Infopark CMS Fiona

XML Interface Referenz

Infopark CMS Fiona

XML Interface Referenz

Inhalt

1 Vorbemerkungen

1.1 Über dieses Handbuch

Dieses Handbuch beschreibt die *Content Retrieval and Update Language* (CRUL). Die CMS-Server-Applikationen verwenden dieses Protokoll, um über ihre XML-Schnittstelle miteinander zu kommunizieren. Das Handbuch wendet sich an Entwickler, die Client-Software oder Skripte schreiben möchten, um auf die Daten und Funktionen des Content Management Servers zuzugreifen. Dafür ist es erforderlich, mit der Metasprache XML und der Funktionsweise des Content Managers und anderer CMS-Komponenten vertraut zu sein.

XML und der Funktionsumfang der CMS-Komponenten werden deshalb in diesem Dokument nicht erläutert. Eine Beschreibung von XML finden Sie in dem Buch *XML Pocket Reference* (Robert Eckstein, O'Reilly & Associates, 1999) sowie auf einer Reihe von Websites, unter anderem auf:

- The XML Cover Pages: http://www.oasis-open.org/cover/sgml-xml.html
- XML at W3C: http://www.w3.org/XML

Die englischen CMS-spezifischen oder Internet-spezifischen Begriffe und Akronyme wie *Body*, *Content*, *Link* oder *Blob* sind in diesem Handbuch nicht ins Deutsche übersetzt, da auch die XML-Elemente englische Wörter oder von englischen Wörtern abgeleitete Kunstwörter sind.

1.2 Konventionen

Bestandteile der DTD sowie Request- und Responsefragmente usw. werden in der Schriftart `Courier` dargestellt.

Der Content Management Server wird in diesem Dokument häufig auch *Content Manager* genannt.

2

2 Mit CMS-Komponenten über HTTP kommunizieren

Mehrere Komponenten des Content Management Systems verfügen über eine XML-Schnittstelle. Über diese Schnittstelle können die Komponenten kommunizieren und unter anderem Informationen über Dateien und ihre Versionen austauschen.

Eine CMS-Komponente oder eine andere Applikation kann über die XML-Schnittstelle beispielsweise eine Anfrage an den Content Management Server senden. Anfragen sind XML-Dokumente, die über HTTP mit einer POST-Anweisung zum Content Manager übertragen werden. Auf diese Weise kann etwa die Benutzerschnittstelle mit HTTP-Requests auf Daten im Content Management Server lesend und schreibend zugreifen. Um dem Content Manager im Body eines HTTP-Requests ein XML-Dokument zu übermitteln, verwendet man eine URL mit folgendem Aufbau:

```
http://mein.nps.server/xml
```

Über den URL-Bestandteil `/xml` ist es einer Applikation oder einer CMS-Komponente möglich, die XML-Schnittstelle einer anderen CMS-Komponente zu erreichen. Anhand dieses URL-Bestandteils kann der Content Manager also erkennen, dass der HTTP-Request an seine XML-Schnittstelle gerichtet ist. Der Content Management Server sendet seine Antwort ebenfalls als XML-Dokument im Body seiner HTTP-Response an die jeweilige Applikation zurück (siehe das Handbuch zur *Systemadministration / Entwicklung*).

In der Regel kann eine Komponente entweder Anfrage-Dokumente oder Antwort-Dokumente kodieren, nicht jedoch beides. Zum Beispiel kann die Benutzerschnittstelle Anfragen an den Content Manager senden und dessen Antworten interpretieren, selbst jedoch keine Antworten auf Anfragen erzeugen. Dies ist auch nicht erforderlich, weil sie keine Daten verwaltet, die für andere CMS-Komponenten interessant wären.

Die XML-Dokumente, die von zwei CMS-Komponenten ausgetauscht werden, folgen einem strikten Anfrage-Antwort-Schema, wobei *Anfrage* und *Antwort* hier nicht Request und Response auf der Ebene des HTTP-Protokolls bedeuten. Vielmehr enthalten die XML-Dokumente entsprechende Anfrage- und Antwort-Elemente. Eine Komponente, die Anfragen in diesem Sinne erzeugt und die Antworten darauf interpretiert, ist ein Client. Komponenten, die Anfragen beantworten können, sind Server (siehe auch das Handbuch zur *Systemadministration / Entwicklung*).

Wie Anfrage- und Antwort-Dokumente, die sogenannten Payloads, aufgebaut sein müssen, damit sie beispielsweise der Content Management Server und die Benutzerschnittstelle akzeptieren und auswerten können, ist durch die *Content Retrieval and Update Language* (CRUL) definiert. CRUL ist eine DTD, die Aufbau und Inhalt der Payloads beschreibt.

Mit CRUL können Applikationsentwickler auf alle im Content Management Server verwalteten Daten zugreifen. Die DTD enthält jedoch keine Elemente, mit denen man auf zusätzliche (benutzerdefinierte)

Versionsfelder oder Benutzerfelder zugreifen kann. Diese Felder sind konfigurationsspezifisch. Die entsprechenden Elemente können bei Bedarf von Entwicklern in die DTD aufgenommen werden.

3

3 Die Content Retrieval and Update Language (CRUL)

3.1 Payloads

Die XML-Dokumente, die CMS-Komponenten über die XML-Schnittstelle austauschen, werden als *Payloads* bezeichnet. Das Element `cm-payload` ist das Wurzelelement aller Anfrage- und Antwort-Dokumente:

```
<!ELEMENT cm-payload (cm-header, (cm-response+ | cm-request+))>
<!ATTLIST cm-payload
  payload-id CDATA #REQUIRED
  timestamp CDATA #REQUIRED
  cm.version CDATA #REQUIRED
>
```

Die Attribute des `cm-payload`-Elements haben die folgende Bedeutung:

- `payload-id`
  Identifikator des Payloads. Dieser Identifikator wird vom Erzeuger des Payloads generiert und muss innerhalb eines Kommunikationskontexts eindeutig sein. Ein solcher Kontext wird durch den Content Management Server (beispielsweise einer Firma) und alle Clients gebildet, die mit dem Server kommunizieren. In der Regel wird die `payload-id` durch einen Algorithmus generiert.
- `timestamp`
  Datum und Uhrzeit (Zeitstempel) der Erzeugung des Payloads. Der Zeitstempel muss in kanonischer Form als 14stelliger String (von links beginnend: Jahr vierstellig, Monat zweistellig, Tag zweistellig, Stunde zweistellig, Minute zweistellig, Sekunde zweistellig) in GMT angegeben sein (Beispiel:: 20010716020223).
- `cm.version`
  Version des XML-Schnittstellenprotokolls. Der Aufbau des Payloads ist von der Protokollversion abhängig. Zum Zeitpunkt der Fertigstellung dieses Handbuchs hat das XML-Schnittstellenprotokoll die Version 6.0.0. Die Versionsnummer entspricht der Version des Content Managers.

Ein Client gibt bei einer Anfrage mit dem Wert des `cm.version`-Attributs an, welche Version des XML-Schnittstellenprotokolls er verwendet.

Der Content Management Server unterstützt neben der aktuellen Version des Protokolls alle Versionen, die bisher gültig waren. Verwendet der Client eine dieser Versionen, so erzeugt der Server einen Antwort-Payload in dieser Version. Andernfalls antwortet er mit einer Fehlermeldung, die die Protokollinkompatibilität mitteilt. Diese Meldung erzeugt der Server in seiner aktuellen Version des XML-Schnittstellenprotokolls.

Die gültigen Werte des `cm.version`-Attributs entsprechen den CMS-Versionsnummern.

3.1.1 Header-Element

Das erste Unterelement aller Payloads ist `cm-header`. Das `cm-header`-Element enthält Informationen über die Identität der beiden Seiten, die das Payload austauschen.

```
<!ELEMENT cm-header (cm-sender, cm-receiver?, cm-authentication?)>
<!ELEMENT cm-sender EMPTY>
<!ATTLIST cm-sender
  sender-id CDATA #REQUIRED
  name CDATA #REQUIRED
>
<!ELEMENT cm-receiver EMPTY>
<!ATTLIST cm-receiver
  receiver-id CDATA #REQUIRED
  name CDATA #REQUIRED
>
<!ELEMENT cm-authentication EMPTY>
<!ATTLIST cm-authentication
  login CDATA #REQUIRED
  password CDATA #REQUIRED
>
```

cm-sender

Das `cm-sender`-Element muss immer als Unterelement des `cm-header`-Elements erscheinen. Es spezifiziert die Identität des Absenders des Payloads. Die Attribute des `cm-sender`-Elements haben die folgende Bedeutung:

- `sender-id`
  Identifikator des Absenders des Payloads. Jedem Content Management Server und den Clients wird bei der Installation jeweils ein Identifikator zugewiesen, der innerhalb des Kommunikationskontextes eindeutig ist. Dieser Identifikator wird bei der Erzeugung des Payloads als `sender-id` verwendet.
- `name`
  Name des Absenders des Payloads. Der Name ist eine Zeichenkette, die die Anwendung bezeichnet, die den Payload überträgt. Der Content Management Server verwendet als Name *CM-Server*.

cm-receiver

Das `cm-receiver`-Element spezifiziert die Identität des gewünschten Empfängers des Payloads. Dieses Element ist optional, sofern es zwischen dem Server und einem Client eine individuelle Netzwerkverbindung gibt. In diesem Fall sind der Absender des Payloads und der Empfänger eindeutig identifiziert. Befindet sich allerdings zwischen dem Server und den Clients ein Proxy-Server, der mehrere Clients oder Server bedient, so können Server und Client das `cm-receiver`-Element nutzen, um dem Proxy-Server den Empfänger mitzuteilen. Die Attribute des `cm-receiver`-Elements haben die folgende Bedeutung:

- `receiver-id`
  Identifikator des Empfängers des Payloads. Dieses Attribut hat dieselbe Semantik wie das `sender-id`-Attribut des `cm-sender`-Elements. Es enthält den innerhalb eines Kommunikationskontextes eindeutigen Identifikator des CMS-Servers oder CMS-Clients, der das Payload empfangen soll. Bei Antwort-Payloads ist dieses Attribut immer eine Kopie des `sender-id`-Attributs des `cm-sender`-Elements im entsprechenden Anfrage-Payload.
- `name`

Name des Empfängers. Der Name ist eine Zeichenkette, die die gewünschte Empfängeranwendung bezeichnet. Bei Antwort-Payloads ist dieses Attribut eine Kopie des `name`-Attributs des `cm-sender`-Elements im entsprechenden Anfrage-Payload.

cm-authentication

Das `cm-authentication`-Element ist optional. Es kann nur bei Anfrage-Payloads verwendet werden. Es enthält Informationen über den Benutzer, für den die Anfrage bearbeitet werden soll. Die Attribute des `cm-authentication`-Elements haben die folgende Bedeutung:

- `login`
  Der Anmeldename (das Login) des Benutzers des Content Management Servers.
- `password`
  Das Passwort des Benutzers (Klartext).

