DOKUMENTATION
PROXESS WEBSERVICE

Stand: PROXESS 10

Dokumentation vom 8. Juni 2020
PROXESS Webservice Version: 10.0.x

Inhaltsverzeichnis

1. Einleitung

Dieses Dokument beschreibt die Installation und Konfiguration von PROXESS Webservice (Version 11.0) und dokumentiert die Funktionen und Datentypen der Webservice-Schnittstelle (WSDL).

Die Dokumentation gliedert sich in je ein Kapitel für

- die Installation und Konfiguration (Kapitel 2),
- die Schnittstellenbeschreibung des *DocumentService* (Kapitel 3) und
- die Schnittstellenbeschreibung des *AdministrationService* (Kapitel 4).

Der überwiegende Teil dieses Dokumentes ist eine Funktionsreferenz. Daher ist es nicht zum Durchlesen geschrieben worden, sondern dient eher als Nachschlagewerk. Als *GettingStarted* für Entwickler von Client-Software eignet sich die Einleitung zu Kapitel 3 sowie der daran anschließende Abschnitt 3.1: *Grundlegende Konzepte und Aufbau*.

1.1. Wichtige Neuerungen in Version 11

1.1.1. Bereitstellung mehrerer Schnittstellen-Versionen über versionierte Endpunkte

PROXESS Webservice stellt ab Version 11 mehrere Endpunkte für unterschiedliche Versionen der Service-Schnittstelle bereit.

- Zukünftige Änderungen/Erweiterungen der Schnittstelle werden jeweils über neue Endpunkte zur Verfügung gestellt.
- Die WSDL-Definition eines einmal veröffentlichen Endpunktes wird nicht mehr verändert.
- Informationen zu den unterschiedlichen Versionen sind in Abschnitt Change-Log Schnittstellen-Versionen beschrieben.

Durch die Einführung der Versionierung haben die Endpunkt-URLs ab Version 11 folgende Form:

`http//[host]:[port]/proxess.svc/[version]/[endpoint_name]`


Der bisher verwendete Endpunkt
`http//[host]:[port]/akzentum/proxessws.svc/[endpoint_name]`
wird weiterhin unterstützt und ist auf die letzte vor PROXESS Webservice 11 veröffentlichte Schnittstellen-Version festgelegt:
`http//[host]:[port]/proxess.svc/v2/[endpoint_name]`

1.1.2. Ablösung der Chunked-Transfer Methoden zur Übertragung von Dateien und Suchergebnissen

Mit PROXESS Webservice 11 wird eine alternative API zur Übertragung von Dateien und Suchergebnissen bereitgestellt.

- Die neuen Funktionen verwenden weiterhin eine blockweise Übertragung, orientieren sich aber an der typischen API für Dateisysteme.
- Erläuterung am Beispiel Datei-Download:
  - Bisher wurde für den Datei-Download ausschließlich die Funktion `DownloadFile` verwendet.
  - Die Funktion musste ggf. mehrfach aufgerufen werden, wobei der Übertragungszustamd über bestimmte Parameter (`TransferStatus`, `ChunkID`) gesteuert wurde.
  - Die neue API stellt eigene Funktionen für die einzelnen Aktionen bereit, so dass der Download z.B. über folgende Aufrufsequenz durchgeführt wird: `FileOpen(...)`, `FileRead(...)`, `FileRead(...)`, `FileClose(...)`
- Die Funktionen `DownloadFile`, `UploadFile`, `UploadNewFile` und `Query` werden abgelöst.
  - Diese Funktionen sind weiterhin Teil der Schnittstelle und funktionieren weiter wie bisher.
  - Die Dokumentation der Funktionen und des Aufrufprotokolls (Zustandssteuerung über Parameter) wurde in den Anhang in dieser Dokumentation verschoben.

Die folgende Tabelle stellt die alten Funktionen ihren Ablösungen gegenüber:

Alte Funktion	Neue Funktionen	Beschreibung

`DownloadFile`	`FileOpen`
	`FileRead`
	`FileClose`
		Download archivierter Dateien

`UploadNewFile`	`FileCreate`
	`FileWrite`
	`FileClose`
		Upload einer neuen Datei.

`UploadFile`	`FileCreateNewVersion`
	`FileWrite`
	`FileClose`
		Upload einer neuen Dateiversion.

`Query`	`ResultOpen`
	`ResultRead`
	`ResultClose`
		Abrufen der Ergebnisliste einer Recherche.

Tabelle 1.1.: Ablösungsmatrix File- und Search-Methoden

1.2. Change-Log Schnittstellen-Versionen

Die folgende Tabelle führt die unterschiedlichen Versionen der Schnittstelle von PROXESS Webservice auf.

Service-Endpunkt	Webservice-Version	Datum	Beschreibung
proxess.svc/v3	ab 11.0	2020-06	- Neue Funktion: GetDocumentTypesFiltered - Neue API zur Übertragung von Dateien und Suchergebnissen
proxess.svc/v2	2.8002.6.2949 - 10.*	2017-09	- Erweiterung Datenmodell: FileDescriptor::LengthDb
proxess.svc/v1	< 2.8002.6.2949	2014-11	- Basisversion, Stand Webservice 2.2.0

Tabelle 1.2.: Versionshistorie Webservice-Schnittstelle

2. Installation und Konfiguration

Dieses Kapitel dokumentiert die Installation von PROXESS Webservice als Self-hosted Application und die Konfiguration der Service-Schnittstelle und der angebotenen Endpunkte. Andere Arten des Hosting (z.B. in IIS, siehe Hosting WCF Services[1]) werden in dieser Dokumentation nicht behandelt, aber der Konfigurations-Teil dieses Kapitels sollte auch in diesen Fällen anwendbar sein[2]. Wird PROXESS Webservice als Self-hosted Application ausgeführt, lautet die Basis-Adresse des Webservice `http://[host]:[port]/akzentum/proxessws.svc`.

Als Self-hosted Application besteht der Webservice selbst lediglich aus einer ausführbaren Datei (`wsphost.exe`) sowie einer Konfigurationsdatei (`wsphost.exe.config`). Die Installation (Abschnitt 2.2) läuft automatisiert ab; nur beim Update von älteren Version müssen evtl. mehrere Schritte von Hand durchgeführt werden.

Die Konfiguration von PROXESS Webservice (Abschnitt 2.3) besteht aus dem Anpassen der Datei `wsphost.exe.config` an die Ausführungsumgebung.

2.1. Voraussetzung

Dieser Abschnitt dokumentiert die Voraussetzungen, die für den Betrieb von PROXESS Webservice erfüllt sein müssen.

- PROXESS Webservice benötigt eine Verbindung zu einem PROXESS Server (mindestens PROXESS 10).
- PROXESS Webservice muss als Modul in der PROXESS Lizenz eingetragen sein.

2.2. Installation

PROXESS Webservice besteht im wesentlichen aus zwei Dateien:

- `wsphost.exe` – enthält den Webservice und kommuniziert über die `krn32.dll` mit PROXESS.
- `wsphost.exe.config` – enthält die spezifische Konfiguration der Service-Adresse und der Service-Endpunkte.

[1] Hosting Services `http://msdn.microsoft.com/en-us/library/ms730158.aspx`
[2] Auf Anfrage kann PROXESS Webservice auch als svc-Datei bereitgestellt werden.

PROXESS Webservice wird mit einem Windows-Installer-Paket[3] verteilt. Das Installerpaket kopiert die Webservice-Dateien in ein frei wählbares Verzeichnis (Standard: `C:\%PROGRAM FILES%\PROXESS Webservice`). Bei einer Aktualisierung des Webservice wird die Konfigurationsdatei `wsphost.exe.config` nicht überschrieben (bzw. bei einer Deinstallation **nicht** gelöscht!).

2.2.1. Ausführung von der Kommandozeile

PROXESS Webservice kann von der Kommandozeile gestartet werden. Beim Aufruf von `wsphost.exe` kann optional der Hostname und der Port angegeben werden. Im Folgenden werden einige Beispiele für gültige Aufrufe von `wsphost.exe` aufgeführt:

- `wsphost.exe` – Hostname und Port werden aus der Konfigurationsdatei gelesen.
- `wsphost.exe srv01` – Verwendet Host `srv01` und Standard-Port (`40400`).
- `wsphost.exe srv01:40401` – Verwendet Host `srv01` und Port `40401`.
- `wsphost.exe srv01 40401` – Verwendet Host `srv01` und Port `40401`.

Die Aufrufvariante ohne Parameter (alle Konfigurationsdaten werden aus der Konfigurationsdatei `wsphost.exe.config` gelesen) eignet sich für die Einrichtung von PROXESS Webservice als Windows-Dienst.

2.2.2. Einrichten als Windows-Dienst

PROXESS Webservice kann als Windows-Dienst registriert werden. Dazu muss der Windows-Kommandozeileninterpreter `cmd.exe` im Installationverzeichnis von PROXESS Webservice mit administrative Rechten ausgeführt werden.

- **Installation.** `wsphost.exe -install [optional Anzeigename]`
  Registriert einen Windows-Dienst mit dem Namen `wsphost` und dem angegebenen Anzeigenamen. Wird der Anzeigename nicht angegeben, wird standardmäßig der Anzeigename “PROXESS Webservice” verwendet. Enthält der übergebene Anzeigenamen Leerzeichen, so muss er in Anführungszeichen gefasst sein.
- **Deinstallation.** `wsphost.exe -uninstall`
  Entefernt den Windows-Dienst mit dem Namen `wsphost` aus der Liste der Windows-Dienste.

2.3. Konfiguration

Die Konfiguration von PROXESS Webservice besteht aus der Anpassung der Datei `wsphost.exe.config` an die Zielumgebung. Die Datei `wsphost.exe.config` ist eine XML-Datei die WCF-spezifische und für PROXESS Webservice

[3] Windows Installer `http://msdn.microsoft.com/en-us/library/cc185688(VS.85).aspx`

spezifische Konfigurationselemente enthält. Im Folgenden werden diejenigen Elemente aufgeführt, die angepasst werden müssen bzw. optional angepasst werden können.

- Zur Erreichbarkeit des Service sollte die `ServiceAddress` (Host-Adresse und Port) (Abschnitt 2.3.1).
- Ggf. ist eine Frieschaltung weiterer Service-Endpunkte (Abschnitt 2.3.3) erforderlich.
- Parameter für internes Verhalten (Abschnitt 2.3.4) oder das Logging (Abschnitt 2.3.5) müssen i.d.R. nicht geändert werden.

2.3.1. Konfiguration von Service-Host und -Port

Durch das Konfigurationelement `ServiceAddress` wird festgelegt, aus welchem Netzwerk (z.B. lokales Netzwerk oder Internet) der Webservice erreichbar ist. Die Service-Adresse wird in der Form

`[host]:[port]`

angegeben, wobei sowohl der Hostname als auch der Port optional sind. Wird der Port nicht angegeben, wird der standardmäßig der Port `40400` verwendet.

Der Standardwert für den Hostnamen ist der Windows-Netzwerkname (NetBIOS) des Host-Computers[4], d.h. wird weder der Hostname noch der Port angegeben, ist der Webservice im lokalen Netz unter Port `40400` erreichbar.

Listing 2.1 zeigt die möglichen Varianten in der entsprechenden Sektion in der Konfigurationsdatei[5].

```
1  <?xml version="1.0"?>
2  <configuration>
3    <!-- [...] -->
4    <!-- custom app settings for web-service -->
5    <appSettings>
6
7      <!-- Listens at www.letsbefast.de:40400 -->
8      <add key="ServiceAddress" value="www.letsbefast.de" />
9
10       <!-- Listens at www.letsbefast.de:40401 -->
11       <add key="ServiceAddress" value="www.letsbefast.de:40001" />
12
13       <!-- Listens at LBFSRV01:40400 (local network) -->
14       <!-- <add key="ServiceAddress" value="COMMENTED" /> -->
15
16     <!-- [...] -->
17   </appSettings>
18   <!-- [...] -->
19 </configuration>
```

Listing 2.1: Konfiguration `ServiceAddress`

[4] Das entspricht dem Wert der .NET Eigenschaft `Environment.MachineName` - `http://msdn.microsoft.com/en-us/library/system.environment.machinename.aspx`

[5] In jeder echten Konfigurationsdatei darf natürlich höchstens ein ServiceAddress-Element deklariert werden.

Zusammensetzung der Endpunkt-URLs. Der Teil `ServiceAddress` enthält wie oben beschrieben den Namen (bzw. die Adresse) des Host-Computers sowie den Port. Der übrige Teil der Service-Adresse setzt sich wie folgt zusammen:

`http:// [SeviceAddress] / proxess.svc / [Version] / [EndpointAddress]`

Die unterschiedlichen Schnittstellen-Versionen werden im folgenden Abschnitt beschrieben.

Die vorgegebenen Endpunkt-Adressen (`EndpointAddress`) werden in Abschnitt 2.3.3 beschrieben.


Bis Version 11.0 von PROXESS Webservice war der Teil `proxess.svc` des Endpunkt-URL frei konfigurierbar und standardmäßig auf `akzentum/proxessws.svc` festgelegt - und der URL enthielt keine Versionsnummer. Dieser Endpunkt wird aus Kompatibilitätsgründen weiterhin veröffentlich und basiert auf der letzten vor PROXESS 11.0 gültigen Interface-Version.
Beispiel für Legacy-Endpunkt: `http://localhost:40400/akzentum/proxessws.svc`

2.3.2. Konfiguration der Service-Kontrakte

Durch die Standard-Konfiguration des Webservice werden unterschiedliche Interface-Versionen des Webservice veröffentlicht.

Folgende Services sind in der Konfiguration enthalten:

Service-Name	Version	Service-Adresse
ProxessDocumentService_v3	ab 11.0.0	http://[host]:[port]/proxess.svc/v3
ProxessDocumentService_v2	2.8002.6.2949 - 10.*	http://[host]:[port]/proxess.svc/v2
ProxessDocumentService_v1	< 2.8802.6.2949	http://[host]:[port]/proxess.svc/v1
ProxessAdministrationService	alle Versionen	http://[host]:[port]/proxess.svc/admin/v1


Eine Anpassung der Service-Definitionen in der Konfigurationsdatei ist ab Version 11.0 nicht mehr erforderlich.


Die Bereitstellung älterer Interface-Versionen erlaubt es Clients, die gegen eine ältere Version der WSDL gebaut wurden, sich mit dem Webservice zu verbinden.
Dafür muss ggf. die Adresse, die bisher zum Verbindungaufbau getauscht werden:
`http://[host]:[port]/akzentum/proxessws.svc`
Clients, die vor Version 2.8002.6.2949 erstellt wurden, verbinden sich jetzt über
`http://[host]:[port]/proxess.svc/v1`.
Client, die von Version 2.8002.6.2949 - 10.* erstellt wurden, verbinden sich jetzt über
`http://[host]:[port]/proxess.svc/v2`.
Die „klassische" Service-Adresse `http://[host]:[port]/akzentum/proxessws.svc` ist für diesen Versionbereich weiterhin erreichbar.

Listing 2.2 zeigt einen exemplarischen Konfigurationsabschnitt für die aktuelle Version der Schnittstelle.

```
<?xml version="1.0"?>
<configuration>
  <!-- [...] -->
  <!-- WCF Service configuration -->
  <system.serviceModel>
       <!-- [...] -->

    <!-- defines service(s) and all endpoints -->
    <services>
      <service behaviorConfiguration="ProxessWebserviceBehavior"
           name="Akzentum.Proxess.Webservice.ServiceImplementation.ProxessDocumentService_v3">
        <!-- Export .wsdl -->
        <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />

        <!-- Secure endpoint with Windows authentication. -->
        <!-- <endpoint address="windows" name="wspWindows"
                binding="wsHttpBinding" bindingConfiguration="wspBindWindows"
          contract="Akzentum.Proxess.Webservice.ServiceContracts.IProxessDocumentService_v3" /> -->

        <!-- Secure endpoint using PROXESS Username/Password authentication. -->
        <!-- <endpoint address="secure" name="wspSecure"
                binding="wsHttpBinding" bindingConfiguration="wspBindSecure"
          contract="Akzentum.Proxess.Webservice.ServiceContracts.IProxessDocumentService_v3" /> -->

        <!-- Insecure endpoint. DISABLED BY DEFAULT.-->
        <!-- <endpoint address="simple" name="wspSimple"
                binding="basicHttpBinding" bindingConfiguration="wspBindSimple"
          contract="Akzentum.Proxess.Webservice.ServiceContracts.IProxessDocumentService_v3" /> -->
      </service>


    </services>
```

Listing 2.2: Konfiguration des Service-Kontrakts und der Service-Endpunkte

2.3.3. Konfiguration der Endpunkte

Die Konfiguration der Endpunkte besteht in der Freischaltung von einzelnen vorgegebenen Endpunkten, die in Listing 2.2 als XML-Elemente unter `configuration/system.servicemodel/services/service/endpoint` aufgelistet sind.

Das Aktivieren (bzw. Deaktivieren) eines Endpunktes geschiet durch Entfernen (bzw. Hinzufügen) eines Kommentars um die Endpunkt-Definition. Standardmäßig sind lediglich die beiden Endpunkte mit den Adressen `mex` und `windows` aktiviert. Im Folgenden werden die einzelnen Endpunkte aufgeführt und ihre Eigenschaften beschrieben.

Endpunkt `mex`. Der Endpunkt mit der Adresse `mex` ist standardmäßig aktiviert und dient zur Veröffentlichung der Metadaten des Service, d.h. der WSDL-Datei. Soll die WSDL-Datei nicht öffentlich abrufbar sein, kann dieser Endpunkt deaktiviert werden.

Enpunkt `windows`. Der Endpunkt mit der Adresse `windows` ist standardmäßig aktiviert und verwendet die in WCF integrierte Windows-Authentifizierung (ActiveDirectory) zur Authentifizierung von Clients.

- Die Kommunikation über diesen Endpunkt ist gesichert (Signiert und Verschlüsselt) und erfordert keine weitere Konfiguration.
- (Nach jetzigem Erfahrungen) nur WCF-WCF, d.h. Clients müssen ebenfalls WCF verwenden.
- PROXESS 5+ ActiveDirectory Authentication kann verwendet werden.

Endpunkt `secure`. Der Endpunkt mit der Adresse `secure` ist standardmäßig deaktiviert und verwendet ein Service-Zertifikat zur Sicherung der Kommunikation.

- Die Kommunikation über diesen Enpunkt ist gesichert (Signiert und Verschlüsselt) und erfordert eine Server- und Client-seitige Einrichtung des Service-Zertifikates (siehe 2.3.3.1).
- Interoperabel mit dem Java Webservice-Stack Metro[6].
- Unterstützt lediglich PROXESS Benutzername/Passwort Authentifizierung.

Endpunkt `simple`. Der Endpunkt mit der Adresse `simple` ist standardmäßig deaktiviert und verwendet keine Sicherheit und ist daher **nicht für Internet-Anwendungsszenarien geeignet**.

- Die Kommunikation über diesen Endpunkt wird in keiner Weise gesichert. Z.B. werden Passwörter in Klartext übertragen.
- Maximale Interoperabilität.
- Unterstützt lediglich PROXESS Benutzername/Passwort Authentifizierung.

[6] Metro `http://metro.java.net/`

2.3.3.1. Konfiguration eines Service-Zertifikates

Wenn der Endpunkt `secure` aktiviert ist, starten der Webservice nur dann, wenn ein Service-Zertifikat konfiguriert ist.

Abbildung 2.3 zeigt diejenige Sektion in der Konfigurationdatei, in der ein Service-Zertifikat registriert werden kann (in der Abbildung auskommentiert, d.h. es wird kein Zertifikat konfiguriert).

```
<?xml version="1.0"?>
<configuration>
  <!-- [...] -->
  <!-- WCF Service configuration -->
  <system.serviceModel>
    <!-- [...] -->
    <behaviors>
      <serviceBehaviors>
        <behavior name="ProxessWebserviceBehavior">
          <serviceDebug includeExceptionDetailInFaults="true" />
          <serviceMetadata httpGetEnabled="true" />
          <!-- serviceCredentials must be specified if the 'secure' is enabled. -->
          <!-- Usually the CN of the certificate must match the DNS name of the service host, e.g.
          www.myproxessservice.com -->
          <!--
          <serviceCredentials>
            <serviceCertificate findValue="CommonNameOfCertificate" x509FindType="FindBySubjectName" />
          </serviceCredentials>
          -->
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <!-- [...] -->
  </system.serviceModel>
  <!-- [...] -->
</configuration>
```

Listing 2.3: Konfiguration des Service-Zertifikates

- Für das Attribut `findValue` (in der Abb. `findValue="CommonNameOfCertificate"`) muss der Wert des Common Name (CN) des Zertifikates eingetragen werden.
- Das Zertifikat (inkl. privater Schlüssel) muss im `Personal`-Store (`Eigene Zertifikate`) des lokalen Computeraccounts installiert sein.
- WCF-Clients suchen standardmäßig nach einem Zertifikat (nur öffentlicher Schlüssel) im `Personal`-Store (`Eigene Zertifikate`) des aktuellen Benutzeraccounts mit einem Common Name (CN), der dem Host-Namen des Webservice entspricht (z.B. `192.168.1.10` oder `prxsrv01` oder `www.letsrocktheboat.com`). Daher ist es zu empfehlen, für den Webservice ein Zertifikat zu konfigurieren, das den gleichen CN wie den in der `ServiceAddress` konfigurierten Host-Namen besitzt.

Zur Erstellung von temporären Zertifikaten für die Entwicklung bietet MSDN einige hilfreiche Artikel[7][8][9] und Tools[10][11].

2.3.4. Konfiguration für internes Verhalten

Listing 2.4 zeigt weitere Webservice-spezifische Konfigurationselemente.

Definition von PROXESS Server Aliasen. Mit Aliasen kann der Adresse eines PROXESS-Servers, der über den Webservice erreicht werden kann, ein kurzer Name gegeben werden. Ohne Konfiguration vorgegeben ist der Alias `Default` als `tcp://localhost`, was für die Fälle, in denen PROXESS auf dem selbsen System wie der Webservice installiert ist, eine Verbindung mit eben diesem PROXESS herstellt.

Clients spezifizieren einen Alias in der Login-Funktion, in dem sie für den PROXESS-Servernamen den Aliasnamen mit einem führenden `@`-Zeichen angeben (z.B. `@Default`). Ist der Parameter für den Servernamen beim Login `null`, wird automatisch der Alias `Default` verwendet.

Konfiguration der Kapazität des DocumentAccessorPool. Der Wert `DAccMaximumPoolSize` legt die maximale Anzahl gleichzeitiger Sessions für diese Webservice-Instanz fest. Ein negativer Wert bedeutet: keine Begrenzung. Standardwert: `-1`.

Konfiguration der Mindest-Lebensdauer einer Session. Der Wert `DAccPoolGuaranteedSessionLifetime` legt fest, wie groß das Keep-Alive-Intervall jeder einzelnen Session zwischen zwei Aufrufen ist (in Sekunden) . Der Standardwert ist `30`. Damit wird das Verhalten in Lastsituationen gesteuert. Ein kleiner Wert bevorzugt neue Clients, die noch keine Verbindung haben (d.h. bestehende Client-Verbindungen werden bei Bedarf früher abgeräumt). Ein großer Wert bevorzugt Clients, die bereits eine Verbindung haben (d.h. neue Clients bekommen häufiger einer Fehlermeldung beim Versuch der Verbindung).

Verwendung der Komponente PROXESS Web Rendr Der Wert `EnablePROXESSWebRendr` legt fest, ob die Komponente PROXESS Web Rendr dazu verwendet wird, bestimmte Dateien (vorrangig COLD-Dokumente) in eine PDF-Datei zu rendern. Diese Funktion ist seit PROXESS Webservice Version 2.1.0 verfügbar. Standardmäßig ist die Komponente PROXESS Web Rendr aktiviert. Durch Setzen des Wertes auf `false` wird die Komponente PROXESS Web Rendr deaktiviert und alle Dateien werden direkt von PROXESS heruntergeladen (damit wird das Verhalten aller Vorgänger-Versionen erreicht).

[7] Securing Services and Clients `http://msdn.microsoft.com/en-us/library/ms734736.aspx`
[8] How to: Create and Install Temporary Certificates in WCF for Message Security During Development `http://msdn.microsoft.com/en-us/library/ff647171.aspx`
[9] How to: Make X.509 Certificates Accessible to WCF `http://msdn.microsoft.com/en-us/library/aa702621.aspx`
[10] MSDN Certificate Creation Tool (`makecert.exe`) `http://msdn.microsoft.com/en-us/library/windows/desktop/aa386968(v=vs.85).aspx`
[11] PVK Digital Certificate Files Importer (`pvkimprt.exe`) `http://www.microsoft.com/en-us/download/details.aspx?id=6563`

```
<?xml version="1.0"?>
<configuration>
  <!-- [...] -->
  <!-- custom app settings for web-service -->
  <appSettings>
    <!-- Listen address for PROXESS Webservice -->
    <add key="ServiceAddress" value="localhost" />
    <!-- Defines the PROXESS server with the alias Default. The alias can be used as a
           replacement for the PROXESS Server with a leading '@'. -->
    <!-- E.g.: LogIn("@Default", "user", "pass")
           => LogIn("tcp://localhost", "user", "pass") -->
    <add key="Alias:Default" value="tcp://PRXSRV"/>
       <!-- define alias Test -->
       <add key="Alias:Test" value="tcp://VM-TESTSRV"/>
    <!-- Maximum pooled connections. The effective value is determined by
           Min( Licensed Sessions for PROXESS Webservice , DAccMaximumPoolSize ). -->
    <!-- A negative value means 'unlimited'. -->
    <add key="DAccMaximumPoolSize" value="-1" />
    <!-- Minimum lifetime of a session in seconds before it can be evicted
           automatically. A negative value means 'unlimited'. -->
    <add key="DAccPoolGuaranteedSessionLifetime" value="30"/>
       <!-- Specifies if the module PROXESS Web Rendr should be used to render files.
       By default, PROXESS Web Rendr is enabled. -->
    <add key="EnablePROXESSWebRendr" value="true"/>
  </appSettings>
  <!-- [...] -->
</configuration>
```

Listing 2.4: Weitere Konfigurationselemente

2.3.5. Konfiguration des Loggings

Konfiguration von `log4net`. Die Konfiguration für das Logging-Framework `log4net` befindet sich am Ende der Konfigurationdatei in der Sektion `configuration/log4net`. Standardmäßig sind ein Consolen- und ein Datei-Appender für `INFO`-Logs und ein EventLog-Appender für `WARN`-Logs konfiguriert.

Trace-Logging von XML-Nachrichten. Außerhalb der `log4net`-Konfiguration kann ein WCF Message Logging konfiguriert werden. Dazu müssen zwei Konfigurations-Sektionen aktiviert werden, die in den Listings 2.5 und 2.6 aufgeführt werden.

```
<?xml version="1.0"?>
<configuration>
  <!-- [...] -->
  <!-- WCF Service configuration -->
  <system.serviceModel>
    <!-- configure trace logging of XML/SOAP messages -->
    <diagnostics>
      <messageLogging logEntireMessage="true"
```

```
                    logMalformedMessages="true"
                    logMessagesAtServiceLevel="true"
                    logMessagesAtTransportLevel="true"
                    maxMessagesToLog="3000"
                    maxSizeOfMessageToLog="32768"/>
    </diagnostics>
        <!-- [...] -->
  </system.serviceModel>
  <!-- [...] -->
</configuration>
```

Listing 2.5: Konfiguration Message Trace Logging

```
<?xml version="1.0"?>
<configuration>
  <!-- [...] -->
  <system.diagnostics>
    <sources>
      <source name ="System.ServiceModel.MessageLogging"
              switchValue="Information, ActivityTracing">
        <listeners>
          <add name="messages"
               type="System.Diagnostics.XmlWriterTraceListener"
               initializeData="msgtrace.svclog"/>
        </listeners>
      </source>
    </sources>
    <trace autoflush="true" />
  </system.diagnostics>
  <!-- [...] -->
 </configuration>
```

Listing 2.6: Konfiguration Message Trace Logging (2)

3. Schnittstellenbeschreibung

Dieses Kapitel beschreibt die primäre Schnittstelle von PROXESS Webservice, den *DocumentService*. Die Beschreibung gliedert sich in drei Teile.

- Abschnitt 3.1 beschreibt den grundlegenden Aufbau und Konzepte der Schnittstelle. Dieser Abschnitt geht auf Eigenschaften und Besonderheiten bei der Verwendung des Webservice ein und nicht, wie die folgenden Abschnitte, auf Operationen oder Datentypen der Webservice-Schnittstelle.
- Abschnitt 3.2 beschreibt die Operationen (bzw. Funktionen, Methoden) des Webservice sowie deren Parameter und Rückgabewerte. Im Allgemeinen gibt es zu jeder Webservice-Operation einen spezifischen Request-Datentyp, der den Eingabeparameter für die Operation darstellt, sowie einen spezifischen Response-Datentyp, der den Rückgabewert der Operation darstellt. Die einzelnen Operationen und deren Request- und Response-Datentypen werden mit Verweisen auf die WSDL-Datei des Webservice beschrieben.
- Abschnitt 3.3 beschreibt alle Datentypen, die im Webservice verwendet werden. Darunter befinden sich sowohl die Datentypen, die vom Webservice in Response-Objekten zurückgegeben werden (z.B. Dokumente und Metadatenbeschreibungen) als auch die Datentypen, die vom Anwender in Request-Objekten an den Webservice geschickt werden (z.B. Suchanfragen und Dokumentänderungen). Im Regelfall[1] sind Eingabe- und Ausgabedatentypen bewusst voneinander getrennt, wobei die Ausgabedatentypen mit dem Ziel entworfen wurden, das Datenschema von PROXESS genau wiederzuspiegeln. Die Eingabetypen sind möglichst einfach aber dennoch genau gehalten.

Zusammenfassend erläutert Abschnitt 3.1 die Funktionsweise des Webservice und die Abschnitte 3.2 und 3.3 stellen eine Dokumentation der WSDL-Datei dar.

[1] Eine Ausnahme stellt der Datentyp `TransferStatus` (B.2.0.1) bei Chunked-Transfers (B.1) dar.

3.1. Grundlegende Konzepte und Aufbau

Dieser Abschnitt führt in die grundlegenden Konzepte und den Aufbau der Schnittstelle von PROXESS Webservice ein.

PROXESS Webservice ist ein Webservice, der WSDL zur Beschreibung seiner Operationen und deren Parameter und Rückgaben verwendet, und SOAP als Kommunikationsprotokoll einsetzt. Daraus ergeben sich einige Eigenschaften der Schnittstelle, die in den folgenden Abschnitten beschrieben werden.

3.1.1. Trennung von Daten und Funktionalität

Alle Operationen des Webservice werden über das Interface `ProxessDocumentService` bereitgestellt. Die Menge von Funktionen dieses Interface (z.Zt. sind das 15 Funktionen) werden in Abschnitt 3.2 dokumentiert.

Die Datenobjekte, die der Service zurückgibt (z.B. `Document`, `DocumentType` oder `SearchResult`) sind reine Datenobjekte, die keine weitere Funktion erfüllen, als die Daten dem Benutzer in einer strukturierten Form zu präsentieren. Alle Objekte können zwar beliebig manipuliert werden (jedes Datenfeld hat öffentlichen Lese- und Schreibzugriff), das hat aber keinerlei Auswirkung auf die in PROXESS gespeicherten Daten.

Die einzige Möglichkeit, PROXESS-Daten zu manipulieren besteht darin, eine passende Interface-Methode des Service zu verwenden. Durch die Funktionen der Schnittstelle werden folgende Aktionen ermöglicht:

- Abrufen von Metadaten (Benutzer und Gruppen, Datenbanken, Dokumenttypen, Dateitypen, Feldbeschreibungen und Validierungsregeln)
- Recherche / Abrufen von Dokumenten
- Anlage, Änderung und Löschung von Dokumenten
- Download, Upload und Löschung von Dateien

Es gibt keine Möglichkeit, mit den Funktionen des *DocumentService* PROXESS Metadaten zu ändern (das sind z.B. Dokumenttypen, Datenbanken, Felddefinitionen, Benutzer, Gruppen, etc.). Diese Funktionalität wird durch dem *AdministrationService* (4) bereitgestellt.

3.1.2. Zustandslosigkeit

PROXESS Webservice soll, soweit das möglich ist, zustandslos sein. Damit ist gemeint, dass auf das Konzept von Object-Handles, wie es bei der API vom PROXESS Document Manager verwendet wird, weitgehend verzichtet wird. So besteht z.B. das Erstellen eines Dokumentes und das Setzen seiner Feldwerte aus einem einzigen Webservice Aufruf (siehe Funktion 3.2.4.2: `CreateDocument`). Mit der klassischen PROXESS Api läuft dieser Vorgang (schematisch) etwa so ab: `CreateDocument()` –> `OpenDocument()` –> `SetDocumentFields()` –> `SaveDocument()` –> `CloseDocument()`.

Das Entwurfsziel der Webservice-Schnittstelle ist es, mit einem einzelnen Aufruf einer Webservice-Funktion eine abgeschlossene Änderung am System vorzunehmen (bzw. eine komplette Abfrage an das System zu formulieren).

Es gibt zwei Bereiche, in denen ein Zustand zwischen Client und Service kommuniziert werden muss:

- Es gibt ein Objekt, dass die PROXESS Session (bzw. den PROXESS Login, das `login_handle`) identifiziert. Dieses Objekt heisst `LoginToken` (siehe 3.3.1.2) und muss bei jedem Web-Service-Aufruf als Parameter übergeben werden. Daher ist vor der Verwendung von PROXESS mit dem Webservice ein Aufruf der `LogIn`-Methode (siehe Funktion 3.2.1.2) nötig, um eben dieses LoginToken zu erhalten.
- Der Chunked-Transfer-Mechanismus (siehe B.1) erfordert ebenfalls einen Zustand. Davon sind folgende Funktionen betroffen:
  - Upload einer Datei (siehe Funktion B.2.0.5: `UploadFile`)
  - Download einer Datei (siehe Funktion B.2.0.4: `DownloadFile`)
  - Abholen von Trefferlisten (siehe Funktion B.2.0.2: `Query`)

Der Client muss also zum einen das `LoginToken` verwalten, mit dem die Session in PROXESS identifiziert wird, und zum anderen bei Suchergebnissen, Datei-Downloads und -Uploads darauf achten, den aktuellen Transfer korrekt zu identifizieren.

Die übrigen Funktionen, wie z.B. das Abrufen von Metadaten und das Anlegen, Ändern und Löschen von Dokumenten, lassen sich mit einem einzelnen Aufruf einer Webservice-Funktion releasieren.

3.1.3. Request-/Response-Methodik

Die Webservice Funktionen sind in den meisten Fällen so entworfen, dass sie eine einheitlich Signatur haben:

```
<FunctionXYZ>Response <FunctionXYZ>( <FunctionXYZ>Request request );
```

Jede Interface-Function erwartet ein spezifisches Request-Objekt, in dem immer alle Funktionsparameter gekapselt sind und gibt immer ein spezifisches Response-Objekt zurück das alle Rückgabedaten abkapselt. Der Typname des Requests bzw. der Response leitet sich aus dem Funktionsnamen ab.

Es gibt folgende Ausnahmen von dieser Regel:

1. Funktionen, die keinen Parameter oder keinen Rückgabewert haben. Diese Funktionen erwarten ein Objekt vom Typ `EmptyMessage` (siehe 3.2) bzw. geben ein solches Objekt zurück. Z.B. Funktion 3.2.1.1: `TestWSConnection` und Funktion 3.2.1.3: `LogOut`.
2. Die Funktion B.2.0.2: `Query` hat einen Eingabeparameter vom Typ `SearchRequest` und gibt ein Objekt des Typs `SearchResponse` zurück.
3. Bei Funktionen zum Download (Funktion B.2.0.4: `DownloadFile`) und Upload (siehe Funktion B.2.0.5: `UploadFile`) heißen die Request- und Responsetypen `FileDownloadRequest` und

`FileDownloadResponse` bzw. `FileUploadRequest` und `FileUploadResponse` (d.h. die Worte `File` und `Download` bzw. `Upload` sind vertauscht).

4. Die Funktion B.2.0.6: `UploadNewFile`hat einen Eingabeparameter vom Typ `NewFileUploadRequest`, gibt aber ein Objekt vom Typ `FileUploadResponse` zurück (und nicht `NewFileUploadResponse`).

Die übrigen Funktionen halten das obige Schema genau ein, verdeutlicht am Beispiel der Funktion 3.2.4.1: `GetDocument`: `GetDocument` hat einen Eingabeparameter vom Typ `GetDocumentRequest` und gibt als Antwort ein Objekt vom Typ `GetDocumentResponse` zurück.

3.1.4. Fehlerbehandlung

Fehlersituationen werden mit SOAP Fault Elements[2] kommuniziert.

- Fehler, die vom Webservice ausgleöst werden, sind daran zu erkennen, dass die Fehlermeldung mit dem Namen der Fehlerquelle beginnt, d.h. die Fehlermeldung beginnt immer mit `PROXESS Document Manager`, `PROXESS DMS API` oder `PROXESS Web Service`.
- Exceptions vom Webservice-Stack des Clients deuten entweder auf eine Fehlkonfiguration oder auf einen Verbindungsabbruch hin.
- Löst der Aufruf einer Webservice-Funktion keine Ausnahme/SOAP Fault aus, ist davon auszugehen, dass die Ausführung auch keine Fehler verursacht hat, d.h. dass alle Aktionen erfolgreich durchgeführt worden sind.

Details zu allen Webservice-Exceptions werden einheitlich mit einem `ProxessWebserviceFaultDetail`-Objekt beschrieben, das genauere Informationen zu der Fehlerquelle und Fehlercodes enthält.
Listing 3.1 zeigt die WSDL-Definition des Datentyps `ProxessWebserviceFaultDetail`.

```
1  <xs:complexType name="DMSFaultContract">
2   <xs:sequence>
3    <xs:element minOccurs="0" name="Message" nillable="true" type="xs:string"/>
4    <xs:element minOccurs="0" name="IsDocumentManagerFault" type="xs:boolean"/>
5    <xs:element minOccurs="0" name="ExceptionCode" type="xs:unsignedInt"/>
6    <xs:element minOccurs="0" name="ExtendedExceptionCode" type="xs:unsignedInt"/>
7    <xs:element minOccurs="0" name="Title" nillable="true" type="xs:string"/>
8    <xs:element minOccurs="0" name="ExceptionText" nillable="true" type="xs:string"/>
9    <xs:element minOccurs="0" name="ExtendedExceptionText" nillable="true" type="xs:string"/>
10   <xs:element minOccurs="0" name="ExceptionSource" type="xs:int"/>
11  </xs:sequence>
12 </xs:complexType>
```

Listing 3.1: Datentyp ProxessWebserviceFaultDetail

[2] http://www.w3schools.com/soap/soap_fault.asp

Achtung. Generell werden Fehler vom PROESS Document Manager direkt an den Client weitergereicht, d.h. es wird nicht versucht

1. Fehlersituationen automatisch zu behebn oder
2. Unmittelbar vor einem Fehler durchgeführte Änderungen rückgängig zu machen.

