätigt werden sollen
- **tstamp_remember** (optional): Zeitstempel, zu dem eine Erinnerung gesendet werden soll
- **expand_groups** (optional): Wenn Sie diesen Parameter aktivieren, werden Benutzer individuell aufgelöst und benachrichtigt (Standard: **false**).
- **ignore_checkout** (optional): Mit diesem Parameter legen Sie fest, ob ein Dokument versendet werden soll, auch wenn der Empfänger abgemeldet ist.
  - Mögliche Werte:
    - **false**: das Dokument wird nicht gesendet
    - **true**: das Dokument wird trotzdem gesendet
  - Standard: **false**
- **date_activate** (optional): Datum, an dem die Wiedervorlage gesendet werden soll
- **type** (optional): Art der Wiedervorlage (**W** für Workflow)
- **sender**: Benutzername des Absenders
- **chain_id** (optional): ID der Wiedervorlagekette
- **remove_immediately**: Wenn der Wert **true** ist, wird die Wiedervorlage nach dem Absenden sofort gekennzeichnet, sodass kein Eintrag unter den gesendeten Wiedervorlagen erscheint.
- **inherit_class_rule**: Erlöschen der Berechtigung nach dem Zeitstempel (in diesem Fall muss **inherit_class_tstamp** festgelegt sein). Wenn der Wert **false** ist, erlischt die Berechtigungen nach Bestätigung im Postfach.
- **inherit_class_tstamp**: Zeitpunkt des Ablaufs für die weitergegebenen Berechtigugen als Zeitstempel
- **inherit_class_right**:

  - Mögliche Werte:
    - **1**: Leserechte
    - **2**: Schreibrechte (beinhaltet auch Leserechte)
- **check_write_access**: Wenn der Wert **true** ist, wird geprüft, ob der Empfänger Schreibrechte für das Dokument hat.

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen
- **-3**: undefinierter Fehler, bitte wenden Sie sich an den d.velop-Support
- **-101**: weder Dokument-ID übergeben noch Kontext festgelegt
- **-110**: Empfänger nicht angegeben
- **-201**: Ungültiger Benutzername angegeben
- **-207**: Ungültiger Empfänger angegeben
- **-208**: Ungültiges Datum angegeben
- **-209**: Ungültige Ketten-ID angegeben
- **-212**: Ungültige Dokument-ID angegeben
- **-244**: Ungültige Regel oder Berechtigung angegeben
- **-301**: Ungültige Dokument-ID angegeben oder Benutzer hat keine Berechtigungen für das Dokument
- **-302**: Fehler beim Ermitteln der Benutzerberechtigungen
- **-303**: Benutzer hat keine Zugriffsberechtigungen für das Dokument
- **-315**: Kein Gruppenmitglied hat Zugriffsberechtigungen für das Dokument, das versendet werden soll.
- **-316**: Empfänger hat keine Zugriffsberechtigungen, aber Absender kann die Berechtigungen erteilen
- **-317**: Absender hat keine Schreibrechte. Die Berechtigungen können nicht an die Empfänger übertragen werden.
- **-318**: Absender hat keine Schreibrechte. Die Berechtigungen können vererbt werden.
- **-450**: Benutzer hat keine Berechtigungen zum Erben
- **-451**: Benutzer verfügt nicht über die entsprechenden Berechtigungen
- **-452**: Empfänger hat bereits die Berechtigungen, die vererbt werden sollen
- **-453**: Dem Benutzer sind keine Dokumentenklassen oder Berechtigungsprofile zugeordnet
- **-454**: Empfänger benötigt Berechtigungen für die Dokumentart, die geerbt werden soll
- **-455**: temporär erstellte Dokumentenklasse wurde dem Benutzer bereits zugewiesen
- **-503**: Empfänger ist abgemeldet

```
import com.dvelop.d3.server.core.D3Interface
import com.dvelop.d3.server.Document
import java.sql.Timestamp;

D3Interface d3 = getProperty("d3")

String recipient = "user01"
String notice = "Please check"
String doc_id = "P000000001"
Timestamp tstamp_acknowledge = new Date().plus(10).toTimestamp() // 10 days
Timestamp tstamp_remember = new Date().plus(30).toTimestamp() // 30 days
boolean expand_groups = false
boolean ignore_checkout = true
Timestamp date_activate = null // Now
String type = ""
String sender = "d3_groovy"
int chain_id = 0
```

```
boolean remove_immediately = false
boolean inherit_class_rule = false
Timestamp inherit_class_tstamp = null;
int inherit_class_right = 0
boolean check_write_access = false

def error = d3.call.hold_file_send(recipient, notice, doc_id,
tstamp_acknowledge, tstamp_remember, expand_groups, ignore_checkout,
date_activate, type, sender, chain_id, remove_immediately,
inherit_class_rule, inherit_class_tstamp, inherit_class_right,
check_write_access)
if (error){
  d3.log.info("$error while sending hold file")
}
else {
  d3.log.info("Sending hold file successful!")
}
```