3.1.2 Request-Element

Bei Anfrage-Payloads folgen auf das `cm-header`-Element im `cm-payload`-Wurzelelement ein oder mehrere `cm-request`-Elemente. Diese Elemente spezifizieren die Operationen, die vom Content Management Server ausgeführt werden sollen.

```
<!ELEMENT cm-request (%cm.cm-request;)>
<!ATTLIST cm-request
  request-id CDATA #REQUIRED
  preclusive (true | false) "false"
>
```

Die Unterelemente des `cm-request`-Elements sind in der DTD definiert, die ab Zum Aufbau der Referenz erläutert wird und im Abschnitt CRUL als DTD komplett abgedruckt ist. Ein `cm-request`-Element hat folgende Attribute:

- `request-id`
  Identifikator des Requests. Dieser Identifikator wird vom Erzeuger des Anfrage-Payloads vergeben. Er muss innerhalb aller Payloads, die in einem bestimmten Kommunikationskontext ausgetauscht werden, eindeutig sein. Es reicht nicht aus, dass die ID innerhalb des Requests eindeutig ist.
- `preclusive`
  Wahrheitswert (`true` oder `false`). Das `preclusive`-Attribut erlaubt dem CMS-Client, die Requests zu markieren, die für die weitere Bearbeitung des Payloads kritisch sind. Wenn die Bearbeitung eines als `preclusive` markierten Requests fehlschlägt, werden alle weitere Requests im Payload nicht bearbeitet, sondern mit einer Fehlermeldung beantwortet.

Alle Anfragen in einem Payload werden sequenziell, beginnend bei der ersten Anfrage, bearbeitet. Dadurch ist es einem Client möglich, auch voneinander abhängende Requests in einem einzigen Anfrage-Payload unterzubringen. So kann beispielsweise ein Client in einem Payload zunächst eine Vorlage anlegen, um anschließend Dateien zu erzeugen, die auf dieser Vorlage beruhen. Mit dem `preclusive`-Attribut kann der Client ferner sicherstellen, dass die abhängige Anfrage nur dann bearbeitet wird, wenn die vorausgehende keinen Fehler erzeugt hat.

3.1.3 Response-Element

Bei Antwort-Payloads enthält das `cm-payload`-Wurzelelement ein oder mehrere `cm-response`-Elemente nach dem `cm-header`-Element, mit denen die jeweiligen Ergebnisse der vom Content Manager ausgeführten Operationen zurückgeliefert werden.

Wenn ein Anfrage-Payload vom Content Management Server vollständig erkannt und bearbeitet wurde, entspricht jedes `cm-response`-Element im Antwort-Payload genau einem `cm-request`-Element des Anfrage-Payloads. In diesem Fall enthalten die `cm-response`-Elemente Meldungen, die sich auf die Inhalte der Requests beziehen (Request-Level-Meldungen).

Erhält der CMS-Server dagegen ein ungültiges Anfrage-Payload (beispielsweise ohne `cm-header`-Element oder mit nicht erkennbaren `cm-request`-Elementen), so liefert der Content Manager ein Antwort-Payload zurück, das nur ein `cm-response`-Element enthält, mit dem der allgemeine Fehler gemeldet wird. In diesem Fall bezieht sich das `cm-response`-Element auf den Payload (Payload-Level-Meldung). Ein `cm-response`-Element ist folgendermaßen aufgebaut.

```
<!ELEMENT cm-response (cm-code*)>
<!ATTLIST cm-response
  response-id CDATA #REQUIRED
  payload-id CDATA #IMPLIED
  request-id CDATA #IMPLIED
  success ("true" | "false") #REQUIRED
>
```

Die einzelnen Antworten auf die Requests werden in `cm-code`-Elementen zurückgegeben:

```
<!ELEMENT cm-code ANY>
<!ATTLIST cm-code
  numeric CDATA #REQUIRED
  phrase CDATA #REQUIRED
>
```

Die Attribute des `cm-response`-Elements haben die folgende Bedeutung:

- `response-id`
  Identifikator der Antwort. Der Identifikator wird vom Erzeuger des Antwort-Payloads gesetzt. Er muss innerhalb aller Payloads, die in einem Kommunikationskontext ausgetauscht werden, eindeutig sein.
- `payload-id`
  Identifikator des Anfrage-Payloads. Er entspricht dem `payload-id`-Attribut des Anfrage-Payloads. Dieses Attribut wird vom Server nur dann eingefügt, wenn das `cm-response`-Element eine Payload-Level-Meldung im ersten `cm-code`-Element enthält. Ein Client kann folglich am Vorhandensein dieses Attributs erkennen, ob sein Anfrage-Payload als solches (nicht die in ihm enthaltenen Anfragen) vom Server interpretiert werden konnte.
- `request-id`
  Identifikator des Requests. Er entspricht dem `receiver-id`-Attribut des `cm-request`-Elements im Anfrage-Payload. Dieses Attribut ist nur vorhanden, wenn das `cm-code`-Element eine Request-Level-Meldung enthält.
- `success`
  Der Wert dieses Attributs ist `true`, wenn der Request erfolgreich bearbeitet werden konnte. Andernfalls ist er `false`.

Bitte beachten Sie, dass innerhalb eines Requests mehrere Operationen ausgeführt werden können. In diesem Fall enthält das `success`-Attribut das logische Und der Ergebnisse aller ausgeführten Operationen. Das heißt, `success` ist nur dann `true`, wenn alle Operationen erfolgreich waren.

Die Ergebnisse der ausgeführten Operationen werden mittels `cm-code`-Elementen innerhalb des `cm-response`-Elements zurückgeliefert. Die `cm-code`-Elemente enthalten eine Erfolgs- oder Fehlermeldung und weitere XML-Elemente, die das Resultat der Operation oder gegebenenfalls Fehlerinformationen darstellen.

Die Attribute des `cm-code`-Elements haben die folgende Bedeutung:

- `numeric`
  Fehlernummer (oder Erfolgsmeldungsnummer). Die den Nummern entsprechenden Meldungen werden im Abschnitt Fehlermeldungen beschrieben.
- `phrase`
  Die Beschreibung des Fehlers.

Der Inhalt des `cm-code`-Elements hängt von der Operation ab, die im Request angegeben war. Bei Operationen, die nicht erfolgreich ausgeführt werden konnten, hängt der Inhalt des `cm-code`-Elements vom aufgetretenen Fehler ab.

Fehlerinformationen in Responses

Im Fehlerfalle liefert die CMS-Applikation als Inhalt des `cm-code`-Elements ein `errorStack` -Element, das je Fehler ein `error`-Element enthält. Dieses wiederum enthält mindestens je ein `numeric`- und ein `phrase`-Element, deren Inhalt die Fehlernummer bzw. die Fehlermeldung ist. Beispiel:

```
<cm-code phrase="[100134] Die Datei kann der Workflowaktion 'Einreichen' nicht unterzogen
 werden." numeric="100134">
  <errorStack>
    <error>
      <numeric>100134</numeric>
      <phrase>[100134] Die Datei kann der Workflowaktion 'Einreichen' nicht unterzogen
 werden.</phrase>
    </error>
    <error>
      <numeric>60029</numeric>
      <phrase>[060029] Es gibt keine Arbeitsversion.</phrase>
    </error>
  </errorStack>
</cm-code>
```

Die Reihenfolge der Fehler im `errorStack`-Element ist umgekehrt zu der Reihenfolge, in der die Fehler aufgetreten sind. Der am weitesten unten stehende Fehler ist also zuerst aufgetreten.

Bei Fehlermeldungen, die weitere Informationen liefern, enthält das `error`-Element zusätzlich ein `info`-Element mit diesen Informationen.

3.2 Mit Elementen auf Daten zugreifen und Funktionen aufrufen

In diesem Abschnitt wird beschrieben, wie Applikationen mit CRUL über die XML-Schnittstelle des Content Management Servers Funktionen aufrufen und auf die Daten zugreifen können, die dieser verwaltet.

Um die Struktur von CRUL besser verstehen zu können, ist es wichtig zu wissen, dass der Content Manager seine Daten in Klassen verwaltet. So sind in der Klasse der Dateien sämtliche Dateien enthalten, in der Klasse der Workflows alle Workflows usw. Die einzelne Datei, der einzelne Workflow wird als Instanz seiner Klasse bezeichnet.