Im *DocumentService* sind die Funktionen zur Anlage und zur Bearbeitung von Dokumenten (`CreateDocument` und `ChangeDocument`) von diesem Verhalten betroffen. Im *AdministrationService* sind die meisten Funktionen zur Anlage oder Bearbeitung von Daten (`Create...` bzw. `Update...`) von diesem Verhalten betroffen. Die Ausführungsreihenfolge der einzelnen Arbeitsschritte werden pro Funktion dokumentiert. Anhand der Fehlermeldung lässt sich somit darauf schließen, an welcher Stelle ein Funktionsaufruf fehlgeschlagen ist und damit, welche Aktionen bereits durchgeführt worden sind.

3.2. Nachrichten und Operationen

Dieser Abschnitt beschreibt die Operation von PROXESS Webservice sowie die zugehörigen Request- und Response-Datentypen.

Die Webservice-Operationen werden in 5 verschiedene Bereiche aufgeteilt:

1. **Verbindungs-Funktionen** (Abschnitt 3.2.1)**.** Enthält die Funktionen zum Aufbauen und Abbauen von PROXESS Verbindungen.

2. **Metadaten-Funktionen** (Abschnitt 3.2.2)**.** Diese Kategorie enthält Funktionen zum Abrufen von Metadaten. Z.Zt. werden die PROXESS Metadaten-Tabellen für Datenbanken, Dokumenttypen, Dateitypen, Benutzer und Gruppen) unterstützt.

3. **Such-Funktionen** (Abschnitt 3.2.3)**.** Diese Kategorie enthält z.Zt. nur eine Funktion, die die Funktionalität der erweiterten Suche von PROXESS (`GeneralSearch`) realisiert. Die erweiterte Suche ist die flexibelste (aber auch komplizierteste) Suchmöglichkeit, die von PROXESS angeboten wird.

4. **Dokument-Funktionen** (Abschnitt 3.2.4)**.** Diese Kategorie enthält Funktionen zum Anlegen, Abrufen, Ändern und Löschen von Dokumenten. Mit diesen Funktionen lassen sich auch Feldwerte von Dokumenten ändern und Querverweise hinzufügen/entfernen.

5. **Dateitransfer-Funktionen** (Abschnitt 3.2.5)**.** Diese Kategorie enthäkt die Funktionen, mit denen der Download bzw. Upload von Dateien realisiert wird.

In diesem Abschnitt werden vorrangig die Operationen beschrieben. Die in den Request- und Response-Objekten enthaltenen komplexen Datentypen werden im folgenden Abschnitt 3.3 beschrieben.

Jede Operation wird in einer C-typischen Syntax für Deklarationen von Funktionen beschrieben, z.B. ist `EmptyMessage TestWSConnection( EmptyMessage )` so zu lesen: Der Webservice definiert in seiner WSDL-Beschreibung eine Operation mit dem Namen `TestWSConnection`, mit einem `wsdl:input` (Eingabeparameter) vom Typ `EmptyMessage` und einem `wsdl:output` (Rückgabewert) vom Typ `EmptyMessage`. Der entsprechende (um Namespaces gekürzter) Auszug aus der WSDL-Datei wird in Listing 3.2 gezeigt.

Jede Operation des PROXESS Webservice beschreibt ebenfalls eine `wsdl:fault`, was mit einer zusätzlichen `throws`-Deklaration beschrieben werden könnte. Darauf wird hier aber verzichtet, da sich die Fehlerbehandlung für alle Operationen gleich gestaltet.

```
1 <wsdl:operation name="TestWSConnection">
2  <wsdl:input wsaw:Action="http://.../TestWSConnection"
3    name="EmptyMessage" message="tns:EmptyMessage"/>
4  <wsdl:output wsaw:Action="http://.../TestWSConnectionResponse"
5    name="EmptyMessage" message="tns:EmptyMessage"/>
6  <wsdl:fault wsaw:Action="http://.../TestWSConnectionProxessWebserviceFault"
7    name="ProxessWebserviceFault" message="tns:..._ProxessWebserviceFault_FaultMessage"/>
8 </wsdl:operation>
```

Listing 3.2: Beispiel für eine WSDL-Operation

Die Eingabe- (`wsdl:input`) bzw. Ausgabedatentypen (`wsdl:output`) werden an der Stelle ihres ersten Auftretens beschrieben. Das bedeutet i.d.R., dass eine Webservice-Operation zusammen mit allen ihren Ausgabe- und Eingabeparametern beschrieben wird.

Die Beschreibung des Datentyps `EmptyMessage` wird an dieser Stelle schon vorweggenommen: Bei `EmptyMessage` handelt es sich um einen XML-Datentyp, der für Operationen ohne Eingabeparameter oder Rückgabe verwendet wird.

Nachrichtentyp `EmptyMessage`. Listing 3.3 zeigt die WSDL-Definition des Datentyps `EmptyMessage`.

```
1 <xs:element name="EmptyMessage">
2  <xs:complexType>
3   <xs:sequence/>
4  </xs:complexType>
5 </xs:element>
```

Listing 3.3: Nachrichtentyp EmptyMessage

Anmerkung. Die meisten Elemente des XML-Schemas der WSDL-Datei haben die Kardinalität `0..1` (d.h. `minOccurs=''0''` und `maxOccurs` ist implizit `1`) und sind damit als optionale Elemente definiert. Im Regelfall sind die in der WSDL beschriebenen Elemente aber **nicht optional**. Außerdem sind die meisten Elemente als nullierbar (`nilable`) definiert, der Webservice weist einigen dieser Elemente aber niemals den Wert `null` zu.

Diese Dokumentation versucht auf alle diejenigen Fälle hinzuweisen, in denen XML-Elemente tatsächlich optional oder nullierbar sind. **Generell gilt für Responses**, die vom Webservice empfangen wurden, **dass alle XML-Element vorhanden sind** (in bestimmten Fällen können Elemente aber den Wert `null` haben).

3.2.1. Verbindungsaufbau und -abbau

3.2.1.1. TestWSConnection

Signatur:

```
EmptyMessage TestWSConnection(
      EmptyMessage request );
```

Diese Methode hat keine Parameter oder Rückgabe und führt keine Aktion aus. Wenn sie aufgerufen werden kann, ohne dass ein SOAP Fault ausgelöst wird, ist der Webservice und der aufrufende Client richtig konfiguriert.

3.2.1.2. LogIn

Signatur:

```
LoginRespone LogIn(
      LoginRequest request );
```

Die Funktion `LogIn` wird dazu verwendet, um ein `LoginToken` (siehe 3.3.1.2) zur Identifikation der PROXESS Session zu erhalten. Im Hintergrund wird eine "konventionelle" Anmeldung an PROXESS durchgeführt und der Webservice bindet im Erfolgsfall das `login_handle` an das zurückgegebene `LoginToken`.

Nachrichtentyp `LoginRequest`. Listing 3.4 zeigt die WSDL-Definition des Datentyps `LoginRequest`.

```
<xs:element name="LoginRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Adresse oder Name des Computers auf dem der PROXESS Document Manager läuft.
        Es ist wichtig, dass der Name aus Sicht des Webservice angegeben wird,
        wenn sich Client und Web-Service in unterschiedlichen Netzwerken befinden. -->
   <xs:element minOccurs="0" name="PROXESSHost" nillable="true" type="xs:string"/>
   <!-- Spezifiziert die Art der PROXESS Authentifizierung. Z.Zt. werden folgende Arten
        der Anmeldung unterstützt:
          * Proxess - klassische Anmeldung mit Benutzername/Passwort
          * Windows - ActiveDirectory Authentifizierung (nur .NET Clients) -->
   <xs:element minOccurs="0" name="Authentication" type="q2:AuthenticationType" />
   <!-- PROXESS Benutzername (nur für PROXESS-Authentifizierung nötig) -->
   <xs:element minOccurs="0" name="Username" nillable="true" type="xs:string"/>
   <!-- Passwort des PROXESS Benutzers (nur für PROXESS-Authentifizierung nötig) -->
   <xs:element minOccurs="0" name="Password" nillable="true" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.4: Nachrichtentyp LoginRequest

Die folgende Tabelle führt Verweise zu allen in Listing 3.4 referenzierten Datentypen auf.

Feldname	Verweis
`AuthenticationType`	3.3.1.1

Nachrichtentyp `LoginResponse`. Listing 3.5 zeigt die WSDL-Definition des Datentyps `LoginResponse`.

```
<xs:element name="LoginResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Repräsentiert die PROXESS Session des Webservice Clients.
       Wird in allen Folgeaufrufen benötigt. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q4:LoginToken"/>
   <!-- Der vollständige Name des angemeldeten Benutzers. -->
   <xs:element minOccurs="0" name="FullName" nillable="true" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.5: Nachrichtentyp LoginResponse

Die folgende Tabelle führt Verweise zu allen in Listing 3.5 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2

3.2.1.3. LogOut

Signatur:

```
EmptyMessage LogOut(
      LogoutRequest request );
```

Die Funktion `LogOut` wird dazu verwendet, einen Benutzer explizit abzumelden. Durch den Aufruf verliert das `LoginToken` seine Gültigkeit.

Nachrichtentyp `LogoutRequest`. Listing 3.6 zeigt die WSDL-Definition des Datentyps `LogoutRequest`.

```
<xs:element name="LogoutRequest">
 <xs:complexType>
  <xs:sequence>
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q6:LoginToken"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.6: Nachrichtentyp LogoutRequest

Die folgende Tabelle führt Verweise zu allen in Listing 3.6 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2

3.2.2. Abfrage von Metadaten

3.2.2.1. GetDatabases

Signatur:

```
GetDatabasesResponse GetDatabases(
       GetDatabasesRequest request );
```

Die Funktion GetDatabases gibt eine Auflistung aller verfügbaren PROXESS-Datenbanken zurück.

Nachrichtentyp GetDatabasesRequest. Listing 3.7 zeigt die WSDL-Definition des Datentyps GetDatabasesRequest.

```
1 <xs:element name="GetDatabasesRequest">
2  <xs:complexType>
3   <xs:sequence>
4    <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
5    <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q7:LoginToken"/>
6   </xs:sequence>
7  </xs:complexType>
8 </xs:element>
```

Listing 3.7: Nachrichtentyp GetDatabasesRequest

Die folgende Tabelle führt Verweise zu allen in Listing 3.7 referenzierten Datentypen auf.

Feldname	Verweis
LoginToken	3.3.1.2

Nachrichtentyp GetDatabasesResponse. Listing 3.8 zeigt die WSDL-Definition des Datentyps GetDatabasesResponse.

```
1 <xs:element name="GetDatabasesResponse">
2  <xs:complexType>
3   <xs:sequence>
4    <!-- Liste aller Datenbanken, für die der angemeldete Benutzer Zugriffsrechte bestitzt. -->
5    <xs:element minOccurs="0" name="Databases" nillable="true" type="q8:DatabaseCollection"/>
6   </xs:sequence>
7  </xs:complexType>
8 </xs:element>
```

Listing 3.8: Nachrichtentyp GetDatabasesResponse

Die folgende Tabelle führt Verweise zu allen in Listing 3.8 referenzierten Datentypen auf.

Feldname	Verweis
DatabaseCollection	3.3

3.2.2.2. GetDocumentTypes

Signatur:

```
GetDocumentTypesResponse GetDocumentTypes(
      GetDocumentTypesRequest request );
```

Die Funktion `GetDocumentTypes` gibt alle verfügbaren Dokumenttypen für eine Datenbank zurück.

Nachrichtentyp `GetDocumentTypesRequest`. Listing 3.9 zeigt die WSDL-Definition des Datentyps `GetDocumentTypesRequest`.

```
<xs:element name="GetDocumentTypesRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q14:LoginToken"/>
   <!-- Name der Datenbank, deren Dokumenttypen abgerufen werden sollen. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.9: Nachrichtentyp GetDocumentTypesRequest

Die folgende Tabelle führt Verweise zu allen in Listing 3.9 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2

Nachrichtentyp `GetDocumentTypesResponse`. Listing 3.13 zeigt die WSDL-Definition des Datentyps `GetDocumentTypesResponse`.

```
<xs:element name="GetDocumentTypesResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Liste der Dokumenttypen, für die der angemeldete Benutzer Zugriffsrechte besitzt. -->
   <xs:element minOccurs="0" name="DocumentTypes" nillable="true" type="q15:DocumentTypeCollection"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.10: Nachrichtentyp GetDocumentTypesResponse

Die folgende Tabelle führt Verweise zu allen in Listing 3.13 referenzierten Datentypen auf.

Feldname	Verweis
`DocumentTypeCollection`	3.3

3.2.2.3. GetDocumentTypesFiltered

Signatur:

```
GetDocumentTypesResponse GetDocumentTypesFiltered(
      GetDocumentTypesFilteredRequest request );
```

Die Funktion `GetDocumentTypes` alle Dokumenttypen für eine Datenbank zurück, für die der Benutzer das angeforderte Recht besitzt.

Nachrichtentyp `GetDocumentTypesFilteredRequest`. Listing 3.11 zeigt die WSDL-Definition des Datentyps `GetDocumentTypesFilteredRequest`.

```
1  <xs:element name="GetDocumentTypesFilteredRequest">
2   <xs:complexType>
3    <xs:sequence>
4     <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
5     <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q36:LoginToken"/>
6     <!-- Name der Datenbank, deren Dokumenttypen abgerufen werden sollen. -->
7     <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
8     <!-- Enumerationwert fÃŒr die Rechte-Filterung (View, Create, Update) -->
9     <xs:element minOccurs="0" name="Filter" type="q37:DocumentUsageFilter"/>
10   </xs:sequence>
11  </xs:complexType>
12 </xs:element>
```

Listing 3.11: Nachrichtentyp GetDocumentTypesFilteredRequest

Die folgende Tabelle führt Verweise zu allen in Listing 3.11 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2

Die Enumeration `DocumentUsageFilter` spezifiziert, anhand welcher Rechte des aktuellen Benutzers die Dokumenttypen gefiltert werden.

Listing 3.12 zeigt die WSDL-Definition des Datentyps `DocumentUsageFilter`.

```
<!-- Rechte-Filter fÃŒr die Abfrage von Dokumenttypen -->
<xs:simpleType name="DocumentUsageFilter">
 <xs:restriction base="xs:string">
  <!-- Keine Filterung, alle Dokumenttypen werden zurÃŒckgegeben -->
  <xs:enumeration value="None"/>
  <!-- Nur Dokumenttypen werden zurÃŒckgegeben fÃŒr die der Benutzer das Ansehen-Recht besitzt -->
  <xs:enumeration value="View"/>
  <!-- Nur Dokumenttypen werden zurÃŒckgegeben fÃŒr die der Benutzer das Erstellen-Recht besitzt -->
  <xs:enumeration value="Create"/>
  <!-- Nur Dokumenttypen werden zurÃŒckgegeben fÃŒr die der Benutzer das Aktualisieren-Recht besitzt -->
  <xs:enumeration value="Update"/>
 </xs:restriction>
</xs:simpleType>
```

Listing 3.12: Datentyp DocumentUsageFilter

Nachrichtentyp `GetDocumentTypesResponse`. Listing 3.13 zeigt die WSDL-Definition des Datentyps `GetDocumentTypesResponse`.

```
<xs:element name="GetDocumentTypesResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Liste der Dokumenttypen, für die der angemeldete Benutzer Zugriffsrechte besitzt. -->
   <xs:element minOccurs="0" name="DocumentTypes" nillable="true" type="q15:DocumentTypeCollection"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.13: Nachrichtentyp GetDocumentTypesResponse

Die folgende Tabelle führt Verweise zu allen in Listing 3.13 referenzierten Datentypen auf.

Feldname	Verweis
`DocumentTypeCollection`	3.3

3.2.2.4. GetFileTypes

Signatur:

```
GetFileTypesResponse GetFileTypes(
       GetFileTypesRequest request );
```

Die Funktion `GetFileTypes` gibt alle verfügbaren Dateitypen für eine Datenbank zurück.

Nachrichtentyp `GetFileTypesRequest`. Listing 3.14 zeigt die WSDL-Definition des Datentyps `GetFileTypesRequest`.

```
<xs:element name="GetFileTypesRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q32:LoginToken"/>
   <!-- Name der Datenbank, deren Dateitypen abgerufen werden sollen. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.14: Nachrichtentyp GetFileTypesRequest

Die folgende Tabelle führt Verweise zu allen in Listing 3.14 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2

Nachrichtentyp `GetFileTypesResponse`. Listing 3.15 zeigt die WSDL-Definition des Datentyps `GetFileTypesResponse`.

```
<xs:element name="GetFileTypesResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Liste der Dateitypen, für die der angemeldete Benutzer Zugriffsrechte besitzt. -->
   <xs:element minOccurs="0" name="FileTypes" nillable="true" type="q33:FileTypeCollection"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.15: Nachrichtentyp GetFileTypesResponse

Die folgende Tabelle führt Verweise zu allen in Listing 3.15 referenzierten Datentypen auf.

Feldname	Verweis
`FileTypeCollection`	3.3

3.2.2.5. GetUsersAndGroups

Signatur:

```
GetUsersAndGroupsResponse GetUsersAndGroups(
      GetUsersAndGroupsRequest request);
```

Die Funktion `GetUsersAndGroups` gibt alle definierten PROXESS-Benutzer und -Gruppen zurück.

Nachrichtentyp `GetUsersAndGroupsRequest`. Listing 3.16 zeigt die WSDL-Definition des Datentyps `GetUsersAndGroupsRequest`.

```
<xs:element name="GetUsersAndGroupsRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q143:LoginToken"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.16: Nachrichtentyp GetUsersAndGroupsRequest

Die folgende Tabelle führt Verweise zu allen in Listing 3.16 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2

Nachrichtentyp `GetUsersAndGroupsResponse`. Listing 3.17 zeigt die WSDL-Definition des Datentyps `GetUsersAndGroupsResponse`.

```
<xs:element name="GetUsersAndGroupsResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Liste von PROXESS-Benutzern. -->
   <xs:element minOccurs="0" name="Users" nillable="true" type="q144:UserCollection"/>
   <!-- Liste von PROXESS-Gruppen. -->
   <xs:element minOccurs="0" name="Groups" nillable="true" type="q145:GroupCollection"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.17: Nachrichtentyp GetUsersAndGroupsResponse

Die folgende Tabelle führt Verweise zu allen in Listing 3.17 referenzierten Datentypen auf.

Feldname	Verweis
`UserCollection`	3.3.2.5
`GroupCollection`	3.3.2.5

3.2.2.6. GetValidationRules

Signatur:

```
GetValidationRulesResponse GetValidationRules(
       GetValidationRulesRequest request);
```

Die Funktion `GetValidationRules` gibt alle für eine Datenbank definierten Validierungsregeln zurück.

Nachrichtentyp `GetValidationRulesRequest`. Listing 3.18 zeigt die WSDL-Definition des Datentyps `GetValidationRulesRequest`.

```
<xs:element name="GetValidationRulesRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q160:LoginToken"/>
   <!-- Name der Datenbank, deren Validierungsregeln abgerufen werden sollen. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.18: Nachrichtentyp GetValidationRulesRequest

Die folgende Tabelle führt Verweise zu allen in Listing 3.18 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2

Nachrichtentyp `GetValidationRulesResponse`. Listing 3.19 zeigt die WSDL-Definition des Datentyps `GetValidationRulesResponse`.

```
<xs:element name="GetValidationRulesResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Liste von Validierungsregeln. -->
   <xs:element minOccurs="0" name="ValidationRules" nillable="true"
    type="q161:ValidationRuleCollection"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.19: Nachrichtentyp GetValidationRulesResponse

Die folgende Tabelle führt Verweise zu allen in Listing 3.19 referenzierten Datentypen auf.

Feldname	Verweis
`ValidationRuleCollection`	3.3.2.6

3.2.2.7. GetDatabaseFields

Signatur:

```
GetDatabaseFieldsResponse GetDatabaseFields(
      GetDatabaseFieldsRequest request);
```

Die Funktion `GetDatabaseFields` gibt alle für eine Datenbank definierten Standardfelder zurück.

Nachrichtentyp `GetDatabaseFieldsRequest`. Listing 3.20 zeigt die WSDL-Definition des Datentyps `GetDatabaseFieldsRequest`.

```
<xs:element name="GetDatabaseFieldsRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q14:LoginToken"/>
   <!-- Name der Datenbank, deren Standardfelder abgerufen werden sollen. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.20: Nachrichtentyp GetDatabaseFieldsRequest

Die folgende Tabelle führt Verweise zu allen in Listing 3.20 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2

Nachrichtentyp `GetDatabaseFieldsResponse`. Listing 3.21 zeigt die WSDL-Definition des Datentyps `GetDatabaseFieldsResponse`.

```
<xs:element name="GetDatabaseFieldsResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Liste der Standardfelder, die für die angefragte Datenbank definiert sind. -->
   <xs:element minOccurs="0" name="DatabaseFields" nillable="true" type="q18:FieldDescriptionCollection"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.21: Nachrichtentyp GetDatabaseFieldsResponse

Die folgende Tabelle führt Verweise zu allen in Listing 3.21 referenzierten Datentypen auf.

Feldname	Verweis
`FieldDescriptionCollection`	3.3

3.2.3. Recherche

Dieser Abschnitt beschreibt die Funktionen zum Ausführen einer Recherche und Abholen der Suchergebnisse.

 Die Funktionen `Query`, die einen anderen Algorithmus zur Datenübertragung verwendet, ist im Anhang Chunked-Transfer beschrieben.

3.2.3.1. ResultOpen

Signatur:

```
ResultOpenResponse ResultOpen(
       ResultOpenRequest request );
```

Die Funktion `ResultOpen` führt eine Recherche in PROXESS aus und gibt ein Handle zur Übertragung der Ergebnisliste zurück.

Der Aufbau einer Suchbedingung ist in den Abschnitten GeneralSearch und SearchCondition beschrieben. Beispiele finden sich in Anhang GeneralSearch/SearchCondition Beispiele.

Nachrichtentyp `ResultOpenRequest`. Listing 3.22 zeigt die WSDL-Definition des Datentyps `ResultOpenRequest`.

```
1  <xs:element name="ResultOpenRequest">
2   <xs:complexType>
3    <xs:sequence>
4     <!-- Login-Token der PROXESS Session. -->
5     <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q212:LoginToken"/>
6     <!-- Beschreibung der Suchbedingung, Trefferzahl, SeitengrÃ¶Ãe. -->
7     <xs:element minOccurs="0" name="Query" nillable="true" type="q213:GeneralSearch"/>
8    </xs:sequence>
9   </xs:complexType>
10 </xs:element>
```

Listing 3.22: Nachrichtentyp ResultOpenRequest

Die folgende Tabelle führt Verweise zu allen in Listing 3.22 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2
`GeneralSearch`	3.3.4.1

Nachrichtentyp `ResultOpenResponse`. Listing 3.23 zeigt die WSDL-Definition des Datentyps `ResultOpenResponse`.

```
<xs:element name="ResultOpenResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Handle fÃŒr die Ãbertragung der Trefferliste. -->
   <xs:element minOccurs="0" name="ResultHandle" nillable="true" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.23: Nachrichtentyp ResultOpenResponse

3.2.3.2. ResultRead

Signatur:

```
ResultReadResponse ResultRead(
       ResultReadRequest request );
```

Die Funktion `ResultRead` ruft den nächsten Block eines Suchergebnisses ab.

Nachrichtentyp `ResultReadRequest`. Listing 3.24 zeigt die WSDL-Definition des Datentyps `ResultReadRequest`.

```
<xs:element name="ResultReadRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Login-Token der PROXESS-Session. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q214:LoginToken"/>
   <!-- Handle fÃŒr die Ãbertragung der Trefferliste. -->
   <xs:element minOccurs="0" name="ResultHandle" nillable="true" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.24: Nachrichtentyp ResultReadRequest

Die folgende Tabelle führt Verweise zu allen in Listing 3.24 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2

Nachrichtentyp `ResultReadResponse`. Listing 3.25 zeigt die WSDL-Definition des Datentyps `ResultReadResponse`.

```
<xs:element name="ResultReadResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Gibt an, ob noch weitere Treffer abzuholen sind. -->
   <xs:element minOccurs="0" name="HasMore" type="xs:boolean"/>
   <!-- Ergebnisblock der Trefferliste. -->
   <xs:element minOccurs="0" name="ResultBlock" nillable="true" type="q215:ResultHitCollection"/>
   <!-- Metadaten-Elemente zu allen EintrÃ€gen der Trefferliste.
              (Zugriff auf Metadaten erfolgt direkt Ã€ber die Trefferlisten-EintrÃ€ge) -->
   <xs:element minOccurs="0" name="Metadata" nillable="true" type="q216:MetadataContainer"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.25: Nachrichtentyp ResultReadResponse

Die folgende Tabelle führt Verweise zu allen in Listing 3.25 referenzierten Datentypen auf.

Feldname	Verweis
`ResultHitCollection`	3.3
`ResultHit`	3.3.4.1

3.2.3.3. ResultClose

Signatur:

```
EmptyMessage ResultClose(
       ResultCloseRequest request );
```

Die Funktion `ResultRead` ruft den nächsten Block eines Suchergebnisses ab.

Nachrichtentyp `ResultCloseRequest`. Listing 3.26 zeigt die WSDL-Definition des Datentyps `ResultCloseRequest`.

```
<xs:element name="ResultCloseRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Login-Token fÃ€r die PROXESS-Session. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q217:LoginToken"/>
   <!-- Handle fÃ€r die Ãbertragung der Trefferliste. -->
   <xs:element minOccurs="0" name="ResultHandle" nillable="true" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.26: Nachrichtentyp ResultCloseRequest

Die folgende Tabelle führt Verweise zu allen in Listing 3.26 referenzierten Datentypen auf.

Feldname	Verweis
LoginToken	3.3.1.2

3.2.3.4. ExtendedResult-Funktionen

Es gibt ein weiteres Triple von Suchfunktionen, die analog zu den oben beschriebenen Funktionen aufgebaut sind: `ResultOpenExtended`, `ResultReadExtended` und `ResultCloseExtended`.

Die *ExtendedSearch* unterscheided sich von der normalen Suche dadurch, dass zusätzlich zu den Dokument-Informationen Informationen über die anhängigen Dateien zurückgegeben werden.

Die Signatur der Methoden `ResultOpenExtended` und `ResultCloseExtended` ist identisch zu den Varianten `ResultOpen` bzw. `ResultClose`. Die Funktion `ResultReadExtended` unterscheidet sich in der Rückgabe.

ResultReadExtended Signatur:

```
ResultReadExtendedResponse ResultClose(
       ResultReadRequest request );
```

Die Funktion `ResultReadExtended` ruft den nächsten Block eines Suchergebnisses einer ExtendedSearch ab.

Nachrichtentyp `ResultReadExtendedResponse`. Listing 3.27 zeigt die WSDL-Definition des Datentyps `ResultReadExtendedResponse`.

```
1  <xs:element name="ResultReadExtendedResponse">
2   <xs:complexType>
3    <xs:sequence>
4     <!-- Gibt an, ob noch weitere Treffer abzuholen sind. -->
5     <xs:element minOccurs="0" name="HasMore" type="xs:boolean"/>
6     <!-- Ergebnisblock der Trefferliste. -->
7     <xs:element minOccurs="0" name="ResultBlock" nillable="true" type="q218:ExtendedResultHitCollection"/>
8     <!-- Metadaten-Elemente zu allen EintrÃ€gen der Trefferliste.
9                (Zugriff auf Metadaten erfolgt direkt Ã€ber die Trefferlisten-EintrÃ€ge) -->
10    <xs:element minOccurs="0" name="Metadata" nillable="true" type="q219:MetadataContainer"/>
11   </xs:sequence>
12  </xs:complexType>
13 </xs:element>
```

Listing 3.27: Nachrichtentyp ResultReadExtendedResponse

Die folgende Tabelle führt Verweise zu allen in Listing 3.27 referenzierten Datentypen auf.

Feldname	Verweis
ExtendedResultHitCollection	3.3
Document	3.3.3.1

3.2.3.5. Code-Beispiel: Abrufen eines Suchergebnisses

Das folgende Beispiel zeigt den Algorithmus zur Abholung eines Suchergebnisses.

 Da - je nach Suchbedingung (z.B. Suche nach Dokumenttyp) - die Ergebnisliste sehr groß werden kann, sollte für den allgemeinen Fall darauf verzichtet werden, das gesamte Suchergebnis in eine Liste/Array zu schreiben. Für bestimmte Anwendungsfälle (z.B. Suche nach Belegnummer) trifft dies widerum nicht zu.
Als Client-Entwickler muss anhand der auszuführenden Suche abgewogen werden, wie Suchergebnisse (en block oder stream-basiert) behandelt werden.

```
List<ResultHit> Search(WebserviceClient wsClient, GeneralSearch searchCondition)
{
  List<ResultHit> result = new List<ResultHit>();
  ResultOpenResponse openResponse = wsClient.ResultOpen(searchCondition);
  string resultHandle = openResponse.ResultHandle;
  bool hasMore = true;
  while(hasMore)
  {
    ResultReadResponse readResponse = wsClient.ResultRead(resultHandle);
    hasMore = readResponse.HasMore;
    result.AddRange(readResponse.ResultBlock);
  }
  wsClient.ResultClose(resultHandle);
  return result;
}
```

3.2.4. Dokumentabfrage, -anlage und -änderung

3.2.4.1. GetDocument

Signatur:

```
GetDocumentResponse GetDocument(
       GetDocumentRequest request );
```

Die Funktion GetDocument gibt das Dokument mit der angegebenen ID zurück.

Nachrichtentyp GetDocumentRequest. Listing 3.28 zeigt die WSDL-Definition des Datentyps GetDocumentRequest.

```
1  <xs:element name="GetDocumentRequest">
2   <xs:complexType>
3    <xs:sequence>
4     <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
5     <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q79:LoginToken"/>
6     <!-- Name der Datenbank, in der das Dokument gespeichert ist. -->
7     <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
8     <!-- ID des Dokuments. -->
9     <xs:element minOccurs="0" name="DocumentID" type="xs:unsignedLong"/>
10   </xs:sequence>
11  </xs:complexType>
12 </xs:element>
```

Listing 3.28: Nachrichtentyp GetDocumentRequest

Die folgende Tabelle führt Verweise zu allen in Listing 3.28 referenzierten Datentypen auf.

Feldname	Verweis
LoginToken	3.3.1.2

Nachrichtentyp GetDocumentResponse. Listing 3.29 zeigt die WSDL-Definition des Datentyps GetDocumentResponse.

```
1  <xs:element name="GetDocumentResponse">
2   <xs:complexType>
3    <xs:sequence>
4     <!-- PROXESS Dokument mit Feldwerten, Datei-Infos und Querverweisen. -->
5     <xs:element minOccurs="0" name="Document" nillable="true" type="q80:Document"/>
6    </xs:sequence>
7   </xs:complexType>
8  </xs:element>
```

Listing 3.29: Nachrichtentyp GetDocumentResponse

Die folgende Tabelle führt Verweise zu allen in Listing 3.29 referenzierten Datentypen auf.

Feldname	Verweis
Document	3.3.3.1

3.2.4.2. CreateDocument

Signatur:

```
CreateDocumentResponse CreateDocument(
       CreateDocumentRequest request );
```

Die Funktion `CreateDocument` erstellt ein neues Dokument mit den angegebenen Eigenschaften. Als Bestätigung wird das neue Dokument zurückgegeben – so, als würde `GetDocument` mit der neuen ID aufgerufen werden.

Nachrichtentyp `CreateDocumentRequest`. Listing 3.30 zeigt die WSDL-Definition des Datentyps `CreateDocumentRequest`.

```
1  <xs:element name="CreateDocumentRequest">
2   <xs:complexType>
3    <xs:sequence>
4     <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
5     <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q40:LoginToken"/>
6     <!-- Name der Datenbank, in der das Dokument angelegt wird. -->
7     <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
8     <!-- ID des Typs des neuen Dokuments. -->
9     <xs:element minOccurs="0" name="DocumentTypeID" type="xs:unsignedLong"/>
10    <!-- Beschreibung bzw. Titel des neuen Dokuments. -->
11    <xs:element minOccurs="0" name="DocumentDescription" nillable="true" type="xs:string"/>
12    <!-- Liste von Feldwerten, die für das neue Dokument gesetzt werden.
13        Fehlt das XML-Element oder ist der Wert null, werden keine Feldwerte gesetzt. -->
14    <xs:element minOccurs="0" name="DocumentFields" nillable="true"
15      type="q41:ArrayOfFieldChange">
16    <!-- Liste von Querverweisen, die für das neue Dokument angelegt werden.
17        Fehlt das XML-Element oder ist der Wert null, werden keine Querverweise gesetzt.-->
18    <xs:element minOccurs="0" name="DocumentLinks" nillable="true"
19      type="q42:ArrayOfDocumentLinkChange"/>
20   </xs:sequence>
21  </xs:complexType>
22 </xs:element>
```

Listing 3.30: Nachrichtentyp CreateDocumentRequest

Die folgende Tabelle führt Verweise zu allen in Listing 3.30 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2
`ArrayOfFieldChange`	3.3
`ArrayOfDocumentLinkChange`	3.3

Nachrichtentyp `CreateDocumentResponse`. Listing 3.31 zeigt die WSDL-Definition des Datentyps `CreateDocumentResponse`.

```
<xs:element name="CreateDocumentResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Das PROXESS Dokument, das mit dem Aufruf angelegt wurde. -->
   <xs:element minOccurs="0" name="Document" nillable="true" type="q51:Document"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.31: Nachrichtentyp CreateDocumentResponse

Die folgende Tabelle führt Verweise zu allen in Listing 3.31 referenzierten Datentypen auf.

Feldname	Verweis
`Document`	3.3.3.1

Anmerkung zu Feldern. Ein Dokumentfeld wird durch die Angabe des Feldnamens und den zu setzenden Wert gesetzt bzw. geändert (siehe Datentyp 3.3.3.5: `FieldChange`). Beim Setzen/Ändern von Feldwerten sollten immer folgende Dinge beachtet werden:

- **Ein Feld wird durch seinen Datenbanknamen identifiziert** und nicht durch die Feld-ID oder den Dokumenttyp-Rollennamen (siehe dazu Abschnitt 3.3.2.3: `FieldDescription`).
- **Feldwerte von Feldern, die nicht in `ArrayOfFieldChange` aufgeführt sind, werden nicht gesetzt/geändert.** Wenn der Wert eines Feldes gelöscht werden soll (beim Ändern eines Dokuments), muss der Feldwert explizit auf `null` gesetzt werden.
- **Clients müssen sicherstellen, dass Feldwerte den richtigen Datentyp haben.** Der Webservice gibt alle Feldwerte als Strings aus. In PROXESS haben diese Felder evtl. einen spezifischen Datentyp, z.B. Integer, Double oder Datetime. Die Strings, die beim Setzen bzw. Ändern von Feldwerten angegeben werden, müssen als Wert des korrekten Datentyps interpretiert werden können[3]. "In sinnvollen Grenzen" funktioniert die Umwandlung automatisch.
- **Clients müssen sicherstellen, dass alle Pflichtfelder gesetzt sind.** Insbesondere dürfen Pflichtfelder nicht den Wert `null` haben.

Informationen zum PROXESS-internen Datentyp eines Feldes und dazu, ob es sich bei einem Feld um ein Pflichtfeld handelt, kann der Beschreibung des Dokumenttyps entnommen werden (siehe Datentyp 3.3.2.2: `DocumentType` bzw. Funktion 3.2.2.2: `GetDocumentTypes`).

[3] O.B.d.A. greifen die Konvertierungsregeln der .NET Basis-Klasse `Convert` `http://msdn.microsoft.com/en-us/library/system.convert(v=vs.100).aspx`. Bei der Interpretation von Werten sind ggf. besondere Regions- und Spracheinstellungen des Computers zu beachten, auf dem der Webservice läuft.

Schlägt das Setzen von Feldwerten beim Anlegen eines Dokumentes fehl, so wird das Dokument verworfen, ohne vorher gespeichert worden zu sein. Ein solcher Fehler hinterlässt keine Spuren in PROXESS.

Anmerkung zu Querverweisen. Beim Setzen bzw. Ändern von Querverweisen müssen folgende Dinge beachtet werden:

- Clients müssen sicherstellen, dass es sich bei den **Link-Tragets** um **gültige Dokument-IDs** handelt.
- Außerdem müssen **alle Zieldokumente in derselben Datenbank** gespeichert sein, in der auch das Quelldokument gespeichert ist.

Schlägt das Hinzufügen einer Dokumentverknüpfung beim Anlegen eines Dokumentes fehl, so wird das Dokument wieder gelöscht (es ist zuvor schon gespeichert worden, und zwar nach dem Setzen der Feldwerte). Ein Fehlerfall beim Hinzufügen von Querverweisen beim Anlegen von Dokumenten hinterlässt also Spuren in PROXESS[4].

3.2.4.3. ChangeDocument

Signatur:

```
ChangeDocumentResponse ChangeDocument(
      ChangeDocumentRequest request );
```

Die Funktion `ChangeDocument` ändert ein bestehendes Dokument und gibt als Bestätigung das geänderte Dokument zurück.

[4] Für den normalen Benutzer sind diese Spuren nicht sichtbar, da die Dokumente sofort wieder gelöscht werden. Ein Administrator kann diese gelöschten Dokumente allerdings sehen und wiederherstellen.

Nachrichtentyp `ChangeDocumentRequest`. Listing 3.32 zeigt die WSDL-Definition des Datentyps `ChangeDocumentRequest`.

```
<xs:element name="ChangeDocumentRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q81:LoginToken"/>
   <!-- Name der Datenbank, in der das Dokument gespeichert ist. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
   <!-- ID des zu ändernden Dokuments. -->
   <xs:element minOccurs="0" name="DocumentID" type="xs:unsignedLong"/>
   <!-- Neue Beschreibung/Titel des Dokuments.
       Fehlt das XML-Element oder ist der Wert null: keine Änderung des Titels. -->
   <xs:element minOccurs="0" name="Description" nillable="true" type="xs:string"/>
   <!-- Liste der zu ändernden Feldwerte.
       Fehlt das XML-Element oder ist der Wert null: keine Änderung an Feldwerten. -->
   <xs:element minOccurs="0" name="Fields" nillable="true" type="q82:ArrayOfFieldChange">
   <!-- Liste der zu ändernden Querverweise.
       Fehlt das XML-Element oder ist der Wert null: keine Änderung an Querverweisen. -->
   <xs:element minOccurs="0" name="Links" nillable="true"
     type="q83:ArrayOfDocumentLinkChange"/>
   <xs:element minOccurs="0" name="DocumentTypeID" nillable="true" type="xs:unsignedLong"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.32: Nachrichtentyp ChangeDocumentRequest

Die folgende Tabelle führt Verweise zu allen in Listing 3.32 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2
`ArrayOfFieldChange`	3.3
`ArrayOfDocumentLinkChange`	3.3

Nachrichtentyp `ChangeDocumentResponse`. Listing 3.33 zeigt die WSDL-Definition des Datentyps `ChangeDocumentResponse`.

```
<xs:element name="ChangeDocumentResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Aktuelle Version des im letzten Aufruf geänderten Dokuments. -->
   <xs:element minOccurs="0" name="Document" nillable="true" type="q84:Document"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.33: Nachrichtentyp ChangeDocumentResponse

Die folgende Tabelle führt Verweise zu allen in Listing 3.33 referenzierten Datentypen auf.

Feldname	Verweis
Document	3.3.3.1

Anmerkung. Hier gelten die gleichen Anmerkungen zum **Ändern/Setzen von Feldwerten** bzw. **Hinzufügen/Entfernen von Querverweisen** wie für das Erstellen von Dokumenten. Schlägt beim Ändern eines Dokumentes einer der beiden Vorgänge fehl, werden alle Änderungen verworfen. Eine fehlgeschlagenes Änderung eines Dokuments hinterlässt in keinem Fall Spuren in PROXESS – anders als beim Erstellen von Dokumenten.

3.2.4.4. DeleteDocument

Signatur:

```
EmptyMessage DeleteDocument(
       DeleteDocumentRequest request );
```

Die Funktion `DeleteDocument` löscht das angegebene Dokument ohne weitere Nachfrage. Die Funktion hat keinen Rückgabewert.

Nachrichtentyp `DeleteDocumentRequest`. Listing 3.34 zeigt die WSDL-Definition des Datentyps `DeleteDocumentRequest`.