hold_file_find

Beschreibung: Die Funktion sucht nach den Benutzern, in deren Wiedervorlage sich das Dokument befindet.

Parameter:

- **recipient** (Pflichtfeld): Empfänger der Wiedervorlage (d.3-Benutzer oder d.3-Gruppe). Trennen Sie mehrere Empfänger mit einem Semikolon.
- **doc_id**: ID des Dokuments, das gesendet werden soll
- **user_name**: Benutzer, in dessen Namen gesucht werden soll

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen
- **-3**: undefinierter Fehler, bitte wenden Sie sich an den d.velop-Support
- **-101**: weder Dokument-ID übergeben noch Kontext festgelegt
- **-201**: Ungültiger Benutzername angegeben
- **-301**: Ungültige Dokument-ID angegeben oder Benutzer hat keine Berechtigungen für das Dokument
- **-302**: Fehler beim Ermitteln der Benutzerberechtigungen
- **-303**: Benutzer hat keine Zugriffsberechtigungen für das Dokument

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

D3Interface d3 = getProperty("d3")

String recipient = null //get all hold_file_recipients of document
String doc_id = "P000000001"
String user_name = "d3_groovy"

String[] hold_file_recipients = d3.call.hold_file_find(recipient, doc_id,
user_name)
if (hold_file_recipients){
  d3.log.info("Finding hold files successful!")
  for(String hold_file_recipient : hold_file_recipients){
```

```
    d3.log.info("hold file recipient -> " + hold_file_recipient)
  }
}
```

hold_file_delete

Beschreibung: Die Funktion bestätigt empfangene Erinnerungen und löscht gesendete Erinnerungen.

Parameter:

- **chain_id** (Pflichtfeld): ID der Wiedervorlagekette, zu der der Eintrag gehört
- **sent_received** (optional):
  - Mögliche Werte:
    - **1**: Wiedervorlagedatei gesendet oder empfangend
    - **2**: Wiedervorlage
  - Standard: **2**
- **workflow_only** (optional): Wenn der Wert **0** oder **1** ist, wird der Wiedervorlageworkflow bestätigt (nur wenn **sent_received** den Wert **2** hat).
- **recipient** (optional): Empfänger (nur wenn **sent_received** den Wert **1** hat oder der Empfänger eine Gruppe ist)
- **doc_id**: Benutzer, in dessen Name gesucht werden soll
- **user_name**: Benutzer, in dessen Name gelöscht oder bestätigt werden soll

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen
- **-3**: undefinierter Fehler, bitte wenden Sie sich an den d.velop-Support
- **-101**: weder Dokument-ID übergeben noch Kontext festgelegt
- **-110**: Empfänger nicht angegeben
- **-124**: Keine Ketten-ID angegeben
- **-528**: Fehler beim Löschen der Absenderliste (falsche Ketten-ID)
- **-529**: Fehler beim Löschen der Empfängerliste (falsche Ketten-ID)

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

D3Interface d3 = getProperty("d3")

long chain_id = 15
Byte sent_received = 2
boolean workflow_only = false
String recipient = "user01"
String doc_id = "P000000001"
String user_name = "d3_groovy"

def error = d3.call.hold_file_delete(chain_id, sent_received,
workflow_only, recipient, doc_id, user_name)
if (error){
  d3.log.info("$error while sending hold file")
}
else {
  d3.log.info("Sending hold file successful!")
}
```

workpath_end_document

Beschreibung: Diese Funktion entfernt das Dokument aus einem Workflow.