Die Instanzen einer Klasse haben in der Regel Parameter. So haben alle Benutzer - also alle Instanzen der Klasse der Benutzer - unter anderem die Parameter *Login* und *E-Mail*. Die Parameter der Instanzen unterscheiden sich natürlich von Klasse zu Klasse. Jeweils einer der Parameter ist stets der sogenannte Primärschlüssel. Über diesen Parameter kann man gezielt auf jede einzelne Instanz einer Klasse zugreifen. Bei Dateien beispielsweise ist der Primärschlüssel die Datei-ID, bei Workflows der Name.

Primärschlüssel sind eindeutig - es kann nicht zwei Dateien mit der gleichen ID oder zwei Benutzer mit dem gleichen Login geben.

CRUL enthält Elemente, mit denen man auf Klassen und Instanzen zugreifen kann. Da diese Elemente Funktionsaufrufe im Content Manager bewirken, werden sie Funktionselemente genannt. Das Element `obj-create` etwa bewirkt, dass die Dateierzeugungsfunktion der Klasse der Dateien (`obj`) aufgerufen wird. Sie legt eine Datei an.

Verwendet man Funktionselemente, um auf eine Instanz zuzugreifen, so ist es meistens erforderlich, Parameter anzugeben. So kann man mit dem Funktionselement `obj-get` Dateiparameter auslesen. Die Parameter werden als Unterelemente von Funktionselementen kodiert und als Parameterelemente bezeichnet.

In den folgenden Abschnitten werden wir näher erläutern, wie die Elemente verwendet werden und die Responses des Content Managers aussehen.

3.2.1 Funktionen aufrufen

Das Tag eines Funktionselements setzt sich aus dem Klassennamen und einem Funktionsnamen zusammen, wobei ein Bindestrich die beiden Komponenten miteinander verbindet:

```
<attribute-create> ... </attribute-create>
```

Mit diesem Element wird ein Feld erzeugt. Die hierfür erforderlichen Parameter und deren Werte werden als Unterelemente angegeben:

```
<cm-request ...>
  <attribute-create>
    <name>Adresse</name>
    <type>string</type>
  </attribute-create>
</cm-request>
```

Die Response auf eine solche Anfrage enthält den Namen des erzeugten Feldes:

```
<cm-response ...>
  <cm-code numeric="0" phrase="ok">
    <attribute>
      <name>Adresse</name>
    </attribute>
  </cm-code>
</cm-response>
```

Der Rückgabewert im Antwortdokument der XML-Schnittstelle des Content Management Servers ist im Klassennamen-Element (hier `attribute`) enthalten. Grundsätzlich liefert die XML-Schnittstelle bei Instanziierungen den Wert des Primärschlüssels der erzeugten Instanz, bei Dateien beispielsweise die ID, bei Benutzern das Login und bei Workflows den Namen.

3.2.2 Instanzen ermitteln und darauf zugreifen

Möchte man Parameterwerte von Instanzen setzen oder auslesen, so benötigt man zunächst die Liste der Primärschlüssel der betreffenden Instanzen. Diese Liste erhält man mit der `where`-Funktion, bei Dateien auch mit der `search`-Funktion. Der folgende Request zeigt, wie alle Dateien ermittelt

werden, die auf der Vorlage `notes` basieren und wie bei diesen Dateien der Wert des Parameters `suppressExport` auf 0 gesetzt wird:

```
<cm-request ...>
  <obj-where>
    <objClass>notes</objClass>
  </obj-where>
  <obj-set>
    <suppressExport>0</suppressExport>
  </obj-set>
</cm-request>
```

Natürlich können im `set`-Element die Werte mehrerer Parameter gleichzeitig gesetzt werden. Das `where`-Element kann bei den meisten Klassen ebenfalls mehrere Unterelemente haben, um die zu modifizierenden Instanzen nach Belieben zu selektieren:

```
<cm-request ...>
  <obj-where>
    <objClass>notes</objClass>
    <nameLike>news</nameLike>
  </obj-where>
  <obj-set>
    <suppressExport>0</suppressExport>
    <workflowName>notesEditors</workflowName>
  </obj-set>
</cm-request>
```

Ferner kann das `where`-Element bei fast allen Klassen auch leer sein. Auf diese Weise können alle Instanzen einer Klasse ermittelt werden:

```
<cm-request ...>
  <obj-where/>
  <obj-get>
    <id/>
    <path/>
  </obj-get>
</cm-request>
```

Schließlich kann im `where`-Element mit dem Attribut `maxResults` die Höchstanzahl der zu liefernden Treffer angegeben werden:

```
<cm-request ...>
  <obj-where maxResults="100"/>
  <obj-get>
    <...>
  </obj-get>
</cm-request>
```

Auf `where`-Elemente muss stets ein Funktionselement unmittelbar folgen, mit dem Instanzen modifiziert oder Parameter von Instanzen ausgelesen werden. Eine solche Verbindung aus einem Instanzen ermittelnden Element und einem Element, das auf die ermittelten Instanzen zugreift, bildet eine Schleife. Das zweite, d. h. das modifizierende oder auslesende Funktionselement (meist ein `set`- oder `get`-Element) wird auf jede der zuvor ermittelten Instanzen angewendet. Ist bei einer `get`-Operation eine Instanz nicht definiert, wird als Ergebnis für diese Instanz ein leeres Element geliefert, das ein Client korrekt verarbeiten können muss. Dagegen wird bei `set`-Operationen in diesem Fall ein Fehler erzeugt.

Bei allen Klassen außer `obj` ist das `where-` Element das einzige, mit dem Instanzen ermittelt werden können. Bei `obj` gibt es neben `obj-where` das Funktionselement `obj-search`, mit dem man die erweiterte Suchfunktion der optionalen Infopark Search Cartridge nutzen kann.

3.2.3 Parameter und ihre Werte

Möchte man mit einem `set`-Element Instanzen modifizieren, so gibt man (wie oben beschrieben) die betreffenden Parameter als Unterelemente an. Die Inhalte der Parameterelemente werden vom Content Management Server als die neuen Werte der Parameter interpretiert.

Greift man dagegen mit `get` (d. h. lesend) auf Instanzen zu, so müssen die Parameter-Elemente leer sein, weil der Content Management Server die Elementhierarchie im `get`-Element wie eine Schablone verarbeitet. Für jede mit dem `where`-Element ermittelte Instanz wird die Schablone mit den entsprechenden Werten ausgefüllt und in den Antwort-Request eingefügt. Dies wird im Abschnitt Requests als Schablonen und die Darstellung der Ergebnis-Daten erläutert.

Einfache und strukturierte Werte

Parameterwerte können - unabhängig davon, ob Sie sie setzen oder auslesen - einfache oder strukturierte Werte sein. Ein einfacher Wert ist eine Zeichenkette, ein strukturierter Wert ist eine Liste oder ein sogenanntes Dictionary. Während *Zeichenkette* ein gängiges Konzept ist, bedürfen *Liste* und *Dictionary* einer Erläuterung. Eine Liste ist eine geordnete Menge beliebig vieler Elemente. Jedes dieser Elemente wird durch ein `listitem`-Element dargestellt:

```
<cm-request ...>
  <attribute-where>
    <name>topics</name>
  </attribute-where>
  <attribute-addEnumValues>
    <listitem>books</listitem>
    <listitem>computing</listitem>
    <listitem>music</listitem>
  </attribute-addEnumValues>
</cm-request>
```

Im obigen Beispiel werden zum Feld `topics` (bei dem wir voraussetzen, dass es vom Typ `enum` oder `multienum` ist) die drei möglichen Werte `books`, `computing` und `music` hinzugefügt. Die Ordnung in einer solchen Liste ergibt sich aus der Reihenfolge der Listenelemente. Sie ist im Allgemeinen nicht relevant.

Ein Dictionary ist eine nicht geordnete Menge beliebig vieler Zuordnungen von Werten zu Namen. Jede Zuordnung wird durch ein `dictitem`-Unterelement repräsentiert, das wiederum aus einem Namen (`key`) und einem Wert (`value`) besteht:

```
<myDictionary>
  <dictitem>
    <key>name</key>
    <value>theName</value>
  </dictitem>
  <dictitem>
  ...
  </dictitem>
</myDictionary>
```

Dictionarys, die mit `dictitem`-Elementen strukturiert sind, kommen im XML-Interface recht selten (beispielsweise bei Zugriffen auf Benutzereinstellungen) vor. Die meisten Name-Wert-Paare werden in CRUL mit dem Namen als Elementname und dem Wert als Inhalt des Elements kodiert.

Der Wert eines Dictionary-Elements und eines Listenelements kann eine Zeichenkette oder wiederum eine Liste oder auch ein Dictionary sein.

Dimensionen von Werten

Im Content Management Server stehen manche Parameterwerte in mehreren Sprachen zur Verfügung. So haben beispielsweise Versionen sowohl ein allgemeines Attribut `title` als auch sprachspezifische Versionen dieses Attributs wie einen englischen Titel usw. Diese „Fähigkeit" einer Instanz, mehrere „Auslegungen" oder Versionen eines Parameters haben zu können, nennt man eine *Dimension*. Das Feld `title` einer Version oder Feldes, einer Vorlage und eines Workflows hat die Dimension *Sprache*. Auf die Dimension können Sie sich mit einem Tag-Attribut im entsprechenden XML-Tag beziehen.