```
1  <xs:element name="DeleteDocumentRequest">
2   <xs:complexType>
3    <xs:sequence>
4     <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
5     <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q85:LoginToken"/>
6     <!-- Name der Datenbank, in der das Dokument gespeichert ist. -->
7     <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
8     <!-- ID des zu löschenden Dokuments. -->
9     <xs:element minOccurs="0" name="DocumentID" type="xs:unsignedLong"/>
10   </xs:sequence>
11  </xs:complexType>
12 </xs:element>
```

Listing 3.34: Nachrichtentyp DeleteDocumentRequest

Die folgende Tabelle führt Verweise zu allen in Listing 3.34 referenzierten Datentypen auf.

Feldname	Verweis
LoginToken	3.3.1.2

3.2.4.5. DeleteFile

Signatur:

```
EmptyMessage DeleteFile(
      DeleteFileRequest request );
```

Die Funktion `DeleteFile` löscht eine Datei aus dem angegebenen Dokument, sofern der Benutzer über die entsprechende Berechtigung verfügt. Diese Funktion hat keinen Rückgabewert.

Nachrichtentyp `DeleteFileRequest`. Listing 3.35 zeigt die WSDL-Definition des Datentyps `DeleteFileRequest`.

```
<xs:element name="DeleteFileRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q86:LoginToken"/>
   <!-- Name der Datenbank, in der das Dokument gespeichert ist. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
   <!-- ID des Dokuments, zu dem die Datei gehört. -->
   <xs:element minOccurs="0" name="DocumentID" type="xs:unsignedLong"/>
   <!-- ID der zu löschenden Datei. -->
   <xs:element minOccurs="0" name="FileID" type="xs:unsignedLong"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.35: Nachrichtentyp DeleteFileRequest

Die folgende Tabelle führt Verweise zu allen in Listing 3.35 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2

3.2.5. Dateidownload und -upload

Dieser Abschnitt beschreibt die Funktionen zur Übertragung von Dateien.

Am Ende dieses Abschnitts finden sich Code-Beispiele für die Implementierung der Dateiübertragung.


Die Funktionen `DownloadFile`, `UploadFile` und `UploadNewFile` die einen anderen Algorithmus zur Dateiübertragung verwenden,sind im Anhang Chunked-Transfer beschrieben.

3.2.5.1. FileOpen

Signatur:

```
FileOpenResponse FileOpen(
       FileOpenRequest request );
```

Die Funktion `FileOpen` wird zum Lesen von Dateien verwendet. Das zurückgegebene Dateihandle kann an die Funktion `FileRead` zum Lesen der Dateidaten übergeben werden. Die Datei muss mit der Funktion `FileClose` wieder geschlossen werden.

Nachrichtentyp `FileOpenRequest`. Listing 3.36 zeigt die WSDL-Definition des Datentyps `FileOpenRequest`.

```
1  <!-- Öffnet eine Datei zum Lesen. -->
2  <xs:element name="FileOpenRequest">
3   <xs:complexType>
4    <xs:sequence>
5     <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
6     <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q141:LoginToken"/>
7     <!-- Name der Datenbank, in der das Dokument gespeichert ist. -->
8     <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
9     <!-- ID des zugehörigen Dokuments. -->
10    <xs:element minOccurs="0" name="DocumentID" type="xs:unsignedLong"/>
11    <!-- ID der Datei. -->
12    <xs:element minOccurs="0" name="FileD" type="xs:unsignedLong"/>
13   </xs:sequence>
14  </xs:complexType>
15 </xs:element>
```

Listing 3.36: Nachrichtentyp FileOpenRequest

Die folgende Tabelle führt Verweise zu allen in Listing 3.36 referenzierten Datentypen auf.

Feldname	Verweis
LoginToken	3.3.1.2

Nachrichtentyp `FileOpenResponse`. Listing 3.37 zeigt die WSDL-Definition des Datentyps `FileOpenResponse`.

```
<xs:element name="FileOpenResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Handle für die geöffnete Datei.
       Wird als Parameter in Folgeaufrufen angegeben. -->
   <xs:element minOccurs="0" name="FileHandle" nillable="true" type="xs:string"/>
   <!-- Falls die Datei zum Lesen geöffnet wurde (FileOpen), enthält RenderInfo
       Informationen darüber, ob die Datei mit der Komponent PROXESS Web Rendr
              in eine PDF-Datei konvertiert worden ist.
              Für die Aufrufe FileCreate und FileCreateNewVersion hat RenderInfo stets
              den Wert NULL. -->
   <xs:element minOccurs="0" name="RenderInfo" nillable="true" type="q139:RenderInfo"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.37: Nachrichtentyp FileOpenResponse

Die folgende Tabelle führt Verweise zu allen in Listing 3.37 referenzierten Datentypen auf.

Feldname	Verweis
`RenderInfo`	3.3.1.4

Anmerkung: `RenderInfo`. Das Datenfeld `RenderInfo` enthält Informationen darüber, ob eine Datei mit der Komponente PROXESS Web Rendr gerendert worden ist, und gibt dem Client die Dateierweiterung an, die beim Speichern der Datei zu verwenden ist. Im Fall von gerenderten Dateien ist die Dateierweiterung im Regelfall `pdf`, für Dateien, die nicht gerendert wurden, wird die Dateierweiterung des zugehörigen PROXESS-Dateityps zurückgegeben, z.B. `txt` oder `docx`.

Weitere Informationen zum Datentyp `RenderInfo` finden sich inAbschnitt B.2.0.4.

3.2.5.2. FileCreate

Signatur:

```
FileCreateResponse FileCreate(
       FileCreateRequest request );
```

Die Funktion `FileCreate` fügt eine neue Datei an ein Dokument an. Das zurückgegebene Dateihandle wird an die Funktion `FileWrite` zur Übertragung der Dateidaten übergeben. Nach erfolgreicher Übertragung muss die Datei mit der Funktion `FileClose` geschlossen werden. Im Falle eines client-seitigen Fehlers (z.B. beim Lesen der Eingabedatei), kann die Übertragung mit der Funktion `FileCancel` abgebrochen werden. Die Datei wird in diesem Fall nicht in PROXESS gespeichert.

Nachrichtentyp `FileCreateRequest`. Listing 3.38 zeigt die WSDL-Definition des Datentyps `FileCreateRequest`.

```
<!-- Fügt einem Dokument eine neue Datei hinzu. -->
<xs:element name="FileCreateRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q141:LoginToken"/>
   <!-- Name der Datenbank, in der das Dokument gespeichert ist. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
   <!-- ID des zugehörigen Dokuments. -->
   <xs:element minOccurs="0" name="DocumentID" type="xs:unsignedLong"/>
   <!-- ID des Dateityps der neuen Datei. -->
   <xs:element minOccurs="0" name="FileTypeID" type="xs:unsignedLong"/>
   <!-- Beschreibung/Titel der neuen Datei. -->
   <xs:element minOccurs="0" name="FileDescription" nillable="true" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.38: Nachrichtentyp FileCreateRequest

Nachrichtentyp `FileCreateResponse`. Listing 3.39 zeigt die WSDL-Definition des Datentyps `FileCreateResponse`.

```
<xs:element name="FileCreateResponse">
  <xs:complexType>
    <xs:sequence>
        <!-- Handle fÃŒr die geÃ¶ffnete Datei. -->
      <xs:element minOccurs="0" name="FileHandle" nillable="true" type="xs:string"/>
      <!-- Neue ID der Datei. -->
        <xs:element minOccurs="0" name="NewFileID" type="xs:unsignedLong"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing 3.39: Nachrichtentyp FileCreateResponse

3.2.5.3. FileCreateNewVersion

Signatur:

```
FileCreateResponse FileCreateNewVersion(
      FileCreateNewVersionRequest request );
```

Die Funktion `FileCreateNewVersion` legt eine neue Version einer bestehnden Datei an. Das zurückgegebene Dateihandle wird an die Funktion `FileWrite` zur Übertragung der Dateidaten übergeben. Nach erfolgreicher

Übertragung muss die Datei mit der Funktion `FileClose` geschlossen werden. Im Falle eines client-seitigen Fehlers (z.B. beim Lesen der Eingabedatei), kann die Übertragung mit der Funktion `FileCancel` abgebrochen werden. Die Datei wird in diesem Fall nicht in PROXESS gespeichert.

Nachrichtentyp `FileCreateNewVersionRequest`. Listing 3.40 zeigt die WSDL-Definition des Datentyps `FileCreateNewVersionRequest`.

```
<!-- Erzeugt eine neue Version einer bestehenden Datei. -->
<xs:element name="FileCreateNewVersionRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q141:LoginToken"/>
   <!-- Name der Datenbank, in der das Dokument gespeichert ist. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
   <!-- ID des Dokuments, zu dem die Datei gehört. -->
   <xs:element minOccurs="0" name="DocumentID" type="xs:unsignedLong"/>
   <!-- ID der Datei, für die eine neue Version erzeugt werden soll. -->
   <xs:element minOccurs="0" name="FileID" type="xs:unsignedLong"/>
   <!-- Beschreibung/Titel der neuen Datei. -->
   <xs:element minOccurs="0" name="FileDescription" nillable="true" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.40: Nachrichtentyp FileCreateNewVersionRequest

3.2.5.4. FileRead

Signatur:

```
FileReadResponse FileRead(
       FileReadRequest request );
```

Die Funktion `FileRead` wird zum Lesen von Dateidaten (ggf. mehrfach) aufgerufen. Es können nur Dateihandles verwendet werden, die mit der Funktion `FileOpen` erzeugt worden sind. Die Übertragung einer Datei ist dann abgeschlossen, wenn der Wert von `FileReadResponse.ReadBytes` kleiner ist als die Anzahl an angeforderten Byten in `FileReadRequest.RequestedBytes`.

Nachrichtentyp `FileReadRequest`. Listing 3.41 zeigt die WSDL-Definition des Datentyps `FileReadRequest`.

```
<xs:element name="FileReadRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q85:LoginToken"/>
   <!-- Handle für die Datei. -->
   <xs:element minOccurs="0" name="FileHandle" nillable="true" type="xs:string"/>
   <!-- Anzahl bytes, die gelesen werden sollen. -->
   <xs:element minOccurs="0" name="RequestedBytes" type="xs:int"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.41: Nachrichtentyp FileReadRequest

Nachrichtentyp `FileReadResponse`. Listing 3.42 zeigt die WSDL-Definition des Datentyps `FileReadResponse`.

```
<xs:element name="FileReadResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Anzahl Bytes die gelesen wurden. -->
   <xs:element minOccurs="0" name="ReadBytes" type="xs:int"/>
   <!-- Puffer mit Dateidaten. -->
   <xs:element minOccurs="0" name="Buffer" nillable="true" type="xs:base64Binary"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

Listing 3.42: Nachrichtentyp FileReadResponse

3.2.5.5. FileWrite

Signatur:

```
EmptyMessage FileWrite(
      FileWriteRequest request );
```

Die Funktion `FileWrite` wird zum Schreiben von Dateidaten (ggf. mehrfach) aufgerufen. Es können nur Dateihandles verwendet werden, die mit einer der Funktionen `FileCreate` und `FileCreateNewVersion` erzeugt worden sind. Damit die Datei nach einer erfolgreichen Übertragung in PROXESS gespeichert wird, muss die Funktion `FileClose` aufgerufen werden. Um die Übertragung im Fehlerfall abzubrechen, steht die Funktion `FileCancel` zur Verfügung.

Nachrichtentyp `FileWriteRequest`. Listing 3.43 zeigt die WSDL-Definition des Datentyps `FileWriteRequest`.

```
<xs:element name="FileWriteRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q85:LoginToken"/>
   <!-- Handle für die Datei. -->
   <xs:element minOccurs="0" name="FileHandle" nillable="true" type="xs:string"/>
   <!-- Puffer mit Dateidaten. -->
   <xs:element minOccurs="0" name="Buffer" nillable="true" type="xs:base64Binary"/>
   <!-- Startposition im Datenpuffer. -->
   <xs:element minOccurs="0" name="Offset" type="xs:int"/>
   <!-- Anzahl bytes, die aus dem Puffer geschrieben werden sollen. -->
   <xs:element minOccurs="0" name="Length" type="xs:int"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.43: Nachrichtentyp FileWriteRequest

3.2.5.6. FileClose

Signatur:

```
EmptyMessage FileClose(
      FileCloseRequest request );
```

Die Funktion `FileClose` wird zum Schließen einer Datei aufgerufen. Die Funktion hat keinen Rückgabewert.

Nachrichtentyp `FileCloseRequest`. Listing 3.44 zeigt die WSDL-Definition des Datentyps `FileCloseRequest`.

```
<xs:element name="FileCloseRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q85:LoginToken"/>
   <!-- Handle für die Datei. -->
   <xs:element minOccurs="0" name="FileHandle" nillable="true" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 3.44: Nachrichtentyp FileCloseRequest

3.2.5.7. FileCancel

Signatur:

```
EmptyMessage FileCancel(
       FileCloseRequest request );
```

Die Funktion `FileCancel` wird zum Abbrechen einer Dateiübertragung verwendet. Die Datei(version) wird in PROXESS nicht gespeichert.

3.2.5.8. Code-Beispiele

Download. Das folgende Beispiel zeigt den Algorithmus für den Download einer Datei.

```
void DownloadFile (WebserviceClient wsClient, string database, ulong docID, ulong fileID, Stream output)
{
  FileOpenResponse openResponse = wsClient.FileOpen(database, docID, fileID);
  string fileHandle = openResponse.FileHandle;
  bool readMore = true;
  while( readMore )
  {
    FileReadResonse readResponse = wsClient.FileRead(fileHandle);
    bytesRead = readResponse.BytesRead;
    if( bytesRead > 0 )
      output.Write(readResponse.Buffer, 0, bytesRead);
    else readMore = false;
  }
  wsClient.FileClose(fileHandle);
}
```

Upload. Das folgende Beispiel zeigt den Algorithmus für den Upload einer neuen Datei am Beispiel der Neuanlage einer Datei.

Zur Versionierung einer bestehenden Datei ist anstelle der Funktion `FileCreate` die Funktion `FileCreateNewVersion` zu verwenden. Darüber hinaus ist das Vorgehen identisch.

```
ulong UploadFile (WebserviceClient wsClient, string database, ulong docID, ulong fileID,
                          ulong fileTypeID, string fileDescription, Stream input)
{
  FileCreateResponse createResponse = wsClient.FileCreate(database, docID, fileID, fileTypeID, fileDescription);
  string fileHandle = createResponse.FileHandle;
  ulong newFileID = createResponse.NewFileID;
  byte[] buffer = new byte[100000];
  bool readMore = true;
  while( readMore )
  {
    int readBytes = input.Read(buffer, 0, buffer.length);
    if(readBytes > 0)
      wsClient.FileWrite(fileHandle, buffer, 0, readBytes);
    else readMore = false;
```

```
  }
  wsClient.FileClose(fileHandle);
  return newFileID;
}
```

3.3. Datenkontrakte

Dieser Abschnitt beschreibt alle Datenkontrakte des Webservice, d.h. alle Datentypen, die kein Parameter einer Webservice-Operation sind. Die Beschreibung gliedert sich – analog zu der Beschreibung von Nachrichten und Operationen – in verscheidene Bereiche:

1. **Webservice-spezifische Datentypen** (Abschnitt 3.3.1)**.** Datentypen, die kein Element des Datenmodells von PROXESS sind.
2. **Datentype für Metadaten** (Abschnitt 3.3.2)**.** Datentypen, die PROXESS Metadaten repräsentieren.
3. **Datenkontrakte für Dokumente und Dateien** (Abschnitt 3.3.3)**.** Datentypen, die PROXESS Dokumente und PROXESS Dateien repräsentieren.
4. **Datenkontrakte für Suchen und Suchbedingunen** (Abschnitt 3.3.4)**.** Datentypen, die zur Spezifikation von Suchabfragen und zur Anzeige von Ergebnissen verwendet werden.

Auch für Datenkontrakte gilt die für Nachrichten und Operationen (Abschnitt 3.2) gemachte Anmerkung, dass alle optionalen XML-Elemente immer vorhanden sind, sofern sie aus einer Response vom Webservice stammen. Die Dokumentation geht explizit auf alle diejenigen XML-Elemente ein, für die der Wert `null` eine besondere Bedeutung hat. Ansonsten ist davon auszugehen, dass der Wert eines XML-Elements niemals `null` sein darf bzw. vom Webservice niemals als `null` zurückgegeben wird.

Anmerkung zu `[...]Collection`-Datentypen. Die WSDL-Datei des Webservice wird aus Klassendefinitionen erzeugt. Das entspricht dem Code-first Ansatz[5] zur Realisierung von SOAP-Webserivces. Alle Datentypen, deren Name mit `Collection` ended sind als spezielle Listenklasse definiert, z.B. `DatabaseCollection` ist von `List<Database>` abgeleitet. Die aus diesem Muster erzeugten WSDL-Definitionen haben eine einheitliche Gestalt, die in Listing 3.45 am Beispiel von `DatabaseCollection` verdeutlicht wird.

```
1 <xs:complexType name="DatabaseCollection">
2  <xs:sequence>
3   <!-- Liste von Datenbanken. -->
4   <xs:element minOccurs="0" maxOccurs="unbounded" name="Database" nillable="true"
5     type="q9:Database"/>
6  </xs:sequence>
7 </xs:complexType>
```

Listing 3.45: Typische WSDL-Definition für einen `Collection`-Typ

Tabelle 3.1 gibt einen Überblick über alle Collection-Datentypen. Alle Datentypen werden in Responses vom Webservice verwendet.

[5] Im Gegensatz zum Contract-first Ansatz, bei dem zuerst die WSDL-Datei geschrieben wird.

Collection-Typ	Element-Typ
DatabaseCollection	Database (3.3.2.1)
DocumentTypeCollection	DocumentType (3.3.2.2)
FieldDescriptionCollection	FieldDescription (3.3.2.3)
FileTypeCollection	FileType (3.3.2.4)
UserCollection	User (3.3.2.5)
GroupCollection	Group (3.3.2.5)
ValidationRuleCollection	ValidationRule (3.3.2.6)
FileCollection	FileDescriptor (3.3.3.3)
FieldCollection	Field (3.3.3.2)
DocumentLinkCollection	DocumentLink (3.3.3.4)
FileHistoryCollection	FileHistoryInfo (3.3.3.3)
ResultHitCollection	ResultHit (3.3.4.1)
ExtendedResultHitCollection	Document (3.3.3.1)

Tabelle 3.1.: Überblick über alle [...]Collection-Datentypen

Collection-Typ	Element-Typ
ArrayOfFieldChange	FieldChange (3.3.3.5)
ArrayOfDocumentLinkChange	DocumentLinkChange (3.3.3.5)
ArrayOfSearchCondition	SearchCondition (3.3.4.2)
ArrayOfunsignedLong	unsigned long
ArrayOfstring	String

Tabelle 3.2.: Überblick über alle [...]Collection-Datentypen

Die ersten sieben Zeilen der Tabelle sind Listen von Datentypen zur Darstellung von PROXESS Metadaten. Die folgenden vier Zeilen sind Listentypen, die zur Darstellung von Dokumenten verwendet werden (Dokument-Dateien (FileDescriptor), Feldwerte (Field), Querverweise (DocumentLink) und alte Versionen von Dateien (FileHistoryInfo)). Die letzte Zeile ist ein Listentyp, der zur Darstellung von Suchergebnissen (ResultHit) verwendet wird.

Anmerkung zu ArrayOf[...]-Datentypen. Es gibt eine zweite Art von Collection-Typen, bei denen der Name immer mit ArrayOf beginnt. Bis auf die Namensgebung unterscheiden sich die ArrayOf[...]-Datentypen nicht von den [...]Collection-Datentypen.

Tabelle 3.2 gibt einen Überblick über alle ArrayOf-Datentypen. Diese Datentypen werden in Requests beim Anlegen oder Ändern von Dokumenten (FieldChange und DocumentLinkChange) oder bei der Spezifikation von Suchbedingungen (SearchCondition) verwendet. In Responses werden ArrayOf-Datentypen beim Abrufen von Benutzern und Gruppen (User, Group) und bei Thesaurus-Validierungsregeln (Thesaurus) verwendet.

3.3.1. Webservice-spezifische Datentypen

3.3.1.1. AuthenticationType

`AuthenticationType` ist eine Enumeration, mit der die Art der Benutzeranmeldung in PROXESS spezifiziert wird. Listing 3.46 zeigt die WSDL-Definition des Datentyps `AuthenticationType`.

```
<xs:simpleType name="AuthenticationType">
 <xs:restriction base="xs:string">
  <!-- "Klassische" Benutzeranmeldung via Benutzername und Passwort. -->
  <xs:enumeration value="PROXESS"/>
  <!-- Benutzeranmeldung über ActiveDirectory. -->
  <xs:enumeration value="Windows"/>
 </xs:restriction>
</xs:simpleType>
```

Listing 3.46: Datentyp AuthenticationType

3.3.1.2. LoginToken

Das `LoginToken` wird benutzt, um dem Client eine PROXESS Session zuzuordnen. Zur Zeit werden lediglich `LoginToken` akzeptiert, die vom Webservice in der Response der `LogIn`-Funktion zurückgegeben wurden. Listing 3.47 zeigt die WSDL-Definition des Datentyps `LoginToken`.

```
<xs:complexType name="LoginToken">
 <xs:sequence>
  <!-- Eindeutiger Session-Identifier, der vom Webservice erteilt wird. -->
  <xs:element minOccurs="0" name="Id" nillable="true" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

Listing 3.47: Datentyp LoginToken

3.3.1.3. ReferableElement

`ReferableElement` ist der XML-Basistyp jedes Metadatentyps.

Listing 3.48 zeigt die WSDL-Definition des Datentyps `ReferableElement`.

```
<xs:complexType name="ReferableElement">
 <xs:sequence/>
 <!-- Wenn Id nicht null ist, dann handelt es sich bei diesem XML-Element um das
      serialisierte Objekt. Der Wert von Id wir in anderen XML-Elementen als Verweise
      auf dieses Element verwendet.
      Wenn Id null ist, dann handelt es sich bei diesem XML-Element um eine Referenz. -->
 <!-- defined as <xs:attribute name="Id" type="xs:ID"/> -->
 <xs:attribute ref="ser:Id"/>
 <!-- Wenn Ref nicht null ist, dann handelt es sich bei diesem XML-Element um eine Referenz.
      Der Wert von Ref entspricht der Id des tatsächlichen Objektes.
      Wenn Ref null ist, dann ist dieses XML-Element keine Referenz. -->
 <!-- definded as <xs:attribute name="Ref" type="xs:IDREF"/> -->
 <xs:attribute ref="ser:Ref"/>
</xs:complexType>
```

Listing 3.48: Datentyp ReferableElement

Anmerkung: Interop. `ReferableElement` wird verwendet, um die (WCF-spezifische) `IsReference`-Eigenschaft[6] eines DataContracts einheitlich zu kennzeichnen. Mit der `IsReference`-Eigenschaft werden Referenzen eines Objekt-Graphen im SOAP-XML korrekt wiedergegeben[7].

Diese Eigenschaft wird vor allem für Ergebnislisten verwendet, um die Dokumenttyp-Informationen (der einzelnen Treffer) nur einmal übertragen zu müssen.

Auf dieses technische Detail wird an dieser Stelle eingegangen, weil die korrekte Deserialisierung der Objekt-Referenzen für andere Plattformen als .NET nicht automatisch funktioniert. Clients auf anderen Platformen – z.B. Java – müssen die Objekt-Referenzen selbst auflösen.

.NET-Clients müssen diese Besonderheit nicht beachten, da die .NET-Runtime den Objekt-Graph bei der Deserialisierung automatisch korrekt aufbaut.

3.3.1.4. RenderInfo

`RenderInfo` gibt dem Anwender Informationen darüber, ob eine Datei beim Herunterladen von der Komponente PROXESS Web Rendr in eine PDF-Datei gerendert wurde.

[6] DataContractAttribute.IsReference [http://msdn.microsoft.com/de-de/library/system.runtime.serialization.datacontractattribute.isreference(v=vs.100).aspx]

[7] Das XML-technische Verfahren dafür wird z.B. hier [http://www.w3.org/TR/xmlschema-0/#schemaConstraintsVsXML1] erläutert.

Listing 3.49 zeigt die WSDL-Definition des Datentyps `RenderInfo`.

```
<xs:complexType name="RenderInfo">
 <xs:sequence>
   <!-- Gibt an, ob das Rendering für den Webservice überhaupt aktiviert ist. -->
   <xs:element minOccurs="0" name="IsRenderingEnabled" type="xs:boolean"/>
   <!-- Gibt an, ob diese Datei gerendert worden ist. -->
   <xs:element minOccurs="0" name="IsFileRendered" type="xs:boolean"/>
   <!-- Gibt die Dateierweiterung an, unter der die Datei gespeichert werden
       sollte. Wurde die Datei gerendert, ist dies die Erweiterung 'pdf'.
              Ansonsten wird die Dateierweiterung des zugehörigen PROXESS-Dateityps
              angegeben. -->
   <xs:element minOccurs="0" name="Extension" nillable="true" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
```

Listing 3.49: Datentyp RenderInfo

3.3.2. Metadaten

3.3.2.1. Database

Listing 3.50 zeigt die WSDL-Definition des Datentyps `Database`.

```
<xs:complexType name="Database">
 <xs:complexContent mixed="false">
  <!-- XML-Element kann als Referenz gespeichert werden (ID/IDREF). -->
  <xs:extension base="q11:ReferableElement">
   <xs:sequence>
    <!-- Name der Datenbank. -->
    <xs:element minOccurs="0" name="Name" nillable="true" type="xs:string"/>
    <!-- Beschreibung der Datenbank. -->
    <xs:element minOccurs="0" name="Description" nillable="true" type="xs:string"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

Listing 3.50: Datentyp Database

3.3.2.2. DocumentType

Listing 3.51 zeigt die WSDL-Definition des Datentyps `DocumentType`.

```
<xs:complexType name="DocumentType">
 <xs:complexContent mixed="false">
  <!-- XML-Element kann als Referenz gespeichert werden (ID/IDREF). -->
  <xs:extension base="q18:ReferableElement">
   <xs:sequence>
    <!-- Eindeutige ID des Dokumenttyps. -->
    <xs:element minOccurs="0" name="Id" type="xs:unsignedLong"/>
    <!-- Name des Dokumenttyps. -->
    <xs:element minOccurs="0" name="Name" nillable="true" type="xs:string"/>
    <!-- Verweis auf die Datenbank, in der dieser Dokumenttyp definiert ist. -->
    <xs:element minOccurs="0" name="Database" nillable="true" type="q18:Database"/>
    <!-- Liste von Felddefinitionen/-beschreibungen. -->
    <xs:element minOccurs="0" name="Fields" nillable="true" type="q18:FieldDescriptionCollection"/>
    <!-- Standard-Medientyp für diesen Dokumenttyp. -->
    <xs:element minOccurs="0" name="DefaultMediaType" type="q18:SMMediaType"/>
    <!-- Standard Lebensdauer für Dokumente dieses Typs. -->
    <xs:element minOccurs="0" name="DefaultLifetimeInDays" type="xs:int"/>
    <!-- Dieser Dokumenttyp ist eine Zwischenablage. -->
    <xs:element minOccurs="0" name="IsPool" type="xs:boolean"/>
    <!-- Volltext-Indexierung ist für diesen Dokumenttyp aktiviert. -->
    <xs:element minOccurs="0" name="IsFulltextActive" type="xs:boolean"/>
    <!-- Ein OCR-Modus ist für diesen Dokumenttyp spezifiziert. -->
    <xs:element minOccurs="0" name="IsOcrActive" type="xs:boolean"/>
    <!-- Dieser Dokumenttyp ist verifiziert (Datenintegritätsprüfung). -->
    <xs:element minOccurs="0" name="IsVerified" type="xs:boolean"/>
    <!-- Dokumente dieses Typs werden verschlüsselt gespeichert. -->
    <xs:element minOccurs="0" name="IsEncrypted" type="xs:boolean"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

Listing 3.51: Datentyp DocumentType

Die folgende Tabelle führt Verweise zu allen in Listing 3.51 referenzierten Datentypen auf.

Feldname	Verweis
`FieldDescription`	3.3.2.3

Enumeration SMMediaType.

Listing 3.52 zeigt die WSDL-Definition des Datentyps `SMMediaType`.

```
1  <xs:simpleType name="SMMediaType">
2   <xs:restriction base="xs:string">
3    <!-- Speichermedium CD/DVD/Blu Ray. -->
4    <xs:enumeration value="CD"/>
5    <!-- Speichermedium Festplatte. -->
6    <xs:enumeration value="Disk"/>
7    <!-- Obsolet. -->
8    <xs:enumeration value="Worm"/>
9    <!-- Obsolet. -->
10   <xs:enumeration value="Eod"/>
11   <!-- Obsolet. -->
12   <xs:enumeration value="Stash"/>
13  </xs:restriction>
14 </xs:simpleType>
```

Listing 3.52: Datentyp SMMediaType

Die Werte `Worm`, `Eod` und `Stash` werden nicht mehr verwendet.

3.3.2.3. FieldDescription

Der Typ `FieldDescription` beschreibt die Metadaten eines Feldes, z.B. Feldname, Datentyp, Sichtbarkeit für den Benutzer. Dokumenttypen definieren die für Dokumente verfügbaren Felder. Pro Dokument kann es für jedes dieser Felder einen Wert geben.

Der Zugriff auf Feldbeschreibungen ist über den Dokumenttyp (`DocumentType`, Abschnitt 3.3.2.2) und über jedes Feld eines Dokumentes (`Field`, Abschnitt 3.3.3.2) möglich.

Listing 3.53 zeigt die WSDL-Definition des Datentyps `FieldDescription`.

```
<xs:complexType name="FieldDescription">
 <xs:complexContent mixed="false">
  <!-- XML-Element kann als Referenz gespeichert werden (ID/IDREF). -->
  <xs:extension base="q22:ReferableElement">
   <xs:sequence>
    <!-- Id des Feldes (nur innerhalb der Datenbank eindeutig). -->
    <xs:element minOccurs="0" name="Id" type="xs:int"/>
    <!-- Physischer Name des Feldes in der Datenbank.
        Wird als eindeutiger Bezeichner verwendet. -->
    <xs:element minOccurs="0" name="DatabaseRolename" nillable="true" type="xs:string"/>
    <!-- Dokumenttyp-spezifischer Anzeigename für das Feld. -->
    <xs:element minOccurs="0" name="Rolename" nillable="true" type="xs:string"/>
    <!-- Beschreibung des Feldes. -->
    <xs:element minOccurs="0" name="Description" nillable="true" type="xs:string"/>
    <!-- Informationen zum Datentyp und Validierungsregeln. -->
    <xs:element minOccurs="0" name="DataFormat" nillable="true" type="q22:FieldDataFormat"/>
    <!-- Informationen zum Layout in der Standard-Dokumentmaske. -->
    <xs:element minOccurs="0" name="Layout" nillable="true" type="q22:FieldLayout"/>
    <!-- Das Feld ist ein Standardfeld (keine Neudefinition durch den Dokumenttyp). -->
    <xs:element minOccurs="0" name="IsDefaultField" type="xs:boolean"/>
    <!-- Das Feld ist für den Benutzer sichtbar. -->
    <xs:element minOccurs="0" name="IsVisible" type="xs:boolean"/>
    <!-- Das Feld ist ein Pflichtfeld (d.h. das Feld darf nicht den Wert null haben). -->
    <xs:element minOccurs="0" name="IsMandatory" type="xs:boolean"/>
    <!-- Das Feld ist verifiziert (Datenintegrität von PROXESS geprüft). -->
    <xs:element minOccurs="0" name="IsVerified" type="xs:boolean"/>
    <!-- Werte dieses Feldes werden verschlüsselt in der Datenbank gespeichert. -->
    <xs:element minOccurs="0" name="IsEncrypted" type="xs:boolean"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

Listing 3.53: Datentyp FieldDescription

Die folgende Tabelle führt Verweise zu allen in Listing 3.53 referenzierten Datentypen auf.

Feldname	Verweis
`FieldDataFormat`	3.3.2.3
`FieldLayout`	3.3.2.3

Anmerkung. Als eindeutiger Bezeichner für Felder wird immer der `DatabaseRolename` verwendet (z.B. in Suchbedingungen oder beim Setzen von Feldwerten).

FieldDataFormat.

Listing 3.54 zeigt die WSDL-Definition des Datentyps `FieldDataFormat`.

```
1  <xs:complexType name="FieldDataFormat">
2   <xs:sequence>
3    <!-- Datentyp des Feldes. -->
4    <xs:element minOccurs="0" name="Type" type="q24:FieldDataType"/>
5    <!-- Maximale Länge des Feldes (bei String-Feldern). -->
6    <xs:element minOccurs="0" name="Length" type="xs:int"/>
7    <!-- Nicht verwendet. -->
8    <xs:element minOccurs="0" name="Precision" type="xs:int"/>
9    <!-- ID der Validierungsregel. -->
10   <xs:element minOccurs="0" name="ValidationRuleId" type="xs:unsignedLong"/>
11   <!-- Nicht verwendet. -->
12   <xs:element minOccurs="0" name="InputFormat" nillable="true" type="xs:string"/>
13  </xs:sequence>
14 </xs:complexType>
```

Listing 3.54: Datentyp FieldDataFormat

Die folgende Tabelle führt Verweise zu allen in Listing 3.54 referenzierten Datentypen auf.

Feldname	Verweis
`FieldDataType`	3.3.2.3

Achtung: Validierungsregeln können erst seit der Webservice-Version 1.6.0 abgerufen werden. Das Anlegen oder Ändern von Validierungsregeln ist mit den in Version 1.9.0 eingeführten administrativen Erweiterungen möglich.

Achtung: Der durch das XML-Element `Type` angegebene Datentyp ist maßgeblich für die Interpretation der Werte beim Setzen/Ändern von Feldwerten (Funktionen 3.2.4.2: `CreateDocument` und 3.2.4.3: `ChangeDocument`). Da alle Feldwerte vom Webservice als String ausgegeben und entgegengenommen werden, müssen Clients sicherstellen, dass die übergebenen Strings in den korrekten Datentyp konvertiert werden können.

Enumeration FieldDataType.

Listing 3.55 zeigt die WSDL-Definition des Datentyps `FieldDataType`.

```
<xs:simpleType name="FieldDataType">
 <xs:restriction base="xs:string">
  <!-- Feld mit Ganzahl-Werten (32-Bit, int). -->
  <xs:enumeration value="Int"/>
  <!-- Nicht verwendet. PROXESS gibt immer Varchar zurück. -->
  <xs:enumeration value="Char"/>
  <!-- Feld mit Textwerten (String).
       Maximale Feldlänge beachten! -->
  <xs:enumeration value="Varchar"/>
  <!-- Feld mit Gleitkommazahl-Werten (64-Bit, double precision). -->
  <xs:enumeration value="Float"/>
  <!-- Feld mit Datums-Werten. -->
  <xs:enumeration value="DateTime"/>
 </xs:restriction>
</xs:simpleType>
```

Listing 3.55: Datentyp FieldDataType

FieldLayout.

Listing 3.56 zeigt die WSDL-Definition des Datentyps `FieldLayout`.

```
<!-- Alle Beschreibung beziehen sich auf die Standard-Dokumentmaske. -->
<xs:complexType name="FieldLayout">
 <xs:sequence>
  <!-- Seite, auf der das Feld angezeigt wird. -->
  <xs:element minOccurs="0" name="Page" type="xs:int"/>
  <!-- Tabulator-Position des Feldes. -->
  <xs:element minOccurs="0" name="TabOrder" type="xs:int"/>
  <!-- Anzeigebereich für die Textbox, die den Feldwert anzeigt. -->
  <xs:element minOccurs="0" name="FieldBounds" nillable="true" type="q27:Rectangle"/>
  <!-- Anzeigebereich für das Label, das den Feldnamen anzeigt. -->
  <xs:element minOccurs="0" name="LabelBounds" nillable="true" type="q28:Rectangle"/>
 </xs:sequence>
</xs:complexType>
```

Listing 3.56: Datentyp FieldLayout

Die folgende Tabelle führt Verweise zu allen in Listing 3.56 referenzierten Datentypen auf.

Feldname	Verweis
`Rectangle`	3.3.2.3

Listing 3.57 zeigt die WSDL-Definition des Datentyps `Rectangle`.

Rectangle.

```
1  <xs:complexType name="Rectangle">
2   <xs:sequence>
3    <!-- X offset. -->
4    <xs:element minOccurs="0" name="X" type="xs:int"/>
5    <!-- Y offset. -->
6    <xs:element minOccurs="0" name="Y" type="xs:int"/>
7    <!-- Breite. -->
8    <xs:element minOccurs="0" name="Width" type="xs:int"/>
9    <!-- Höhe. -->
10   <xs:element minOccurs="0" name="Height" type="xs:int"/>
11  </xs:sequence>
12 </xs:complexType>
```

Listing 3.57: Datentyp Rectangle

3.3.2.4. FileType

Listing 3.58 zeigt die WSDL-Definition des Datentyps `FileType`.

```
<xs:complexType name="FileType">
 <xs:complexContent mixed="false">
  <!-- XML-Element kann als Referenz gespeichert werden (ID/IDREF). -->
  <xs:extension base="q36:ReferableElement">
   <xs:sequence>
    <!-- Id des Dateityps. -->
    <xs:element minOccurs="0" name="Id" type="xs:unsignedLong"/>
    <!-- Name des Dateityps. -->
    <xs:element minOccurs="0" name="Name" nillable="true" type="xs:string"/>
    <!-- Dateierweiterung des Dateityps. -->
    <xs:element minOccurs="0" name="Extension" nillable="true" type="xs:string"/>
    <!-- Verweis auf die Datenbank, in der der Dateityp definiert ist. -->
    <xs:element minOccurs="0" name="Database" nillable="true" type="q36:Database"/>
    <!-- Wenn nicht 0: ID der Vorlagendatei bei Neuanlage. -->
    <xs:element minOccurs="0" name="TemplateFileId" type="xs:unsignedLong"/>
    <!-- Dateityp ist "universell" (kein spezifischer Dateityp) -->
    <xs:element minOccurs="0" name="IsUniversal" type="xs:boolean"/>
    <!-- Volltext-Indexierung ist für diesen Dateityp aktiviert. -->
    <xs:element minOccurs="0" name="IsFulltextActive" type="xs:boolean"/>
    <!-- Art der Text-Extraktion für die Volltext-Indexierung. -->
    <xs:element minOccurs="0" name="FulltextType" type="q36:FulltextType"/>
    <!-- OCR Modus -->
    <xs:element minOccurs="0" name="OcrMode" type="q36:OcrMode"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

Listing 3.58: Datentyp FileType

Die folgende Tabelle führt Verweise zu allen in Listing 3.58 referenzierten Datentypen auf.

Feldname	Verweis
`FulltextType`	3.3.2.4

Enumeration FulltextType.

Listing 3.59 zeigt die WSDL-Definition des Datentyps `FulltextType`.