Parameter:

- **doc_id**: Dokument-ID, sofern nichts angegeben ist
- **user_name**: Benutzer, für den gesucht werden soll
- **delete_jobs**:
  - Mögliche Werte:
    - **false**: Workflowjobs, die sich noch in der asynchronen Jobwarteschlange befinden, werden nicht entfernt.
    - **true**: Alle entsprechenden Workflowjobs werden entfernt, die sich noch in der asynchronen Jobwarteschlange befinden.
  - Standard: **false**

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen

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

D3Interface d3 = getProperty("d3")

String doc_id = "P000000001"
String user_name = "d3_groovy"
boolean delete_jobs = false

def error = d3.call.workpath_end_document(doc_id, user_name, delete_jobs)
if (error){
  d3.log.info("$error while ending workpath")
} else {
  d3.log.info("Ending workpath successful!")
}
```

workpath_start_document

Beschreibung: Diese Funktion fügt das Dokument im Workflow ein.

Parameter:

- **wfl_id**: ID für den Workflow, in den das Dokument eingefügt werden soll
- **doc_id**: Dokument-ID
- **user_name**: Benutzer, für den gesucht werden soll. Wenn kein Wert angegeben ist, wird der ausführende Benutzer verwendet.

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen

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

```
D3Interface d3 = getProperty("d3")

String wfl_id = "398t9mgq983q44jg"
String doc_id = "P000000001"
String user_name = "d3_groovy"

def error = d3.call.workpath_start_document(wfl_id, doc_id, user_name)
if (error){
  d3.log.info("$error while starting workpath")
} else {
  d3.log.info("Starting workpath successful!")
}
```

workpath_go_to_next_step

Beschreibung: Die Funktion verschiebt das Dokument im Workflow, entweder durch Angeben eines Entscheidungswertes oder durch Angeben des nächsten Schritts.

Parameter:

- **exit_value**: Exit, mit dem der Schritt abgeschlossen werden soll
  - 1: noch im Startelement
  - 2: TRUE
  - 3: FALSE
  - 100: manuelles Verschieben
- **next_step_id**: ID des Schritts, mit dem der Workflow weitergeführt werden soll. Dieser Parameter wird für **exit_value** = 100 benötigt.
- **doc_id**: ID des Dokuments.
- **user_name**: Benutzer in dessen Name die Aktion ausgeführt werden soll.

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen

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

D3Interface d3 = getProperty("d3")

int exit_value = 100
String next_step_id = ""
String doc_id = "P000000001"
String user_name = "d3_groovy"

def error = d3.call.workpath_go_to_next_step(exit_value, next_step_id,
doc_id, user_name)
if (error){
  d3.log.info("$error while going to next workpath step")
} else {
  d3.log.info("Going to next workpath step successful!")
}
```

roll_get_names

Beschreibung: Die Funktion fragt alle Rollennamen ab.

Parameter: Keine

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen

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

D3Interface d3 = getProperty("d3")

def role_names = d3.call.roll_get_names()
if (role_names){
  d3.log.info("Getting role names successful!")
  for(String role_name : role_names){
    d3.log.info("role_name -> " + role_name)
  }
}
```

roll_get_users

Beschreibung: Die Funktion zeigt dem Benutzer an, wem die Rolle zugewiesen ist.

Parameter:

- **roll_id**: ID der Rolle, optional
- **rollname**: Name der Rolle, optional

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen
- **-118**: Weder Rollen-ID, noch Rollenname angegeben
- **-217**: Ungültige Rollen-ID oder ungültiger Name angegeben

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

D3Interface d3 = getProperty("d3")

Long roll_id = 6
String roll_name = ""

String[] assigned_users = d3.call.roll_get_users(roll_id, roll_name)
if (assigned_users){
  d3.log.info("Getting assigned users successful!")
  for(String assigned_user : assigned_users){
    d3.log.info("assigned_user -> " + assigned_user)
  }
}
```

document_render

Beschreibung: Erzeugt TIFF, PDF, OCR

Parameter:

- **source**: Quelldatei; erforderlich, wenn das Dokument doc_id (letzte Version) nicht gerendert werden soll
- **destination**: Zieldatei (Pflichtfeld), wenn die gerenderte Datei nicht als abhängige t1- bzw. p1-Datei für doc_id gespeichert werden soll.
- **render_option**:
  - 0: Keine Erstellung einer Renderdatei, z.B. nur die OCR-Datei erstellen
  - 1: Erstellen einer TIFF-Datei als Renderdatei (Standard)
  - 2: Erstellen einer PDF-Datei als Renderdatei
  - 3: Erstellen einer TIFF- und PDF-Datei als Renderdatei
- **ocr**:
  - **false:** Indexschlüsselwörter nicht extrahieren (Voreinstellung)
  - **true:** Indexschlüsselwörter extrahieren
- **asynchronous**: Nur der Wert **true** ist möglich
- **replace_doc**:
  - **false:** Generiertes TIFF/PDF ersetzt nicht das Quelldokument in d.velop documents (Standard)
  - **true:** Generiertes TIFF/PDF ersetzt das Quelldokument in d.velop documents
- **overwrite**:
  - **false:** Zieldatei nicht ersetzen, wenn sie bereits existiert (Standard)
  - **true:** Zieldatei ersetzen, wenn sie bereits existiert
- **doc_id**: ID des Dokuments
- **user_name**: Benutzer, in dessen Namen der Befehl ausgeführt werden soll.
- **doc_status**: Status der Version, die gerendert werden soll. Wenn Sie nichts angeben, wird die aktuelle Version verwendet. (**B** für **Bearbeitung, P** für **Prüfung, F** für **Freigabe, A** für **Archiv**)
- **archiv_index**: Index für die Version im Statusarchiv, die gerendert werden soll (nur für **=A/Archiv** notwendig)
- **prio**: Gültige Werte sind **low, normal** (Standard), **high**

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen
- **-118**: Weder Rollen-ID, noch Rollenname angegeben
- **-217**: Ungültige Rollen-ID oder ungültiger Name angegeben

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

D3Interface d3 = getProperty("d3")

String source = null
String destination = null
int render_option = 2
boolean ocr = true
boolean asynchronous = false
boolean replace_doc = false
boolean overwrite = true
String doc_id = "P000000001"
```

```
String user_name = "d3_groovy"
String doc_status = null
int archiv_index = 0
String prio = "Low"

def error = d3.call.document_render(source, destination, render_option,
ocr, asynchronous, replace_doc, overwrite, doc_id, user_name, doc_status,
archiv_index, prio)
if (error){
  d3.log.info("$error while sending to d.ecs rendition service")
}
else {
  d3.log.info("Sending to d.ecs rendition service successful!")
}
```

tiff_concat

Beschreibung: Die Funktion fügt ein TIFF-Dokument an ein anderes Dokument an.

Parameter:

- **source**: TIFF-Quelldatei
- **destination**: TIFF-Zieldatei
- **source_rdl**: Quelldatei für Kommentare, optional
- **destination_rdl**: Zieldatei für Kommentare, optional

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen
- **-3:** Undefinierter Fehler, kontaktieren Sie den d.velop-Support.
- **-151**: Quelldateiname ist nicht angegeben
- **-152**: Zieldateiname ist nicht angegeben
- **-559**: Fehler beim Anhängen der TIFF-Dateien
- **-560**: Anzahl der Zielseiten abweichend von der Summe der Quellenseiten
- **-561**: TIFF-Quelldatei existiert nicht
- **-562**: TIFF-Quelldatei wird nicht unterstützt (möglicherweise beschädigt oder falsches Format)
- **-563**: TIFF-Zieldatei kann nicht geöffnet werden
- **-564**: Redlining-Quelldatei wird nicht unterstützt (möglicherweise beschädigt oder falsches Format)
- **-565**: Redlining-Quelldatei enthält nicht zugeordnete Seiten

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

D3Interface d3 = getProperty("d3")

String source = "C:\\temp\\myTiff1.tif"
String destination = "C:\\temp\\myTiff2.tif"
String source_rdl = ""
String destination_rdl = ""

def error = d3.call.document_render(source, destination, render_option,
source_rdl, destination_rdl)
if (error){
```

```
  d3.log.info("$error while concating tif files")
}
else {
  d3.log.info("Concating tif files successful!")
}
```