Sprache ist zurzeit die einzige Parameter-Dimension im Content Manager. Sie steht bei den `title`- und `helpText`-Parametern der meisten Klassen zur Verfügung. Die Sprachdimension wird mit dem `lang`-Tag-Attribut kodiert:

```
<cm-request ...>
  <obj-where>
    <objClass>document</objClass>
  </obj-where>
  <obj-get>
    <title lang="en"/>
  </obj-get>
</cm-request>
```

3.3 Requests als Schablonen und die Darstellung der Ergebnis-Daten

Die Kommunikation mit dem Content Management Server über die XML-Schnittstelle findet mit Payloads statt. Payloads enthalten Anfragen (Requests) und Antworten (Responses). Für jede Anfrage in einem Payload wird genau eine Antwort im Ergebnis-Payload erzeugt.

```
<cm-request ...>
  <obj-create>
    <name>simplePub</name>
    <parent>/</parent>
    <objClass>publication</objClass>
  </obj-create>
</cm-request>
```

Der obige Request, der eine Datei unmittelbar unterhalb des Basisordners anlegt, führt zu folgender Response:

```
<cm-response ...>
  <cm-code numeric="0" phrase="ok">
    <obj>
      <id>32191</id>
    </obj>
  </cm-code>
</cm-response>
```

Bei Anfragen, mit denen Parameterwerte von Instanzen mit `where` und folgendem `get` ermittelt werden sollen, gibt die Struktur der Anfrage die Struktur der Antwort vor. Eine Anfrage ist gewissermaßen eine Schablone, die der Content Management Server ausfüllt, um die Antwort zu

erzeugen. Im folgenden Beispiel gibt der Request vor, dass die Response den Namen und die ID der selektierten Dateien enthalten soll.

```
<cm-request ...>
  <obj-where>
    <objClass>document</objClass>
  </obj-where>
  <obj-get>
    <name/>
    <id/>
  </obj-get>
</cm-request>
```

Das Antwort-Dokument zu diesem Request wird eine Response enthalten, in der für jede auf das `obj-where`-Element passende Datei der Name und die ID geliefert werden:

```
<cm-response ...>
  <cm-code numeric="0" phrase="ok">
    <obj>
      <name>mary</name>
      <id>92674</id>
    </obj>
  </cm-code>
  <cm-code numeric="0" phrase="ok">
    <obj>
      <name>john</name>
      <id>72941</id>
    </obj>
  </cm-code>...
</cm-response>
```

In den beiden vorausgehenden Beispiel-Responses ist zu erkennen, dass für jede in der Anfrage mit `where` selektierte Instanz ein `cm-code`-Element erzeugt wird. Innerhalb eines solchen Elements sind die abgefragten Parameterwerte je Instanz als Inhalt des Klassennamenelements kodiert. In den obigen Beispielen ist dies das `obj`-Element.

Manche Parameterwerte sind Referenzen auf Instanzen der gleichen oder einer anderen Klasse. So ist beispielsweise die übergeordnete Datei (`parent`) eines Ordners (außer beim Basisordner) eine Referenz auf eine Vorlage. Die Mitglieder (`users`) einer Benutzergruppe (`group`) sind Benutzer-Referenzen, und die obligatorischen Felder (`mandatoryAttributes`) einer Vorlage (`objClass`) sind Referenzen auf Felder (`attribute`). Wie später gezeigt wird, können die Instanzen, auf die sich die Referenzen beziehen, durch eine Erweiterung der Abfrage-Schablone im gleichen `get`-Aufruf ebenfalls nach Werten ihrer Parameter befragt werden.

Hier folgt jedoch zunächst ein Beispiel, das zeigt, wie die im Basisordner enthaltenen Dateien abgefragt und in der Antwort kodiert werden:

```
<cm-request ...>
  <obj-where>
    <path>/</path>
  </obj-where>
  <obj-get>
    <children/>
  </obj-get>
</cm-request>
```

Die Antwort hat das folgende Schema:

```
<cm-response ...>
```

```
  <cm-code numeric="0" phrase="ok">
    <obj>
      <children>
        <listitem>92674</listitem>
        <listitem>72941</listitem>
      </children>
    </obj>
  </cm-code>
</cm-response>
```

Zurückgegeben werden in diesen Fällen die eindeutigen Identifikatoren der Instanzen, bei Dateien, Versionen, Links und Tasks die IDs, bei den Instanzen aller anderen Klassen die Namen. Ein Identifikator wird nicht als Inhalt des entsprechenden Parameterelements kodiert, d. h. die ID einer Dateireferenz ist nicht der Inhalt eines `id`-Elements und der Name eines Feldes ist nicht der Inhalt eines `name`-Elements. Stattdessen werden die Identifikatoren mit `listitem` als Listenelemente kodiert. Hier ein Beispiel, in dem die Gruppen ermittelt werden, in denen ein Benutzer Mitglied ist. Auf den Request

```
<cm-request ...>
  <user-where>
    <login>mustermann</login>
  </user-where>
  <user-get>
    <groups/>
  </user-get>
</cm-request>
```

folgt eine Antwort nach diesem Schema:

```
<cm-response ...>
  <cm-code numeric="0" phrase="ok">
    <user>
      <groups>
        <listitem>cmadmins</listitem>
        <listitem>sysadmins</listitem>
      </groups>
    </user>
  </cm-code>
</cm-response>
```

3.3.1 Erweiterte Abfrage

Wenn die in einer Anfrage ermittelten Daten Instanzen einer Klasse sind, so kann man die Anfrage erweitern, um auch Daten dieser Instanzen zu ermitteln. So sind die im obigen Beispiel abgefragten Gruppen Instanzen der Klasse `group`, und jede dieser Instanzen kann im gleichen Request nach Parameterwerten befragt werden:

```
<cm-request ...>
  <user-where>
    <login>mustermann</login>
  </user-where>
  <user-get>
    <groups>
      <group>
        <realName/>
        <users/>
      </group>
    </groups>
  </user-get>
</cm-request>
```

Der obige Request erzeugt eine Antwort nach dem folgenden Muster:

```
<cm-response ...>
  <cm-code numeric="0" phrase="ok">
    <user>
      <groups>
        <group>
          <realName>CM-Administratoren</realName>
          <users>
            <listitem>musterfrau</listitem>
            <listitem>mustermann</listitem>
          </users>
        </group>
        <group>
          <realName>System-Administratoren</realName>
          <users>
            <listitem>mustermann</listitem>
          </users>
        </group>
      </groups>
    </user>
  </cm-code>
</cm-response>
```

Mit einer einzigen Anfrage können also Daten über die ursprünglichen und die von ihnen referenzierten Instanzen in beliebiger Tiefe ermittelt werden. Zu diesem Zweck kodiert man wie im obigen Beispiel als Inhalt des `get`-Elements den abzufragenden Parameter nicht als leeres Element, sondern gibt als Inhalt dieses Elements die Schablone für die weitere Abfrage vor.

Das folgende Beispiel zeigt, wie man bei Ordnern den jeweiligen Namen und die Vorlage ermittelt. Von den Vorlagen werden die erlaubten Vorlagen für die in einem Ordner enthaltenen Dateien abgefragt und von diesen wiederum der Name und der Titel in deutscher Sprache:

```
<cm-request ...>
  <obj-where>
    <objType>publication</objType>
  </obj-where>
  <obj-get>
    <name/>
    <objClass>
      <name/>
      <validSubObjClasses>
        <name/>
        <title lang="de"/>
      </validSubObjClasses>
    </objClass>
  </obj-get>
</cm-request>
```

Diese Anfrage erzeugt eine Antwort nach dem folgenden Schema:

```
<cm-response ..>
  <cm-code numeric="0" phrase="ok">
    <obj>
      <name>ROOTPUB</name>
      <objClass>
        <name>publication<name/>
        <validSubObjClasses>
          <objClass>
            <name>publication</name>
            <title lang="de">Standard-Ordner</title>
          </objClass>
          <objClass>
```

```
            <name>document</name>
            <title lang="de">Standard-Dokument</title>
          </objClass>
          ...
        </validSubObjClasses>
      </objClass>
    <obj>
  </cm-code>
  <cm-code numeric="0" phrase="ok">
  ...
  </cm-code>
  <cm-code numeric="0" phrase="ok">
  ...
  </cm-code>
  ...
</cm-response>
```

Bitte beachten Sie, dass ein Parameterwert nur dadurch zu einer weiterzuverfolgenden Referenz wird, dass das entsprechende Element nicht leer ist, sondern die zu ermittelnden Parameter in seinem Inhalt kodiert sind. Die folgende Anfrage liefert zu jeder Datei des Typs `publication` die ID der darüber liegenden Datei und die Identifikatoren (IDs) der Dateien in dem Ordner (also die ursprüngliche Datei und deren Geschwister-Dateien):

```
<cm-request ...>
  <obj-where>
    <objType>publication</objType>
  </obj-where>
  <obj-get>
    <parent/>
    <parent>
      <children/>
    </parent>
  </obj-get>
</cm-request>
```

Während das erste `parent`-Element nur die ID liefert, weil es leer ist, liefert das zweite die Werte der in seinem Inhalt spezifizierten Parameter der darüber liegenden Datei, also die `children`.

3.4 Beispiele für Payloads

Im Folgenden sind ein Anfrage-Payload mit zwei Anfragen und ein entsprechendes Anwort-Payload abgedruckt. Die Unterelemente, die in `cm-request`- und `cm-response`-Elementen auftreten dürfen, werden im Abschnitt Die Content Retrieval and Update Language (CRUL) beschrieben.