```
<xs:simpleType name="FulltextType">
 <xs:restriction base="xs:string">
  <!-- Volltext-Indexierung deaktiviert. -->
  <xs:enumeration value="NoFulltext"/>
  <!-- Datei liegt als Plain-Text vor. -->
  <xs:enumeration value="PlainText"/>
  <!-- Universelle Text-Extraktion verwenden. "Catch-All" -->
  <xs:enumeration value="Formatted"/>
  <!-- Dateityp ist Adobe PDF. -->
  <xs:enumeration value="Pdf"/>
  <!-- Dateityp ist ein Microsoft OpenXML (Office 2007 and later) Format. -->
  <xs:enumeration value="Office2007"/>
 </xs:restriction>
</xs:simpleType>
```

Listing 3.59: Datentyp FulltextType

3.3.2.5. Benutzer und Gruppen

Die Datentypen `User` und `Group` beschreiben PROXESS-Benutzer und -Gruppen.

Listing 3.60 zeigt die WSDL-Definition des Datentyps `User`.

```
<xs:complexType name="User">
  <xs:sequence>
    <!-- System-weit eindeutiger Identifier des Benutzers. -->
    <xs:element name="Id" type="xs:unsignedLong"/>
    <!-- Anmeldename des Benutzers. -->
    <xs:element name="Name" nillable="true" type="xs:string"/>
    <!-- Vollständiger Name des Benutzers. -->
    <xs:element name="FullName" nillable="true" type="xs:string"/>
    <!-- Zeitpunkt der letzten Anmeldung. -->
    <xs:element name="LastLogOn" type="xs:dateTime"/>
    <!-- Zeitpunkt der letzten fehlgeschlagenen Anmeldung. -->
    <xs:element name="LastFailedLogOn" type="xs:dateTime"/>
    <!-- Besondere Privilegien des Benutzers. -->
    <xs:element name="Privilege" type="q148:Privilege"/>
    <!-- Der Benutzeraccount ist deaktiviert. -->
    <xs:element name="AccountDisabled" type="xs:boolean"/>
    <!-- Zeitpunkt der letzten Änderung des Passworts. -->
    <xs:element name="PasswordLastChange" type="xs:dateTime"/>
    <!-- Das Passwort läuft niemals ab. -->
    <xs:element name="PasswordNeverExpires" type="xs:boolean"/>
    <!-- Die Datenintegrität des Benutzers wurde überprüft. -->
    <xs:element name="IsVerified" type="xs:boolean"/>
    <!-- Art der Benutzerauthentifizierung. -->
    <xs:element name="IdentityType" type="q149:IdentityType"/>
    <!-- ID des Benutzers in einem externen Authentifizierungssystem. -->
    <xs:element minOccurs="0" name="ExternalID" nillable="true" type="xs:string"/>
    <!-- ??? -->
    <xs:element minOccurs="0" name="SingleSignOnEnabled" type="xs:boolean"/>
    <!-- Liste von IDs aller Gruppen, in denen dieser Benutzer Mitglied ist. -->
    <xs:element name="GroupIDs" nillable="true" type="q150:ArrayOfunsignedLong"/>
  </xs:sequence>
</xs:complexType>
```

Listing 3.60: Datentyp User

Die folgende Tabelle führt Verweise zu allen in Listing 3.60 referenzierten Datentypen auf.

Feldname	Verweis
`Privilege`	3.3.2.5
`IdentityType`	3.3.2.5

Listing 3.61 zeigt die WSDL-Definition des Datentyps `Group`.

```
<xs:complexType name="Group">
  <xs:sequence>
    <!-- System-weit eindeutiger Identifier der Gruppe. -->
    <xs:element name="Id" type="xs:unsignedLong"/>
    <!-- Name der Gruppe. -->
    <xs:element name="Name" nillable="true" type="xs:string"/>
    <!-- Beschreibunh der Gruppe. -->
    <xs:element name="Description" nillable="true" type="xs:string"/>
    <!-- Besondere Privilegien der Gruppe. -->
    <xs:element name="Privilege" type="q156:Privilege"/>
    <!-- Datenintegrität der Gruppe wurde überprüft. -->
    <xs:element name="IsVerified" type="xs:boolean"/>
    <!-- Art der Benutzerauthentifizierung für alle Mitglieder der Gruppe. -->
    <xs:element name="IdentityType" type="q157:IdentityType"/>
    <!-- Identifier in einem externen Benutzersystem. -->
    <xs:element minOccurs="0" name="ExternalID" nillable="true" type="xs:string"/>
    <!-- Liste der IDs aller Mitgliedern der Gruppe. -->
    <xs:element name="UserIDs" nillable="true" type="q158:ArrayOfunsignedLong"/>
  </xs:sequence>
</xs:complexType>
```

Listing 3.61: Datentyp Group

Die folgende Tabelle führt Verweise zu allen in Listing 3.61 referenzierten Datentypen auf.

Feldname	Verweis
`Privilege`	3.3.2.5
`IdentityType`	3.3.2.5

Enumeration Privilege. Die Enumeration `Privilege` stellt besonderen Rechte eines Benutzers oder einer Gruppe dar.

Listing 3.62 zeigt die WSDL-Definition des Datentyps `Privilege`.

```
<xs:simpleType name="Privilege">
 <xs:restriction base="xs:string">
  <!-- Benutzer/Gruppe hat keine besonderen Rechte. -->
  <xs:enumeration value="None"/>
  <!-- Benutzer/Gruppe besitzt administrative Rechte. -->
  <xs:enumeration value="Admin"/>
  <!-- Benutzer/Gruppe besitzt Supervisor Privilegien. -->
  <xs:enumeration value="Supervisor"/>
 </xs:restriction>
</xs:simpleType>
```

Listing 3.62: Datentyp Privilege

Enumeration IdentityType. Die Enumeration `IdentityType` stellt das Authentifizierungssystem eines Benutzers oder einer Gruppe dar.

Listing 3.63 zeigt die WSDL-Definition des Datentyps `IdentityType`.

```
<xs:simpleType name="IdentityType">
  <xs:restriction base="xs:string">
    <!-- Benutzer wird/werden über Benutzername/Passwort authentifiziert. -->
    <xs:enumeration value="Proxess"/>
    <!-- Benutzer wird/werden über ActiveDirectory authentifiziert. -->
    <xs:enumeration value="Windows"/>
  </xs:restriction>
</xs:simpleType>
```

Listing 3.63: Datentyp IdentityType

3.3.2.6. Validierungsregeln

Der Datentyp `ValidationRule` stellt eine Validierungsregel für Felder dar.

Listing 3.64 zeigt die WSDL-Definition des Datentyps `ValidationRule`.

```
<xs:complexType name="ValidationRule">
  <xs:sequence>
    <!-- Eindeutiger Identifier der Validierungsregel. -->
    <xs:element name="Id" type="xs:unsignedLong"/>
    <!-- Name der Validierungsregel. -->
    <xs:element name="Name" nillable="true" type="xs:string"/>
    <!-- Typ der Validierungsregel. -->
    <xs:element name="ValidationType" type="q164:ValidationRuleType"/>
    <!-- Null für Thesaurus-VR; Andernfalls:
           Spezifiziert die untere und obere Schranke für eine Bereichs-VR. -->
    <xs:element minOccurs="0" name="ValidationRange" nillable="true"
        type="q165:ValidationRange"/>
    <!-- Null für Bereichs-Validierungsregeln; Andernfalls:
           Spezifiziert die Wortliste und weitere Eigenschaften der Thesaurus-VR. -->
    <xs:element minOccurs="0" name="Thesaurus" nillable="true" type="q166:Thesaurus"/>
  </xs:sequence>
</xs:complexType>
```

Listing 3.64: Datentyp ValidationRule

Die folgende Tabelle führt Verweise zu allen in Listing 3.64 referenzierten Datentypen auf.

Feldname	Verweis
`ValidationRuleType`	3.3.2.6
`ValidationRange`	3.3.2.6
`Thesaurus`	3.3.2.6

Anmerkung. Das Feld `ValidationRuleType` gibt an, welchen Typ eine Validierungregel hat. Der Wert dieses Feldes legt die weiteren Zugriffsmöglichkeiten fest. Hat das Feld `ValidationRuleType` den Wert `Thesaurus`, dann besitzt das Feld `Thesaurus` einen Wert und das Feld `ValidationRange` hat keinen Wert. Andernfalls (`IntegerRange`, `DatetimeRange` oder `DoubleRange`) besitzt das Feld `ValidationRange` einen Wert und das Feld `Thesaurus` hat keinen Wert.

Enumeration ValidationRuleType. Die Enumeration `ValidationRuleType` gibt den Typ einer Validierungsregel an.
Listing 3.65 zeigt die WSDL-Definition des Datentyps `ValidationRuleType`.

```
<xs:simpleType name="ValidationRuleType">
  <xs:restriction base="xs:string">
    <!-- Bereichs-Validierungsregel mit ganzen Zahlen. -->
    <xs:enumeration value="IntegerRange"/>
    <!-- Bereichs-Validierungsregel mit Datumsangaben. -->
    <xs:enumeration value="DatetimeRange"/>
    <!-- Bereichs-Validierungsregel für Gleitkommazahlen. -->
    <xs:enumeration value="DoubleRange"/>
    <!-- Thesaurus-Validierungsregel (explizit gültige Werte für ein String-Feld). -->
    <xs:enumeration value="Thesaurus"/>
  </xs:restriction>
</xs:simpleType>
```

Listing 3.65: Datentyp ValidationRuleType

ValidationRange. Der Datentyp `ValidationRange` gibt die untere und obere Grenze eine Bereichs-Validierungsregel (`IntegerRange`, `DatetimeRange` oder `DoubleRange`) an.
Listing 3.66 zeigt die WSDL-Definition des Datentyps `ValidationRange`.

```
<xs:complexType name="ValidationRange">
  <xs:sequence>
    <!-- Untere Schranke einer Bereichs-Validierungsregel. -->
    <xs:element name="Lower" nillable="true" type="xs:string"/>
    <!-- Obere Schranke einer Bereichs-Validierungsregel. -->
    <xs:element name="Upper" nillable="true" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Listing 3.66: Datentyp ValidationRange

Thesaurus. Der Datentyp `Thesaurus` beschreibt die Eigenschaften einer Thesaurus-Validierungsregel.

Listing 3.67 zeigt die WSDL-Definition des Datentyps `Thesaurus`.

```
<xs:complexType name="Thesaurus">
  <xs:sequence>
    <!-- Wortliste kann durch (Standard-)Benutzer aktualisiert werden. -->
    <xs:element name="UserUpdateAllowed" type="xs:boolean"/>
    <!-- Maximale Länge für Thesaurus-Wörter (max: 119). -->
    <xs:element name="MaxWordLength" type="xs:int"/>
    <!-- Definiert die Menge gültiger Feldwerte. -->
    <xs:element name="WordList" nillable="true" type="q170:ArrayOfstring"/>
  </xs:sequence>
</xs:complexType>
```

Listing 3.67: Datentyp Thesaurus

3.3.3. Dokumente und Dateien

Dieser Abschnitt beschreibt alle Datentypen, die für das Abrufen, Anlegen und Ändern von Dokumenten verwendet werden. Die ersten Abschnitte (3.3.3.1: `Document`, 3.3.3.2: `Field`, 3.3.3.3: `FileDescriptor` und 3.3.3.4: `DocumentLink`) beschreiben alle Datentypen, die in der Response der Funktion 3.2.4.1: `GetDocument` enthalten sind.

Der letzte Abschnitt 3.3.3.5 beschreibt die Datentypen, die bei der Anlage (Funktion 3.2.4.2: `CreateDocument`) bzw. Änderung (Funktion 3.2.4.3: `ChangeDocument`) eines Dokuments verwendet werden.

3.3.3.1. Document

Ein Dokument stellt die zentrale Dateneinheit von PROXESS dar. Jedes Dokument hat einen Typ (`DocumentType`), einige feste Felder (z.B. Dokumenttitel, Erstellungsdatum), eine Menge von Feldwerten (für alle in der Datenbank/-dem Dokumenttyp definierten Felder), eine Menge von Dateien und Querverweise zu anderen Dokumenten.

Listing 3.68 zeigt die WSDL-Definition des Datentyps `Document`.

```
<xs:complexType name="Document">
 <xs:sequence>
  <!-- Verweis auf den Typ des Dokuments. -->
  <xs:element minOccurs="0" name="DocumentType" nillable="true" type="q52:DocumentType"/>
  <!-- Eindeutiger Identifier des Dokuments. -->
  <xs:element minOccurs="0" name="Id" type="xs:unsignedLong"/>
  <!-- Beschreibung/Titel des Dokuments. -->
  <xs:element minOccurs="0" name="Description" nillable="true" type="xs:string"/>
  <!-- Name des Benutzers, der das Dokument erstellt hat. -->
  <xs:element minOccurs="0" name="Creator" nillable="true" type="xs:string"/>
  <!-- Erstellungsdatum. -->
  <xs:element minOccurs="0" name="CreateDate" type="xs:dateTime"/>
  <!-- Name des Benutzers, der das Dokument zuletzt geämdert hat. -->
  <xs:element minOccurs="0" name="Updater" nillable="true" type="xs:string"/>
  <!-- Datum der letzten Änderung. -->
  <xs:element minOccurs="0" name="UpdateDate" type="xs:dateTime"/>
  <!-- Liste von Feldwerten. -->
  <xs:element minOccurs="0" name="Fields" nillable="true" type="q53:FieldCollection"/>
  <!-- Liste aller Dateitypen, die von Dateien dieses Dokuments verwendet werden. -->
  <xs:element minOccurs="0" name="FilesTypes" nillable="true" type="q54:FileTypeCollection"/>
  <!-- Liste der Dateien des Dokuments. -->
  <xs:element minOccurs="0" name="Files" nillable="true" type="q55:FileCollection"/>
  <!-- Liste der Querverweise dieses Dokuments. -->
  <xs:element minOccurs="0" name="LinkedDocuments" nillable="true" type="q56:DocumentLinkCollection"/>
 </xs:sequence>
</xs:complexType>
```

Listing 3.68: Datentyp Document

Die folgende Tabelle führt Verweise zu allen in Listing 3.68 referenzierten Datentypen auf.

Feldname	Verweis
DocumentType	3.3.2.2
FieldCollection	3.3
FileTypeCollection	3.3
FileCollection	3.3
DocumentLinkCollection	3.3

In den folgenden Abschnitten (3.3.3.2, 3.3.3.3, 3.3.3.4) werden die inneren Datentypen eines Dokuments beschrieben. In Abschnitt 3.3.3.5 werden spezielle Datentypen für die Änderung von Feldwerten und Querverweisen dokumentiert.

3.3.3.2. Field

Ein `Field` beschreibt den Wert eines Dokumentfeldes und besitzt eine Referenz auf die Definition des Feldes im Dokumenttyp.

Listing 3.69 zeigt die WSDL-Definition des Datentyps `Field`.

```
<xs:complexType name="Field">
 <xs:sequence>
  <!-- Verweis auf die Feldbeschreibung des Dokumenttyps. -->
  <xs:element minOccurs="0" name="MetaData" nillable="true" type="q60:FieldDescription"/>
  <!-- Wert des Feldes. -->
  <xs:element minOccurs="0" name="Value" nillable="true" type="xs:string"/>
  <!-- Aus den Metadaten: Name des Feldes in der Datenbank. -->
  <xs:element minOccurs="0" name="DatabaseRolename" nillable="true" type="xs:string"/>
  <!-- Aus den Metadaten: Rollenname des Feldes für den Dokumenttyp. -->
  <xs:element minOccurs="0" name="Rolename" nillable="true" type="xs:string"/>
  <!-- Aus den Metadaten: Datentyp des Feldes. -->
  <xs:element minOccurs="0" name="DataType" type="q61:FieldDataType"/>
 </xs:sequence>
</xs:complexType>
```

Listing 3.69: Datentyp Field

Die folgende Tabelle führt Verweise zu allen in Listing 3.69 referenzierten Datentypen auf.

Feldname	Verweis
`FieldDescription`	3.3.2.3

Anmerkung: Aus den Metadaten wurden der Datenbank- und Rollen-Name sowie der Datentyp des Feldes direkt in das XML-Element `Field` aufgenommen, um einen schnellen Zugriff (ohne Umweg über `ReferableElement` bzw. ID/IDREF; siehe Abschnitt 3.3.1.3: `ReferableElement`) auf die wichtigsten Informationen aus den Metadaten zu erlauben.

3.3.3.3. **FileDescriptor**, FileVersionInfo, FileCheckOutInfo, FileHistoryInfo

Ein `FileDescriptor` beschreibt die Metadaten einer PROXESS-Datei. Eine Datei wird beim Download/Upload über seine ID identifiziert.

Listing 3.70 zeigt die WSDL-Definition des Datentyps `FileDescriptor`.

```
<xs:complexType name="FileDescriptor">
 <xs:sequence>
  <!-- ID der Datei. -->
  <xs:element minOccurs="0" name="Id" type="xs:unsignedLong"/>
  <!-- Verweis auf den Typ dieser Datei. -->
  <xs:element minOccurs="0" name="Type" nillable="true" type="q65:FileType"/>
  <!-- Beschreibung der Datei. -->
  <xs:element minOccurs="0" name="Description" nillable="true" type="xs:string"/>
  <!-- Länge der Datei in Bytes. -->
  <xs:element minOccurs="0" name="Length" type="xs:unsignedLong"/>
  <!-- Information zur aktuellen Version der Datei. -->
  <xs:element minOccurs="0" name="VersionInfo" nillable="true" type="q66:FileVersionInfo"/>
  <!-- Information zum Check-Out Status der Datei. -->
  <xs:element minOccurs="0" name="CheckOutInfo" nillable="true" type="q67:FileCheckOutInfo"/>
  <!-- Information zu alten Versionen dieser Datei. -->
  <xs:element minOccurs="0" name="History" nillable="true" type="q68:FileHistoryCollection"/>
  <!-- Information für externe Storage-Systeme. -->
  <xs:element minOccurs="0" name="ExternInfo" nillable="true" type="xs:string"/>
  <!-- [Deprecated] Länge der Datei in Bytes (liefert den gleichen Wert wie Length). -->
  <xs:element minOccurs="0" name="LengthDb" nillable="true" type="xs:unsignedLong"/>
 </xs:sequence>
</xs:complexType>
```

Listing 3.70: Datentyp FileDescriptor

Die folgende Tabelle führt Verweise zu allen in Listing 3.70 referenzierten Datentypen auf.

Feldname	Verweis
FileType	3.3.2.4
FileVersionInfo	3.3.3.3
FileCheckOutInfo	3.3.3.3
FileHistoryCollection	3.3
FileHistoryInfo	3.3.3.3

Listing 3.71 zeigt die WSDL-Definition des Datentyps `FileVersionInfo`.

```
<xs:complexType name="FileVersionInfo">
 <xs:sequence>
  <!-- Hauptversionsnummer. -->
  <xs:element minOccurs="0" name="MajorVersion" type="xs:int"/>
  <!-- Unterversionsnummer. -->
  <xs:element minOccurs="0" name="MinorVersion" type="xs:int"/>
  <!-- Kommentar zur Dateiversion. -->
  <xs:element minOccurs="0" name="Comment" nillable="true" type="xs:string"/>
  <!-- Name des Benutzers, der diese Dateiversion erzeugt hat. -->
  <xs:element minOccurs="0" name="Creator" nillable="true" type="xs:string"/>
  <!-- Datum der Erstellung dieser Dateiversion. -->
  <xs:element minOccurs="0" name="CreateDate" type="xs:dateTime"/>
 </xs:sequence>
</xs:complexType>
```

Listing 3.71: Datentyp FileVersionInfo

Listing 3.72 zeigt die WSDL-Definition des Datentyps `FileCheckOutInfo`.

```
<xs:complexType name="FileCheckOutInfo">
 <xs:sequence>
  <!-- Die Datei ist aktuell ausgecheckt. -->
  <xs:element minOccurs="0" name="IsCheckedOut" type="xs:boolean"/>
  <!-- Check-Out Kommentar. -->
  <xs:element minOccurs="0" name="Comment" nillable="true" type="xs:string"/>
  <!-- Name des Computers, auf dem die Datei ausgecheckt wurde. -->
  <xs:element minOccurs="0" name="Host" nillable="true" type="xs:string"/>
  <!-- Pfadangabe, wohin die Datei ausgecheckt worden ist. -->
  <xs:element minOccurs="0" name="Path" nillable="true" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

Listing 3.72: Datentyp FileCheckOutInfo

Listing 3.73 zeigt die WSDL-Definition des Datentyps `FileHistoryInfo`.

```
<xs:complexType name="FileHistoryInfo">
 <xs:sequence>
  <!-- ID dieser Dateiversion. -->
  <xs:element minOccurs="0" name="Id" type="xs:unsignedLong"/>
  <!-- Beschreibung der Dateiversion. -->
  <xs:element minOccurs="0" name="Description" nillable="true" type="xs:string"/>
  <!-- Name des Benutzers, der diese Version der Datei erstellt hat. -->
  <xs:element minOccurs="0" name="CreatorId" type="xs:unsignedLong"/>
  <!-- Versionnummer dieser Dateiversion. -->
  <xs:element minOccurs="0" name="VersionInfo" nillable="true" type="q74:FileVersionInfo"/>
  <!-- Status der Datei. Siehe PROXESS Dokumentation. -->
  <xs:element minOccurs="0" name="FileStatus" type="xs:int"/>
  <!-- Wenn nicht 0: ID des Benutzers, der diese Version der Datei ausgecheckt hat. -->
  <xs:element minOccurs="0" name="CheckedOutByUser" type="xs:unsignedLong"/>
 </xs:sequence>
</xs:complexType>
```

Listing 3.73: Datentyp FileHistoryInfo

3.3.3.4. DocumentLink

Ein Querverwies ist ein unidirektionaler Verweis auf ein Dokument in derselben Datenbank.
Listing 3.74 zeigt die WSDL-Definition des Datentyps `DocumentLink`.

```
<xs:complexType name="DocumentLink">
 <xs:sequence>
  <!-- ID des Dokuments, auf das verwiesen wird. -->
  <xs:element minOccurs="0" name="DocumentID" type="xs:unsignedLong"/>
  <!-- Name des Dokumenttyps Dokuments, auf das verwiesen wird. -->
  <xs:element minOccurs="0" name="DocumentTypeName" nillable="true" type="xs:string"/>
  <!-- Titel/Beschreibung des Dokuments, auf das verwiesen wird. -->
  <xs:element minOccurs="0" name="DocumentDescription" nillable="true" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

Listing 3.74: Datentyp DocumentLink

3.3.3.5. Datentypen für Dokumentänderungen

Die Datentypen für Dokumentänderungen werden beim Anlegen eines Dokumentes (siehe Funktion 3.2.4.2: `CreateDocument`) oder beim Verändern eines Dokumentes (siehe Funktion 3.2.4.3: `ChangeDocument`) verwendet.

FieldChange. Der Datentyp `FieldChange` wird verwendet, um einen Feldwert zu setzen.
Listing 3.75 zeigt die WSDL-Definition des Datentyps `FieldChange`.

```
<xs:complexType name="FieldChange">
 <xs:sequence>
  <!-- Datenbankname des Feldes, dessen Wert gesetzt werden soll. -->
  <xs:element minOccurs="0" name="DatabaseRolename" nillable="true" type="xs:string"/>
  <!-- Wert, der gesetzt werden soll.
      null bedeutet, dass der Feldwert auf no_value (kein Wert) gesetzt wird. -->
  <xs:element minOccurs="0" name="Value" nillable="true" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

Listing 3.75: Datentyp FieldChange

Achtung: Beim Setzen oder Feldwerte sind die Anmerkungen zu Funktion 3.2.4.2: `CreateDocument` zu beachten.

DocumentLinkChange. Der Datentyp `DocumentLinkChange` wird verwendet, um Dokumentverknüpfungen hinzuzufügen oder zu löschen.
Listing 3.76 zeigt die WSDL-Definition des Datentyps `DocumentLinkChange`.

```
<xs:complexType name="DocumentLinkChange">
 <xs:sequence>
  <!-- Gibt an, ob ein Querverweis hinzugefügt oder entfernt werden soll. -->
  <xs:element minOccurs="0" name="Operation" type="q48:DocumentLinkOperation"/>
  <!-- Gibt die ID des Dokuments an, auf das der zu ändernde Querverweis zeigt.
      Der Querverweis wird durch die ID des verknüpften Dokumentd identifiziert. -->
  <xs:element minOccurs="0" name="LinkedDocumentID" type="xs:unsignedLong"/>
 </xs:sequence>
</xs:complexType>
```

Listing 3.76: Datentyp DocumentLinkChange

Die folgende Tabelle führt Verweise zu allen in Listing 3.76 referenzierten Datentypen auf.

Feldname	Verweis
`DocumentLinkOperation`	3.3.3.5

Achtung: Beim Hinzufügen oder Löschen von Querverweisen sind die Anmerkungen zu Funktion 3.2.4.2: `CreateDocument` zu beachten.

Enumeration DocumentLinkOperation. Die Enumeration `DocumentLinkOperation` spezifiziert die Aktion, die mit einem Querverweis durchgeführt werden soll.

Listing 3.77 zeigt die WSDL-Definition des Datentyps `DocumentLinkOperation`.

```
<xs:simpleType name="DocumentLinkOperation">
 <xs:restriction base="xs:string">
  <!-- Querverweis wird hinzugefügt. -->
  <xs:enumeration value="Create"/>
  <!-- Querverweis wird entfernt. -->
  <xs:enumeration value="Remove"/>
 </xs:restriction>
</xs:simpleType>
```

Listing 3.77: Datentyp DocumentLinkOperation

3.3.4. Suchbedingungen, Abfragen und Ergebnisse

Dieser Abschnitt beschreibt alle Datentypen, die zur Formulierung von Abfragen (3.3.4.1: `GeneralSearch` und 3.3.4.2: `SearchCondition`) und zur Darstellung von Ergebnislisten (B.2.0.3: `SearchResult`) verwendet werden.

3.3.4.1. GeneralSearch

`GeneralSearch` implementiert die erweiterte Suche von PROXESS. Eine Suche besteht im wesentlichen aus der Datenbank, in der gesucht wird, sowie aus einer (benötigten) Suchbedingung (`RootCondition`) und einem (optionalen) Sortierkriterium (`OrderBy`). Eine Suche ist immer auf eine bestimmte Datenbank eingeschränkt, d.h. Suchen über mehrere Datenbanken sind nicht möglich.

Zusätzlich werden Angaben über die maximal abzurufenden Treffer (`MaxHits`) und der Größe eines einzelnen Ergebnis-Blocks (`HitsPerPage`) gemacht.

Listing 3.78 zeigt die WSDL-Definition des Datentyps `GeneralSearch`.

```
<xs:complexType name="GeneralSearch">
 <xs:sequence>
  <!-- Name der Datenbank, in der gesucht wird. -->
  <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
  <!-- Maximale Anzahl an Treffern, die zurückgegeben werden sollen. -->
  <xs:element minOccurs="0" name="MaxHits" type="xs:int"/>
  <!-- Größe eines einzelnen Chunks. -->
  <xs:element minOccurs="0" name="HitsPerPage" type="xs:int"/>
  <!-- Spezifiziert die Suchbedingung für die Suche. -->
  <xs:element minOccurs="0" name="RootCondition" nillable="true" type="q91:SearchCondition"/>
  <!-- Spezifiziert die Sortierung der Ergebnisse. -->
  <xs:element minOccurs="0" name="OrderBy" nillable="true" type="q92:OrderBy"/>
 </xs:sequence>
</xs:complexType>
```

Listing 3.78: Datentyp GeneralSearch

Die folgende Tabelle führt Verweise zu allen in Listing 3.78 referenzierten Datentypen auf.

Feldname	Verweis
`SearchCondition`	3.3.4.2
`OrderBy`	3.3.4.3

Anmerkung zu `HitsPerPage`. Der Wert für `HitsPerPage` sollte nicht zu groß gewählt werden, da sonst möglicherweise die maximale Nachrichtengröße des Webservice überschritten wird. Ein Wert in der Größenordnung von 100 ist ein guter Richtwert.

ResultHit. Ein `ResultHit` repräsentiert ein einzelnes Trefferdokument einer Ergebnisliste. Dokumenttitel, Dokumenttyp und Feldwerte werden übermittelt. Einige Informationen des Dokuments sind an dieser Stelle nicht verfügbar, z.B. die Dateiliste oder die Querverweisliste des Dokuments. Um diese Informationen zu erhalten, muss das Dokument abgerufen werden (siehe Funktion 3.2.4.1: `GetDocument`).

Listing 3.79 zeigt die WSDL-Definition des Datentyps `ResultHit`.

```
<xs:complexType name="ResultHit">
 <xs:sequence>
  <!-- Name der Datenbank, in der das Trefferdokument gespeichert ist. -->
  <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
  <!-- Verweis auf den Typ des Trefferdokuments. -->
  <xs:element minOccurs="0" name="DocumentType" nillable="true" type="q132:DocumentType"/>
  <!-- ID des Trefferdokuments. -->
  <xs:element minOccurs="0" name="DocumentId" type="xs:unsignedLong"/>
  <!-- Titlel/Beschreibung des Trefferdokuments. -->
  <xs:element minOccurs="0" name="DocumentDescription" nillable="true" type="xs:string"/>
  <!-- Status des Trefferdokuments. 0 = normal, 1 = gelöscht -->
  <xs:element minOccurs="0" name="DocumentState" type="xs:int"/>
  <!-- Feldwerte des Trefferdokuments. -->
  <xs:element minOccurs="0" name="Fields" nillable="true" type="q133:FieldCollection"/>
  <!-- Aus den Dokumenttyp-Metadaten: ID des Dokumenttyps. -->
  <xs:element minOccurs="0" name="DocTypeID" type="xs:unsignedLong"/>
  <!-- Aus den Dokumenttyp-Metadaten: Name des Dokumenttyps. -->
  <xs:element minOccurs="0" name="DocTypeName" nillable="true" type="xs:string"/>
  <!-- Name des Benutzers, der das Dokument erstellt hat. -->
  <xs:element minOccurs="0" name="Creator" nillable="true" type="xs:string"/>
  <!-- Anlagedatum des Dokuments. -->
  <xs:element minOccurs="0" name="CreateDate" nillable="true" type="xs:dateTime"/>
  <!-- Falls nicht NULL, Name des Benutzers, der das Dokument zuletzt aktualisiert hat. -->
  <!-- Falls NULL, wurde das Dokument noch nicht aktualisiert. -->
  <xs:element minOccurs="0" name="Updater" nillable="true" type="xs:string"/>
  <!-- Falls nicht NULL, Datum der letzten Änderung des Dokuments. -->
  <!-- Fall NULL, wurde das Dokument nach der Erstellung noch nicht geändert. -->
  <xs:element minOccurs="0" name="UpdateDate" nillable="true" type="xs:dateTime"/>
 </xs:sequence>
</xs:complexType>
```

Listing 3.79: Datentyp ResultHit

Die folgende Tabelle führt Verweise zu allen in Listing 3.79 referenzierten Datentypen auf.

Feldname	Verweis
`DocumentType`	3.3.2.2
`FieldCollection`	3.3
`Field`	3.3.3.2

Anmerkung: Der Dokumenttyp-Name und die Dokumenttyp-ID wurden direkt in das XML-Element `ResultHit` aufgenommen, um einen schnellen Zugriff (ohne Umweg über `ReferableElement` bzw. ID/IDREF; siehe Abschnitt 3.3.1.3: `ReferableElement`) auf die wichtigsten Informationen aus den Metadaten zu erlauben.

Anmerkung: Die Felder `Creator`, `CreateDate`, `Updater` und `UpdateDate` wurden mit Version 1.9.1 eingeführt.

MetadataContext. Der Datentyp `MetadataContext` speichert alle Metadaten einer Webservice-Response. Z.Zt wird dieser Datentyp nur für Suchergebnisse verwendet und speichert in diesem Fall alle Datenbanken und Dokumenttypen der Trefferdokumente. Die Trefferdokumente haben lediglich Referenzen auf die (im `MetadataContext` gespeicherten) Metadaten-Objekte (siehe auch 3.3.1.3: `ReferableElement`).
Listing 3.80 zeigt die WSDL-Definition des Datentyps `MetadataContext`.

```
1  <xs:complexType name="MetadataContext">
2   <xs:sequence>
3    <!-- Menge von Metadaten. -->
4    <xs:element minOccurs="0" name="Metadata" nillable="true" type="q124:MetadataContainer"/>
5   </xs:sequence>
6  </xs:complexType>
7
8  <xs:complexType name="MetadataContainer">
9   <xs:sequence>
10   <!-- Liste von Datenbanken. -->
11   <xs:element minOccurs="0" name="Databases" nillable="true" type="q126:DatabaseCollection"/>
12   <!-- Liste von Dokumenttypen. -->
13   <xs:element minOccurs="0" name="DocumentTypes" nillable="true" type="q127:DocumentTypeCollection"/>
14   <!-- Liste von Dateitypen. -->
15   <xs:element minOccurs="0" name="FileTypes" nillable="true" type="q128:FileTypeCollection"/>
16  </xs:sequence>
17 </xs:complexType>
```

Listing 3.80: Datentyp MetadataContext

Anmerkung: Dieser Bereich eines Suchergebnisses kann von Clients i.d.R. ignoriert werden. Dieses XML-Element wurde nur eingeführt, damit alle Metadaten an eine klar definierten Stelle in der SOAP-Nachricht auftauchen. Der ID/IDREF-Mechanismus ermöglicht es, ausgehend von den einzelnen Trefferdokumenten auf die korrekten Metadaten zuzugreifen.

3.3.4.2. SearchCondition

Dieser Abschnitt beschreibt die Struktur von Suchbedingungen und Sortierkriterien.

Die unterschiedlichen Elemente einer Suchbedingung werden durch verschiedene Klassen abgebildet, die allesamt von der abstrakten Basisklasse `SearchCondition` abgeleitet sind. Die `SearchCondition`-Klassenhierarchie implementiert das Kompositum-Pattern[8] wobei die Klasse `BranchCondition` das Kompositum ist und alle von `LeafCondition` abgeleiteten Klassen die Blätter sind. Diese Struktur wird in Abbildung 3.1 als UML-Klassendiagramm dargestellt.

Im Folgenden werden alle konkreten `SearchCondition`-Klassen beschrieben. Im Anhang A finden sich Code-Beispiele für das Zusammensetzen von Suchbedingungen.

[8] `http://de.wikipedia.org/wiki/Kompositum_(Entwurfsmuster)`

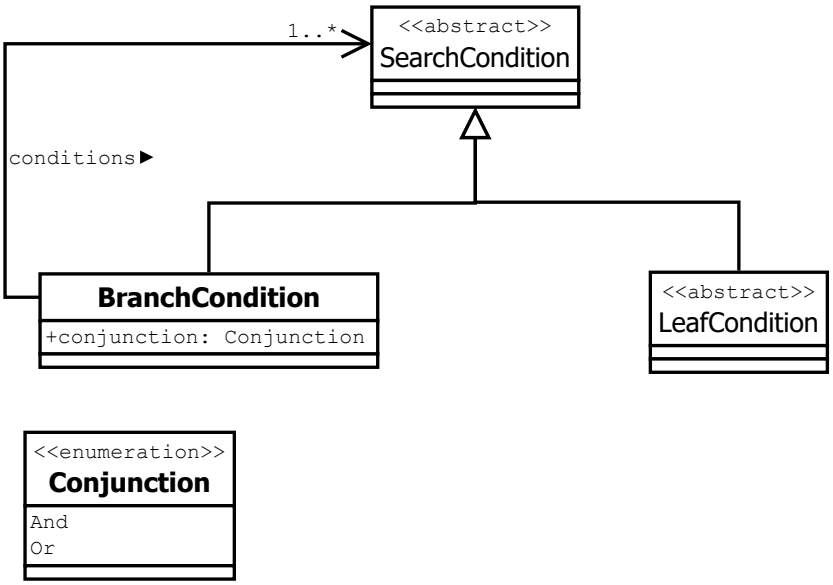

Abbildung 3.1.: `SearchCondition` setzt das Composite Pattern um

BranchCondition. Eine `BranchCondition` ist eine logische (UND-/ODER-)Verknüpfung von mehreren Suchbedingungen, d.h. (`B1 AND B2 AND ... AND BN`) oder (`B1 OR B2 OR ... OR BN`), wobei `B1` bis `BN` beliebige Suchbedingungen sind.
Listing 3.81 zeigt die WSDL-Definition des Datentyps `BranchCondition`.

```
1  <xs:complexType name="BranchCondition">
2   <xs:complexContent mixed="false">
3    <xs:extension base="q95:SearchCondition">
4     <xs:sequence>
5      <!-- Spezifiziert die Art der logischen Verknüpfung. -->
6      <xs:element minOccurs="0" name="Conjunction" type="q95:Conjunction"/>
7      <!-- Liste von Suchbedingungen die verknüpft werden. -->
8      <xs:element minOccurs="0" name="Conditions" nillable="true"
9        type="q95:ArrayOfSearchCondition"/>
10    </xs:sequence>
11   </xs:extension>
12  </xs:complexContent>
13 </xs:complexType>
```

Listing 3.81: Datentyp BranchCondition

Enumeration Conjunction. Die Enumeration `Conjunction` bestimmt den Typ einer `BranchCondition`.

Listing 3.82 zeigt die WSDL-Definition des Datentyps `Conjunction`.