document_render_wfl_prot

Beschreibung: Übermittelt die Workflowprotokolle an eine Dokument-ID und erstellt eine abhängige W1-Datei. Wenn bereits eine W1-Datei vorhanden ist, führt die Funktion keine Aktion aus (Rückgabe 1). Wenn eine neue TIFF-Datei übermittelt werden soll, weil z.B. ein weiteres Protokoll für die Dokument-ID hinzugefügt wurde, müssen Sie die abhängige W1-Datei entfernen.

Parameter:

- **doc_id**: Dokument, dessen Workflow-Protokolle übermittelt werden sollen
- **user_name**: Name des ausführenden Benutzers

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen

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

D3Interface d3 = getProperty("d3")

String doc_id = "P000000001"
String  user_name = "d3_groovy"

def error = d3.call.document_render_wfl_prot(doc)
if (error){
  d3.log.info("$error while rendering wfl protocol")
} else {
  d3.log.info("Rendering wfl protocol successful!")
}
```

object_property_set

Beschreibung: Legt die Eigenschaftswerte für Objekte fest. Wenn Sie einen Null-String für den Wert festlegen, wird der Eigenschaftswert für das Objekt gelöscht.

Parameter:

- **property_name**: Interner Name der Objekteigenschaft
- **object_id**: Objektname (z.B. Benutzername, Gruppenname abhängig von **object_class_id)**
- **object_class_id**: Objektklassen-ID
  - **1**: Benutzer
  - **2**: Gruppen
  - **3**: Organisationseinheit
  - **4**: Aktivitätsprofile
  - **5**: Datensätze
  - **6**: Repository-Felder
  - **7**: Documenttypen

- **8**: Documentklassen
- **9**: Berechtigungsprofile
- **10:** Sets
- **11:** Workflows

- **object_info**: Eigenschaftswert

Rückgabewerte:

- **0**: alles in Ordnung
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen

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

D3Interface d3 = getProperty("d3")

String property_name = "myProperty"
String object_id = "d3_groovy"
int object_class_id = 1
String object_info = "myValue"

def error = d3.call.object_property_set(property_name, object_id,
object_class_id, object_info)
if (error){
  d3.log.info("$error while setting object property")
} else {
  d3.log.info("Setting object property successful!")
}
```

d3set_add_filter

Beschreibung: Fügt einen Filter zu einer bestehenden Berechtigungsmenge hinzu. Wenn die Zuordnung für Berechtigungsmengen (set_name, object_id) nicht existiert, wird die Zuordnung erstellt.

Parameter:

- **user_name**: Name des d.3-Benutzers in dessen Namen die Änderung ausgeführt werden soll.
- **doc_id**: Dokument-ID des Berechtigungsmengen-Dokuments. Wenn die Dokument-ID festgelegt ist, dann können **set_name** und **object_id** leer bleiben.
- **set_name**: Name der Berechtigungsmenge.
- **object_id**: Benutzername oder Gruppen-ID. Wenn der Parameter leer ist, wird ein Set ohne Zuordnung gesucht.
- **filter**: Filterwerte, die dem Set hinzugefügt werden sollen. Sie können mehrere Filterwerte durch ein Semikolon (;) voneinander trennen. Die maximale Länge der Filter beträgt 150 Zeichen.