3.4.1 Anfrage-Payload

Mit dem ersten Request im Anfrage-Payload werden die ID und der Name aller Dateien abgefragt, deren Vorlage `Bericht` ist. Die zweite Anfrage legt das Feld `farbe` an.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE request SYSTEM "http://www.infopark.com/cm.dtd">
<cm-payload payload-id="B42TE241"
            timestamp="20000705020223" cm.version="5.0.0">
  <cm-header>
    <cm-sender sender-id="FX45RTDT" name="HTMLUI"/>
    <cm-authentication login="holmes" password="apple"/>
  </cm-header>
  <cm-request request-id="H4BPBYE3">
    <obj-where>
      <objClass>Bericht</objClass>
```

```
    </obj-where>
    <obj-get>
      <id/>
      <name/>
    </obj-get>
  </cm-request>
  <cm-request request-id="BH423MXA">
    <attribute-create>
      <name>farbe</name>
      <type>string</type>
    </attribute-create>
  </cm-request>
</cm-payload>
```

3.4.2 Antwort-Payload

Die erste Antwort im Antwort-Dokument enthält als Ergebnis die ID und den Namen zweier Dateien. Die zweite Antwort enthält die Fehlermeldung, dass das Feld bereits existiert.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cm-payload SYSTEM "http://www.infopark.com/cm.dtd">
  <cm-payload payload-id="B3BWPOIU"
              timestamp="20000705020224" cm.version="5.0.0">
    <cm-header>
      <cm-sender sender-id="G33Z4GZU" name="CM SERVER"/>
    </cm-header>
    <cm-response response-id="BR12TI5X"
                 request-id="H4BPBYE3"
                 success="true">
      <cm-code numeric="0" phrase="ok">
        <obj>
          <id>3123</id>
          <name>BerichtMai</name>
        </obj>
      </cm-code>
      <cm-code numeric="0" phrase="ok">
        <obj>
          <id>4831</id>
          <name>BerichtJuni</name>
        </obj>
      </cm-code>
    </cm-response>
    <cm-response response-id="AQP3L24V"
                 request-id="BH423MXA"
                 success="false">
      <cm-code numeric="1743" phrase="attribute already exits">
        <attribute>
          <name>farbe</name>
        </attribute>
      </cm-code>
    </cm-response>
  </cm-payload>
```

4

4 Zum Aufbau der Referenz

Die Request- und Response-Elemente sind in diesem Handbuch nach Anwendungsbereichen geordnet. In den meisten Fällen entspricht ein Anwendungsbereich einer Klasse (wie der Klasse der Benutzer oder der Dateien). Jeder Bereich beginnt mit einem Definitionsabschnitt, der den zum Bereich gehörenden Ausschnitt aus CRUL enthält. An dieser Stelle werden gegebenenfalls auch Besonderheiten der CMS-Daten in diesem Bereich genannt. Der nächste Abschnitt führt die Parameterelemente des Bereichs auf. Neben einer Beschreibung jedes Parameters findet man hier auch die entsprechenden Parameter-Definitionen aus der CRUL-DTD.

Anschließend werden die zum jeweiligen Anwendungsbereich gehörenden Funktionselemente beschrieben. Auch hier ist für jedes Element der entsprechende Ausschnitt aus CRUL abgedruckt. Die Aufgaben der Elemente werden erläutert und die Rückgabewerte in Responses spezifiziert. Viele Einzelbeschreibungen der Funktionselemente sind zum besseren Verständnis mit Zusatzinformationen oder Erläuterungen zu speziellen Unterelementen versehen. Ferner sind die Rechte aufgeführt, die ein Benutzer haben muss, um das jeweilige Element in einem Request verwenden zu können. Für jedes Element wird ein Beispiel für seine Verwendung in einem Request und in einer Response gegeben.

4.1 Anwendungsbereiche

Die folgende Aufstellung gibt Ihnen einen Überblick über die Anwendungsbereiche, denen die CRUL-Elemente zugeordnet sind.

- Applikationsbefehle - app: Applikationsbefehle ausführen (ab Applikationsbefehle - app).
- Felder - attribute: Felder anlegen, definieren und löschen (ab Felder - attribute).
- Feldergruppen - attributeGroup: Feldergruppen erzeugen, definieren und löschen (ab Feldergruppen - attributeGroup).
- Benutzer - user: Benutzernamen anlegen, ändern und löschen, globale Benutzerrechte definieren, den Verwalter eines Benutzers festlegen (ab Benutzer - user).
- Benutzerfelder - userAttribute: Benutzerfelder anlegen und löschen (ab Benutzerfelder - userAttribute).
- Benutzergruppen - group: Benutzergruppen definieren (ab Benutzergruppen - group).
- Benutzerkonfiguration - userConfig: Die benutzerspezifische Systemkonfiguration auslesen oder ändern (ab Benutzerkonfiguration - userConfig).
- Benutzermanager-Proxy - userProxy und groupProxy: Lesend auf die Daten eines externen Benutzermanagers zugreifen (ab Benutzermanager-Proxy - userProxy und groupProxy).
- Versionen - content: Feldwerte einer Version definieren, freie Links setzen, den Blob laden, eine Version löschen (ab Versionen - content).

- Inkrementeller Export - incrExport: Den Status des Inkrementellen Exports ermitteln (ab Inkrementeller Export - incrExport).
- Jobs - job: Stapelverarbeitung von Aufgaben steuern (ab Jobs - job).
- Kundenspezifische Menübefehle - customCommand: kundenspezifische Menübefehle ausführen (ab Kundenspezifische Menübefehle - customCommand).
- Links - link: Links erzeugen, Link-Parameter setzen und auslesen, Links löschen (ab Links - link).
- Lizenzmanager - licenseManager: Lizenzinformationen abrufen (ab Lizenzmanager - licenseManager).
- Log-Einträge - logEntry: Log-Einträge suchen und löschen (ab Log-Einträge - logEntry).
- Dateien - obj: Dateien anlegen, löschen und suchen, Feldwerte und Zugriffsrechte setzen und löschen, Workflow-Befehle ausführen (ab Dateien - obj).
- Vorlagen - objClass: Dateivorlagen anlegen, ändern und löschen (ab Vorlagen - objClass).
- Systemkonfiguration - systemConfig: Lesend auf die Systemkonfiguration zugreifen (ab Systemkonfiguration - systemConfig).
- Tasks - task: Tasks suchen, Task-Parameter auslesen (ab Tasks - task).
- Workflows - workflow: Workflows erzeugen und löschen, Workflow-Parameter auslesen und definieren (ab Workflows - workflow).

5

5 Datenstrukturelemente, Bezeichner und Formate

5.1 Elemente für einfache und strukturierte Werte

Die Eingabe- oder Rückgabewerte in CRUL-Payloads können einfache oder strukturierte Werte sein. Ein einfacher Wert ist eine Zeichenkette; eine Liste oder ein Dictionary ist jeweils ein strukturierter Wert (siehe Parameter und ihre Werte).

Eine Zeichenkette wird in Payloads zwischen dem Start- und dem Ende-Tag eines Elements angegeben. Die Werte einer Liste werden jeweils zwischen dem `listitem`-Start-Tag und dem `listitem`-Ende-Tag eingetragen, wenn der Wert eine Zeichenkette ist. Ist der Listenwert wieder eine Liste oder ein Dictionary, so wird zwischen dem Start- und dem Ende-Tag eine weitere Liste oder ein Dictionary angegeben.

Ein Dictionary-Eintrag enthält dagegen immer Unterelemente. Das bedeutet, dass zwischen dem `dictitem`-Start- und dem `dictitem`-Ende-Tag keine Zeichenkette enthalten sein kann, sondern Unterelemente angegeben werden müssen, wobei diese wiederum eine Zeichenkette enthalten können. Listen und Dictionaries können beliebig oft ineinander verschachtelt werden.

Zeichenkette (`#PCDATA`)

Bedeutung: Der Eingabe- oder Rückgabewert ist eine Zeichenkette, also ein `string`, eine Zahl (`number`), ein Datum mit oder ohne Uhrzeit (`datetime`), ein logischer Wert (`bool`) oder eine leere Zeichenkette (`void`).

Definition:

```
<!ENTITY % cm.atom "#PCDATA">
<!ELEMENT elementName (&cm.atom;)>
```

wird in der Referenz ersetzt durch:

```
<!ELEMENT elementName (#PCDATA)>
```

Beispiel:

```
<attribute-create>
  <name>Auswahl</name>
  <type>enum</type>
</attribute-create>
```

Liste (`listitem`)

Bedeutung: Eine Liste ist eine geordnete Menge beliebig vieler Elemente. Jedes Element wird durch ein `listitem`-Element dargestellt. Die Ordnung der Listenelemente ergibt sich aus ihrer Reihenfolge. Der Wert eines `listitem`-Elements kann wiederum eine Liste oder ein Dictionary sein.

Definition:

```
<!ENTITY % cm.atom "#PCDATA">
<!ENTITY % cm.dictitem "dictitem">
<!ENTITY % cm.listitem "listitem">
<!ELEMENT listitem (%cm.atom; | %cm.listitem; | %cm.dictitem;)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | %cm.listitem; | %cm.dictitem;)*>
```

wird in der Referenz ersetzt durch:

```
<!ELEMENT listitem (#PCDATA | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (#PCDATA)>
<!ELEMENT value (#PCDATA | listitem | dictitem)*>
```

Beispiel:

```
<attribute-addEnumValues>
  <listitem>books</listitem>
  <listitem>music</listitem>
</attribute-addEnumValues>
```

Dictionary (`dictitem`)

Bedeutung: Ein Dictionary ist eine nicht geordnete Menge beliebig vieler Zuordnungen von Werten zu Namen (Bezeichnern). Jede Zuordnung wird durch ein `dictitem`-Element (*engl.* "dictionary item") repräsentiert, das wiederum aus einem Namen (`key`) und einem Wert (`value`) besteht. Der Wert eines `dictitem`-Elements kann eine Liste oder auch ein Dictionary sein.

Definition:

```
<!ENTITY % cm.atom "#PCDATA">
<!ENTITY % cm.dictitem "dictitem">
<!ENTITY % cm.listitem "listitem">
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | %cm.listitem; | %cm.dictitem;)*>
```

wird in der Referenz ersetzt durch:

```
<!ELEMENT dictitem (key, value)>
```

```
<!ELEMENT key (#PCDATA)>
<!ELEMENT value (#PCDATA | listitem | dictitem)*>
```

Beispiel:

```
<dictitem>
  <key>type</key>
  <value>enum</value>
</dictitem>
<dictitem>
  <key>values</key>
  <value>
    <listitem>SWEET</listitem>
    <listitem>SALTY</listitem>
  </value>
</dictitem>
```

5.2 Formate und Bezeichner

5.2.1 Datumsformat

Alle Datums- und Zeitstempelangaben werden im Content Management Server in kanonischer Form als 14stelliger String (von links beginnend: Jahr vierstellig, Monat zweistellig, Tag zweistellig, Stunde zweistellig, Minute zweistellig, Sekunde zweistellig) in GMT gespeichert.