```
1 <xs:simpleType name="Conjunction">
2  <xs:restriction base="xs:string">
3   <!-- Die Suchbedingungen werden UND-verknüpft. -->
4   <xs:enumeration value="And"/>
5   <!-- Die Suchbedingungen werden ODER-verknüpft. -->
6   <xs:enumeration value="Or"/>
7  </xs:restriction>
8 </xs:simpleType>
```

Listing 3.82: Datentyp Conjunction

LeafCondition. Abbildung 3.2 zeigt die von Blätter des `SearchCondition`-Kompositums als UML-Klassendiagramm. In den folgenden Abschnitten werden die einzelnen Klassen beschrieben.

Anmerkungen zu Feldern. Die meisten Suchbedingungen erforden einen Feldnamen für den die Suchbedingung gilt. Das kann entweder der Datenbankname eines im Dokumenttyp definierten Feldes sein, oder ein sog. Kernfeld des Dokuments. Folgende wichtige “versteckte” Feldnamen gibt es:

Feldname	Beschreibung
`DocDes`	Titel/Beschreibung des Dokuments.
`DocID`	ID des Dokuments.
`DocsDocTypeName`	Name des Dokumenttyps.
`DocTypeID`	ID des Dokumenttyps.
`Creator`	Name des Benutzers, der das Dokument erstellt hat.
`CreateDate`	Datum der Erstellung.
`Updater`	Name des Benutzers, der das Dokument zuletzt aktualisiert hat.
`UpdateDate`	Datum der letzten Aktualisierung.

FieldCondition. Eine `FieldCondition` spezifiziert eine Suchbedingung, die ein Feld mit einem vorgegebenen Wert in Relation setzt, z.B. `DocDes='Rechnung 134889'` oder `KdNr<100000`.

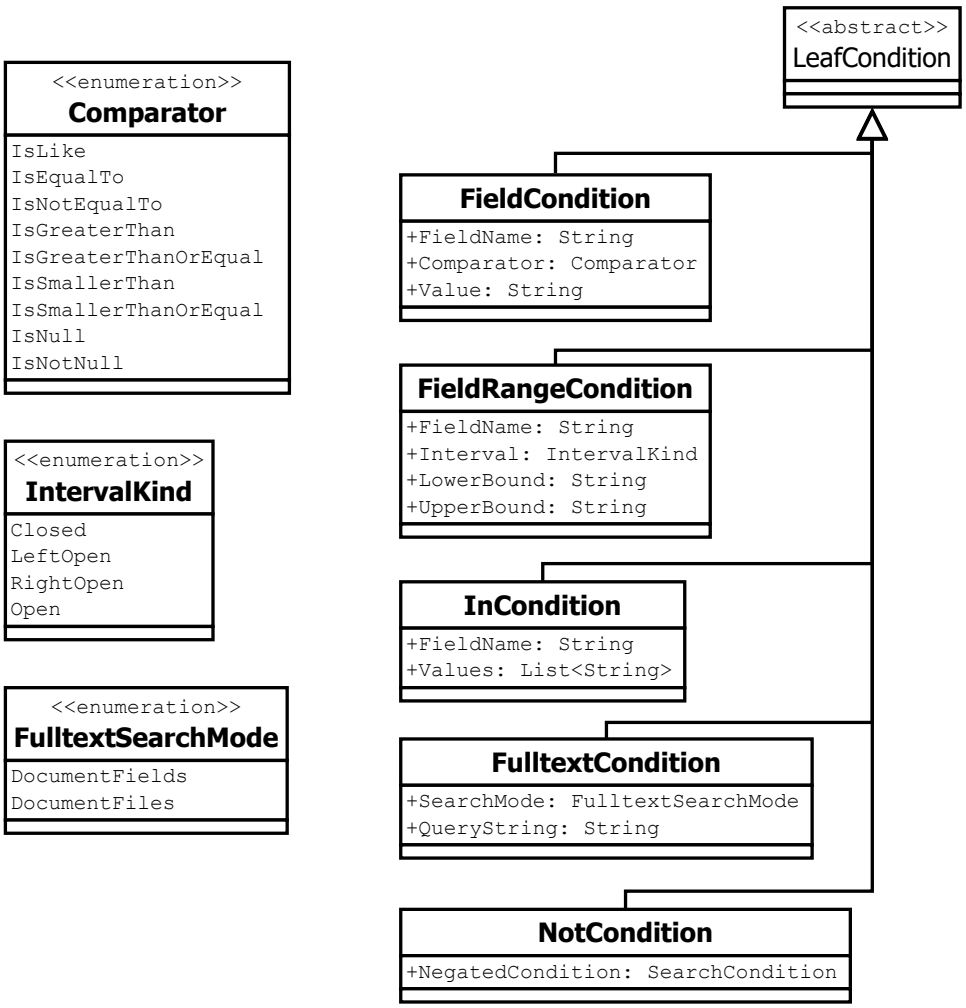

Abbildung 3.2.: Verschiedene Arten von Suchbedingungen

Listing 3.83 zeigt die WSDL-Definition des Datentyps `FieldCondition`.

```
<xs:complexType name="FieldCondition">
 <xs:complexContent mixed="false">
  <xs:extension base="q113:LeafCondition">
   <xs:sequence>
    <!-- Datenbankname des Feldes dieser Suchbedingung. -->
    <xs:element minOccurs="0" name="FieldName" nillable="true" type="xs:string"/>
    <!-- Vergleichsoperator. -->
    <xs:element minOccurs="0" name="Comparator" type="q113:Comparator"/>
    <!-- Vergleichswert. -->
    <xs:element minOccurs="0" name="Value" nillable="true" type="xs:string"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

Listing 3.83: Datentyp FieldCondition

Enumeration Comparator. Die Enumeration `Comparator` spezifiziert den Vergleichsoperator eine Suchbedingung.

Listing 3.84 zeigt die WSDL-Definition des Datentyps `Comparator`.

```
<!-- a -- Feldname der Suchbedingung.
    b -- Vergleichswert der Suchbedingung. -->
<xs:simpleType name="Comparator">
 <xs:restriction base="xs:string">
  <!-- a == b (SQL: a = b) -->
  <xs:enumeration value="IsEqualTo"/>
  <!-- a > b -->
  <xs:enumeration value="IsGreaterThan"/>
  <!-- a => b -->
  <xs:enumeration value="IsGreaterOrEqualThan"/>
  <!-- a < b -->
  <xs:enumeration value="IsSmallerThan"/>
  <!-- a <= b -->
  <xs:enumeration value="IsSmallerOrEqualThan"/>
  <!-- a LIKE b (SQL: LIKE mit % und _ Wildcards) -->
  <xs:enumeration value="IsLike"/>
  <!-- a != b (SQL: a <> b) -->
  <xs:enumeration value="IsNotEqualTo"/>
  <!-- a == null (SQL: a IS NULL) -->
  <xs:enumeration value="IsNull"/>
  <!-- a != null (SQL: a IS NOT NULL) -->
  <xs:enumeration value="IsNotNull"/>
 </xs:restriction>
</xs:simpleType>
```

Listing 3.84: Datentyp Comparator

FieldRangeCondition. Eine `FieldRangeCondition` spezifiert eine Suchbedingung, die ein Interval von gültigen Werten für ein Feld vorgibt, z.B. `(PLZ>=50000 AND PLZ<60000)`.[9]
Listing 3.85 zeigt die WSDL-Definition des Datentyps `FieldRangeCondition`.

```
<xs:complexType name="FieldRangeCondition">
 <xs:complexContent mixed="false">
  <xs:extension base="q100:LeafCondition">
   <xs:sequence>
    <!-- Datenbankname des Feldes für die Suchbedingung. -->
    <xs:element minOccurs="0" name="FieldName" nillable="true" type="xs:string"/>
    <!-- Spezifiziert die Art des Intervalls. -->
    <xs:element minOccurs="0" name="Interval" type="q100:IntervalKind"/>
    <!-- Untere Grenze des Intervalls. -->
    <xs:element minOccurs="0" name="Lower" nillable="true" type="xs:string"/>
    <!-- Obere Grenze des Intervalls. -->
    <xs:element minOccurs="0" name="Upper" nillable="true" type="xs:string"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

Listing 3.85: Datentyp FieldRangeCondition

`IntervalKind` spezifiziert die Art eines Intervals (rechts/links offen/geschlossen).
Listing 3.86 zeigt die WSDL-Definition des Datentyps `IntervalKind`.

```
<xs:simpleType name="IntervalKind">
 <xs:restriction base="xs:string">
  <!-- Geschlossenes Intervall [a,b] -->
  <xs:enumeration value="Closed"/>
  <!-- Links-offenes Intervall ]a,b] -->
  <xs:enumeration value="LeftOpen"/>
  <!-- Rechts-offenes Intervall [a,b[ -->
  <xs:enumeration value="RightOpen"/>
  <!-- Offenes Intervall ]a,b[ -->
  <xs:enumeration value="Open"/>
 </xs:restriction>
</xs:simpleType>
```

Listing 3.86: Datentyp IntervalKind

InCondition. Eine `InCondition` spezifiziert eine Suchbedingung für ein Feld, die mehrere gültige Werte für das Feld vorgibt, z.B. `ReNr IN (4556, 4563, 4564, 4670)`

[9] In den Beispielen werden SQL-Bedingungen angegeben. Eine `FieldRangeCondition` könnte ganz frei auch so geschreiben werden: `PLZ IN [50000, 60000[`

Listing 3.87 zeigt die WSDL-Definition des Datentyps `InCondition`.

```
<xs:complexType name="InCondition">
 <xs:complexContent mixed="false">
  <xs:extension base="q105:LeafCondition">
   <xs:sequence>
    <!-- Datenbankname des Feldes für die Suchbedingung. -->
    <xs:element minOccurs="0" name="FieldName" nillable="true" type="xs:string"/>
    <!-- Liste von Strings, die gültige Werte für das Feld spezifizieren. -->
    <xs:element minOccurs="0" name="Values" nillable="true" type="q106:ArrayOfstring"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

Listing 3.87: Datentyp InCondition

FulltextCondition. Eine `FulltextCondition` spezifiziert eine Suchbedingung an den Volltext-Index. Für eine Volltextsuche wird ein Suchmodus `SearchMode` und ein Suchstring `QueryString` spezifiziert.

Es gibt zwei Arten von Volltext-Suchmodi:

1. Volltext der Dokument-Felder (`DocumentFields`) – d.h. Volltextsuche in Dokumenttitel, Merkmalsfeldern, etc. – oder
2. Volltext der Dokument-Dateien (`DocumentFiles`) – d.h. Volltextsuche in den Dateien, die zu dem Dokument gehören.

Der `QueryString` beschreibt einen Text, der im Volltext gefunden werden soll. Es können auch die Wildcards % (beliebig viele Zeichen) und _ (genau ein Zeichen) verwendet werden. Sequenzen von mehreren Wörtern können in Anführungszeichen zusammengefasst werden. Beispiele für gültige `QueryStrings` sind: *Me_er*, *1338%*, *Müller* oder ``*Peter Müller*''.
Listing 3.88 zeigt die WSDL-Definition des Datentyps `FulltextCondition`.

```
<xs:complexType name="FulltextCondition">
 <xs:complexContent mixed="false">
  <xs:extension base="q108:LeafCondition">
   <xs:sequence>
    <!-- Spezifiziert den Volltext-Index, in dem gesucht werden soll. -->
    <xs:element minOccurs="0" name="SearchMode" type="q108:FulltextSearchMode"/>
       <!-- Gibt den Text an, der im Volltext gefunden werden soll. -->
    <xs:element minOccurs="0" name="QueryString" nillable="true" type="xs:string"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

Listing 3.88: Datentyp FulltextCondition

Enumeration FulltextSearchMode. Die Enumeration `FulltextSearchMode` spezifitiert den Suchmodus im Volltext-Index.
Listing 3.89 zeigt die WSDL-Definition des Datentyps `FulltextSearchMode`.

```
<xs:simpleType name="FulltextSearchMode">
 <xs:restriction base="xs:string">
  <!-- Suche im Dokument-Volltext (Werte von (Kern- und Merkmals-)Feldern). -->
  <xs:enumeration value="DocumentFields"/>
  <!-- Suche im Datei-Volltext. -->
  <xs:enumeration value="DocumentFiles"/>
 </xs:restriction>
</xs:simpleType>
```

Listing 3.89: Datentyp FulltextSearchMode

NotCondition. Eine `NotCondition` spezifiert eine Suchbedingung, die eine andere Suchbedingung negiert, d.h. `NOT( X )`, wobei `X` eine beliebige Suchbedingung ist.
Listing 3.90 zeigt die WSDL-Definition des Datentyps `NotCondition`.

```
<xs:complexType name="NotCondition">
 <xs:complexContent mixed="false">
  <xs:extension base="q111:SearchCondition">
   <xs:sequence>
    <!-- Suchbedingung, die negiert werden soll. -->
    <xs:element minOccurs="0" name="Condition" nillable="true" type="q111:SearchCondition"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

Listing 3.90: Datentyp NotCondition

3.3.4.3. OrderBy

`OrderBy` spezifiziert die Sortierung von Suchergebnissen. Mehrere Sortierkriterien können miteinander verkettet werden. Die Struktur von `OrderBy` wird in Abbildung 3.3 als UML-Klassendiagramm dargestellt.

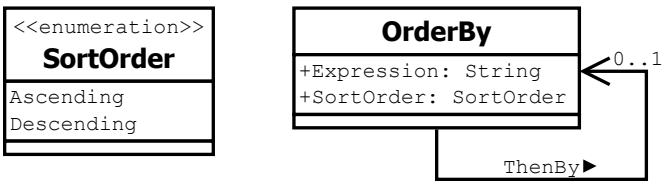

Abbildung 3.3.: Rekursive Struktur von `OrderBy`.

Listing 3.91 zeigt die WSDL-Definition des Datentyps `OrderBy`.

```
<xs:complexType name="OrderBy">
 <xs:sequence>
  <!-- Ausdruck, der das Sortierkriterium angibt, z.B. Name eines Feldes. -->
  <xs:element minOccurs="0" name="Expression" nillable="true" type="xs:string"/>
  <!-- Aufsteigende oder absteigende Sortierung. -->
  <xs:element minOccurs="0" name="SortOrder" type="q116:SortOrder"/>
  <!-- Nächstes, weniger relevantes, Sortierkriterium. -->
  <xs:element minOccurs="0" name="ThenBy" nillable="true" type="q117:OrderBy"/>
 </xs:sequence>
 </xs:complexType>
```

Listing 3.91: Datentyp OrderBy

`SortOrder` spezifiziert, ob für ein Sortierkriterium eine aufsteigende oder absteigende Sortierung verwendet wird.
Listing 3.92 zeigt die WSDL-Definition des Datentyps `SortOrder`.

```
<xs:simpleType name="SortOrder">
 <xs:restriction base="xs:string">
  <!-- Aufsteigende Sortierung. -->
  <xs:enumeration value="Ascending"/>
  <!-- Absteigende Sortierung. -->
  <xs:enumeration value="Descending"/>
 </xs:restriction>
</xs:simpleType>
```

Listing 3.92: Datentyp SortOrder

4. Administrative Erweiterungen

Dieses Kapitel beschreibt die administrativen Erweiterungen von PROXESS Webservice, die in der Konfiguration durch Verwendung des *AdminstrationService* aktiviert werden können.

Abschnitt 4.1 gibt einen Überblick über die Gemeinsamkeiten der Funktionen in ihrem Aufbau und ihrem Verhalten. Im wesentlichen besitzen alle Request-Nachrichten einen Teil, der das anzulegende/zu änderne Objekt identifiziert und einen Teil, der die Eigenschaften beschreibt, die für das Objekt gesetzt werden sollen.

Die darauf folgenden Abschnitte dokumentieren die Funktionen und Datentypen des *AdminstrationService*.

4.1. Allgemeines Verhalten der administrativen Funktionen

Die administrativen Funktionen des Webservice sind so aufgebaut, dass sie eine Request-Nachricht als Eingabeparameter erwarten und eine Response-Nachricht als Rückgabewert zurückgeben.

Die Request-Nachricht besteht i.d.R. aus zwei Teilen.

1. Ein erster Teil, der das Objekt identifiziert. Dieser Teil ist elementar für die Ausführung im Webservice und muss in jedem Fall korrekt angegeben werden. Bei Neuanlage wird der Kontext identifiziert, in dem das Objekt angelegt werden soll (z.B. ist das bei `CreateDocumentTypeField` die PROXESS-Datenbank sowie der Dokumenttyp). Bei Aktualisierung wird das konkrete Objekt identifiziert (z.B. ist das bei `UpdateFileType` die PROXESS-Datenbank sowie die ID des zu ändernden Dateityps).

2. Ein zweiter Teil, der die Eigenschaften des Objekts spezifiziert, die entweder bei der Neuanlage gesetzt werden sollen oder bei der Aktualiserung geändert werden sollen. Die Elemente aus diesem Teil sind i.d.R. optional (im Folgenden *optionale Parameter* oder auch *optionale Felder* genannt, s.u.), was bedeutet, dass die entsprechenden XML-Elemente vom Client nicht spezifiziert werden müssen. Generell werden nur Eigenschaften geändert, die vom Anwender auch spezifiert worden sind. Wird z.B. bei der Änderung eines Dokumenttyps nur die Beschreibung spezifiziert, bleiben alle übrigen Eigenschaften (z.B. Anzeigename, Zwischenablage, Volltext-Indexierung) unverändert.

Optionale Parameter und ihre Darstellung

- Wird im Folgenden von Parametern gesprochen, sind damit XML-Elemente einer SOAP-Nachricht gemeint, die einen Teil einer Request-Nachricht oder eines Datentyps darstellen. Hat ein **Paramter den Wert** `null`,

Funktionsname	Verweis
`UpdateUser`	4.2.2
`UpdateGroup`	4.2.5
`CreateDocumentType`	4.5.1
`UpdateDocumentType`	4.5.2
`CreateFileType`	4.6.1
`UpdateFileType`	4.6.2
`CreateValidationRule`	4.8.1
`UpdateValidationRule`	4.8.2

Tabelle 4.1.: Funktionen mit einer möglicherweise teilweisen Durchführung im Fehlerfall

dann ist damit gemeint, dass 1.) das entsprechende XML-Element in der SOAP-Nachricht fehlt oder 2.) das XML-Element in der SOAP-Nachricht vorhanden ist und mit `xsi:nil="true"` annotiert ist.

- In der Dokumentation werden einige Paramter als optional deklariert. Ein *optionaler Paramter* darf den Wert `null` haben und wird in diesem Fall bei der Ausführung im Webservice ignoriert. Nur wenn ein optionaler Parameter einen von `null` verschiedenen Wert hat, wird er im Webservice ausgewertet.
- Paramter, die keine optionalen Parameter sind, sind *nicht-optionale Paramter*. Nicht-optionale Parameter dürfen nicht den Wert `null` haben, d.h. das entsprechende XML-Element muss immer in der SOAP-Nachricht vorhanden sein und muss einen gültigen Wert haben.

Anmerkungen zur Fehlerbehandlung

- Generell gilt – wie auch für die Funktionen des *DocumentService* –, dass alle Aktionen einer administrativen Funktion fehlerfrei durchgeführt wurden, wenn der Webservice-Aufruf keine Ausnahme auslöst.
- Im Fehlerfall ist in manchen Fällen allerdings eine teilweise Durchführung möglich; d.h. diejenigen Aktionen, die zeitlich vor dem Auftreten des Fehlers bereits durchgeführt worden sind, werden nicht rückgängig gemacht. Alle Funktionen, für die eine teilweise Durchführung im Fehlerfall möglich ist, werden in Tabelle 4.1 aufgeführt. In jeden Fall ist die Ausführungsreihenfolge der Aktionen fest, d.h. anhand der Fehlermeldung lässt sich immer darauf schließen,
  1. welche Aktionen bereits durchgeführt worden sind,
  2. welche Aktion den Fehler ausgelöst hat und
  3. welche Aktionen gar nicht ausgeführt worden sind (weil zuvor der Fehler aufgetreten ist).

  Alle Funktionen, die nicht in Tabelle 4.1 aufgeführt sind, werden in jedem Fall „ganz oder gar" nicht durchgeführt.

4.2. Benutzer- und Gruppen-Verwaltung

Dieser Abschnitt dokumentiert die Funktionen zur Verwaltung von Benutzern, Gruppen und Gruppenzugehörigkeit.

4.2.1. Funktion `CreateUser`

Signatur:

```
CreateUserResponse CreateUser(
       CreateUserRequest request);
```

Die Funktion `CreateUser` legt einen neuen PROXESS-Benutzer an. Beim Anlegen eines Benutzers werden der Anmeldename, der vollständige Name und das Passwort festgelegt. Kann der Benutzer erfolgreich angelegt werden, wird der zugehörige `User` in der Response zurückgegeben.

Nachrichtentyp `CreateUserRequest`. Listing 4.1 zeigt die WSDL-Definition des Datentyps `CreateUserRequest`.

```
1  <xs:element name="CreateUserRequest">
2   <xs:complexType>
3    <xs:sequence>
4     <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
5     <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q195:LoginToken"/>
6     <!-- Anmeldename des Benutzers. -->
7     <xs:element minOccurs="0" name="LoginName" nillable="true" type="xs:string"/>
8     <!-- Vollständiger Name des Benutzers. -->
9     <xs:element minOccurs="0" name="FullName" nillable="true" type="xs:string"/>
10    <!-- Passwort des Benutzers. -->
11    <xs:element minOccurs="0" name="Password" nillable="true" type="xs:string"/>
12    <!-- Optional: Das Passwort des Benutzers läuft niemals ab. -->
13    <xs:element minOccurs="0" name="PasswordNeverExpires" type="xs:boolean"/>
14   </xs:sequence>
15  </xs:complexType>
16 </xs:element>
```

Listing 4.1: Nachrichtentyp CreateUserRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.1 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2

Anmerkung. Beim Anlegen eines Benutzers ist nur das Feld `PasswordNeverExpires` optional, d.h. Anmeldename (`LoginName`), vollständer Name (`FullName`) und das Passwort (`Password`) müssen auf jeden Fall festgelegt werden. Standardmäßig wird für den Parameter `PasswordNeverExpires` der Wert `false` verwendet.

Nachrichtentyp `CreateUserResponse`. Listing 4.2 zeigt die WSDL-Definition des Datentyps `CreateUserResponse`.

```
1 <xs:element name="CreateUserResponse">
2  <xs:complexType>
3   <xs:sequence>
4    <!-- Neuer Benutzer, der durch den zugehörigen Request angelegt worden ist. -->
5    <xs:element minOccurs="0" name="User" nillable="true" type="q196:User"/>
6   </xs:sequence>
7  </xs:complexType>
8 </xs:element>
```

Listing 4.2: Nachrichtentyp CreateUserResponse

Die folgende Tabelle führt Verweise zu allen in Listing 4.2 referenzierten Datentypen auf.

Feldname	Verweis
`User`	3.3.2.5

4.2.2. Funktion `UpdateUser`

Signatur:

```
UpdateUserResponse UpdateUser(
       UpdateUserRequest request);
```

Die Funktion `UpdateUser` ändert die Daten eines bestehenden PROXESS-Benutzers. Kann die Änderung an den Daten des Benutzers erfolgreich durchgeführt werden, wird der geänderte `User` in der Response zurückgegeben.

Achtung. Dieser Aufruf wird intern in mehreren Einzelschritten verarbeitet, von denen jeder einzelne Schritt fehlschlagen kann. Unmittelbar vor einem Fehler durchgeführte Änderungen werden nicht rückgängig gemacht. Die Teilschritte werden in der folgenden Reihenfolge abgearbeitet:

1. Ggf. ändern des Anmeldenamens (Parameter `LoginName`).
2. Ggf. ändern des vollständigen Namens (Parameter `FullName`).
3. Ggf. ändern der Passwortgültigkeit (Parameter `PasswordNeverExpired`).
4. Ggf. ändern des Passworts (Parameter `Password`).
5. Ggf. aktivieren/deaktivieren des Benutzeraccounts (Parameter `AccountDisabled`).

Nachrichtentyp `UpdateUserRequest`. Listing 4.3 zeigt die WSDL-Definition des Datentyps `UpdateUserRequest`.

```
<xs:element name="UpdateUserRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q197:LoginToken"/>
   <!-- Identifier des zu ändernden Benutzers. -->
   <xs:element minOccurs="0" name="UserID" type="xs:unsignedLong"/>
   <!-- Optional: Neuer Anmeldename des Benutzers. -->
   <xs:element minOccurs="0" name="LoginName" nillable="true" type="xs:string"/>
   <!-- Optional: Neuer vollständiger Name des Benutzers. -->
   <xs:element minOccurs="0" name="FullName" nillable="true" type="xs:string"/>
   <!-- Optional: Neues Passwort des Benutzers. -->
   <xs:element minOccurs="0" name="Password" nillable="true" type="xs:string"/>
   <!-- Optional: Neue Angabe, ob das Passwort niemals abläuft. -->
   <xs:element minOccurs="0" name="PasswordNeverExpired" nillable="true" type="xs:boolean"/>
   <!-- Optional: Benutzeraccount ist (nicht) deaktiviert. -->
   <xs:element minOccurs="0" name="AccountDisabled" nillable="true" type="xs:boolean"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.3: Nachrichtentyp UpdateUserRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.3 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2

Nachrichtentyp `UpdateUserResponse`. Listing 4.4 zeigt die WSDL-Definition des Datentyps `UpdateUserResponse`.

```
<xs:element name="UpdateUserResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Benutzer, der durch den zugehörigen Request verändert wurde. -->
   <xs:element minOccurs="0" name="User" nillable="true" type="q198:User"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.4: Nachrichtentyp UpdateUserResponse

Die folgende Tabelle führt Verweise zu allen in Listing 4.4 referenzierten Datentypen auf.

Feldname	Verweis
`User`	3.3.2.5

4.2.3. Funktion `DeleteUser`

Signatur:

```
EmptyMessage DeleteUser(
      DeleteUserRequest request);
```

Die Funktion `DeleteUser` löscht einen vorhandenen Benutzer. Wenn die Funktion keine Ausnahme auslöst, wurde der Benutzer erfolgreich gelöscht.

Nachrichtentyp `DeleteUserRequest`. Listing 4.5 zeigt die WSDL-Definition des Datentyps `DeleteUserRequest`.

```
1  <xs:element name="DeleteUserRequest">
2   <xs:complexType>
3    <xs:sequence>
4     <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
5     <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q199:LoginToken"/>
6     <!-- ID des Benutzers, der gelöscht werden soll. -->
7     <xs:element minOccurs="0" name="UserID" type="xs:unsignedLong"/>
8    </xs:sequence>
9   </xs:complexType>
10 </xs:element>
```

Listing 4.5: Nachrichtentyp DeleteUserRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.5 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2

4.2.4. Funktion `CreateGroup`

Signatur:

```
CreateGroupResponse CreateGroup(
      CreateGroupRequest request);
```

Die Funktion `CreateGroup` legt eine neue Gruppe (zunächst ohne Mitglieder) an. Dabei werden der Name und eine Beschreibung der Gruppe festgelegt. Kann die Gruppe erfolgreich angelegt werden, wird die entsprechende `Group` in der Response zurückgegeben.

Nachrichtentyp `CreateGroupRequest`. Listing 4.6 zeigt die WSDL-Definition des Datentyps `CreateGroupRequest`.

```
<xs:element name="CreateGroupRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q200:LoginToken"/>
   <!-- Name der neuen Gruppe. -->
   <xs:element minOccurs="0" name="Name" nillable="true" type="xs:string"/>
   <!-- Beschreibung der neuen Gruppe. -->
   <xs:element minOccurs="0" name="Description" nillable="true" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.6: Nachrichtentyp CreateGroupRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.6 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2

Anmerkung. Beim Anlegen einer Gruppe muss sowohl der Name der Gruppe (`Name`) als auch die Beschreibung (`Description`) gesetzt werden.

Nachrichtentyp `CreateGroupResponse`. Listing 4.7 zeigt die WSDL-Definition des Datentyps `CreateGroupResponse`.

```
<xs:element name="CreateGroupResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Gruppe, die durch den zugehörigen Request erzeugt wurde. -->
   <xs:element minOccurs="0" name="Group" nillable="true" type="q201:Group"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.7: Nachrichtentyp CreateGroupResponse

Die folgende Tabelle führt Verweise zu allen in Listing 4.7 referenzierten Datentypen auf.

Feldname	Verweis
`Group`	3.3.2.5

4.2.5. Funktion `UpdateGroup`

Signatur:

```
UpdateGroupResponse UpdateGroup(
       UpdateGroupRequest request);
```

Die Funktion `UpdateGroup` aktualisiert die Daten einer bestehenden Gruppe. Können die Änderungen an der Gruppe erfolgreich durchgeführt werden, wird die entsprechende `Group` in der Response zurückgegeben.

Achtung. Dieser Aufruf wird intern in mehreren Einzelschritten verarbeitet, von denen jeder einzelne Schritt fehlschlagen kann. Unmittelbar vor einem Fehler durchgeführte Änderungen werden nicht rückgängig gemacht. Die Teilschritte werden in der folgenden Reihenfolge abgearbeitet:

1. Ggf. ändern des Gruppennamens (Parameter `Name`).
2. Ggf. ändern der Gruppenbeschreibung (Parameter `Description`).

Nachrichtentyp `UpdateGroupRequest`. Listing 4.8 zeigt die WSDL-Definition des Datentyps `UpdateGroupRequest`.

```
1  <xs:element name="UpdateGroupRequest">
2   <xs:complexType>
3    <xs:sequence>
4     <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
5     <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q202:LoginToken"/>
6     <!-- ID der Gruppe, die aktualisiert werden soll. -->
7     <xs:element minOccurs="0" name="GroupID" type="xs:unsignedLong"/>
8     <!-- Optional: Neuer Name der Gruppe. -->
9     <xs:element minOccurs="0" name="Name" nillable="true" type="xs:string"/>
10    <!-- Optional: Neue Beschreibung der Gruppe. -->
11    <xs:element minOccurs="0" name="Description" nillable="true" type="xs:string"/>
12   </xs:sequence>
13  </xs:complexType>
14 </xs:element>
```

Listing 4.8: Nachrichtentyp UpdateGroupRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.8 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2

Nachrichtentyp `UpdateGroupResponse`. Listing 4.9 zeigt die WSDL-Definition des Datentyps `UpdateGroupResponse`.

```
<xs:element name="UpdateGroupResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Gruppe, die durch den zugehörigen Request verändert wurde. -->
   <xs:element minOccurs="0" name="Group" nillable="true" type="q203:Group"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.9: Nachrichtentyp UpdateGroupResponse

Die folgende Tabelle führt Verweise zu allen in Listing 4.9 referenzierten Datentypen auf.

Feldname	Verweis
`Group`	3.3.2.5

4.2.6. Funktion `DeleteGroup`

Signatur:

```
EmptyMessage DeleteGroup(
       DeleteGroupRequest request);
```

Die Funktion `DeleteGroup` löscht eine bestehende Gruppe. Wenn die Funktion keine Ausnahme auslöst, dann wurde die Gruppe erfolgreich gelöscht.

Nachrichtentyp `DeleteGroupRequest`. Listing 4.10 zeigt die WSDL-Definition des Datentyps `DeleteGroupRequest`.

```
<xs:element name="DeleteGroupRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q204:LoginToken"/>
   <!-- ID der Gruppe, die gelöscht werden soll. -->
   <xs:element minOccurs="0" name="GroupID" type="xs:unsignedLong"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.10: Nachrichtentyp DeleteGroupRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.10 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2

4.2.7. Funktion `GroupAddUser`

Signatur:

```
GroupAddUserResponse GroupAddUser(
       GroupAddUserRequest request);
```

Die Funktion `GroupAddUser` fügt einen Benutzer zu einer Gruppe hinzu. Wurde der Benutzer erfolgreich zu der Gruppe hinzugefügt, wird die Gruppe mit aktualisierter Benutzerliste in der Response zurückgegeben.

Nachrichtentyp `GroupAddUserRequest`. Listing 4.11 zeigt die WSDL-Definition des Datentyps `GroupAddUserRequest`.

```
<xs:element name="GroupAddUserRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q205:LoginToken"/>
   <!-- ID der Gruppe, der ein Benutzer hinzugefügt werden soll. -->
   <xs:element minOccurs="0" name="GroupID" type="xs:unsignedLong"/>
   <!-- ID des Benutzers, der einer Gruppe hinzugefügt werden soll. -->
   <xs:element minOccurs="0" name="UserID" type="xs:unsignedLong"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.11: Nachrichtentyp GroupAddUserRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.11 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2

Nachrichtentyp `GroupAddUserResponse`. Listing 4.12 zeigt die WSDL-Definition des Datentyps `GroupAddUserResponse`.

```
<xs:element name="GroupAddUserResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Gruppe, der im zugehörigen Request ein Benutzer hinzugefügt wurde. -->
   <xs:element minOccurs="0" name="Group" nillable="true" type="q206:Group"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.12: Nachrichtentyp GroupAddUserResponse

Die folgende Tabelle führt Verweise zu allen in Listing 4.12 referenzierten Datentypen auf.

Feldname	Verweis
Group	3.3.2.5

4.2.8. Funktion GroupRemoveUser

Signatur:

```
GroupRemoveUserResponse GroupRemoveUser(
       GroupRemoveUserRequest request);
```

Die Funktion GroupRemoveUser entfernt einen Benutzer aus einer Gruppe. Wurde der Benutzer erfolgreich aus der Gruppe entfernt, wird die Gruppe mit aktualisierter Benutzerliste in der Response zurückgegeben.

Nachrichtentyp GroupRemoveUserRequest. Listing 4.13 zeigt die WSDL-Definition des Datentyps GroupRemoveUserRequest.

```
1  <xs:element name="GroupRemoveUserRequest">
2   <xs:complexType>
3    <xs:sequence>
4     <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
5     <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q207:LoginToken"/>
6     <!-- ID der Gruppe, aus der ein Benutzer entfernt werden soll. -->
7     <xs:element minOccurs="0" name="GroupID" type="xs:unsignedLong"/>
8     <!-- ID des Benutzers, der aus einer Gruppe entfernt werden soll. -->
9     <xs:element minOccurs="0" name="UserID" type="xs:unsignedLong"/>
10   </xs:sequence>
11  </xs:complexType>
12 </xs:element>
```

Listing 4.13: Nachrichtentyp GroupRemoveUserRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.13 referenzierten Datentypen auf.

Feldname	Verweis
LoginToken	3.3.1.2

Nachrichtentyp `GroupRemoveUserResponse`. Listing 4.14 zeigt die WSDL-Definition des Datentyps `GroupRemoveUserResponse`.

```
<xs:element name="GroupRemoveUserResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Gruppe, aus der im zugehörigen Request ein Benutzer entfernt worden ist. -->
   <xs:element minOccurs="0" name="Group" nillable="true" type="q208:Group"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.14: Nachrichtentyp GroupRemoveUserResponse

Die folgende Tabelle führt Verweise zu allen in Listing 4.14 referenzierten Datentypen auf.

Feldname	Verweis
`Group`	3.3.2.5

4.3. Zugriffsrechte

Dieser Abschnitt beschreibt die Funktionen und Datentypen zum Abfragen und Ändern von Zugriffsrechten.

Ein Zugriffsrecht bezieht sich immer auf ein Rechte-Objekt (d.h. ein Objekt, für das Zugriffrechte erteilt oder entzogen werden können) und ein Rechte-Subjekt (ein Benutzer oder eine Gruppe, der/die Rechte für ein bestimmtes Objekt erteilt oder entzogen bekommt). Mögliche Rechte-Objekte sind

- **Datenbanken.** Für Datenbanken gibt es ein Zugriffsrecht und ein sog. Database-Grant-Recht (das Grant-Recht erlaubt einem Benutzer/Gruppe die Vergabe weiterer Rechte für diese Datenbank).
- **Dokumente.** Für Dokumente gibt es die klassischen CRUD-Rechte (Create, Read, Update, Delete) sowie ein Dokument-Grant-Recht, also das Recht, weitere Zugriffsrechte für dieses Dokument zu vergeben.
- **Dokumenttypen.** Für Dokumenttypen gibt es die gleichen Rechte wie für Dokumente und zusätzlich ein Dokumenttyp-Grant-Recht, das die Vergabe weiterer Zugriffsrechte für diesen Dokumenttyp erlaubt. Das Dokument-Grant-Recht (s.o.) bezieht sich in diesem Fall auf alle Dokumente dieses Dokumenttyps.

Jedes einzelne Zugriffsrecht kann einem Rechte-Subjekt für ein bestimmtes Rechte-Objekt erteilt oder entzogen werden. Ist ein Recht nicht gesetzt (also weder erteilt noch entzogen), ergeben sich die effektive Zugriffsrechte aus der Gruppenzugehörigkeit eines Benutzers. Die effektiven Rechte können nicht explizit abgefragt werden.

4.3.1. Funktion `GetAccessRights`

Signatur:

```
GetAccessRightsResponse GetAccessRights(
       GetAccessRightsRequest request);
```

Die Funktion `GetAccessRights` ruft eine Zugriffsrechte-Liste für ein bestimmtes Rechte-Objekt ab. Optional kann das Ergebnis durch ein bestimmtes Rechte-Subjekte gefiltert werden.

Nachrichtentyp `GetAccessRightsRequest`. Listing 4.15 zeigt die WSDL-Definition des Datentyps `GetAccessRightsRequest`.

```
<xs:element name="GetAccessRightsRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q173:LoginToken"/>
   <!-- Datenbank, in der sich das Rechte-Objekt befindet. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
   <!-- Art des Rechte-Objekts (Datenbank, Dokumenttyp, Dokument). -->
   <xs:elem-ent minOccurs="0" name="RightObjectType" type="q174:RightObjectType"/>
   <!-- ID des Rechte-Objekts (im Fall Dokumenttyp oder Dokument), andernfalls null.-->
   <xs:element minOccurs="0" name="RightObjectIdentifier" nillable="true"
     type="xs:unsignedLong"/>
   <!-- Optional: ID der Gruppe oder des Benutzers, deren/dessen Rechte abgefragt werden.
       Wenn null: Rechte für alle Gruppen und Benutzer werden abgerufen. -->
   <xs:element minOccurs="0" name="RightSubjectIdentifier" nillable="true"
     type="xs:unsignedLong"/>
   <!-- Optional: Gibt an, ob die Rechte der zugehörigen Benutzer (bzw. der Gruppen eines
        Benutzers) zusätzlich abgefragt werden sollen. Dieser Parameter wird ignoriert,
           wenn kein RightSubjectIdentifier angegeben wurde. -->
   <xs:element minOccurs="0" name="IncludeGroupsOfUserOrMembersOfGroup" nillable="true"
     type="xs:boolean"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.15: Nachrichtentyp GetAccessRightsRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.15 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2
`RightObjectType`	4.3.5

Anmerkung. Um alle Rechte für ein Rechte-Objekt abzurufen, dürfen die beiden optionalen Parameter `RightSubjectIdentifier` und `IncludeGroupsOfUserOrMembersOfGroup` keinen Wert haben. Wenn `RightSubjectIdentifier` einen Wert hat (d.h. die Rechte-Liste wird gefiltert) wird für `IncludeGroupsOfUserOrMembersOfGroup` standardmäßig der Wert `false` verwendet, d.h. es werden exakt die Rechte für einen einzelnen Benutzer (bzw. eine einzelne Gruppe) ausgegeben.

Nachrichtentyp `GetAccessRightsResponse`. Listing 4.16 zeigt die WSDL-Definition des Datentyps `GetAccessRightsResponse`.

```
<xs:element name="GetAccessRightsResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Liste von Zugriffrechten entsprechend dem zugehörigen GetAccessRightsRequest. -->
   <xs:element minOccurs="0" name="AccessControlPolicy" nillable="true"
     type="q176:AccessControlPolicy"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.16: Nachrichtentyp GetAccessRightsResponse

Die folgende Tabelle führt Verweise zu allen in Listing 4.16 referenzierten Datentypen auf.

Feldname	Verweis
`AccessControlPolicy`	4.3.3

4.3.2. Funktion `SetAccessRight`

Signatur:

```
EmptyMessage SetAccessRight(
       SetAccessRightRequest request);
```

Die Funktion `SetAccessRight` setzt ein Zugriffsrecht für ein einzelnes Rechte-Subjekt und ein bestimmtes Rechte-Objekt. Z.B. werden für Benutzer `hugo` für den Dokumenttyp `Lieferschein` die Rechte `Create`, `Read` und `Update` erteilt, das Recht `Delete` entzogen und beide Grant-Rechte auf neutral gesetzt. Löst der Aufruf dieser Funktion keine Ausnahme aus, wurden die neuen Rechte erfolgreich gesetzt.

Nachrichtentyp `SetAccessRightRequest`. Listing 4.17 zeigt die WSDL-Definition des Datentyps `SetAccessRightRequest`.