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen

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

D3Interface d3 = getProperty("d3")
```

```
String user_name = "d3_groovy"
String doc_id = ""
String set_name = "myRestrictionSet"
String object_id = "user01"
String filter = "myValue1;myValue2"
boolean overwrite = true

def error = d3.call.d3set_add_filter(user_name, doc_id, set_name,
object_id, filter, overwrite)
if (error){
  d3.log.info("$error while adding filter")
} else {
  d3.log.info("Adding filter successful!")
}
```

d3set_remove_filter

Beschreibung: Entfernt einen Filter aus einer bestehenden Berechtigungsmenge.

Parameter:

- **user_name**: Name des d.3-Benutzers in dessen Namen die Änderung ausgeführt werden soll.
- **doc_id**: Dokument-ID des Berechtigungsmengen-Dokuments. Wenn die Dokument-ID festgelegt ist, dann können **set_name** und **object_id** leer bleiben.
- **set_name**: Name der Berechtigungsmenge
- **object_id**: Benutzername oder Gruppen-ID. Wenn der Parameter leer ist, wird ein Set ohne Zuordnung gesucht.
- **filter**: Filterwerte, die dem Set hinzugefügt werden sollen. Sie können mehrere Filterwerte durch ein Semikolon (;) voneinander trennen. Die maximale Länge der Filter beträgt 150 Zeichen.

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen

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

D3Interface d3 = getProperty("d3")

String user_name = "d3_groovy"
String doc_id = ""
String set_name = "myRestrictionSet"
String object_id = "user01"
String filter = "myValue1"

def error = d3.call.d3set_remove_filter(user_name, doc_id, set_name,
object_id, filter)
if (error){
  d3.log.info("$error while removing filter")
} else {
  d3.log.info("Removing filter successful!")
}
```

d3set_remove_set

Beschreibung: Entfernt eine bestehende Berechtigungsmenge.

Parameter:

- **user_name**: Name des d.3-Benutzers in dessen Namen die Änderung ausgeführt werden soll.
- **doc_id**: Dokument-ID des Berechtigungsmengen-Dokuments. Wenn die Dokument-ID festgelegt ist, dann können **set_name** und **object_id** leer bleiben.
- **set_name**: Name des Berechtigungsmengen-Sets.
- **object_id**: Benutzername oder Gruppen-ID. Wenn der Parameter leer ist, wird ein Set ohne Zuordnung gesucht.

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen

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

D3Interface d3 = getProperty("d3")

String user_name = "d3_groovy"
String doc_id = ""
String set_name = "myRestrictionSet"
String object_id = "user01"

def error = d3.call.d3set_remove_set(user_name, doc_id, set_name, object_id)
if (error){
  d3.log.info("$error while removing set")
} else {
  d3.log.info("Removing set successful!")
}
```

d3async_job_open

Beschreibung: Öffnet einen neuen d.3 async-Job

Parameter:

- **doc_id_ref**: Dokument-ID des entsprechenden Dokuments
- **job_type**: Art des Jobs
- **user_name**: Name des d.3-Benutzers in dessen Namen die Änderung ausgeführt werden soll.

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen

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

D3Interface d3 = getProperty("d3")
```

```
String doc_id_ref = "P000000001"
String job_type = "CUJ001"
String user_name = "d3_groovy"

d3.call.d3async_job_open( doc_id_ref ,  job_type, user_name);
d3.call.d3async_job_set_attribute("hook_function", "myJplFunction", 0);
d3.call.d3async_job_set_attribute("custom_job_par[1]", "valueForParam1", 0);
d3.call.d3async_job_set_attribute("custom_job_par[2]", "valueForParam2", 0);
d3.call.d3async_job_set_attribute("custom_job_par[3]", "valueForParam3", 0);
d3.call.d3async_job_close();
```

d3async_job_set_attribute

Beschreibung: Legt die Parameter des zuvor geöffneten d.3 async-Auftrags fest.