Zur Konvertierung von Datums- und Zeitstempelangaben von der kanonischen in eine konventionelle Form können Funktionselemente für die Systemkonfiguration verwendet werden (siehe Systemkonfiguration - systemConfig). Sollen bei der Umwandlung eines Datums oder einer Zeitangabe die Benutzer-Voreinstellungen für die Zeitzone und das Ausgabeformat berücksichtigt werden, so verwendet man stattdessen die entsprechenden `userConfig`-Elemente.

5.2.2 Anforderungen an Bezeichner

Sämtliche Bezeichner, die im Content Manager (für Feldnamen, Logins, Gruppennamen, Dateinamen usw.) verwendet werden, dürfen weder Leerzeichen noch Sonderzeichen enthalten.

5.2.3 Rechte

CRUL-Funktionselemente sollten in XML-Anfragen an den Content Management Server in der Regel nur dann angegeben werden, wenn der über das `cm-authentication`-Element identifizierte Benutzer das globale oder dateispezifische Recht hat, das benötigt wird, damit die von dem Element initiierte Funktion vom Content Manager ausgeführt werden kann. Verfügt er nicht über das Recht, so gibt der Content Management Server eine Fehlermeldung in seiner Response zurück. Welches Recht benötigt wird, erfahren Sie in der Beschreibung des jeweiligen Funktionselements. Im Folgenden sind alle vergebbaren Rechte aufgeführt:

Dateispezifische Rechte	Bedeutung
`permissionRead`	Leserecht

permissionWrite	Schreibrecht
permissionCreateChildren	Recht, in einem Ordner Dateien anzulegen
permissionRoot	Dateiadministrationsrecht
permissionLiveServerRead	Leserecht auf dem Liveserver

Globale Rechte	Bedeutung
permissionGlobalRoot	Administrationsrecht
permissionGlobalUserEdit	Benutzer und Gruppen anlegen und bearbeiten
permissionGlobalUserAttributeEdit	Benutzerfelder anlegen und bearbeiten
permissionGlobalRTCEdit	Felder, Workflows und Vorlagen bearbeiten
permissionGlobalExport	Das Recht, Dateien zu exportieren
permissionGlobalMirrorHandling	Das Recht, Spiegeldateien anzulegen

Die im Content Manager verwendbaren globalen Rechte sind im Systemkonfigurationseintrag `content.globalPermissions` definiert. `globalPermissions` ist eine Liste, die Sie bei Bedarf ergänzen können, wenn Sie globale Rechte zum Content Management Server hinzufügen möchten.

6

6 Applikationsbefehle - app

6.1 Definition

Mit Requests kann man applikations- und kundenspezifische Befehle ausführen lassen. Der folgende Ausschnitt aus der DTD des Content Management Servers zeigt die Elemente, mit denen dies möglich ist:

```
<!ENTITY % cm.cm-request "
  ...
  (app-get) |
  (app-execute) |
  ...
">
```

Diese Elemente müssen sich unmittelbar unterhalb des `cm-request`-Elements befinden. Sie werden im Abschnitt Funktionselemente für Felder erläutert.

6.2 Parameterelemente für Applikationsbefehle

Mit Hilfe von Funktionselementen kann man applikations- und kundenspezifische Befehle ausführen. So kann man beispielsweise mit dem `app-get`-Element unter anderem die Server-Zeitzone ermitteln. Um zu spezifizieren, welcher Wert ausgelesen werden soll, verwendet man Parameterelemente. Im Folgenden werden die Parameterelemente für Applikationsbefehle aufgeführt.

now

Bedeutung: Die Server-Zeit.

Definition:

```
<!ENTITY % cm.date "
 (%cm.atom; |
  isoDateTime |
  systemConfigFormattedTime |
  userConfigFormattedTime)*
">
<!ELEMENT now %cm.date;>
<!ATTLIST now
  type CDATA #IMPLIED
>
```

timezone

Bedeutung: Die Server-Zeitzone.

Definition:

```
<!ELEMENT timezone (%cm.atom;)>
```

appName

Bedeutung: Der Name der Applikation.

Definition:

```
<!ELEMENT appName (%cm.atom;)>
```

version

Bedeutung: Die Versionszeichenkette.

Definition:

```
<!ELEMENT version (%cm.atom;)>
```

today

Bedeutung: das aktuelle Datum (14 Stellen), 0.00 Uhr, in GMT.

Definition:

```
<!ELEMENT today (%cm.atom;)>
```

rootConfigPath

Bedeutung: der Pfad zur Hauptkonfigurationsdatei `nps.xml`.

Definition:

```
<!ELEMENT rootConfigPath (%cm.atom;)>
```

command

Bedeutung: Das auszuführende Server-Kommando.

Definition:

```
<!ELEMENT command (%cm.atom;)>
```

arguments

Bedeutung: Die Argumente zum auszuführenden Server-Kommando.

Definition:

```
<!ELEMENT arguments (%cm.listitem;)>
```

baseDir

Bedeutung: Das Installationsverzeichnis von Infopark CMS Fiona.

Definition:

```
<!ELEMENT baseDir (%cm.atom;)>
```

binDir

Bedeutung: Das Verzeichnis unterhalb des Instanzenverzeichnisses, in dem sich die Startskripte befinden.

Definition:

```
<!ELEMENT binDir (%cm.atom;)>
```

commonScriptDir

Bedeutung: Der absolute Pfad des instanzenspezifischen Verzeichnisses `script/common`.

Definition:

```
<!ELEMENT commonScriptDir (%cm.atom;)>
```

configDir

Bedeutung: Das Verzeichnis unterhalb des Instanzenverzeichnisses, in dem sich die Konfigurationsdateien befinden.

Definition:

```
<!ELEMENT configDir (%cm.atom;)>
```

dataDir

Bedeutung: Das Verzeichnis unterhalb des Instanzenverzeichnisses, in dem sich Daten wie Blobs und Streaming-Tickets befinden.

Definition:

```
<!ELEMENT dataDir (%cm.atom;)>
```

instanceDir

Bedeutung: Das Instanzenverzeichnis.

Definition:

```
<!ELEMENT instanceDir (%cm.atom;)>
```

libDir

Bedeutung: Das Verzeichnis unterhalb des Installationsverzeichnisses, in dem sich die Bibliotheken und ausführbaren Dateien befinden.

Definition:

```
<!ELEMENT libDir (%cm.atom;)>
```

logDir

Bedeutung: Das Verzeichnis unterhalb des Instanzenverzeichnisses, in dem sich die Protokolldateien befinden.

Definition:

```
<!ELEMENT logDir (%cm.atom;)>
```

scriptDir

Bedeutung: Das Verzeichnis unterhalb des Instanzenverzeichnisses, in dem sich die Skripte befinden. Das Verzeichnis `cm` wird vom Content Manager und der Template Engine gemeinsam verwendet, das Verzeichnis `ses` vom Search Engine Server.

Definition:

```
<!ELEMENT scriptDir (%cm.atom;)>
```

shareDir

Bedeutung: Das Verzeichnis unterhalb des Installationsverzeichnisses, in dem sich die von allen Instanzen gemeinsam benötigten Daten befinden, beispielsweise die Dokumentation.

Definition:

```
<!ELEMENT shareDir (%cm.atom;)>
```

tmpDir

Bedeutung: Das Verzeichnis unterhalb des Instanzenverzeichnisses, in dem sich temporäre Dateien wie beispielsweise PID-Dateien befinden.

Definition:

```
<!ELEMENT tmpDir (%cm.atom;)>
```

6.3 Funktionselemente für Applikationsbefehle

6.3.1 <app-execute>

Definition:

```
<!ENTITY % cm.app-execute "
 (command,
  arguments)
">
<!ELEMENT app-execute %cm.app-execute;>
```

Aufgabe: Führt das kundenspezifische Kommando `command` mit den angegebenen Argumenten `arguments` aus. Kundenspezifische Kommandos müssen in der Systemkonfiguration registriert sein. Bitte entnehmen Sie die Details dem Handbuch zur *Systemadministration / Entwicklung*.

Rückgabewert bei Erfolg: Der Pfadbestandteil der URL, unter der der Rückgabewert des kundenspezifischen Befehls ermittelt werden kann.

Erforderliche Rechte: Falls ein globales Recht in der Definition eines kundenspezifischen Kommandos angegeben wurde, so ist dieses Recht erforderlich, um das Kommando auszuführen.

Beispiel:

```
<cm-request...>
  <app-execute>
    <command>procAlias</command>
    <arguments>
      <listitem>65382</listitem>
      <listitem>text</listitem>
    </arguments>
  </app-execute>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">/stream/87629</cm-code>
</cm-response>
```

6.3.2 <app-get>

Definition:

```
<!ENTITY % cm.app-get "
  (appName |
  baseDir |
  binDir |
  commonScriptDir |
  configDir |
  dataDir |
  debugChannels |
  debugLevel |
  getKeys |
  instanceDir |
  libDir |
  logDir |
  now |
  rootConfigPath |
  scriptDir |
  shareDir |
  timeZone |
  tmpDir |
  today |
  version)*
">
<!ELEMENT app-get %cm.app-get;>
```

Aufgabe: Ermittelt einen applikationsspezifischen Wert.

Rückgabewert bei Erfolg: Der applikationsspezifische Wert.