```
<xs:element name="SetAccessRightRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q192:LoginToken"/>
   <!-- Name der Datenbank, in der das Rechte-Objekt gespeichert ist. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
   <!-- Art des Rechte-Objekts (Datenbank, Dokumenttyp, Dokument). -->
   <xs:element minOccurs="0" name="RightObjectType" type="q193:RightObjectType"/>
   <!-- ID des Rechte-Objekts (im Fall Dokumenttyp oder Dokument) oder null. -->
   <xs:element minOccurs="0" name="RightObjectIdentifier" nillable="true" type="xs:unsignedLong"/>
   <!-- Einzelnes Zugriffsrecht, das gesetzt werden soll. -->
   <xs:element minOccurs="0" name="AccessRule" nillable="true" type="q194:AccessRule"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.17: Nachrichtentyp SetAccessRightRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.17 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2
`RightObjectType`	4.3.5
`AccessRule`	4.3.4

Im Folgenden werden die Datentypen zur Darstellung von Zugriffsrechten beschrieben.

4.3.3. Datentyp `AccessControlPolicy`

Eine `AccessControlPolicy` stellt eine Menge von Zugriffsrechten dar, die für ein bestimmtes Rechte-Objekt gelten.

Listing 4.18 zeigt die WSDL-Definition des Datentyps `AccessControlPolicy`.

```
<xs:complexType name="AccessControlPolicy">
 <xs:sequence>
  <!-- Name der Datenbank, in der das Rechte-Objekt gespeichert ist. -->
  <xs:element name="DatabaseName" nillable="true" type="xs:string"/>
  <!-- Art des Rechte-Objekts (Datenbank, Dokumenttyp oder Dokument). -->
  <xs:element name="RightObjectType" type="q177:RightObjectType"/>
  <!-- Null für Datenbankrechte, ansonsten ID des Dokumenttyps oder des Dokuments,
      für den/das die Rechte gelten. -->
  <xs:element minOccurs="0" name="RightObjectIdentifier" nillable="true"
    type="xs:unsignedLong"/>
  <!-- Liste von Benutzer-/Gruppen-spezifischen Zugriffsrechten für das Rechte-Objekt
      dieser AccessControlPolicy. -->
  <xs:element name="AccessRules" nillable="true" type="q178:AccessRuleCollection"/>
 </xs:sequence>
</xs:complexType>
```

Listing 4.18: Datentyp AccessControlPolicy

Die folgende Tabelle führt Verweise zu allen in Listing 4.18 referenzierten Datentypen auf.

Feldname	Verweis
`RightObjectType`	4.3.5
`AccessRuleCollection`	4.3.4

4.3.4. Datentyp `AccessRule`

Eine `AccessRule` stellt ein Zugriffsrecht für ein bestimmtes Rechte-Subjekt dar. Eine `AccessRule` ist unabhängig von einem Rechte-Objekt. Das Rechte-Objekt für das eine bestimmte Zugriffsregel gültig ist, ergibt sich ggf. aus dem Zusammenhang (z.B. `AccessControlPolicy` oder `SetAccessRightRequest`).

Listing 4.19 zeigt die WSDL-Definition des Datentyps `AccessRule`.

```
<xs:complexType name="AccessRule">
 <xs:sequence>
  <!-- Art des Rechte-Subjekts (Benutzer/User oder Gruppe/Group). -->
  <xs:element name="RightSubjectType" type="q182:RightSubjectType"/>
  <!-- ID des Benutzers bzw. der Gruppe. -->
  <xs:element name="RightSubjectIdentifier" type="xs:unsignedLong"/>
  <!-- Für Dokumenttypen/Dokumente: Erstellen-Recht erteilt/entzogen/neutral. -->
  <!-- Für Datenbanken: nicht anwendbar -->
  <xs:element minOccurs="0" name="CreatePermission" type="q183:Permission"/>
  <!-- Für Dokumenttypen/Dokumente: Lesen-Recht erteilt/entzogen/neutral. -->
  <!-- Für Datenbanken: Zugriffs-Recht erteilt/entzogen/neutral. -->
  <xs:element minOccurs="0" name="ReadPermission" type="q184:Permission"/>
  <!-- Für Dokumenttypen/Dokumente: Editieren-Recht erteilt/entzogen/neutral. -->
  <!-- Für Datenbanken: nicht anwendbar -->
  <xs:element minOccurs="0" name="UpdatePermission" type="q185:Permission"/>
  <!-- Für Dokumenttypen/Dokumente: Löschen-Recht erteilt/entzogen/neutral. -->
  <!-- Für Datenbanken: nicht anwendbar -->
  <xs:element minOccurs="0" name="DeletePermission" type="q186:Permission"/>
  <!-- Für Dokumenttypen/Dokumente: Dokument-Grant-Recht erteilt/entzogen/neutral. -->
  <!-- Für Datenbanken: nicht anwendbar -->
  <xs:element minOccurs="0" name="GrantDocumentPermission" type="q187:Permission"/>
  <!-- Für Dokumente: nicht anwendbar -->
  <!-- Für Dokumenttypen: Dokumenttyp-Grant-Recht erteilt/entzogen/neutral. -->
  <!-- Für Datenbanken: Database-Grant-Recht erteilt/entzogen/neutral. -->
  <xs:element minOccurs="0" name="GrantPermission" type="q188:Permission"/>
 </xs:sequence>
</xs:complexType>
```

Listing 4.19: Datentyp AccessRule

Die folgende Tabelle führt Verweise zu allen in Listing 4.19 referenzierten Datentypen auf.

Feldname	Verweis
`RightSubjectType`	4.3.6
`Permission`	4.3.7

4.3.5. Enumeration `RightObjectType`

Ein `RightObjectType` spezifiert den Typ eines Rechte-Objekts.

Listing 4.20 zeigt die WSDL-Definition des Datentyps `RightObjectType`.

```
<xs:simpleType name="RightObjectType">
 <xs:restriction base="xs:string">
  <!-- Rechte beziehen sich auf eine Datenbank. -->
  <xs:enumeration value="Database"/>
  <!-- Rechte beziehen sich auf einen Dokumenttyp. -->
  <xs:enumeration value="DocumentType"/>
  <!-- Rechte beziehen sich auf ein Dokument. -->
  <xs:enumeration value="Document"/>
 </xs:restriction>
</xs:simpleType>
```

Listing 4.20: Datentyp RightObjectType

4.3.6. Enumeration `RightSubjectType`

Ein `RightSubjectType` spezifiziert den Typ eines Rechte-Subjekts.

Listing 4.21 zeigt die WSDL-Definition des Datentyps `RightSubjectType`.

```
<xs:simpleType name="RightSubjectType">
 <xs:restriction base="xs:string">
  <!-- Rechte beziehen sich auf einen Benutzer. -->
  <xs:enumeration value="User"/>
  <!-- Rechte beziehen sich auf eine Gruppe. -->
  <xs:enumeration value="Group"/>
 </xs:restriction>
</xs:simpleType>
```

Listing 4.21: Datentyp RightSubjectType

4.3.7. Enumeration `Permission`

Eine `Permission` gibt an, ob ein bestimmtes Recht erteilt oder entzogen ist.

Listing 4.22 zeigt die WSDL-Definition des Datentyps `Permission`.

```
<xs:simpleType name="Permission">
 <xs:restriction base="xs:string">
  <!-- Das Recht wurde nicht spezifiert. -->
  <xs:enumeration value="Neutral"/>
  <!-- Das Recht wurde erteilt. -->
  <xs:enumeration value="Allow"/>
  <!-- Das Recht wurde entzogen. -->
  <xs:enumeration value="Deny"/>
 </xs:restriction>
</xs:simpleType>
```

Listing 4.22: Datentyp Permission

4.4. Datenbanken

Dieser Abschnitt dokumentiert die Funktionen zur Verwaltung von PROXESS-Datenbanken.

Diese Funktionen dienen nicht zur Verwaltung von relationalen Datenbanken des Datenbank-Backends, sondern geben lediglich die administrativen Funktionen von PROXESS wieder. Diese Funktionen werden dazu verwendet, um relationale Datenbanken in der Master-Konfiguration von PROXESS zu registrieren bzw. wieder aus der Master-Konfiguration zu entfernen.

4.4.1. Funktion `CreateDatabase`

Signatur:

```
CreateDatabaseResponse CreateDatabase(
      CreateDatabaseRequest request);
```

Die Funktion `CreateDatabase` legt eine neue PROXESS-Datenbank an. Dabei wird der Name der Datenbank und eine Beschreibung der Datenbank festgelegt. Im Erfolgsfall wird die neu angelegte Datenbank in der Response zurückgegeben.

Nachrichtentyp `CreateDatabaseRequest`. Listing 4.23 zeigt die WSDL-Definition des Datentyps `CreateDatabaseRequest`.

```
<xs:element name="CreateDatabaseRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q209:LoginToken"/>
   <!-- Name der neuen Datenbank. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
   <!-- Beschreibung der neuen Datenbank. -->
   <xs:element minOccurs="0" name="Description" nillable="true" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.23: Nachrichtentyp CreateDatabaseRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.23 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2

Nachrichtentyp `CreateDatabaseResponse`. Listing 4.24 zeigt die WSDL-Definition des Datentyps `CreateDatabaseResponse`.

```
<xs:element name="CreateDatabaseResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Neue angelegte Datenbank. -->
   <xs:element minOccurs="0" name="Databases" nillable="true" type="q210:DatabaseCollection"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.24: Nachrichtentyp CreateDatabaseResponse

Die folgende Tabelle führt Verweise zu allen in Listing 4.24 referenzierten Datentypen auf.

Feldname	Verweis
`Database`	3.3.2.1

4.4.2. Funktion `UpdateDatabase`

Signatur:

```
UpdateDatabaseResponse UpdateDatabase(
      UpdateDatabaseRequest request);
```

Die Funktion `UpdateDatabase` aktualisiert eine PROXESS-Datenbank. Im Erfolgsfall wird die geänderte Datenbank in der Response zurückgegeben.

Nachrichtentyp `UpdateDatabaseRequest`. Listing 4.25 zeigt die WSDL-Definition des Datentyps `UpdateDatabaseRequest`.

```
<xs:element name="UpdateDatabaseRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q211:LoginToken"/>
   <!-- Name der Datenbank, die aktualisiert werden soll. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
   <!-- Optional: Neue Beschreibung der Datenbank. -->
   <xs:element minOccurs="0" name="Description" nillable="true" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.25: Nachrichtentyp UpdateDatabaseRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.25 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2

Anmerkung. Der Name einer Datenbank kann nicht geändert werden, und daher kann mit dem Aufruf von `UpdateDatabase` lediglich die Beschreibung der Datenbank geändert werden. Es macht folglich keinen Sinn, dass der optionale Parameter `Description` keinen Wert hat.

Nachrichtentyp `UpdateDatabaseResponse`. Listing 4.26 zeigt die WSDL-Definition des Datentyps `UpdateDatabaseResponse`.

```
<xs:element name="UpdateDatabaseResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Aktualisierte Datenbank. -->
   <xs:element minOccurs="0" name="Databases" nillable="true" type="q212:DatabaseCollection"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.26: Nachrichtentyp UpdateDatabaseResponse

Die folgende Tabelle führt Verweise zu allen in Listing 4.26 referenzierten Datentypen auf.

Feldname	Verweis
`Database`	3.3.2.1

4.4.3. Funktion `DeleteDatabase`

Signatur:

```
EmptyMessage DeleteDatabase(
      DeleteDatabaseRequest request);
```

Die Funktion `DeleteDatabase` löscht eine PROXESS-Datenbank. Wenn der Aufruf keine Ausnahme auslöst, wurde die Datenbank erfolgreich gelöscht.

Nachrichtentyp `DeleteDatabaseRequest`. Listing 4.27 zeigt die WSDL-Definition des Datentyps `DeleteDatabaseRequest`.

```
1  <xs:element name="DeleteDatabaseRequest">
2   <xs:complexType>
3    <xs:sequence>
4     <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
5     <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q213:LoginToken"/>
6     <!-- Name der Datenbank, die gelöscht werden soll. -->
7     <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
8    </xs:sequence>
9   </xs:complexType>
10 </xs:element>
```

Listing 4.27: Nachrichtentyp DeleteDatabaseRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.27 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2

4.5. Dokumenttypen

Dieser Abschnitt beschreibt die Funktionen zur Verwaltung von PROXESS Dokumenttypen.

4.5.1. Funktion `CreateDocumentType`

Signatur:

```
CreateDocumentTypeResponse CreateDocumentType(
      CreateDocumentTypeRequest request);
```

Die Funktion `CreateDocumentType` legt einen Dokumenttyp mit den angegebenen Eigenschaften an. Wenn alle Eigenschaften erfolgreich angewendet werden können, wird der neue Dokumenttyp in der Response zurückgegeben.

Achtung. Der Aufruf wird intern in mehreren Teilschritten durchgeführt, die im Fehlerfall zu einer nur teilweisen Durchführung der gesamten Aktion führen können. Die Teilschritte und deren Ausführungsreihenfolge werden bei der Dokumentation des Datentyps `DocumentTypeSpec` (4.5.4) beschrieben.

Nachrichtentyp `CreateDocumentTypeRequest`. Listing 4.28 zeigt die WSDL-Definition des Datentyps `CreateDocumentTypeRequest`.

```
<xs:element name="CreateDocumentTypeRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q214:LoginToken"/>
   <!-- Name der Datenbank, in der der Dokumenttyp angelegt wird. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
   <!-- Beschreibung des anzulegenden Dokumenttyps. -->
   <xs:element minOccurs="0" name="DocumentType" nillable="true" type="q215:DocumentTypeSpec"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.28: Nachrichtentyp CreateDocumentTypeRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.28 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2
`DocumentTypeSpec`	4.5.4

Nachrichtentyp `CreateDocumentTypeResponse`. Listing 4.29 zeigt die WSDL-Definition des Datentyps `CreateDocumentTypeResponse`.

```
<xs:element name="CreateDocumentTypeResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Neu angelegter Dokumenttyp. -->
   <xs:element minOccurs="0" name="DocumentType" nillable="true" type="q217:DocumentType"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.29: Nachrichtentyp CreateDocumentTypeResponse

Die folgende Tabelle führt Verweise zu allen in Listing 4.29 referenzierten Datentypen auf.

Feldname	Verweis
`DocumentType`	3.3.2.2

4.5.2. Funktion `UpdateDocumentType`

Signatur:

```
UpdateDocumentTypeResponse UpdateDocumentType(
      UpdateDocumentTypeRequest request);
```

Die Funktion `UpdateDocumentType` aktualisiert einen Dokumenttyp. Wenn alle Änderungen erfolgreich durchgeführt werden können, wird der geänderte Dokumenttyp in der Response zurückgegeben.

Achtung. Der Aufruf wird intern in mehreren Teilschritten durchgeführt, die im Fehlerfall zu einer nur teilweisen Durchführung der gesamten Aktion führen können. Die Teilschritte und deren Ausführungsreihenfolge werden bei der Dokumentation des Datentyps `DocumentTypeSpec` (4.5.4) beschrieben.

Nachrichtentyp `UpdateDocumentTypeRequest`. Listing 4.30 zeigt die WSDL-Definition des Datentyps `UpdateDocumentTypeRequest`.

```
1  <xs:element name="UpdateDocumentTypeRequest">
2   <xs:complexType>
3    <xs:sequence>
4     <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
5     <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q218:LoginToken"/>
6     <!-- Datenbank, in der der Dokumenttyp gespeichert ist. -->
7     <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
8     <!-- ID des Dokumenttyps, der aktualisiert werden soll. -->
9     <xs:element minOccurs="0" name="DocumentTypeID" type="xs:unsignedLong"/>
10    <!-- Beschreibung der Änderungen am Dokumenttyp. -->
11    <xs:element minOccurs="0" name="DocumentTypeChanges" nillable="true" type="q219:DocumentTypeSpec"/>
12   </xs:sequence>
13  </xs:complexType>
14 </xs:element>
```

Listing 4.30: Nachrichtentyp UpdateDocumentTypeRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.30 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2
`DocumentTypeSpec`	4.5.4

Nachrichtentyp `UpdateDocumentTypeResponse`. Listing 4.31 zeigt die WSDL-Definition des Datentyps `UpdateDocumentTypeResponse`.

```
<xs:element name="UpdateDocumentTypeResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Aktualisierter Dokumenttyp. -->
   <xs:element minOccurs="0" name="DocumentType" nillable="true" type="q220:DocumentType"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.31: Nachrichtentyp UpdateDocumentTypeResponse

Die folgende Tabelle führt Verweise zu allen in Listing 4.31 referenzierten Datentypen auf.

Feldname	Verweis
DocumentType	3.3.2.2

4.5.3. Funktion `DeleteDocumentType`

Signatur:

```
EmptyMessage DeleteDocumentType(
       DeleteDocumentTypeRequest request);
```

Die Funktion `DeleteDocumentType` löscht einen Dokumenttyp. Wenn der Aufruf keine Ausnahme auslöst, wurde der Dokumenttyp erfolgreich gelöscht.

Nachrichtentyp `DeleteDocumentTypeRequest`. Listing 4.32 zeigt die WSDL-Definition des Datentyps `DeleteDocumentTypeRequest`.

```
<xs:element name="DeleteDocumentTypeRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q221:LoginToken"/>
   <!-- Name der Datenbank, in der der Dokumenttyp gespeichert ist. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
   <!-- ID des Dokumenttyps, der gelöscht werden soll. -->
   <xs:element minOccurs="0" name="DocumentTypeID" type="xs:unsignedLong"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.32: Nachrichtentyp DeleteDocumentTypeRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.32 referenzierten Datentypen auf.

Feldname	Verweis
LoginToken	3.3.1.2

4.5.4. Datentyp DocumentTypeSpec

Der Datentyp DocumentTypeSpec definiert die Werte, die bei Neuanlage oder Aktualisierung eines Dokumenttyps gesetzt werden sollen.
Listing 4.33 zeigt die WSDL-Definition des Datentyps DocumentTypeSpec.

```
<xs:complexType name="DocumentTypeSpec">
 <xs:sequence>
  <!-- Name des Dokumenttyps. -->
  <!-- Bei Neuanlage verpflichtend; Bei Update optional. -->
  <xs:element minOccurs="0" name="DocumentTypeName" nillable="true" type="xs:string"/>
  <!-- Optional: Dokumenttyp ist eine Zwischenablage (Pool). -->
  <xs:element minOccurs="0" name="ClipboardEnabled" nillable="true" type="xs:boolean"/>
  <!-- Optional: Volltext-Indexierung ist für diesen Dokumenttyp aktiviert. -->
  <xs:element minOccurs="0" name="FulltextEnabled" nillable="true" type="xs:boolean"/>
  <!-- Optional: OCR ist für diesen Dokumenttyp aktiviert. -->
  <xs:element minOccurs="0" name="OcrEnabled" nillable="true" type="xs:boolean"/>
  <!-- Optional: Dateiverschlüsselung ist für diesen Dokumnettyp aktiviert. -->
  <!-- Ab PROXESS 5; Erfordert Hochsicherheitsdatenbank. -->
  <xs:element minOccurs="0" name="FileEncryptionEnabled" nillable="true" type="xs:boolean"/>
  <!-- Optional: Dokumenten-Historie ist für diesen Dokumenttyp aktiviert. -->
  <!-- Ab PROXESS 5+. -->
  <xs:element minOccurs="0" name="DocumentHistoryEnabled" nillable="true" type="xs:boolean"/>
 </xs:sequence>
</xs:complexType>
```

Listing 4.33: Datentyp DocumentTypeSpec

Das Anlegen oder Ändern von Dokumenttypen wird intern in mehreren Schritten durchgeführt, von denen jeder einzelne Fehlschlagen kann.

1. Bei Neuanlage werden zunächst drei Parameter auf einmal verwendet, die bei einer Aktualisierung eines Dokumenttyps in drei einzelnen Aufrufen gesetzt werden:
   a) Bei Neuanlage: Anlegen des Dokumenttyps mit den Werten DocumentTypeName, ClipboardEnabled (default: false) und FulltextEnabled (default: false).
   b) Bei Update:
      i. Ggf. Namen des Dokumenttyps aktualisieren (Parameter DocumentTypeName).
      ii. Ggf. Zwischenablage des Dokumenttyps ändern (Parameter ClipboardEnabled).

iii. Ggf. Volltext-Indexierung des Dokumenttyps ändern (Parameter `FulltextEnabled`).

2. Ggf. OCR-Modus ändern/setzen (Parameter `OcrEnabled`).
3. Ggf. Dateiverschlüsselung aktivieren/deaktivieren (Parameter `FileEncryptionEnabled`).
4. Ggf. Dokumenten-Historie aktivieren/deaktivieren (Parameter `DocumentHistoryEnabled`).

Wenn z.B. bei der Dokumenttypanlage der interne Aufruf für `FileEncryption` fehlschlägt – z.B. weil keine Hochsicherheitsdatenbank gegeben ist –, wurde der Dokumenttyp bereits angelegt und die Aktion für `DocumentHistory` noch nicht ausgeführt.

Bei der Aktualisierung ist es möglich, dass Schritt 1.b.i ausgeführt wurde und Schritt 1.b.ii fehlgeschlagen ist. Der Fehler in Schritt 1.b.ii wird dem Anwender als Ausnahme übermittelt; alle evtl. folgenden Schritte wurden nicht ausgeführt.

4.6. Dateitypen

Dieser Abschnitt beschreibt die Funktionen zur Verwaltung von PROXESS Dateitypen.

4.6.1. Funktion `CreateFileType`

Signatur:

```
CreateFileTypeResponse CreateFileType(
      CreateFileTypeRequest request);
```

Die Funktion `CreateFileType` erzeugt einen neuen Dateityp mit den angegebenen Eigenschaften. Können alle Teilaktionen erfolgreich durchgeführt werden, wird der neu angelegte Dateityp in der Response zurückgegeben.

Achtung. Der Aufruf wird intern in mehreren Teilschritten durchgeführt, die im Fehlerfall zu einer nur teilweisen Durchführung der gesamten Aktion führen können. Die Teilschritte und deren Ausführungsreihenfolge werden bei der Dokumentation des Datentyps `FileTypeSpec` (4.6.4) beschrieben.

Nachrichtentyp `CreateFileTypeRequest`. Listing 4.34 zeigt die WSDL-Definition des Datentyps `CreateFileTypeRequest`.

```
<xs:element name="CreateFileTypeRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q222:LoginToken"/>
   <!-- Name der Datenbank, in der der Dateityp angelegt werden soll. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
   <!-- Beschreibt den Dateityp, der angelegt werden soll. -->
   <xs:element minOccurs="0" name="FileType" nillable="true" type="q223:FileTypeSpec"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.34: Nachrichtentyp CreateFileTypeRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.34 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2
`FileTypeSpec`	4.6.4

Nachrichtentyp `CreateFileTypeResponse`. Listing 4.35 zeigt die WSDL-Definition des Datentyps `CreateFileTypeResponse`.

```
<xs:element name="CreateFileTypeResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Neu angelegter Dateityp. -->
   <xs:element minOccurs="0" name="FileType" nillable="true" type="q235:FileType"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.35: Nachrichtentyp CreateFileTypeResponse

Die folgende Tabelle führt Verweise zu allen in Listing 4.35 referenzierten Datentypen auf.

Feldname	Verweis
`FileType`	3.3.2.4

4.6.2. Funktion `UpdateFileType`

Signatur:

```
UpdateFileTypeResponse UpdateFileType(
      UpdateFileTypeRequest request);
```

Die Funktion `UpdateFileType` aktualisiert die angegebenen Eigenschaften eines existierenden Dateityps. Können alle Teilaktionen erfolgreich durchgeführt werden, wird der geänderte Dateityp in der Response zurückgegeben.

Achtung. Der Aufruf wird intern in mehreren Teilschritten durchgeführt, die im Fehlerfall zu einer nur teilweisen Durchführung der gesamten Aktion führen können. Die Teilschritte und deren Ausführungsreihenfolge werden bei der Dokumentation des Datentyps `FileTypeSpec` (4.6.4) beschrieben.

Nachrichtentyp `UpdateFileTypeRequest`. Listing 4.36 zeigt die WSDL-Definition des Datentyps `UpdateFileTypeRequest`.

```
<xs:element name="UpdateFileTypeRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q236:LoginToken"/>
   <!-- Name der Datenbank, in der der Dateityp definiert ist. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
   <!-- ID des Dateitype, der geändert werden soll. -->
   <xs:element minOccurs="0" name="FileTypeID" type="xs:unsignedLong"/>
   <!-- Spezifiziert Änderungen, die am Dateityp durchgeführt werden sollen. -->
   <xs:element minOccurs="0" name="FileTypeChanges" nillable="true" type="q237:FileTypeSpec"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.36: Nachrichtentyp UpdateFileTypeRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.36 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2
`FileTypeSpec`	4.6.4

Nachrichtentyp `UpdateFileTypeResponse`. Listing 4.37 zeigt die WSDL-Definition des Datentyps `UpdateFileTypeResponse`.

```
<xs:element name="UpdateFileTypeResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Geänderter Dateityp. -->
   <xs:element minOccurs="0" name="FileType" nillable="true" type="q238:FileType"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.37: Nachrichtentyp UpdateFileTypeResponse

Die folgende Tabelle führt Verweise zu allen in Listing 4.37 referenzierten Datentypen auf.

Feldname	Verweis
FileType	3.3.2.4

4.6.3. Funktion DeleteFileType

Signatur:

```
EmptyMessage DeleteFileType(
      DeleteFileTypeRequest request);
```

Die Funktion DeleteFileType löscht einen Dateityp. Wenn der Aufruf keine Ausnahme auslöst, dann wurde der Dateityp erfolgreich gelöscht.

Nachrichtentyp DeleteFileTypeRequest. Listing 4.38 zeigt die WSDL-Definition des Datentyps DeleteFileTypeRequest.

```
1  <xs:element name="DeleteFileTypeRequest">
2   <xs:complexType>
3    <xs:sequence>
4     <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
5     <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q239:LoginToken"/>
6     <!-- Name der Datenbank, in der der Dateityp gespeichert ist. -->
7     <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
8     <!-- ID des Dateityps, der gelöscht werden soll. -->
9     <xs:element minOccurs="0" name="FileTypeID" type="xs:unsignedLong"/>
10   </xs:sequence>
11  </xs:complexType>
12 </xs:element>
```

Listing 4.38: Nachrichtentyp DeleteFileTypeRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.38 referenzierten Datentypen auf.

Feldname	Verweis
LoginToken	3.3.1.2

4.6.4. Datentyp FileTypeSpec

Der Datentyp FileTypeSpec beschreibt Eigenschaften, die bei der Neuanlage oder Aktualisierung eines Dateityps relevant sind.

Listing 4.39 zeigt die WSDL-Definition des Datentyps `FileTypeSpec`.

```
<xs:complexType name="FileTypeSpec">
 <xs:sequence>
  <!-- Name des Dateityps. -->
  <!-- Bei Neuanlage verpflichtend; Bei Aktualisierung optional. -->
  <xs:element minOccurs="0" name="FileTypeName" nillable="true" type="xs:string"/>
  <!-- Dateityperweiterung. -->
  <!-- Bei Neuanlage verpflichtend; Bei Aktualisierung optional. -->
  <xs:element minOccurs="0" name="Extension" nillable="true" type="xs:string"/>
  <!-- Optional: Universeller Dateityp. -->
  <xs:element minOccurs="0" name="UniversalEnabled" nillable="true" type="xs:boolean"/>
  <!-- Optional: Ocr-Modus. -->
  <xs:element minOccurs="0" name="OcrMode" nillable="true" type="q224:OcrMode"/>
  <!-- Optional: Indikator für Volltext-Parser. -->
  <xs:element minOccurs="0" name="FulltextType" nillable="true" type="q225:FulltextType"/>
  <!-- Optional: ID der Vorlagendatei für neue Dateien. -->
  <xs:element minOccurs="0" name="TemplateFileID" nillable="true" type="xs:unsignedLong"/>
  <!-- Optional: Liste von geänderten oder neuen Editoren. -->
  <xs:element minOccurs="0" name="Editors" nillable="true" type="q226:ArrayOfEditor"/>
  <!-- Optional: Liste von Editoren, die gelöscht werden sollen. -->
  <!-- Wird bei Neuanlage ignoriert. -->
  <xs:element minOccurs="0" name="DeleteEditors" nillable="true" type="q227:ArrayOfEditorType"/>
 </xs:sequence>
</xs:complexType>
```

Listing 4.39: Datentyp FileTypeSpec

Die folgende Tabelle führt Verweise zu allen in Listing 4.39 referenzierten Datentypen auf.

Feldname	Verweis
`OcrMode`	4.6.4
`FulltextType`	3.3.2.4
`Editor`	4.6.5
`EditorType`	4.6.5

Listing 4.40 zeigt die WSDL-Definition des Datentyps `OcrMode`.

```
<xs:simpleType name="OcrMode">
 <xs:restriction base="xs:string">
  <!-- Kein OCR Modus aktiviert. -->
  <xs:enumeration value="NoOcr"/>
  <!-- OCR Eingabe. -->
  <xs:enumeration value="Input"/>
  <!-- OCR Ausgabe. -->
  <xs:enumeration value="Output"/>
 </xs:restriction>
</xs:simpleType>
```

Listing 4.40: Datentyp OcrMode

Die Anlage und Aktualisierung eines Dateityps läuft intern in mehreren Einzelschritten ab, von denen jeder einzelne Fehlschlagen kann. Im Fehlerfall werden bereits durchgeführte Änderungen nicht rückgängig gemacht. Die Teilschritte werden in der folgenden Reihenfolge durchgeführt:

1. Setzen/Ändern der Basiseigenschaften `FileTypeName`, `Extension`, `FulltextType` (default: `FulltextType.None`), `UniversalEnabled` (default: `false`).
2. Ggf. Setzen/Ändern des OCR-Modus (Parameter `OcrMode`).
3. Ggf. Setzen/Ändern der ID der Vorlagendatei (Parameter `TemplateFileID`).
4. Ggf. sequentielles Abarbeiten der Editoren-Definitionen (Parameter `Editors`).
5. Ggf. sequentielles Abarbeiten der zu löschenden Editoren (Parameter `DeleteEditors`).

Es wird empfohlen, die optionalen Parameter für die Schritte 4 und 5 entweder gar nicht zu setzen oder nur isoliert in einzelnen Aufrufen.

4.6.5. Datentyp `Editor`

Der Datentyp `Editor` wird zur Spezifikation eines Anzeige-, Editier- bzw. Printout-Programmes für einen bestimmten Dateityp verwendet. Für eine genaue Dokumentation der einzelnen Parameter wird an dieser Stelle auf die Dokumentation von PROXESS DocumentManger bzw. PROXESS Administrator verwiesen.
Listing 4.41 zeigt die WSDL-Definition des Datentyps `Editor`.

```
<xs:complexType name="Editor">
 <xs:sequence>
  <!-- Typ der Editors (lesen, erstellen, drucken) xD -->
  <xs:element minOccurs="0" name="Type" type="q231:EditorType"/>
  <!-- Beschreibung des Editors. -->
  <xs:element minOccurs="0" name="Description" nillable="true" type="xs:string"/>
  <!-- Aufrufmethode des Editors. -->
  <xs:element minOccurs="0" name="CallConvention" type="q232:InvocationMethod"/>
  <xs:element minOccurs="0" name="CommandLine" nillable="true" type="xs:string"/>
  <xs:element minOccurs="0" name="DdeCommand" nillable="true" type="xs:string"/>
  <xs:element minOccurs="0" name="ServiceName" nillable="true" type="xs:string"/>
 </xs:sequence>
</xs:complexType>
```

Listing 4.41: Datentyp Editor

Listing 4.42 zeigt die WSDL-Definition des Datentyps `EditorType`.

```
<xs:simpleType name="EditorType">
 <xs:restriction base="xs:string">
  <!-- Editor zum Erstellen von Dateien. -->
  <xs:enumeration value="Create"/>
  <!-- Editor zum Ansehen von Dateien. -->
  <xs:enumeration value="View"/>
  <!-- Editor zum Ausdrucken von Dateien. -->
  <xs:enumeration value="Print"/>
 </xs:restriction>
</xs:simpleType>
```

Listing 4.42: Datentyp EditorType

Listing 4.43 zeigt die WSDL-Definition des Datentyps `InvocationMethod`.

```
<xs:simpleType name="InvocationMethod">
 <xs:restriction base="xs:string">
  <xs:enumeration value="Program"/>
  <xs:enumeration value="DDE"/>
  <xs:enumeration value="Quickview"/>
  <xs:enumeration value="Scanning"/>
  <xs:enumeration value="Diaclip"/>
  <xs:enumeration value="LocalRegistry"/>
 </xs:restriction>
</xs:simpleType>
```

Listing 4.43: Datentyp InvocationMethod

Im folgenden werden Einschränkungen für die Kombination von Editor-Parametern angegeben, die das Verhalten von PROXESS AdminFE wiedergeben. Vom Benutzer übergebene Editoren, die diese Einschränkungen nicht einhalten, werden einfach ignoriert.

Der Webservice prüft in Abhängigkeit von der Aufrufmethode, dass nur erlaubte Editor-Typen verwendet werden. Lediglich die Kombination von `QuickView` mit `EditorType.Create` ist verboten:

	Create	View	Print
Program	YES	YES	YES
DDE	YES	YES	YES
Quickview	**NO**	YES	YES
Scanning	YES	YES	YES
Diaclip	YES	YES	YES
LocalRegistry	YES	YES	YES

Der Webservice prüft in Abhängigkeit von der Aufrufmethode, dass folgende Bedingungen für die Parameter `CommandLine`, `DdeCommand` und `ServiceName` eingehalten werden. NULL bedeutet dass der Parameter den Wert `null` haben muss; REQUIRED bedeutet, dass das der Parameter einen von `null` verschiedenen Wert haben muss.

	CommandLine	DdeCommand	ServiceName
Program	REQUIRED	NULL	NULL
DDE	REQUIRED	REQUIRED	REQUIRED
Quickview	NULL	NULL	NULL
Scanning	NULL	NULL	NULL
Diaclip	REQUIRED	NULL	NULL
LocalRegistry	NULL	NULL	NULL

4.7. Felder

Dieser Abschnitt beschreibt die Funktionen zur Verwaltung von Datenbank- und Dokumenttyp-Feldern.

Die Menge von Datenbankfeldern einer PROXESS-Datenbank stellt den Grundstock an Merkmalsfeldern für alle Dokumenttypen dieser Datenbank dar. Jedes Datenbankfeld kann pro Dokumenttyp mit Dokumenttyp-spezifischen Eigenschaften (z.B. Name, Sichtbarkeit, Pflichteingabe) versehen werden. Ein auf diese Weise markiertes Feld heißt Dokumenttyp-Feld.

Das Erstellen und Löschen von Datenbankfelder schlägt sich im Datenbanklayout des Backend-Systems nieder. Das Erstellen und Löschen von Dokumenttypfelder besteht nur aus dem Hinzufügen und Löschen von Markern.

4.7.1. Funktion `CreateDatabaseField`

Signatur:

```
CreateDatabaseFieldResponse CreateDatabaseField(
      CreateDatabaseFieldRequest request);
```

Die Funktion `CreateDatabaseField` erzeugt ein neues Datenbankfeld mit den angegebenen Eigenschaften. Wenn das Datenbankfeld angelegt werden und alle Eigenschaften gesetzt werden konnten, wird das Datenbankfeld in der Response zurückgegeben.

Nachrichtentyp `CreateDatabaseFieldRequest`. Listing 4.44 zeigt die WSDL-Definition des Datentyps `CreateDatabaseFieldRequest`.

```
<xs:element name="CreateDatabaseFieldRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q240:LoginToken"/>
   <!-- Name der Datenbank, in der das Feld angelegt werden soll. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
   <!-- Beschreibt die Eigenschaften des anzulegenden Feldes. -->
   <xs:element minOccurs="0" name="Field" nillable="true" type="q241:FieldSpec"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.44: Nachrichtentyp CreateDatabaseFieldRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.44 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2
`FieldSpec`	4.7.7

Nachrichtentyp `CreateDatabaseFieldResponse`. Listing 4.45 zeigt die WSDL-Definition des Datentyps `CreateDatabaseFieldResponse`.

```
<xs:element name="CreateDatabaseFieldResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Neu angelegtes Feld. -->
   <xs:element minOccurs="0" name="Field" nillable="true" type="q245:FieldDescription"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.45: Nachrichtentyp CreateDatabaseFieldResponse

Die folgende Tabelle führt Verweise zu allen in Listing 4.45 referenzierten Datentypen auf.

Feldname	Verweis
`FieldDescription`	3.3.2.3

4.7.2. Funktion `UpdateDatabaseField`

Signatur:

```
UpdateDatabaseFieldResponse UpdateDatabaseField(
      UpdateDatabaseFieldRequest request);
```

Die Funktion `UpdateDatabaseField` aktualisiert ein bestehendes Datenbankfeld mit den angegebenen Eigenschaften. Wenn alle Eigenschaften gesetzt werden konnten, wird das geänderte Datenbankfeld in der Response zurückgegeben.

Nachrichtentyp `UpdateDatabaseFieldRequest`. Listing 4.46 zeigt die WSDL-Definition des Datentyps `UpdateDatabaseFieldRequest`.

```
<xs:element name="UpdateDatabaseFieldRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q246:LoginToken"/>
   <!-- Name der Datenbank, in der das Feld gespeichert ist. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
   <!-- Datenbankname des Felds, das aktualisiert werden soll. -->
   <xs:element minOccurs="0" name="PhysicalName" nillable="true" type="xs:string"/>
   <!-- Beschreibt die Eigenschaften des Feldes, die aktualisiert werden sollen. -->
   <xs:element minOccurs="0" name="FieldChanges" nillable="true" type="q247:FieldSpec"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.46: Nachrichtentyp UpdateDatabaseFieldRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.46 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2
`FieldSpec`	4.7.7

Nachrichtentyp `UpdateDatabaseFieldResponse`. Listing 4.47 zeigt die WSDL-Definition des Datentyps `UpdateDatabaseFieldResponse`.

```
<xs:element name="UpdateDatabaseFieldResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Aktualisiertes Datenbankfeld. -->
   <xs:element minOccurs="0" name="Field" nillable="true" type="q248:FieldDescription"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.47: Nachrichtentyp UpdateDatabaseFieldResponse

Die folgende Tabelle führt Verweise zu allen in Listing 4.47 referenzierten Datentypen auf.

Feldname	Verweis
`FieldDescription`	3.3.2.3

4.7.3. Funktion `DeleteDatabaseField`

Signatur:

```
EmptyMessage DeleteDatabaseField(
      DeleteDatabaseFieldRequest request);
```

Die Funktion `DeleteDatabaseField` löscht ein Datenbank-Feld. Wenn diese Funktion keine Ausnahme auslöst, wurde das Datenbank-Feld erfolgreich gelöscht.

Nachrichtentyp `DeleteDatabaseFieldRequest`. Listing 4.48 zeigt die WSDL-Definition des Datentyps `DeleteDatabaseFieldRequest`.

```
1  <xs:element name="DeleteDatabaseFieldRequest">
2   <xs:complexType>
3    <xs:sequence>
4     <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
5     <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q249:LoginToken"/>
6     <!-- Name der Datenbank, in der das Feld gespeichert ist. -->
7     <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
8     <!-- Name des Feldes, das gelöscht werden soll. -->
9     <xs:element minOccurs="0" name="PhysicalName" nillable="true" type="xs:string"/>
10   </xs:sequence>
11  </xs:complexType>
12 </xs:element>
```

Listing 4.48: Nachrichtentyp DeleteDatabaseFieldRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.48 referenzierten Datentypen auf.