Parameter:

- **attr_name**: Name des Attributs
- **attr_value**: Wert des Attributs
- **attr_type**: Typ des Attributs

Rückgabewerte:

- **0**: alles in Ordnung
- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen

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

D3Interface d3 = getProperty("d3")

String doc_id_ref = "P000000001"
String job_type = "CUJ001"
String user_name = "d3_groovy"

d3.call.d3async_job_open( doc_id_ref ,  job_type, user_name);
d3.call.d3async_job_set_attribute("hook_function", "myJplFunction", 0);
d3.call.d3async_job_set_attribute("custom_job_par[1]", "valueForParam1", 0);
d3.call.d3async_job_set_attribute("custom_job_par[2]", "valueForParam2", 0);
d3.call.d3async_job_set_attribute("custom_job_par[3]", "valueForParam3", 0);
d3.call.d3async_job_close();
```

d3async_job_set_lin002_attribute

```
currently not supported
```

d3async_job_close

Beschreibung: Schließt d.3 async-Jobs

Parameter: Keine

Rückgabewerte:

- **0**: alles in Ordnung

- **> 0**: Datenbankfehler
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen

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

D3Interface d3 = getProperty("d3")

String doc_id_ref = "P000000001"
String job_type = "CUJ001"
String user_name = "d3_groovy"

d3.call.d3async_job_open( doc_id_ref ,  job_type, user_name);
d3.call.d3async_job_set_attribute("hook_function", "myJplFunction", 0);
d3.call.d3async_job_set_attribute("custom_job_par[1]", "valueForParam1", 0);
d3.call.d3async_job_set_attribute("custom_job_par[2]", "valueForParam2", 0);
d3.call.d3async_job_set_attribute("custom_job_par[3]", "valueForParam3", 0);
d3.call.d3async_job_close();
```

send_email

Beschreibung: Diese Funktion ermöglicht das Senden von E-Mails.

Parameter:

- **recipient**: Empfänger der E-Mail (d.3-Benutzername oder d.3-Gruppenname)
- **notice**: Betreff
- **date**: Sendedatum (kann auch in der Zukunft liegen), optional
- **doc_id**: ID des Dokuments, das als Link gesendet werden soll, optional
- **user_name**: Absender der E-Mail (d.3-Benutzername)
- **body_file**: Name und Pfad der Datei, die den Text für den Body enthält
- **mail_format**: HTML-Format, sonst: Textformat (Standard)
- **attach**: Dokument als Anhang senden (**0** (Standard) oder **1**).
- **doc_status**: Version im Status doc_status anhängen.
- **archiv_index**: Index für die Archivversion, die angehängt werden soll.
- **attach_abh**: Liste der abhängigen Dateien (z.B. “T1,P1”)
- **attach_file**: Liste der Dateien, die angehängt werden sollen (z.B.“C:\file1.jpg,c:\file2.txt")

Rückgabewerte:

- **0**: alles in Ordnung
- **-1**: Panther-Syntaxfehler, für Details siehe d.3 logviewer
- **-2**: Funktion wurde falsch aufgerufen
- **-3**: Undefinierter Fehler, kontaktieren Sie den d.velop-Support.
- **-110**: Kein Empfänger angegeben
- **-201**: ungültiger Benutzername angegeben
- **-301**: ungültige Dokument-ID angegeben oder Benutzer hat keine Rechte darauf
- **-302**: Fehler bei der Ermittlung der Benutzerrechte
- **-303**: Benutzer hat kein Zugriffsrecht auf Dokument
- **-540**: SMTP_SUPPORT ist nicht aktiviert
- **-541**: Die SMTP DLL ist nicht verfügbar
- **-542**: Fehler beim Senden der E-Mail