Feldname	Verweis
LoginToken	3.3.1.2

4.7.4. Funktion `CreateDocumentTypeField`

Signatur:

```
CreateDocumentTypeFieldResponse CreateDocumentTypeField(
      CreateDocumentTypeFieldRequest request);
```

Die Funktion `CreateDocumentTypeField` erzeugt ein neues Dokumenttyp-Feld mit den Angegebenen Eigenschaften. Konnte das Dokumenttyp-Feld erzeugt werden und alle Eigenschaften gesetzt werden, wird das Dokumenttyp-Feld in der Response zurückgegeben.

Der Begriff *Erzeugen* ist in diesem Fall so zu verstehen, dass ein bestehendes Datenbank-Feld (mit festem Namen und Datentyp) mit einer Reihe von Dokumenttyp-spezifischen Eigenschaften markiert wird. Es wird kein neues physisches Feld in der Datenbank erzeugt; jedes Dokumenttyp-Feld ist von einem bestimmten Datenbank-Feld abhängig.

Nachrichtentyp `CreateDocumentTypeFieldRequest`. Listing 4.49 zeigt die WSDL-Definition des Datentyps `CreateDocumentTypeFieldRequest`.

```
<xs:element name="CreateDocumentTypeFieldRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q250:LoginToken"/>
   <!-- Name der Datenbank, in der sich das Feld befindet. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
   <!-- ID des Dokumenttyps, der das Feld spezialisieren soll. -->
   <xs:element minOccurs="0" name="DocumentTypeID" type="xs:unsignedLong"/>
   <!-- Name des Feldes, das spezialisiert werden soll. -->
   <xs:element minOccurs="0" name="PhysicalName" nillable="true" type="xs:string"/>
   <!-- Beschreibung weiterer Eigenschaften des Dokumenttyp-Feldes. -->
   <xs:element minOccurs="0" name="FieldChanges" nillable="true" type="q251:FieldSpec"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.49: Nachrichtentyp CreateDocumentTypeFieldRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.49 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2
`FieldSpec`	4.7.7

Nachrichtentyp `CreateDocumentTypeFieldResponse`. Listing 4.50 zeigt die WSDL-Definition des Datentyps `CreateDocumentTypeFieldResponse`.

```
<xs:element name="CreateDocumentTypeFieldResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Neu erstelltes Dokumenttyp-Feld. -->
   <xs:element minOccurs="0" name="Field" nillable="true" type="q252:FieldDescription"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.50: Nachrichtentyp CreateDocumentTypeFieldResponse

Die folgende Tabelle führt Verweise zu allen in Listing 4.50 referenzierten Datentypen auf.

Feldname	Verweis
`FieldDescription`	3.3.2.3

4.7.5. Funktion `UpdateDocumentTypeField`

Signatur:

```
UpdateDocumentTypeFieldResponse UpdateDocumentTypeField(
      UpdateDocumentTypeFieldRequest request);
```

Die Funktion `UpdateDocumentTypeField` aktualisiert ein bestehendes Dokumenttyp-Feld mit den angegebenen Eigenschaften. Konnten alle Eigenschaften gesetzt werden, wird das geänderte Dokumenttyp-Feld in der Response zurückgegeben.

Nachrichtentyp `UpdateDocumentTypeFieldRequest`. Listing 4.51 zeigt die WSDL-Definition des Datentyps `UpdateDocumentTypeFieldRequest`.

```
1  <xs:element name="UpdateDocumentTypeFieldRequest">
2   <xs:complexType>
3    <xs:sequence>
4     <!-- Identifiziert die PROXESS-Session des Webservice-Clients. -->
5     <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q253:LoginToken"/>
6     <!-- Name der Datenbank, in der das Feld gespeichert ist. -->
7     <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
8     <!-- ID des Dokumenttyps, zu dem das Dokumenttypfeld gehört. -->
9     <xs:element minOccurs="0" name="DocumentTypeID" type="xs:unsignedLong"/>
10    <!-- Datenbankname des Dokumenttypfeldes. -->
11    <xs:element minOccurs="0" name="PhysicalName" nillable="true" type="xs:string"/>
12    <!-- Spezifiziert Eigenschaften des Feldes, die aktualisiert werden sollen. -->
13    <xs:element minOccurs="0" name="FieldChanges" nillable="true" type="q254:FieldSpec"/>
14   </xs:sequence>
15  </xs:complexType>
16 </xs:element>
```

Listing 4.51: Nachrichtentyp UpdateDocumentTypeFieldRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.51 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2
`FieldSpec`	4.7.7

Nachrichtentyp `UpdateDocumentTypeFieldResponse`. Listing 4.52 zeigt die WSDL-Definition des Datentyps `UpdateDocumentTypeFieldResponse`.

```
<xs:element name="UpdateDocumentTypeFieldResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Aktualisiertes Dokumenttypfeld. -->
   <xs:element minOccurs="0" name="Field" nillable="true" type="q255:FieldDescription"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.52: Nachrichtentyp UpdateDocumentTypeFieldResponse

Die folgende Tabelle führt Verweise zu allen in Listing 4.52 referenzierten Datentypen auf.

Feldname	Verweis
`FieldDescription`	3.3.2.3

4.7.6. Funktion `DeleteDocumentTypeField`

Signatur:

```
EmptyMessage DeleteDocumentTypeField(
       DeleteDocumentTypeFieldRequest request);
```

Die Funktion `DeleteDocumentTypeField` löscht ein Dokumenttyp-Feld. *Löschen* bedeutet in diesem Zusammenhang, dass das Feld (aus Sicht des Dokumenttyps) wieder zu einem Datenbank-Feld wird. Wenn der Aufruf dieser Funktion keine Ausnahme auslöst, ist das Dokumenttyp-Feld erfolgreich gelöscht worden.

Nachrichtentyp `DeleteDocumentTypeFieldRequest`. Listing 4.53 zeigt die WSDL-Definition des Datentyps `DeleteDocumentTypeFieldRequest`.

```
<xs:element name="DeleteDocumentTypeFieldRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q256:LoginToken"/>
   <!-- Name der Datenbank, in der das Dokumenttyp-Feld gespeichert ist. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
   <!-- ID des Dokumenttyps. -->
   <xs:element minOccurs="0" name="DocumentTypeID" type="xs:unsignedLong"/>
   <!-- Name des Feldes, das wieder zu einem Standard-Feld gemacht werden soll. -->
   <xs:element minOccurs="0" name="PhysicalName" nillable="true" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.53: Nachrichtentyp DeleteDocumentTypeFieldRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.53 referenzierten Datentypen auf.

Feldname	Verweis
LoginToken	3.3.1.2

4.7.7. Datentyp `FieldSpec`

Der Datentyp `FieldSpec` spezifiziert die Eigenschaften, die bei der Anlage oder Aktualisierung eines Feldes gesetzt werden sollen.
Listing 4.54 zeigt die WSDL-Definition des Datentyps `FieldSpec`.

```
<xs:complexType name="FieldSpec">
 <xs:sequence>
  <!-- Name des Feldes. -->
  <!-- Bei Neuanlage verpflichtend; bei Update null/kein Wert. -->
  <xs:element name="PhysicalName" nillable="true" type="xs:string"/>
  <!-- Optional: Anzeigename des Feldes. -->
  <xs:element minOccurs="0" name="DisplayName" nillable="true" type="xs:string"/>
  <!-- Optional: Beschreibung des Feldes. -->
  <xs:element minOccurs="0" name="Description" nillable="true" type="xs:string"/>
  <!-- Datentyp des Feldes. -->
  <!-- Bei Neuanlage verpflichtend; bei Update null/kein Wert. -->
  <xs:element minOccurs="0" name="DataType" nillable="true" type="q242:FieldDataType"/>
  <!-- Länge des Feldes. -->
  <!-- Bei Neuanlage optional; bei Update null/kein Wert. -->
  <xs:element minOccurs="0" name="Length" nillable="true" type="xs:int"/>
  <!-- Optional: Das Feld ist sichtbar (in der Standardmaske). -->
  <xs:element minOccurs="0" name="Visible" nillable="true" type="xs:boolean"/>
  <!-- Optional: Das Feld ist ein Pflichtfeld. -->
  <xs:element minOccurs="0" name="Mandatory" nillable="true" type="xs:boolean"/>
  <!-- Optional: Werte des Feldes werden verschlüsselt gespeichert. -->
  <xs:element minOccurs="0" name="Encrypted" nillable="true" type="xs:boolean"/>
  <!-- Optional: Layout des Feldes. -->
  <xs:element minOccurs="0" name="Layout" nillable="true" type="q243:ArrayOfNullableOfint"
  xmlns:q243="http://schemas.datacontract.org/2004/07/System"/>
  <!-- Optional: ID der Validierungsregel des Feldes. -->
  <xs:element minOccurs="0" name="ValidationRuleID" nillable="true" type="xs:unsignedLong"/>
 </xs:sequence>
</xs:complexType>
```

Listing 4.54: Datentyp FieldSpec

Achtung. Bei der Anlage/Aktualisierung von Datenbank- oder Dokumenttyp-Feldern ist es generell möglich, dass eine Ausnahme auftritt und nicht alle spezifizierten Aktionen durchgeführt worden sind. In allen Fällen werden Aktionen mit einem Feld in der folgenden Reihenfolge durchgeführt:

1. Feld erstellen/aktualisieren.
    a) Ggf. Anzeigenamen setzen (`DisplayName`).
    b) Ggf. Beschreibung setzen (`Description`).
    c) Ggf. Sichtbarkeit setzen (`IsVisible`).
    d) Ggf. Pflichteingabe setzen (`IsMandatory`).
    e) Ggf. Verschlüsselung setzen (`IsEncrypted`).
    f) Ggf. Feldlayout setzen (`Layout`).
2. Validierungsregel aktualisieren.

Für die einzelnen Anwendungsfälle werden einige der Schritte 1.a bis 1.f in einem einzelnen Aufruf durchgeführt:

- `CreateDatabaseField`. Alle Schritte 1.a bis 1.f werden in einem einzelnen Aufruf durchgeführt.
- `UpdateDatabaseField`. Alle Schritte werden in jeweils eigenen Aufrufen durchgeführt.
- `CreateDocumentTypeField`. Schriit 1.a und 1.b werden zusammen in einem Aufruf durchgeführt. Die übrigen Schritte werden in jeweils eigenen Aufrufen durchgeführt.
- `UpdateDocumentTypeField`. Alle Schritte werden in jeweils eigenen Aufrufen durchgeführt.

4.8. Validierungsregeln

Dieser Abschnitt dokumentiert die Funktionen zur Verwaltung von Validierungsregeln.

4.8.1. Funktion `CreateValidationRule`

Signatur:

```
CreateValidationRuleResponse CreateValidationRule(
      CreateValidationRuleRequest request);
```

Die Funktion `CreateValidationRule` erstellt eine neue Validierungsregel mit den angegebenen Eigenschaften. Kann die Validierungsregel erzeugt werden und können alle Eigenschaften gesetzt werden, wird die neue Validierungsregel in der Response zurückgegeben.

Nachrichtentyp `CreateValidationRuleRequest`. Listing 4.55 zeigt die WSDL-Definition des Datentyps `CreateValidationRuleRequest`.

```
<xs:element name="CreateValidationRuleRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q257:LoginToken"/>
   <!-- Name der Datenbank, in der die Validierungsregel angelegt werden soll. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
   <!-- Beschreibung der Eigenschaften der neuen Validierungsregel. -->
   <xs:element minOccurs="0" name="ValidationRule" nillable="true"
    type="q258:ValidationRuleSpec"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.55: Nachrichtentyp CreateValidationRuleRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.55 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2
`ValidationRuleSpec`	4.8.4

Nachrichtentyp `CreateValidationRuleResponse`. Listing 4.56 zeigt die WSDL-Definition des Datentyps `CreateValidationRuleResponse`.

```
<xs:element name="CreateValidationRuleResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Neu angelegte Validierungsregel. -->
   <xs:element minOccurs="0" name="ValidationRule" nillable="true" type="q264:ValidationRule"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.56: Nachrichtentyp CreateValidationRuleResponse

Die folgende Tabelle führt Verweise zu allen in Listing 4.56 referenzierten Datentypen auf.

Feldname	Verweis
`ValidationRule`	3.3.2.6

4.8.2. Funktion `UpdateValidationRule`

Signatur:

```
UpdateValidationRuleResponse UpdateValidationRule(
```

```
    UpdateValidationRuleRequest request);
```

Die Funktion `UpdateValidationRule` aktualisiert eine Validierungsregel mit den angegebenen Eigenschaften. Können alle Änderungen erfolgreich durchgeführt werden, wird die geänderte Validierungsregel in der Response zurückgegeben.

Nachrichtentyp `UpdateValidationRuleRequest`. Listing 4.57 zeigt die WSDL-Definition des Datentyps `UpdateValidationRuleRequest`.

```
<xs:element name="UpdateValidationRuleRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q265:LoginToken"/>
   <!-- Name der Datenbank, in der die Validierungsregel gespeichert ist. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
   <!-- ID der Validierungsregel, die aktualisiert werden soll. -->
   <xs:element minOccurs="0" name="ValidationRuleID" type="xs:unsignedLong"/>
   <!-- Beschreibt die zu ändernden Eigenschaften der Validierungsregel. -->
   <xs:element minOccurs="0" name="ValidationRuleChanges" nillable="true"
     type="q266:ValidationRuleSpec"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.57: Nachrichtentyp UpdateValidationRuleRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.57 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2

Nachrichtentyp `UpdateValidationRuleResponse`. Listing 4.58 zeigt die WSDL-Definition des Datentyps `UpdateValidationRuleResponse`.

```
<xs:element name="UpdateValidationRuleResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Aktualisierte Validierungsregel. -->
   <xs:element minOccurs="0" name="ValidationRule" nillable="true" type="q267:ValidationRule"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing 4.58: Nachrichtentyp UpdateValidationRuleResponse

Die folgende Tabelle führt Verweise zu allen in Listing 4.58 referenzierten Datentypen auf.

Feldname	Verweis
`ValidationRule`	3.3.2.6

4.8.3. Funktion `DeleteValidationRule`

Signatur:

```
EmptyMessage DeleteValidationRule(
      DeleteValidationRuleRequest request);
```

Die Funktion `DeleteValidationRule` löscht eine Validierungsregel. Wenn der Aufruf der Funktion keine Ausnahme auslöst, ist die Validierungsregel erfolgreich gelöscht worden.

Nachrichtentyp `DeleteValidationRuleRequest`. Listing 4.59 zeigt die WSDL-Definition des Datentyps `DeleteValidationRuleRequest`.

```
1  <xs:element name="DeleteValidationRuleRequest">
2   <xs:complexType>
3    <xs:sequence>
4     <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
5     <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q268:LoginToken"/>
6     <!-- Name der Datenbank, in der die Validierungsregel gespeichert ist. -->
7     <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
8     <!-- ID der Validierungsregel, die gelöscht werden soll. -->
9     <xs:element minOccurs="0" name="ValidationRuleID" type="xs:unsignedLong"/>
10   </xs:sequence>
11  </xs:complexType>
12 </xs:element>
```

Listing 4.59: Nachrichtentyp DeleteValidationRuleRequest

Die folgende Tabelle führt Verweise zu allen in Listing 4.59 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2

4.8.4. Datentyp `ValidationRuleSpec`

Der Datentyp `ValidationRuleSpec` spezifiziert die Eigenschaften einer Validierungsregel, die bei der Neuanlage oder Aktualisierung gesetzt werden sollen. Für Bereichs-VR und Thesaurus-VR müssen verschiedene Felder gesetzt werden.

Listing 4.60 zeigt die WSDL-Definition des Datentyps `ValidationRuleSpec`.

```
<xs:complexType name="ValidationRuleSpec">
 <xs:sequence>
  <!-- Name der Validierungsregel. -->
  <!-- Beu Neuanlage verpflichtend; bei Update optional. -->
  <xs:element minOccurs="0" name="Name" nillable="true" type="xs:string"/>
  <!-- Typ der Validierungsregel (Bereichs-VR oder Thesaurus-VR). -->
  <!-- Bei Neuanlage verpflichtend; bei Update null/kein Wert -->
  <xs:element minOccurs="0" name="Type" nillable="true" type="q259:ValidationRuleType"/>
  <!-- Für Bereichs-VR: Untere Bereichsgrenze. -->
  <!-- Für Thesaurus-VR: null/kein Wert. -->
  <!-- Bei Neuanlage verpflichtend; bei Update optional -->
  <xs:element minOccurs="0" name="VRangeLower" nillable="true" type="xs:string"/>
  <!-- Für Bereichs-VR: Obere Bereichsgrenze. -->
  <!-- Für Thesaurus-VR: null/kein Wert. -->
  <!-- Bei Neuanlage verpflichtend; bei Update optional -->
  <xs:element minOccurs="0" name="VRangeUpper" nillable="true" type="xs:string"/>
  <!-- Für Bereichs-VR: null/kein Wert. -->
  <!-- Für Thesaurus-VR: Wortliste kann durch Benutzer editiert werden. -->
  <!-- Bei Neuanlage verpflichtend; bei Update optional -->
  <xs:element minOccurs="0" name="ThesUserUpdateAllowed" nillable="true" type="xs:boolean"/>
  <!-- Für Bereichs-VR: null/kein Wert. -->
  <!-- Für Thesaurus-VR: Maximale Länge von Wörtern in der Wortliste. -->
  <!-- Bei Neuanlage verpflichtend; bei Update optional -->
  <xs:element minOccurs="0" name="ThesMaxWordLength" nillable="true" type="xs:int"/>
  <!-- Für Bereichs-VR: null/kein Wert. -->
  <!-- Für Thesaurus-VR: Spezifiziert die Art der Änderung der Wortliste. -->
  <!-- Bei Neuanlage ignoriert; bei Update optional -->
  <xs:element minOccurs="0" name="ThesChangeBehaviour" nillable="true"
    type="q260:ThesaurusChangeBehaviour"/>
  <!-- Für Bereichs-VR: null/kein Wert. -->
  <!-- Für Thesaurus-VR: Wortliste bzw. zusätzliche Wörter für die Wortliste. -->
  <!-- Bei Neuanlage verpflichtend; bei Update optional -->
  <xs:element minOccurs="0" name="ThesWords" nillable="true" type="q261:ArrayOfstring"/>
 </xs:sequence>
</xs:complexType>
```

Listing 4.60: Datentyp ValidationRuleSpec

Achtung. Die Verwendung des Datentyps `ValidationRuleSpec` (durch die Funktionen `CreateValidationRule` und `UpdateValidationRule`) führt für Thesaurus-Validierungsregeln zu einer internen schrittweisen Ausführung. Dies kann im Fehlerfall zu einer teilweisen Ausführung führen.

1. Anlegen der Thesaurus-Validierungsregel bzw. Ändern der Basiseigenschaften `Name`, `Type`, `ThesUserUpdateAllowed`, `ThesMaxWordLength`.

2. Sequentielles Abarbeiten der Thesaurus-Wortliste.

Wenn bei der Verarbeitung der Wortliste an einer beliebigen Stelle ein Fehler auftritt bedeutet das,

1. dass die nachfolgenden Worte in der übergebenen Wortliste nicht mehr verarbeitet werden und
2. dass der Thesaurus bereits angelegt worden ist bzw. das die Basiseigenschaften bereits geändert wurden.

Die Änderungsaktionen für alle Bereichs-Validierungsregeln werden intern mit einem einzelnen Aufruf durchgeführt und sind damit nicht von der unvollständigen Verarbeitung im Fehlerfall betroffen.

Die Enumeration `ThesaurusChangeBehaviour` gibt bei der Aktualisierung von Thesaurus-Validierungsregeln an, wie die übergebene Wortliste ausgewertet werden soll. Es ist möglich

1. ausschließlich neue Worte in die Wortliste aufzunehmen (`AddOnly`; Vereinigung von übergebener und gespeicherter Wortmenge) und
2. die übergebene Wortliste genau so zu übernehmen (`AddAndRemove`; fehlende Worte werden entfernt, neue Worte werden hinzugefügt).

Listing 4.61 zeigt die WSDL-Definition des Datentyps `ThesaurusChangeBehaviour`.

```
1 <xs:simpleType name="ThesaurusChangeBehaviour">
2  <xs:restriction base="xs:string">
3   <!-- Fügt nur neue Worte zu der Wortliste hinzu. -->
4   <xs:enumeration value="AddOnly"/>
5   <!-- Fügt neue Wörter zu der Wortliste hinzu und
6        entfernt fehlende Worte aus der Wortliste. -->
7   <xs:enumeration value="AddAndRemove"/>
8  </xs:restriction>
9 </xs:simpleType>
```

Listing 4.61: Datentyp ThesaurusChangeBehaviour

A. GeneralSearch/SearchCondition Beispiele

Im Folgenden werden einige Beispiele für den Aufbau von initialen `SearchRequests` gegeben, mit denen Suchkriterien spezifiziert werden. Die Beispiele zeigen jeweils den konkreten Aufbau der XML-Daten eines `SearchRequest`. Die Daten stammen aus geloggten Aufrufen eines Testclients für den Webservice. Der durch `...` abgekürzte Namespace ist in allen Beispielen `http://schemas.datacontract.org/2004/07/-Akzentum.Proxess.Webservice.MessageContracts`.

Die folgende Tabelle gibt einen Überblick über die Abbildungen.

Beinhaltet	Verweis
`BranchCondition, FieldCondition`	A.1
`FulltextCondition`	A.2
`FieldRangeCondition`	A.3
`InCondition`	A.4
`NotCondition, FieldCondition`	A.5

```
<SearchRequest xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="...">
 <LoginToken xmlns:d2p1="http://schemas.akzentum.de/2013/07/proxessws">
  <d2p1:Id>b96886b2-edb0-4d5d-a568-5fefa7986f22</d2p1:Id>
 </LoginToken>
 <ChunkID>0</ChunkID>
 <Query xmlns:d2p1="http://schemas.akzentum.de/2013/07/proxessws">
  <d2p1:DatabaseName>ArchivDB</d2p1:DatabaseName>
  <d2p1:MaxHits>0</d2p1:MaxHits>
  <d2p1:HitsPerPage>100</d2p1:HitsPerPage>
  <!-- CreateDate >= 2013-03-01 OR DocDes LIKE 'Super%' -->
  <d2p1:RootCondition i:type="d2p1:BranchCondition">
   <d2p1:Conjunction>Or</d2p1:Conjunction>
   <d2p1:Conditions>
    <d2p1:SearchCondition i:type="d2p1:FieldCondition">
     <d2p1:FieldName>CreateDate</d2p1:FieldName>
     <d2p1:Comparator>IsGreaterOrEqualThan</d2p1:Comparator>
     <d2p1:Value>2013-03-01</d2p1:Value>
    </d2p1:SearchCondition>
    <d2p1:SearchCondition i:type="d2p1:FieldCondition">
     <d2p1:FieldName>DocDes</d2p1:FieldName>
     <d2p1:Comparator>IsLike</d2p1:Comparator>
     <d2p1:Value>Super%</d2p1:Value>
    </d2p1:SearchCondition>
   </d2p1:Conditions>
  </d2p1:RootCondition>
  <!-- -->
```

```
  <d2p1:OrderBy i:nil="true" />
 </Query>
 <QueryKey i:nil="true" />
 <Status>Initiate</Status>
</SearchRequest>
```

Listing A.1: XML-Aufbau einer `BranchCondition`

```
<SearchRequest xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="...">
 <LoginToken xmlns:d2p1="http://schemas.akzentum.de/2013/07/proxessws">
  <d2p1:Id>388a5927-db4d-4e7e-9607-71d4c8a9c4e4</d2p1:Id>
 </LoginToken>
 <ChunkID>0</ChunkID>
 <Query xmlns:d2p1="http://schemas.akzentum.de/2013/07/proxessws">
  <d2p1:DatabaseName>ArchivDB</d2p1:DatabaseName>
  <d2p1:MaxHits>0</d2p1:MaxHits>
  <d2p1:HitsPerPage>100</d2p1:HitsPerPage>
  <!-- [@r, Rechnung] -->
  <d2p1:RootCondition i:type="d2p1:FulltextCondition">
   <d2p1:SearchMode>DocumentFields</d2p1:SearchMode>
   <d2p1:QueryString>Rechnung</d2p1:QueryString>
  </d2p1:RootCondition>
  <!-- -->
  <d2p1:OrderBy i:nil="true" />
 </Query>
 <QueryKey i:nil="true" />
 <Status>Initiate</Status>
</SearchRequest>
```

Listing A.2: XML-Aufbau einer `FulltextCondition`

```
<SearchRequest xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="...">
 <LoginToken xmlns:d2p1="http://schemas.akzentum.de/2013/07/proxessws">
  <d2p1:Id>dabb7a4a-d4f4-4e15-943d-d7f04e26337d</d2p1:Id>
 </LoginToken>
 <ChunkID>0</ChunkID>
 <Query xmlns:d2p1="http://schemas.akzentum.de/2013/07/proxessws">
  <d2p1:DatabaseName>ArchivDB</d2p1:DatabaseName>
  <d2p1:MaxHits>0</d2p1:MaxHits>
  <d2p1:HitsPerPage>100</d2p1:HitsPerPage>
  <!-- UpdateDate >= 2000-01-01 AND UpdateDate < 2001-01-01 -->
  <d2p1:RootCondition i:type="d2p1:FieldRangeCondition">
   <d2p1:FieldName>UpdateDate</d2p1:FieldName>
   <d2p1:Interval>RightOpen</d2p1:Interval>
   <d2p1:Lower>2000-01-01</d2p1:Lower>
   <d2p1:Upper>2001-01-01</d2p1:Upper>
  </d2p1:RootCondition>
  <!-- -->
  <d2p1:OrderBy i:nil="true" />
 </Query>
 <QueryKey i:nil="true" />
 <Status>Initiate</Status>
```

```
</SearchRequest>
```

Listing A.3: XML-Aufbau einer `FieldRangeCondition`

```
<SearchRequest xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="...">
 <LoginToken xmlns:d2p1="http://schemas.akzentum.de/2013/07/proxessws">
  <d2p1:Id>388a5927-db4d-4e7e-9607-71d4c8a9c4e4</d2p1:Id>
 </LoginToken>
 <ChunkID>0</ChunkID>
 <Query xmlns:d2p1="http://schemas.akzentum.de/2013/07/proxessws">
  <d2p1:DatabaseName>ArchivDB</d2p1:DatabaseName>
  <d2p1:MaxHits>0</d2p1:MaxHits>
  <d2p1:HitsPerPage>100</d2p1:HitsPerPage>
  <!-- KD_NAME IN ( 'Rodman', 'Pippen', 'Jordan' ) -->
  <d2p1:RootCondition i:type="d2p1:InCondition">
   <d2p1:FieldName>KD_NAME</d2p1:FieldName>
   <d2p1:Values xmlns:d4p1="http://schemas.microsoft.com/2003/10/Serialization/Arrays">
    <d4p1:string>Rodman</d4p1:string>
    <d4p1:string>Pippen</d4p1:string>
    <d4p1:string>Jordan</d4p1:string>
   </d2p1:Values>
  </d2p1:RootCondition>
  <!-- -->
  <d2p1:OrderBy i:nil="true" />
 </Query>
 <QueryKey i:nil="true" />
 <Status>Initiate</Status>
</SearchRequest>
```

Listing A.4: XML-Aufbau einer `InCondition`

```
<SearchRequest xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="...">
 <LoginToken xmlns:d2p1="http://schemas.akzentum.de/2013/07/proxessws">
  <d2p1:Id>388a5927-db4d-4e7e-9607-71d4c8a9c4e4</d2p1:Id>
 </LoginToken>
 <ChunkID>0</ChunkID>
 <Query xmlns:d2p1="http://schemas.akzentum.de/2013/07/proxessws">
  <d2p1:DatabaseName>ArchivDB</d2p1:DatabaseName>
  <d2p1:MaxHits>0</d2p1:MaxHits>
  <d2p1:HitsPerPage>100</d2p1:HitsPerPage>
  <!-- NOT ( KD_NAME = 'Ratzinger' ) -->
  <d2p1:RootCondition i:type="d2p1:NotCondition">
   <d2p1:Condition i:type="d2p1:FieldCondition">
    <d2p1:FieldName>KD_NAME</d2p1:FieldName>
    <d2p1:Comparator>IsEqualTo</d2p1:Comparator>
    <d2p1:Value>Ratzinger</d2p1:Value>
   </d2p1:Condition>
  </d2p1:RootCondition>
  <!-- -->
  <d2p1:OrderBy i:nil="true" />
 </Query>
 <QueryKey i:nil="true" />
```

```
22  <Status>Initiate</Status>
23 </SearchRequest>
```

Listing A.5: XML-Aufbau einer `NotCondition`

B. Chunked-Transfer

B.1. Allgemeine Beschreibung

Zur Übertragung von potentiell großen Datenmengen verwendet PROXESS Webservice eine stückweise Übertragen, den sog. Chunked Transfer. Die stückweise Übertragung wird dadurch motiviert, dass zum einen von technischer Seite die Nachrichtenpuffer beim Sender und Empfänger eine beschränkte Größe besitzen und dass es zum anderen aus benutzungstechnischer Sicht an einer Rückmeldung über den Fortschritt der Übertragung fehlt. Die Übertragung von Daten in kleinen Teilen erlaubt die Übertragung von sehr großen Dateien (z.B. mehrere MB große PDF-Dateien) ohne an die Grenzen der Nachrichtenpuffer zu stoßen und ermöglicht es, dem Benutzer den Fortschritt der Übertragung anzuzeigen (z.B. mit ProgressBars in "klassischen" GUI-Anwendungen).

Insgesamt sind vier Funktionen der Webservice-Schnittstelle von diesem Mechanismus betroffen:

- Datei-Download (siehe Funktion B.2.0.4: `DownloadFile`),
- Datei-Upload (siehe Funktion B.2.0.5: `UploadFile` bzw. Funktion B.2.0.6: `UploadNewFile`) und
- Das Abholen von Trefferlisten (siehe Funktion B.2.0.2: `Query`).

Im folgenden werden die allgemeingültigen Regeln für den Chunked Transfer aufgeführt und erläutert. Die folgenden Abschnitte beleuchten den Ablauf für die einzelnen Varianten detailiert.

Generell wird eine Übertragung immer vom Client initiiert und je nach dem, ob Daten vom Server abgeholt werden oder Daten zum Server gesendet werden entscheidet der Server bzw. der Client, wann die Übertragung abgeschlossen ist. Die Schnittstellen und die Implementierung der oben aufgeführten Funktionen sind analog aufgebaut und es gibt einige einfache Regeln:

- Ein Transfer wird immer vom Client initiiert. Ein laufender Transfer wird immer durch einen Transferschlüssel identifiziert. Der Transferschlüssel setzt sich für die Übertragung von Dateien und die Übertragung von Suchergebnissen unterschiedlich zusammen (Details in der folgenden Abschnitten).
- Der Status der Übertragung wird (in beide Richtungen) durch bestimmte Enumerationswerte (siehe `TransferStatus` B.2.0.1) signalisiert. So wird ein Transfer entweder initialisiert (`Initiate`), befindet sich im aktiven Zustand (`InProgress`), ist erfolgreich abgeschlossen (`Complete`) oder wurde vom Client abgebrochen (`Cancel`).

- Server-seitige Fehler werden dem Client immer als SOAP Fault mitgeteilt, d.h. der Webservice schickt keine `Cancel`-Nachrichten von sich aus. Lediglich vom Client empfangene `Cancel`-Nachrichten werden vom Webservice mit einer `Cancel`-Nachricht quittiert.
- Der Service organisiert die Reihenfolge der Chunks und teilt dem Client immer mit, welcher Chunk als nächstes abgerufen bzw. gesendet werden muss. In der aktuellen Implementierung werden die Chunks immer sequentiell (d.h. in der Reigenfolge `0`, `1`, `2`, `...`) angefordert.
- Sobald einmal eine Nachricht mit dem Status `Complete` oder `Cancel` gesendet oder empfangen wurde, ist das Transferobjekt abgelaufen und damit ungültig.
- Der Status `Initiate` darf nur genau einmal und zwar beim ersten Aufruf verwendet werden. Wird dieser Status öfter gesetzt erlischt die Gültigkeit des Transferobjektes, d.h. `Initiate` hat dann die selbe Bedeutung wie `Cancel`.
- Wird durch eine der vier Transfer-Methoden ein SOAP Fault (siehe 3.1.4) ausgelöst, erlischt die Gültigkeit des Transferobjektes. Wenn der Client den Fehler übermittelt bekommt, hat der Service das Transferobjekt bereits gelöscht.

In den folgenden Abschnitten wird der Ablauf der einzlnen Varianten detaillierter beleuchtet. Im Anhang dieses Dokuments (siehe Anhang B) finden sich Beispiel für die Implementierung des Algorithmus.

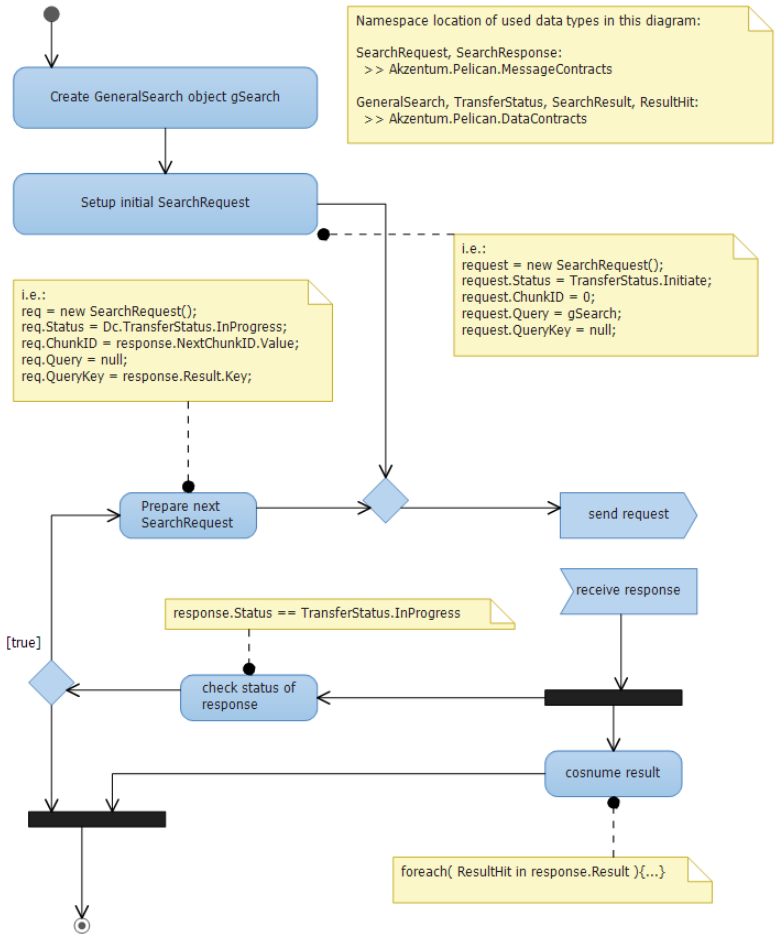

Abbildung B.1.: Ablauf für das Abrufen von Suchergebnissen

B.1.0.1. Ablauf: Abrufen von Suchergebnissen

Dieser Abschnitt beschreibt einen Schritt-für-Schritt-Vorgang für das Abrufen von Suchergebnissen. Dieser Vorgang wird in Abbildung B.1 begleitend als UML-Aktivitätsdiagramm dargestellt.

1. Zuerst muss die Suchanfrage in einem `GeneralSearch`-Objekt formuliert werden. Die Struktur der Abfragebedingung(en) wird in Abschnitt 3.3.4 beschrieben.
2. Der initiale Request `req` wird vorbereitet:
   a) Der `TransferStatus` wird auf `Initiate` gesetzt.
   b) Die `ChunkID` wird auf `0` gesetzt.
   c) `Query` wird auf die in Schritt 1 erzeugte `GeneralSearch` gesetzt.

d) `QueryKey` wird auf `null` (kein Wert) gesetzt.

3. `req` wird an den Webservice gesendet und die Response `resp` empfangen (synchron). Die in `resp` enthaltenen Werte werden ausgewertet.

   a) Das Feld `Result` enthält eine Liste von Treffern, die z.B. dem Benutzer angezeigt werden können oder dazu verwendet werden können, das zugehörige Dokument abzurufen.

   b) Das Feld `Status` enthält den serverseitigen Transferstatus. Anhand dieses Feldes wird entschieden, wie weiter zu verfahren ist.

      i. Wenn `Status` den Wert `InProgress` hat, wird **mit Schritt 4 fortgefahren.**

      ii. Sonst: **Der Transfer ist erfolgreich beendet worden** (Status hat den Wert `Complete`; die Werte `Initiate` und `Cancel` werden vom Service nicht verwendet).

4. Nächsten Request `req` vorbereiten.

   a) `req.Status = InProgress`

   b) Das Feld `resp.NextChunkID` enthält die ID des nächsten abzurufenden Chunks. Es muss immer in den Folge-Request kopiert werden, also `req.ChunkID = resp.NextChunkID`.

   c) Das Feld `resp.QueryKey` enthält den vom Webservice zugewiesenen Transferschlüssel. Er muss immer in den Folge-Request kopiert werden, also `req.QueryKey = resp.QueryKey`.

   d) Das Feld `req.Query` wird auf `null` (kein Wert) gesetzt.

   e) **Mit Schritt 3 fortfahren.**

B.1.0.2. Ablauf: Download von Dateien

Dieser Abschnitt beschreibt einen Schritt-für-Schritt-Vorgang für den Download von Dateien. Dieser Vorgang wird in Abbildung B.2 begleitend als UML-Aktivitätsdiagramm dargestellt.

1. Gegen ist die Datei (`FileDescriptor`) eines Dokuments (`Dokument`). Damit ist der Datenbankname `dbName`, die Dokument-ID `docID` und die Datei-ID `fileID` bekannt. Der Transferschlüssel für den Download ist die Kombination aus `dbName`, `docID` und `fileID`.

2. Initialen Request `req` vorbereiten:

   a) Dem Feld `req.Database` den Wert von `dbName` zuweisen.

   b) Dem Feld `req.DocumentID` den Wert von `docID` zuweisen.

   c) Dem Feld `req.FileID` den Wert von `fileID` zuweisen.

   d) Den Status `req.Status` auf `Initiate` setzen.

   e) Dam Feld `req.ChunkID` den Wert `0` zuweisen.