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

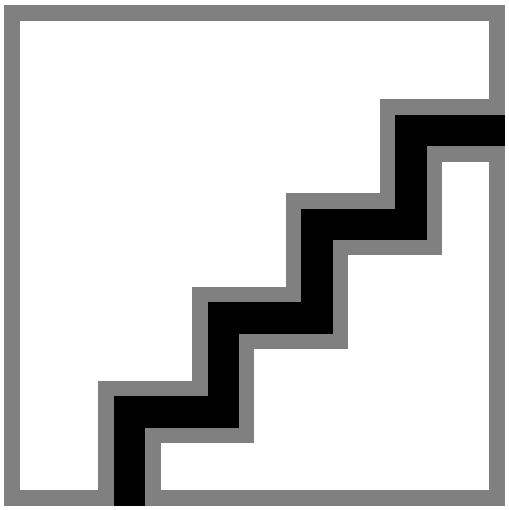
```
import java.sql.Timestamp;

D3Interface d3 = getProperty("d3")

String recipient = "dvelop"
String notice = "Please check this document"
Timestamp date = null
String doc_id = "P000000001"
String user_name = "d3_groovy"
String body_file = "<html>Please check this document</html>"
String mail_format = "html"
boolean attach = true
String doc_status = ""
int archiv_index = 0
String attach_abh = ""
String attach_file = ""

def error = d3.call.send_email (recipient, notice, date, doc_id, user_name,
body_file, mail_format, attach, doc_status, archiv_index, attach_abh,
attach_file)
if (error){
  d3.log.info(error + " while sending mail")
} else {
  d3.log.info("Sending mail successful! ->"+error)
}
```

1.9.5. 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.



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

```
// Getting the ID of the DBMS and writing it to the log file
String dbServer = d3.config.value("db_server");
d3.log.info( "Database: ${dbServer}");

// Get the first hostimport directory
String hostimpDir1 = d3.config.value("HOSTIMP_IMPORT_DIR", 1)
d3.log.info( "Hostimport directory: ${hostimpDir1}");

// Get the d.3 serverId of d.3 server process
String serverId = d3.config.value("d3fc_server_id")
d3.log.info("serverId " + serverId)

// Set variables depending on the serverId
String myTechnicalUser
if(serverId == "A"){
        myTechnicalUser = "techUser1"
}else if(serverId == "B"){
        myTechnicalUser = "techUser2"
} else {
        d3.log.error("Unknown server id " + serverId)
}
d3.log.info("Using user " + myTechnicalUser)
```

1.9.6. 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);
    }
```

> **Anmerkung**
> 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.

Beispiel

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

D3Interface d3 = getProperty("d3");

d3.log.critical("My critical message")
d3.log.error("My error message")
d3.log.warn("My warn message")
d3.log.info("My info message")
d3.log.debug("My debug message")

if(d3.log.isDebugEnabled()){
  d3.log.debug("My debug message")
} else {
```

```
  //no message
}

d3.log.message("My critical message", 0)
d3.log.message("My error message", 1)
d3.log.message("My start/stop message", 2)
d3.log.message("My warn message", 3)
d3.log.message("My warn message", 4)
d3.log.message("My warn message", 5)
d3.log.message("My info message", 6)
d3.log.message("My debug message", 7)
d3.log.message("My debug message", 8)
d3.log.message("My debug message", 9)
```

1.9.7. 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);
    }
```

1.9.8. 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
```

Beispiel

```
import com.dvelop.d3.server.core.D3Interface
import com.dvelop.d3.server.exceptions.D3Exception
import com.dvelop.d3.server.Document;
D3Interface d3 = getProperty("d3");

//D3Exception
try{
   Document tmpDoc = d3.archive.getDocument("P000000001",
"thisUserDoesNotExist")
} catch(D3Exception ex){
   d3.log.error('Error "Failed to find user" ' + ex.getMessage())
}

//General exception
try{
   def arr = new int[3];
   arr[5] = 5;
} catch(Exception ex){
   d3.log.error('Error "Index out of bounds" ' + ex.getMessage())
}
```

1.9.9. 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();
    }
```