3. Den Request `req` synchron an den Webservice senden und die Response `resp` erhalten. In Abhängigkeit des Serverstatus `req.Status` fortfahren:

   a) Im Fall von `InProgress` oder `Complete` enthält die Response Daten aus der Datei. **Mit Schritt 4 fortfahren.**

   b) Im Fall von `Cancel` quittiert der Service eine `Cancel`-Nachricht des Clients. **Der Transfer ist fehlerhaft beendet worden.**

4. Die Nutzdaten (payload, `req.ChunkData`) verarbeiten, z.B. in eine lokale Datei schreiben. Die (lokale) Verarbeitung der Daten kann evtl. einen IO-Fehler verursachen.

   a) **Bei fehlerhafter Verarbeitung** wird eine `Cancel`-Nachricht `req` an den Webservice gesendet werden. Damit wird dem Webservice signalisiert, den laufenden Transfer zu beenden.

      i. Transferschlüssen kopieren (`req.Database`, `req.DocumentID`, `req.FileID`).

      ii. Transferstatus `req.Status` auf den Wert `Cancel` setzen.

      iii. `req.ChunkID` auf `null` (kein Wert) setzen.

      iv. **Mit Schritt 3 fortfahren.**

   b) **Bei fehlerfreier Verarbeitung** wird in Abhängigkeit des Transferstatus `resp.Status` fortgefahren:

      i. Im Fall `InProgress` sind noch mehr Daten abzuholen: Der nächste Request wird vorbereitet. **Mit Schritt 5 fortfahren.**

ii. Im Fall `Complete` wurde die Datei vollständig und fehlerfrei heruntergeladen. **Der laufende Transfer ist erfolgreich beendet worden.**

5. Nächsten Request `req` vorbereiten:

   a) Transferschlüssen kopieren (`req.Database`, `req.DocumentID`, `req.FileID`).

   b) Transferstatus `req.Status` auf `InProgress` setzen.

   c) Dem Feld `req.ChunkID` den Wert von `resp.NextChunkID` zuweisen.

   d) **Mit Schritt 3 fortfahren.**

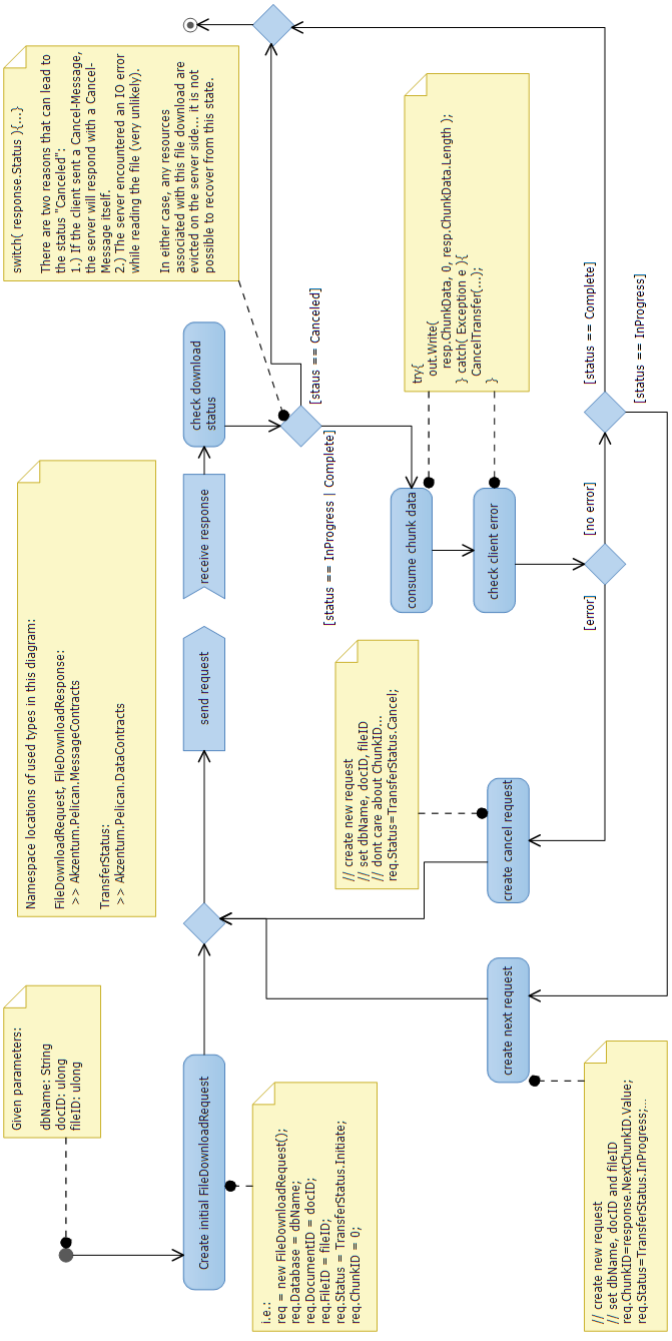

Abbildung B.2.: Ablauf für Dateidownload

B.1.0.3. Ablauf: Upload von Dateien

Dieser Abschnitt beschreibt einen Schritt-für-Schritt-Vorgang für den Upload von Dateien. Dieser Vorgang wird in Abbildung B.3 begleitend als UML-Aktivitätsdiagramm dargestellt.

Der erste Request für einen Datei-Upload unterscheidet sich je nach dem, ob es sich um eine **neue Datei** (Schritt 1) oder um eine **neue Version einer bestehenden Datei** (Schritt 2) handelt. Der Transferschlüssel eines Uploads besteht – genau so wie der Transferschlüssel für Downloads – aus der Kombination von Datenbankname `dbName`, Dokument-ID `docID` und der Datei-ID `fileID`.

Beim Upload einer Datei muss der Client wissen, wie groß ein Chunk maximal sein darf. Standardmäßig ist die maximale Größe auf 1024 * 128 byte eingestellt. Ob evtl. Abweichungen vorliegen, muss mit dem Systemadministrator abgesprochen werden. Im folgenden wird die vom Client verwendete Chunkgröße mit `n` bezeichnet.

1. Upload einer neuen Datei. Gegeben ist der Datenbankname und die Dokument-ID. Eine Datei-ID ist noch nicht vorhanden.
   a) Initialen Request `req` vorbereiten:
      i. Transferschlüssel initialisieren: `req.DatabaseName = dbName; req.DocumentID = docID; req.FileID = 0;`
      ii. Transferstatus `req.Status` auf `Initiate` setzen.
      iii. Chunk-ID `req.ChunkID` auf den Wert `0` setzen.
      iv. Die ersten `n` bytes der hochzuladenden Datei im Feld `req.ChunkData` speichern.
      v. Dateityp-ID `req.FileTypeID` setzen (siehe Funktion 3.2.2.4: `GetFileTypes`). I.d.R. gibt es PROXESS Dateitypen, die "realen" Dateitypen entsprechen, z.B. `pdf`, `doc` oder `txt`.
      vi. Dateibeschreibung (Titel) `req.FileDescription` setzen. Das ist ein String, der den Inhalt der Datei beschreibt, z.B.: "Scan Rechnung Maier 23.05.2012"
   b) Request an Webservice senden und Response `resp` erhalten.
      - Die Response auf einen `NewFileUploadRequest` enthält immer die ID der neu angelegten Datei.
      - Die neue Datei-ID wird in den folgenden Requests benötigt, um den Transferschlüssel zu bilden.
      - Die ID der neuen Datei ist in dem Feld `resp.NewFileID` gespeichert.
   c) **Mit Schritt 4 fortfahren.**
2. Upload einer neuen Version einer bereits bestehenden Datei. Gegeben ist der Datenbankname, die Dokument-ID und die Datei-ID.
   a) Initialen Request `req` vorbereiten:
      i. Transferschlüssel initialisieren: `req.DatabaseName = dbName; req.DocumentID = docID; req.FileID = fileID;`

ii. Transferstatus `req.Status` auf `Initiate` setzen.

iii. Chunk-ID `req.ChunkID` auf den Wert `0` setzen.

iv. Die ersten `n` bytes der hochzuladenden Datei im Feld `req.ChunkData` speichern.

3. Request `req` an den Webservice senden und Response `resp` erhalten. Anhand des Transferstatus `resp.Status` wird unterschiedlich fortgefahren:

   a) Standardfall: `InProgress`; der Server hat den letzten gesendeten Chunk akzeptiert. Der nächste Request wird vom Client vorbereitet. **Mit Schritt 4 fortfahren.**

   b) Im Fall `Complete` erhält die Response die neue Datei-ID im Feld `resp.NewFileID`. Beim Upload einer neuen Versionen einer bestehenden Dateien ist das die erste Stelle, an der die neue Datei-ID bekannt ist. **Der Transfer ist erfolgreich beendet worden.**

   c) Im Fall `Cancel` wurde eine Client-seitige `Cancel`-Nachricht quittiert. **Der Transfer wurde fehlerhaft beendet.**

4. Nächsten Request `req` vorbereiten:

   a) Transferschlüssel initialisieren: `req.DatabaseName = dbName; req.DocumentID = docID; req.FileID = fileID;` Die Datei-ID ist entweder von Beginn an bekannt (neue Version einer Datei) oder wurde in der ersten Response des Webservice ermittelt (neue Datei).

   b) Es gibt noch weitere Daten, die noch nicht hochgeladen worden sind.

      i. Transferstatus `req.Status` auf `InProgress` setzen.

      ii. Chunk-ID `req.ChunkID` auf den Wert `resp.NextChunkID` setzen.

      iii. Die nächsten `n` bytes der hochzuladenden Datei im Feld `req.ChunkData` speichern.
      Achtung: Wenn nur noch weniger als `n` bytes hochgeladen werden müssen, dann ist es wichtig, dass das Array `req.ChunkData` genau die passende Größe hat, d.h. wenn z.B. noch 144 Bytes hochgeldaen werden müssen, muss das Array `req.ChunkData` auch genau 144 bytes lang sein. Werden zu viele Daten in die PROXESS-Datei geschrieben, kann das leicht zu korrupten Daten führen, die sich von der passenden Anwendung nicht mehr öffnen lassen.

   c) Es gibt keine weiteren Daten mehr. Der Transfer ist aus Sicht des Clients vollständig.

      i. Transferstatus `req.Status` auf `Complete` setzen.

      ii. Chunk-ID `req.ChunkID` auf `null` (kein Wert) setzen.

      iii. Nutzdaten `req.ChunkData` auf `null` (kein Wert) setzen.

   d) **Mit Schritt 3 fortfahren.**

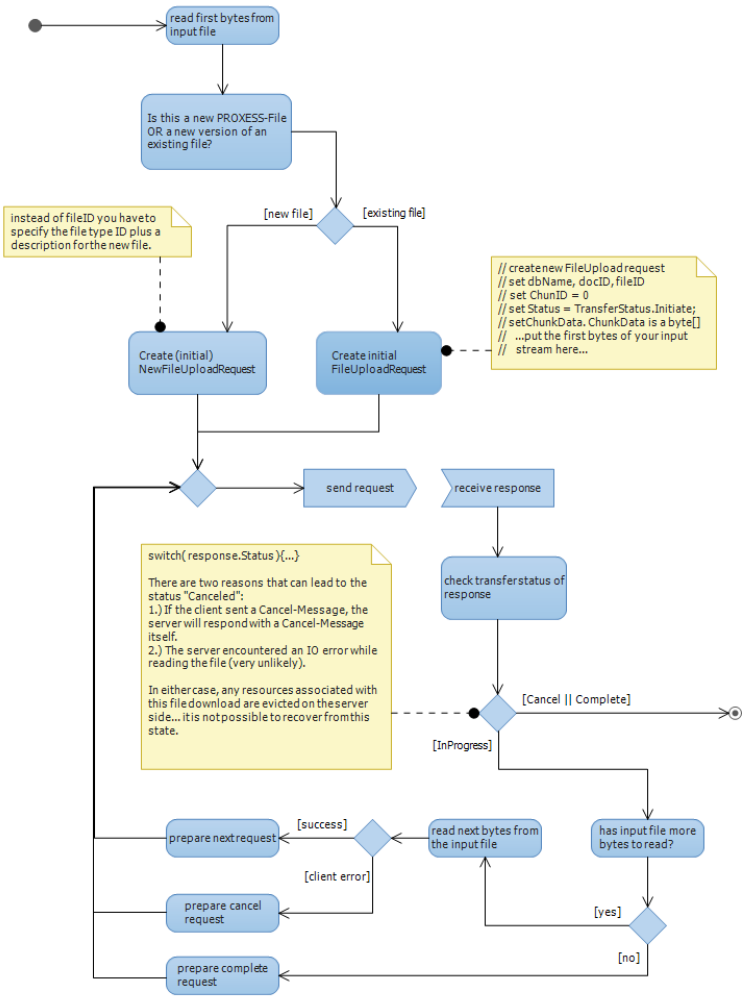

Abbildung B.3.: Ablauf für Dateiupload

B.2. Funktionen und Datenkontrakte

B.2.0.1. TransferStatus (Datenkontrakt)

`TransferStatus` ist eine Enumeration, mit der der Zustand einer Datenübertragung zwischen dem Client und dem Service kommuniziert wird.
Listing B.1 zeigt die WSDL-Definition des Datentyps `TransferStatus`.

```
<xs:simpleType name="TransferStatus">
 <xs:restriction base="xs:string">
  <!-- Wird ausschließlich vom Client zum initiieren einer Datenübertragung verwendet. -->
  <xs:enumeration value="Initiate"/>
  <!-- Wird von Webservice und Client verwendet, solange eine Übertragung noch nicht
      abgeschlossen ist. -->
  <xs:enumeration value="InProgress"/>
  <!-- Wird vom Webservice (bei Ergebnislisten und Downloads) und vom Client (bei Uploads)
      dazu verwendet, um eine erfolgreich abgeschlossene Übertragung zu kennzeichnen. -->
  <xs:enumeration value="Complete"/>
  <!-- Wird vom Client verwendet, um eine Übertragung abzubrechen.
      Der Server schickt Cancel-Nachrichten, um eine Client-seitige Cancel-Nachricht zu
      quittieren. Server-seitige Fehler werden über SOAP-Faults kommuniziert. -->
  <xs:enumeration value="Cancel"/>
 </xs:restriction>
</xs:simpleType>
```

Listing B.1: Datentyp TransferStatus

Der zu Grunde liegende Übertragungmechnismus (Chunked Transfer) wurde in Abschnitt B.1 eingeführt.

B.2.0.2. Query (Funktion)

Signatur:

```
SearchResponse Query(
       SearchRequest request );
```

Die Funktion `Query` führt eine Suche in PROXESS durch und holt die Ergebnisse ab. Sie wird auf zwei Weisen verwendet:

1. Eine bestimmte Abfrage[1] an eine Datenbank wird ausgeführt und die Übertragung des Ergebnis wird begonnen.
2. Anhand eines (vom Webservice generierten) Transferschlüssels wird die Übertragung eines Ergebnisses fortgesetzt.

[1] Erweiterte Suche, `General Search` (siehe 3.3.4)

Die Übertragung von Suchergebnissen ist analog zu der Beschreibung in Abschnitt B.1: Chunked Transfer zu implementieren.

Nachrichtentyp `SearchRequest`. Listing B.2 zeigt die WSDL-Definition des Datentyps `SearchRequest`.

```
<xs:element name="SearchRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q87:LoginToken"/>
   <!-- Client-seitiger Status des Transfers. -->
   <xs:element minOccurs="0" name="Status" type="q88:TransferStatus">
   <!-- ID des aktuell angeforderten Chunks. -->
   <xs:element minOccurs="0" name="ChunkID" type="xs:long"/>
   <!-- Spezifiziert bei neuen Suchen die Suchbedingung. Ansonsten null. -->
   <xs:element minOccurs="0" name="Query" nillable="true" type="q89:GeneralSearch">
   <!-- Spezifiziert bei laufenden Transfers den Transferschlüssel. Ansonsten null. -->
   <xs:element minOccurs="0" name="QueryKey" nillable="true" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing B.2: Nachrichtentyp SearchRequest

Die folgende Tabelle führt Verweise zu allen in Listing B.2 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2
`TransferStatus`	B.2.0.1
`GeneralSearch`	3.3.4.1

Nachrichtentyp `SearchResponse`. Listing B.3 zeigt die WSDL-Definition des Datentyps `SearchResponse`.

```
<xs:element name="SearchResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Server-seitiger Status des Transfers. -->
   <xs:element minOccurs="0" name="Status" type="q120:TransferStatus"/>
   <!-- ID des nächsten Chunks, das vom Client abgeholt werden muss. -->
   <xs:element minOccurs="0" name="NextChunkID" nillable="true" type="xs:long"/>
   <!-- Liste der Suchergebnisse des zuletzt angeforderten Chunks. -->
   <xs:element minOccurs="0" name="Result" nillable="true" type="q121:SearchResult"/>
   <!-- Wenn der Transfer abgeschlossen ist:
        * true -- Alle Dokumente sind übermittelt worden.
        * false -- Es gibt mehr Treffer-Dokumente als angefordert wurden.
      Ansonsten hat das Feld den Wert null. -->
   <xs:element minOccurs="0" name="ContainsAll" nillable="true" type="xs:boolean"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing B.3: Nachrichtentyp SearchResponse

Die folgende Tabelle führt Verweise zu allen in Listing B.3 referenzierten Datentypen auf.

Feldname	Verweis
`TransferStatus`	B.2.0.1
`SearchResult`	B.2.0.3

B.2.0.3. SearchResult (Datenkontrakt)

Ein `SearchResult` repräsentiert einen Chunk bei der Übertragung eines Suchergebnisses. Es enthält den Transferschlüssel für Folgerequests und eine Liste von Trefferdokumenten.

Listing B.4 zeigt die WSDL-Definition des Datentyps `SearchResult`.

```
<xs:complexType name="SearchResult">
 <xs:complexContent mixed="false">
  <!-- Basistyp speichert alle Metadaten-Objekte. -->
  <xs:extension base="q122:MetadataContext">
   <xs:sequence>
    <!-- Vom Webservice generierter Transferschlüssel.
         Wird für die Übertragung weiterer Chunks benötigt. -->
    <xs:element minOccurs="0" name="Key" nillable="true" type="xs:string"/>
    <!-- Liste von Trefferdokumenten des zuletzt angeforderten Chunks. -->
    <xs:element minOccurs="0" name="ResultHits" nillable="true" type="q122:ResultHitCollection"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

Listing B.4: Datentyp SearchResult

Die folgende Tabelle führt Verweise zu allen in Listing B.4 referenzierten Datentypen auf.

Feldname	Verweis
`MetadataContext`	3.3.4.1
`ResultHitCollection`	3.3
`ResultHit`	3.3.4.1

B.2.0.4. DownloadFile (Funktion)

Signatur:

```
FileDownloadResponse DownloadFile(
       FileDownloadRequest request );
```

Die Funktion `DownloadFile` lädt eine Datei von PROXESS herunter.

- Die Übertragung der Datei ist analog zu der Beschreibung in Abschnitt B.2: *Chunked Transfer (Download)* zu implementieren.
- Der Transferschlüssel setzt sich aus den drei Feldern `DatabaseName`, `DocumentID` und `FileID` des Requests zusammen.
- Die Standardgröße eines Chunks ist `131072` (oder: `1024 * 128`) bytes (der Client muss keine Puffer o.ä. bereitstellen/allokieren).

Nachrichtentyp `FileDownloadRequest`. Listing B.5 zeigt die WSDL-Definition des Datentyps `FileDownloadRequest`.

```
<xs:element name="FileDownloadRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q135:LoginToken"/>
   <!-- Name der Datenbank, in der das Dokument gespeichert ist.
       Teil des Transferschlüssels. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
   <!-- ID des Dokuments, zu dem die Datei gehört.
       Teil des Transferschlüssels. -->
   <xs:element minOccurs="0" name="DocumentID" type="xs:unsignedLong"/>
   <!-- ID der Datei, die heruntergeladen wird.
       Teil des Transferschlüssels. -->
   <xs:element minOccurs="0" name="FileID" type="xs:unsignedLong"/>
   <!-- Client-seitiger Status des Transfers. -->
   <xs:element minOccurs="0" name="Status" type="q136:TransferStatus"/>
   <!-- ID des aktuell angeforderten Chunks. -->
   <xs:element minOccurs="0" name="ChunkID" type="xs:long"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing B.5: Nachrichtentyp FileDownloadRequest

Die folgende Tabelle führt Verweise zu allen in Listing B.5 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2
`TransferStatus`	B.2.0.1

Nachrichtentyp `FileDownloadResponse`. Listing B.6 zeigt die WSDL-Definition des Datentyps `FileDownloadResponse`.

```
<xs:element name="FileDownloadResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Server-seitiger Status des Transfers. -->
   <xs:element minOccurs="0" name="Status" type="q137:TransferStatus"/>
   <!-- ID des nächsten Chunks, das vom Client abgeholt werden muss. -->
   <xs:element minOccurs="0" name="NextChunkID" nillable="true" type="xs:long"/>
   <!-- byte[], in dem die Daten des zuletzt angeforderten Chunks gespeichert sind. -->
   <xs:element minOccurs="0" name="ChunkData" nillable="true" type="xs:base64Binary"/>
   <!-- Wenn nicht null: Gesamtlänge der Datei in Bytes.
       Ansonsten: Dateilänge unbekannt oder nicht übermittelt. -->
   <xs:element minOccurs="0" name="FileLength" nillable="true" type="xs:long"/>
   <!-- Wenn nicht null: Gibt dem Client Informationen darüber, ob die Datei vor
       dem Download mit der Komponente PROXESS Web Rendr in eine PDF-Datei gerendert
            worden ist, bzw. unter welcher Dateierweiterung die Datei zu speichern ist. -->
   <xs:element minOccurs="0" name="RenderInfo" nillable="true" type="q139:RenderInfo"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing B.6: Nachrichtentyp FileDownloadResponse

Die folgende Tabelle führt Verweise zu allen in Listing B.6 referenzierten Datentypen auf.

Feldname	Verweis
`TransferStatus`	B.2.0.1
`RenderInfo`	3.3.1.4

Anmerkung: `RenderInfo`. Das Datenfeld `RenderInfo` ist seit PROXESS Webservice Version 2.1.0 in der `FileDownloadResponse` enthalten.

Dieses Feld enthält Informationen darüber, ob eine Datei mit der Komponente PROXESS Web Rendr in eine PDF-Datei gerendert worden ist, und gibt dem Client die Dateierweiterung an, die beim Speichern verwendet werden sollte. Im Fall von gerenderten Dateien ist die Dateierweiterung stets `pdf`, für Dateien, die nicht gerendert wurden, wird die Dateierweiterung des zugehörigen PROXESS-Dateityps zurückgegeben, z.B. `txt` oder `docx`.

B.2.0.5. UploadFile (Funktion)

Signatur:

```
FileUploadResponse UploadFile(
       FileUploadRequest request );
```

Die Funktion `UploadFile` wird zum Hochladen von Dateien an PROXESS verwendet. Die Funktion wird in folgenden Fällen verwendet:

- Der Upload einer **neue Version einer existierenden PROXESS-Datei** soll begonnen werden.
- Der nächste Chunk eines Upload[2] soll übertragen werden.
- Ein Upload soll abgeschlossen oder abgebrochen werden.

Im Zusammenhang mit der stückweisen Übertragung sind folgende Dinge zu beachten:

- Die Übertragung der Datei ist analog zu der Beschreibung in Abschnitt B.3: *Chunked Transfer (Upload)* zu implementieren.
- Der Transferschlüssel setzt sich auf den drei Feldern `DatabaseName`, `DocumentID` und `FileID` des Requests zusammen.
- Der Client darf mit den Requests die maximale Nachrichtengröße nicht überschreiten. Der Webservice ist standardmäßig so konfiguriert, dass Chunks mit einer Größe von 131072 bytes (d.h. 128 KB) übertragen werden können.

Nachrichtentyp `FileUploadRequest`. Listing B.7 zeigt die WSDL-Definition des Datentyps `FileUploadRequest`.

```
<xs:element name="FileUploadRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q138:LoginToken"/>
   <!-- Name der Datenbank, in der das Dokument gespeichert ist. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
   <!-- ID des Dokuments, zu dem die Datei gehört. -->
   <xs:element minOccurs="0" name="DocumentID" type="xs:unsignedLong"/>
   <!-- ID der Datei, die aktualisiert werden soll. -->
   <xs:element minOccurs="0" name="FileID" type="xs:unsignedLong"/>
   <!-- Client-seitiger Status des Transfers. -->
   <xs:element minOccurs="0" name="Status" type="q139:TransferStatus"/>
   <!-- ID des aktuell gesendeten Chunks. -->
   <xs:element minOccurs="0" name="ChunkID" type="xs:long"/>
   <!-- Daten des aktuellen Chunks, d.h. die nächsten bytes aus der Quelldatei. -->
   <xs:element minOccurs="0" name="ChunkData" nillable="true" type="xs:base64Binary"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing B.7: Nachrichtentyp FileUploadRequest

[2] Egal, ob es sich um eine **neue PROXESS-Datei** oder um eine **neue Version einer PROXESS-Datei** handelt.

Die folgende Tabelle führt Verweise zu allen in Listing B.7 referenzierten Datentypen auf.

Feldname	Verweis
LoginToken	3.3.1.2
TransferStatus	B.2.0.1

Nachrichtentyp `FileUploadResponse`. Listing B.8 zeigt die WSDL-Definition des Datentyps `FileUploadResponse`.

```
<xs:element name="FileUploadResponse">
 <xs:complexType>
  <xs:sequence>
   <!-- Server-seitiger Status des Transfers. -->
   <xs:element minOccurs="0" name="Status" type="q140:TransferStatus"/>
   <!-- ID des nächsten Chunks, das vom Client gesendet werden muss. -->
   <xs:element minOccurs="0" name="NextChunkID" nillable="true" type="xs:long"/>
   <!-- Wenn nicht null: Neue Datei-ID bzw. ID der neuen Datei.
       Dieses Feld ist in den folgenden Fällen nicht null:
          1. Antwort auf einen NewFileUploadRequest (d.h. neue Datei wird angelegt)
          2. Antwort auf einen FileUploadRequest mit dem TransferStatus Complete, d.h.
              die Übertragung wurde vom Client erfolgreich beendet (und der Webservice
              konnte den Vorgang in PROXESS korrekt durchführen). -->
   <xs:element minOccurs="0" name="NewFileID" nillable="true" type="xs:unsignedLong"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing B.8: Nachrichtentyp FileUploadResponse

Die folgende Tabelle führt Verweise zu allen in Listing B.8 referenzierten Datentypen auf.

Feldname	Verweis
TransferStatus	B.2.0.1

B.2.0.6. UploadNewFile (Funktion)

Signatur:

```
FileUploadResponse UploadNewFile(
       NewFileUploadRequest request );
```

Die Funktion `UploadNewFile` wird zum Hochladen von Dateien an PROXESS verwendet, wenn eine **neue Datei** erstellt werden soll. Diese Funktion wird nur für den ersten Webservice-Aufruf der Übertragung verwendet. Für alle Folgeaufrufe wird Funktion B.2.0.5: `UploadFile` verwendet.

Es gelten die gleichen Anmerkungen zur Übertragung wie für Funktion B.2.0.5: `UploadFile`; zusätzlich ist zu beachten, dass das Feld `FileID` des Requests auf den Wert `0` gesetzt werden muss.

Nachrichtentyp `NewFileUploadRequest`. Listing B.9 zeigt die WSDL-Definition des Datentyps `NewFileUploadRequest`.

```
<xs:element name="NewFileUploadRequest">
 <xs:complexType>
  <xs:sequence>
   <!-- Identifiziert die PROXESS Session des Webservice-Clients. -->
   <xs:element minOccurs="0" name="LoginToken" nillable="true" type="q141:LoginToken"/>
   <!-- Name der Datenbank, in der das Dokument gespeichert ist. -->
   <xs:element minOccurs="0" name="DatabaseName" nillable="true" type="xs:string"/>
   <!-- ID des Dokuments, zu dem die Datei gehört. -->
   <xs:element minOccurs="0" name="DocumentID" type="xs:unsignedLong"/>
   <!-- Muss den Wert 0 haben. -->
   <xs:element minOccurs="0" name="FileID" type="xs:unsignedLong"/>
   <!-- Client-seitiger Status des Transfers.
       (bei korrekter Anwendung immer TansferStatus.Initiate) -->
   <xs:element minOccurs="0" name="Status" type="q142:TransferStatus"/>
   <!-- ID des aktuell gesendeten Chunks.
       (bei korrekter Anwendung immer 0) -->
   <xs:element minOccurs="0" name="ChunkID" type="xs:long"/>
   <!-- Daten des aktuellen Chunks, d.h. die nächsten bytes aus der Quelldatei. -->
   <xs:element minOccurs="0" name="ChunkData" nillable="true" type="xs:base64Binary"/>
   <!-- Dateityp-ID (siehe FileType, GetFileTypes) der neuen Datei. -->
   <xs:element minOccurs="0" name="FileTypeID" type="xs:unsignedLong"/>
   <!-- Beschreibung/Titel der neuen Datei. -->
   <xs:element minOccurs="0" name="FileDescription" nillable="true" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Listing B.9: Nachrichtentyp NewFileUploadRequest

Die folgende Tabelle führt Verweise zu allen in Listing B.9 referenzierten Datentypen auf.

Feldname	Verweis
`LoginToken`	3.3.1.2
`TransferStatus`	B.2.0.1
`FileType`	3.3.2.4

Für den Datentyp `UploadFileResponse` siehe Funktion B.2.0.5: `UploadFile`.

B.3. Beispiele

Im Folgenden werden drei Beispiel-Implementierungen für den Chunked-Transfer in der Programmiersprache Java aufgeführt. Diese Beispiele demonstrieren lediglich die prinzipielle Funktionsweise und wurden nicht mit allen möglichen Randfällen getestet.

Listing B.10 zeigt eine Methode, die den Chunked-Transfer für Suchergebnisse implementiert.

Listing B.11 zeigt eine Methode, die den Chunked-Transfer für Dateidownloads implementiert.

Listing B.12 zeigt eine Methode, die den Chunked-Transfer für Dateiuploads implementiert. Die Methode behandelt beide Fälle von Uploads: Den Upload einer neuen Datei und den Upload einer neuen Version einer existierenden Datei. Listing B.13 ist eine Hilfsfunktion zur Ausführung eines Upload-Requests.

```
/**
 * Executes {@code search} on {@code session} and logs the result hits at level INFO.
 * @param wsclient Endpoint proxy.
 * @param factory Factory for PROXESS Webservice requests.
 * @param session PROXESS sesion.
 * @param search Search that will be executed.
 * @throws ProxessWebserviceFault If anything goes wrong...
 */
public static void printSearchResult(ProxessDocumentServiceContract wsclient, RequestFactory factory, LoginToken
     session, GeneralSearch search) throws ProxessWebserviceFault {

   // record total hits. for output only..
   int totalHits = 0;
   // display the search result with a nicely formatted text
   StringBuilder resultBuilder = new StringBuilder();


   SearchRequest request;
   SearchResponse response;
   Long chunkID = 0L;

   // prepare initial request. queryKey (type: String) must be null and query (type: GeneralSearch) must not be
        null
   request = factory.createSearchRequest(session, TransferStatus.INITIATE, chunkID, null, search);

   while (request != null) {
      response = wsclient.query(request);

      SearchResult result = response.getResult();
      String queryKey = result.getKey();
      List<ResultHit> hits = result.getResultHits().getResultHits();

      totalHits += hits.size();
      resultBuilder.append(String.format("Printing result hits of chunk %d for result %s.\n", chunkID, queryKey))
           ;
      for (ResultHit rh : hits) {
         resultBuilder.append(String.format(" [Doc %d: '%s'; Type %d: %s]\n", rh.getDocumentId(), rh.
              getDocumentDescription(), rh.getDocTypeID(), rh.getDocTypeName()));
      }

      switch (response.getStatus()) {
         case IN_PROGRESS:
            // prepare next request: get next chunk ID and queryKey from response.
            // if queryKey is set, parameter query will be ignored...
            chunkID = response.getNextChunkID();
            request = factory.createSearchRequest(session, TransferStatus.IN_PROGRESS, chunkID, queryKey, null);
            break;
```

```
            case COMPLETE:
               resultBuilder.append(String.format("Result transfer completed. Found %d documents in %d chunks. ",
                    totalHits, chunkID+1));
               // if(debug) assert response.getContainsAll().getValue() != null
               boolean containsAll = response.isContainsAll();
               if (containsAll) {
                  resultBuilder.append("The result contains all documents.\n");
               } else {
                  resultBuilder.append("The result was truncated at maxHits.\n");
               }
               // exit transfer loop
               request = null;
               break;
            case CANCEL:
            case INITIATE:
            default:
               throw new RuntimeException("Invalid state.");
         }
      } // while(request != null)

      Logger.getAnonymousLogger().info(resultBuilder.toString());
   }
```

Listing B.10: Chunked-Transfer SearchResult

```
/**
 * Downloads a file from PROXESS Webservice and writes the downloaded data to output stream
 * {@code toStream}.
 *
 * @param wsclient Web service endpoint proxy.
 * @param msgFactory Factory for PROXES Webservice requests.
 * @param session Token that identifies the PROXESS session.
 * @param database Name of the parent database.
 * @param docID ID of the parent document of the file.
 * @param fileID ID of the file to be downloaded.
 * @param toStream Sink for downloaded binary data.
 * @throws Exception Any web service fault or client-side IOException
 */
public static void downloadFile(ProxessDocumentServiceContract wsclient, RequestFactory msgFactory, LoginToken session
     , String database, BigInteger docID, BigInteger fileID, OutputStream toStream) throws Exception {

   FileDownloadRequest request;
   FileDownloadResponse response;

   try {
      // prepare initial request
      request = msgFactory.createFileDownloadRequest(
        session, database, docID, fileID, TransferStatus.INITIATE, 0L);

      // download loop
      while (request != null) {

         // execute web service call
```

```
            response = wsclient.downloadFile(request);

            // write downloaded binary data to output stream
            byte[] downloadChunk = response.getChunkData();
            try {
                toStream.write(downloadChunk, 0, downloadChunk.length);
            } catch (IOException ex) {
                wsclient.downloadFile(msgFactory.createFileDownloadRequest(
                  session, database, docID, fileID, TransferStatus.CANCEL, null));
                throw ex;
            }

            // check transer status
            switch (response.getStatus()) {
                case IN_PROGRESS: // download not complete. prepare next request
                    Long nextChunkID = response.getNextChunkID();
                    request = msgFactory.createFileDownloadRequest(
                      session, database, docID, fileID, TransferStatus.IN_PROGRESS,
                      nextChunkID);
                    break;
                case COMPLETE: // download complete. cancel download loop
                    request = null;
                    break;
                case CANCEL: // service does not return these states when downloading...
                case INITIATE:
                default:
                    throw new Exception("Invalid state.");
            }
        }
    } catch (Exception e) {
        Logger.getLogger(DownloadDemo.class.getName()).log(Level.SEVERE, null, e);
        throw e;
    }
}
```

Listing B.11: Chunked-Transfer Download

```
/**
 * Uploads a file to PROXESS Webservice reading the data from stream {@code toStream}.
 *
 * @param wsclient Web service endpoint proxy.
 * @param msgFactory Factory for PROXES Webservice requests.
 * @param session Token that identifies the PROXESS session.
 * @param database Name of the parent database.
 * @param docID ID of the parent document of the file.
 * @param fileID if not null: ID of the file to be updated. otherwise: new file
 * @param fileTypeID if new file: ID of the file type. otherwise: must be null
 * @param fileDesc if new file: Description of the new file. otherwise: ignored
 * @param toStream Sink for downloaded binary data.
 * @throws Exception Any web service fault or client-side IOException
 */
public static BigInteger Upload(ProxessDocumentServiceContract wsClient, RequestFactory msgFactory, LoginToken session
     , String database, BigInteger docID, BigInteger fileID, BigInteger fileTypeID, String fileDesc, InputStream
```

```
 fromStream) throws Exception {
// init local variables
Object request; // Most common ancestor of FileUploadRequest and NewFileUploadRequest
FileUploadResponse resp = null;
byte[] buffer = new byte[1024 * 128];
byte[] chunkData;
boolean isNewFile;
boolean fileIsEmpty = true;
int read = 0;

// prepare initial request (the actual byte-data will be set later)
if (fileID == null && fileTypeID != null) {
   // case 1: fileTypeID!=null => create a new file of type with ID fileTypeID
   isNewFile = true;
   request = msgFactory.createNewFileUploadRequest(
    session, database, docID, fileTypeID, fileDesc, TransferStatus.INITIATE, 0L, null);
} else if (fileID != null && fileTypeID == null) {
   // case 2: fileID != null ==>
          // update existing file / make new version of file with ID fileID
   isNewFile = false;
   request = msgFactory.createFileUploadRequest(
            session, database, docID, fileID, TransferStatus.INITIATE, 0L, null);
} else {
   // else: invalid combination
   throw new Exception("fileID XOR fileTypeID must be null.");
}

// upload loop
while (request != null) {

   // read next chunk from source stream
   try {
      read = fromStream.read(buffer, 0, buffer.length);
      if (read < 0) {
         chunkData = new byte[0];
      } else if (read < buffer.length) {
         chunkData = Arrays.copyOf(buffer, read);
      } else {
         chunkData = buffer;
      }
   } catch (IOException e) {
      Logger.getAnonymousLogger().info(
                  String.format("Client IO error: %s", e.getMessage()));
      if (fileID != null) {
         wsClient.uploadFile(msgFactory.createFileUploadRequest(
                        session, database, docID, fileID, TransferStatus.CANCEL, null, null));
      }
      throw e;
   }

   if (read > 0) {
      // input stream has more data: send next chunk in a 'IN_PROGRESS'-message
```

```
            // to the service
            fileIsEmpty = false;
            resp = executeRequest(wsClient, request, chunkData);
            // if(debug) assert resp.getStatus()==TransferStatus.IN_PROGRESS
            if (isNewFile) {
               // when we upload a new file (UploadNewFileRequest), the first response
               // contains the ID of the new file the ID is needed to identify the upload
               // handle in following upload requests
               fileID = resp.getNewFileID();
               isNewFile = false;
            }
         } else {
            // special case: empty file; initiate request wasn't sent yet
            if (fileIsEmpty) {
               resp = executeRequest(wsClient, request, chunkData);
               // if(debug) assert resp.getStatus()==TransferStatus.IN_PROGRESS
               if (isNewFile) {
                  // when we upload a new file (UploadNewFileRequest), the first response
                  // contains the ID of the new file. the file ID is needed to identify
                  // the upload handle in following upload requests
                  fileID = resp.getNewFileID();
                  isNewFile = false;
               }
            }
            // send 'Complete'-message to the service
            resp = wsClient.uploadFile(msgFactory.createFileUploadRequest(
               session, database, docID, fileID, TransferStatus.COMPLETE, null, null));
            // if(debug) assert resp.getStatus()==TransferStatus.COMPLETE
         }
         request = null;
         if (resp.getStatus() == TransferStatus.IN_PROGRESS) {
            request = msgFactory.createFileUploadRequest(
              session, database, docID, fileID,
              TransferStatus.IN_PROGRESS, resp.getNextChunkID(), null);
         }
      }
      // the last response (that is the response to the COMPLETE-request)
      // returns the ID of the new file / version
      return resp == null ? null : resp.getNewFileID();
   }
```

Listing B.12: Chunked-Transfer Upload

```
private static FileUploadResponse executeRequest(ProxessDocumentServiceContract wsclient,
  Object request, byte[] chunkData) throws Exception {
    if (request == null) {
        throw new NullPointerException("request");
    }
    if (request instanceof FileUploadRequest) {
        FileUploadRequest req = (FileUploadRequest) request;
        req.setChunkData(chunkData);
        return wsclient.uploadFile(req);
    } else if (request instanceof NewFileUploadRequest) {
```

```
        NewFileUploadRequest req = (NewFileUploadRequest) request;
        req.setChunkData(chunkData);
        return wsclient.uploadNewFile(req);
    } else {
        throw new RuntimeException("Unsupported request type: " +
                request.getClass().getName());
    }
}
```

Listing B.13: Chunked-Transfer Upload - Hilfsfunktion executeRequest()