Erforderliche Rechte: Keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <app-get>
    <timezone/>
  </app-get>
</cm-request>

<cm-response...>
```

```
   <cm-code numeric="0" phrase="ok">MET</cm-code>
</cm-response>
```

7

7 Felder - attribute

7.1 Definition

In Requests kann man auf Felder lesend und schreibend zugreifen. Der folgende Ausschnitt aus der DTD des Content Management Servers zeigt die Elemente, mit denen dies möglich ist:

```
<!ENTITY % cm.cm-request "
  ...
  (attribute-create) |
  (attribute-types) |
  (attribute-where+, attribute-addEnumValues) |
  (attribute-where+, attribute-delete) |
  (attribute-where+, attribute-description) |
  (attribute-where+, attribute-get) |
  (attribute-where+, attribute-getValidEditFieldKeys) |
  (attribute-where+, attribute-removeEnumValues) |
  (attribute-where+, attribute-set) |
  ...
">
<!ATTLIST attribute-where
  maxResults CDATA #IMPLIED
>
```

Diese Elemente müssen sich unmittelbar unterhalb des `cm-request`-Elements befinden. Sie werden im Abschnitt Funktionselemente für Felder erläutert.

Mit dem Attribut `maxResults`, dessen Wert eine ganze Zahl ist, kann die maximale Anzahl der Treffer festgelegt werden. Ist der Wert von `maxResults` kleiner oder gleich null, ist die Anzahl nicht begrenzt.

7.1.1 Feldtypen

Feldwerte sind im Content Management Server je nach Datentyp größenbeschränkt. In der folgenden Tabelle sind die Datentypen, ihre Bedeutung sowie die maximale Größe eines Feldwerts für den jeweiligen Typ aufgeführt:

Datentyp	Größe (Bytes)	Inhalt
`date`	14	Datum
`enum`	250	Aufzählungswerte
`html`	unbegrenzt	HTML-Code
`linklist`	unbegrenzt	Liste mit IDs freier Links

`markdown`	unbegrenzt	Mit Markdown-Auszeichnungen versehener Text (wird nur bei Einsatz des Infopark Rails Connectors ausgewertet)
`multienum`	250	Aufzählungswerte mit Mehrfachauswahl
`signature`	250	Signatur, bestehend aus Gruppenname und Zeitstempel
`string`	abhängig von *`maxLength`* (`editFieldSpec`)	Zeichenkette (der voreingestellte Datentyp)
`text`	unbegrenzt	Text, der in HTML-Seiten genau so erscheint wie der Feldwert (d. h. HTML-spezifische Zeichen wie "<" werden umgewandelt)

7.1.2 Eingabefelder

Zu jedem Feld kann im Content Manager die Art seines Eingabefeldes angegeben werden. Eingabefelder für Felder können folgende Datentypen haben:

Datentyp	Inhalt
`textfield`	Einzeiliges Textfeld für `string`-Felder
`textarea`	Mehrzeiliges Textfeld für `string`-, `html`-, `text`- und `markdown`-Felder
`passwordfield`	Einzeiliges Textfeld mit verdeckter Eingabe für `string`-Felder
`multiselect`	Auswahlliste mit Möglichkeit der Mehrfachauswahl für `multienum`-Felder
`popup`	Aufklappmenü für `enum`-Felder
`radio`	Auswahlschalter für `enum`-Felder
`html`	HTML-Editor *edit-on pro* für `html`-Felder
`checkbox`	Auswahlkästchen für `multienum`-Felder
`hidden`	Verstecktes Eingabefeld (nicht vom Benutzer editierbar) für alle Feldtypen
`external`	Vom Benutzer festgelegte lokale Anwendung für den `body` und `html`-Felder
`linklistfield`	Standarddialog zur Bearbeitung von `linklist`-Feldern
`wizard`	Assistent für alle Felder außer `signature`

7.2 Parameterelemente für Felder

Mit Hilfe von Funktionselementen kann man auf Felder zugreifen. So kann man beispielsweise mit dem `attribute-get`-Element die Werte sämtlicher Feldparameter ermitteln. Um jedoch zu spezifizieren, welche Feldeigenschaften, ausgelesen oder gesetzt werden sollen, verwendet man Parameterelemente. Im Folgenden werden die Parameterelemente für Felder aufgeführt.

`callback`

Bedeutung: Ein Tcl-Script, das aufgerufen wird, nachdem dem Feld ein Wert zugewiesen wurde.

Definition:

```
<!ELEMENT callback (%cm.atom;)>
```

`displayTitle`

Bedeutung: Der in der HTML-Benutzerschnittstelle angezeigte Titel des Feldes (eine Kombination aus Titel und Namen).

Definition:

```
<!ELEMENT displayTitle (%cm.atom;)>
```

`displayValueCallback`

Bedeutung: Tcl-Script, das aufgerufen wird, um den Anzeigewert eines kundenspezifischen Feldes zu berechnen.

Definition:

```
<!ELEMENT displayValueCallback (%cm.atom;)>
```

`editField`

Bedeutung: Definition des bei Wertzuweisung zu dem Feld zu verwendenden Eingabefeldes.

Definition:

```
<!ELEMENT editField (%cm.atom;)>
<!ATTLIST editField
          parameter (length |
                     maxlength |
                     nilAllowed |
                     objClasses |
                     rows |
                     startPub |
```

```
                    type |
                    wizard) CDATA #REQUIRED
>
```

Bedeutung der Attribute:

- `length`: Die angezeigte Länge eines Feldes (als Parameter verfügbar nur bei `textfield`, `passwordfield`, `textarea`).
- `maxlength`: Die maximale Länge des in ein Feld eingebbaren Textes (als Parameter verfügbar nur bei `textfield` und `passwordfield`).
- `nilAllowed`: Gibt bei Eingabefeldern vom Typ `multiselect` und `popup` an, ob neben den Aufzählungswerten des Feldes auch der leere Wert ausgewählt werden kann.
- `objClasses`: wenn `type` gleich `linklistfield` ist, beschränkt diese Liste (durch Leerzeichen separierter Vorlagennamen), die im Dateiauswahldialog auswählbaren Dateien (verfügbar ab Version 6.7.0).
- `rows`: Die Anzahl der darzustellenden Zeilen des Feldes (als Parameter verfügbar nur bei `textarea`, `multiselect`, `popup` bis Version 6.0.x).
- `startPub`: wenn `type` gleich `linklistfield` ist, legt dieser Wert den Pfad des für die Linkzielauswahl zu öffnenden Ordners fest (verfügbar ab Version 6.7.0).
- `type`: Der Typ des Eingabefeldes. Folgende Typen sind verfügbar: `textfield`, `passwordfield`, `textarea`, `multiselect`, `popup`, `radio`, `custom`, `html`, `checkbox`, `hidden`, `external`, `linklistfield` (ab Version 6.5.0), `wizard`.
- `wizard`: wenn `type` gleich `wizard` ist, legt dieser Wert den Namen des Assistenten fest, der zur Erfassung des Feldwertes verwendet werden soll.

Beispiel:

```
<cm-request...>
  <attribute-where>
    <name>fruit</name>
  </attribute-where>
  <attribute-get>
    <editfield parameter="type"/>
  </attribute-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <attribute>
      <editfield parameter="type">multiselect</editfield>
    </attribute>
  </cm-code>
</cm-response>
```

`editFieldSpec`

Bedeutung: Liefert die vollständige Spezifikation des Eingabefeldes - enthält auch Feldnamen und Aufzählungswerte.

Definition:

```
<!ELEMENT editFieldSpec (%cm.atom;
```

```
  | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom;
  | listitem | dictitem)*>
```

`getKeys`

Bedeutung: Liste der mit `attribute-get` abfragbaren Parameter.

Definition:

```
<!ELEMENT getKeys (listitem)*>
<!ELEMENT listitem (%cm.atom;
  | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom;
  | listitem | dictitem)*>
```

`helpText`

Bedeutung: Beschreibung des Feldes.

Definition:

```
<!ELEMENT helpText (%cm.atom;)>
<!ATTLIST helpText
          lang (en | de | it | fr | es) #IMPLIED
>
```

Bedeutung der Attribute:

- `lang`: Kennzeichnet die Sprache einer Feldbeschreibung. Zu Dimensionen von Werten im CMS siehe Dimensionen von Werten.

Beispiel:

```
<cm-request...>
  <attribute-where>
    <name>fruit</name>
  </attribute-where>
  <attribute-get>
    <helpText lang="en"/>
  </attribute-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <attribute-get>
      <helpText lang="en">summer fruit</helpText>
    </attribute-get>
  </cm-code>
</cm-response>
```

isSearchableInCM

Bedeutung: Gibt an, ob die Werte des Feldes bei laufendem Search Engine Server im Content Manager durchsucht werden können.

Definition:

```
<!ELEMENT isSearchableInCM (%cm.atom;)>
```

isSearchableInTE

Bedeutung: Gibt an, ob die Werte des Feldes bei laufendem Search Engine Server auf dem Live-Server durchsucht werden können.

Definition:

```
<!ELEMENT isSearchableInTE (%cm.atom;)>
```

localizedTitle

Bedeutung: Der Titel in der Sprache, die der authentifizierte Benutzer eingestellt hat. Ist dieser leer, wird `title` zurückgegeben. Ist auch dieser Titel leer, wird `name` zurückgegeben.

Definition:

```
<!ELEMENT localizedTitle (%cm.atom;)>
```

localizedHelpText

Bedeutung: Die Feldbeschreibung in der Sprache, die der authentifizierte Benutzer eingestellt hat. Ist diese leer, wird `helpText` zurückgegeben. Ist auch diese leer, so wird nichts zurückgegeben.

Definition:

```
<!ELEMENT localizedHelpText (%cm.atom;)>
```

maxSize

Bedeutung: Bei Feldern vom Typ `linklist` die maximale Anzahl Links, die in der Linkliste enthalten sein darf (ab Version 6.7.0).

Definition:

```
<!ELEMENT maxSize (%cm.atom;)>
```

minSize

Bedeutung: Bei Feldern vom Typ `linklist` die minimale Anzahl Links, die in der Linkliste enthalten sein muss (ab Version 6.7.0).

Definition:

```
<!ELEMENT minSize (%cm.atom;)>
```

name

Bedeutung: Der Name des Feldes.

Definition:

```
<!ELEMENT name (%cm.atom;)>
```

setKeys

Bedeutung: Liste der mit`attribute-set` setzbaren Parameter.

Definition:

```
<!ELEMENT setKeys (listitem)*>
<!ELEMENT listitem (%cm.atom;
 | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom;
 | listitem | dictitem)*>
```

title

Bedeutung: Der Titel des Feldes.

Definition:

```
<!ELEMENT title (%cm.atom;)>
<!ATTLIST title
          lang (en | de | it | fr | es) #IMPLIED
>
```

Bedeutung der Attribute:

- `lang`: Kennzeichnet die Sprache des Feldtitels. Zu Dimensionen von Werten im CMS siehe Dimensionen von Werten.

Beispiel:

```
<cm-request...>
  <attribute-where>
    <name>fruit</name>
  </attribute-where>
  <attribute-get>
    <title lang="en"/>
  </attribute-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <attribute-get>
      <title lang="en">fruits and berries</title>
    </attribute-get>
  </cm-code>
</cm-response>
```

`type`

Bedeutung: Typ des Feldes (zulässig: `string`(voreingestellt), `date`, `enum`, `html`, `multienum`, `signature`, `text`).

Definition:

```
<!ELEMENT type (%cm.atom;)>
```

`validEditFieldKeys`

Bedeutung: Liste der möglichen Parameternamen in der Eingabefelddefinition (abhängig vom Eingabefeldtyp).

Definition:

```
<!ELEMENT validEditFieldKeys (listitem)*>

<!ELEMENT listitem (%cm.atom;
 | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom;
 | listitem | dictitem)*>
```

validEditFieldTypes

Bedeutung: Liste der möglichen Eingabefeldtypen (abhängig vom Feldtyp).

Definition:

```
<!ELEMENT validEditFieldTypes (listitem)*>
<!ELEMENT listitem (%cm.atom;
 | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom;
 | listitem | dictitem)*>
```

values

Bedeutung: Aufzählungswerte (nur bei `enum`- und `multienum`-Feldern).

Definition:

```
<!ELEMENT values (listitem)*>
<!ELEMENT listitem (%cm.atom;
 | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom;
 | listitem | dictitem)*>
```

wantedTags

Bedeutung: Liste der im Feldwert erlaubten HTML-Tags (nur bei Feldern vom Typ `html`).

Definition:

```
<!ELEMENT wantedTags (listitem)*>
<!ELEMENT listitem (%cm.atom;
 | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom;
 | listitem | dictitem)*>
```

7.3 Funktionselemente für Felder

7.3.1 <attribute-create>

Definition:

```
<!ENTITY % cm.attribute-create "
```

```
   (callback |
   helpText |
   isSearchableInCM |
   isSearchableInTE |
   maxSize |
   minSize |
   name |
   title |
   type |
   values |
   wantedTags |
   editField)*
">
<!ELEMENT attribute-create %cm.attribute-create;>
```

Aufgabe: Erzeugt ein neues kundenspezifisches Feld mit den angegebenen Werten - die Runtime-Konfiguration wird neu geschrieben.

Rückgabewert bei Erfolg: der Name des Feldes.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalRTCEdit` haben.

Beispiel:

```
<cm-request...>
  <attribute-create>
    <name>fruit</name>
    <type>multienum</type>
    <values>
      <listitem>apple</listitem>
      <listitem>orange</listitem>
      <listitem>banana</listitem>
    </values>
  </attribute-create>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <attribute>
      <name>fruit</name>
    </attribute>
  </cm-code>
</cm-response>
```

7.3.2 <attribute-getValidEditFieldKeys>

Definition:

```
<!ENTITY % cm.attribute-getValidEditFieldKeys "
  (listitem)*
">
```

```
<!ELEMENT attribute-getValidEditFieldKeys
          %cm.attribute-getValidEditFieldKeys;>
<!ATTLIST attribute-getValidEditFieldKeys
          type CDATA #REQUIRED
>
```

Aufgabe: Liefert die zulässigen Eingabefeldparameter für den angegebenen Eingabefeldtyp.

Rückgabewert bei Erfolg: die Liste der zulässigen Eingabefeldparameter.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <attribute-getValidEditFieldKeys type="popup"/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <attribute-getValidEditFieldKeys type="enum">
      <listitem>rows</listitem>
      <listitem>nilAllowed</listitem>
    </attribute-getValidEditFieldKeys>
  </cm-code>
</cm-response>
```

7.3.3 <attribute-types>

Definition:

```
<!ELEMENT attribute-types %cm.attribute-types;>
```

Aufgabe: attribute-types erzeugt eine Liste der zulässigen Feldtypen.

Rückgabewert bei Erfolg: die Liste der Feldtypen.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <attribute-types/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <attribute-types>
      <listitem>date</listitem>
      <listitem>enum</listitem>
      <listitem>html</listitem>
      <listitem>multienum</listitem>
      <listitem>signature</listitem>
      <listitem>string</listitem>
      <listitem>text</listitem>
    </attribute-types>
  </cm-code>
</cm-response>
```

7.3.4 <attribute-where>

Ausschnitt aus der DTD:

```
<!ENTITY % cm.cm-request "
  ...
```

```
  (attribute-where+, attribute-addEnumValues) |
  (attribute-where+, attribute-delete) |
  (attribute-where+, attribute-get) |
  (attribute-where+, attribute-removeEnumValues) |
  (attribute-where+, attribute-set) |
  ...
">
```

Definition:

```
<!ENTITY % cm.attribute-where "
  (name |
  (nameLike?,
  type?))?
">
<!ELEMENT attribute-where %cm.attribute-where;>
```

Aufgabe: attribute-where ermöglicht die Suche nach den Feldern, bei denen der Wert der angegebenen Parameter den jeweils spezifizierten String enthält. Werden keine Parameter angegeben, so werden die Namen sämtlicher Felder zurückgegeben.

Spezielle Parameter dieses Funktionselements:

- `nameLike`: bewirkt, dass die Namen der Felder zurückgegeben werden, bei denen die angegebene Zeichenkette im Feldnamen enthalten ist.

Verwendung: `attribute-where` ist ein Funktionselement, das nicht ohne ein weiteres unmittelbar darauf folgendes Funktionselement verwendet werden kann. Mit `attribute-where` erhält man die Liste der Feldnamen, die dem Suchkriterium entsprechen. Erst durch das folgende Funktionselement wird jedoch bestimmt, wie diese Feldnamen zu behandeln sind. Jedes der Elemente, die auf `attribute-where` folgen können, entspricht dabei einer Schablone, die nacheinander auf jeden der mit `attribute-where` ermittelten Feldnamen angewendet wird (siehe Requests als Schablonen und die Darstellung der Ergebnis-Daten).

Erforderliche Rechte: keine Einschränkungen.

7.3.5 <attribute-where> <attribute-addEnumValues>

Definition:

```
<!ENTITY % cm.attribute-addEnumValues "
  (listitem+)
">
<!ELEMENT attribute-addEnumValues %cm.attribute-addEnumValues;>
```

Aufgabe: attribute-addEnumValues fügt einem `enum-` oder `multienum`-Attribut die angegebenen Aufzählungswerte hinzu, falls diese nicht bereits existieren.

Verwendung: Zunächst wird mit `attribute-where` das gewünschte Feld ermittelt. Anschließend wird `attribute-addEnumValues` verwendet, um Aufzählungswerte zu dem Feld hinzuzufügen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalRTCEdit` haben.

Beispiel:

```
<cm-request...>
  <attribute-where>
    <name>fruit</name>
  </attribute-where>
  <attribute-addEnumValues>
    <listitem>blackberry</listitem>
    <listitem>blueberry</listitem>
    <listitem>strawberry</listitem>
  </attribute-addEnumValues>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

7.3.6 <attribute-where> <attribute-delete>

Definition:

```
<!ENTITY % cm.attribute-delete "EMPTY">
<!ELEMENT attribute-delete %cm.attribute-delete;>
```

Aufgabe: `attribute-delete` löscht Felder.

Verwendung: Zunächst wird mit `attribute-where` das Attribut ermittelt, das anschließend mit `attribute-delete` gelöscht wird.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalRTCEdit` haben.

Beispiel:

```
<cm-request...>
  <attribute-where>
    <name>old_fruit</name>
  </attribute-where>
  <attribute-delete/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

7.3.7 <attribute-where> <attribute-description>

Definition:

```
<!ENTITY % cm.atom "#PCDATA">
<!ELEMENT attribute-description %cm.atom;>
```

Aufgabe: `attribute-description` liefert eine Zeichenketten-Repräsentation der wichtigsten Feldparameter.

Verwendung: Zunächst wird mit `attribute-where` das Feld ermittelt, dessen Zeichenketten-Repräsentation anschließend mit `attribute-description` ausgelesen wird.

Rückgabewert bei Erfolg: die Zeichenketten-Repräsentation.

Erforderliche Rechte: keine.

Beispiel:

```
<cm-request...>
  <attribute-where>
    <name>datenblatt</name>
  </attribute-where>
  <attribute-description/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok>
    {
      editField = {
        length = 65;
        type = textfield;
      };
      name = datenblatt;
      title = datenblatt;
      type = string;
    }
  </cm-code>
</cm-response>
```

7.3.8 <attribute-where> <attribute-get>

Definition:

```
<!ENTITY % cm.attribute-get "
  (callback |
  displayTitle |
  displayValueCallback |
  editField |
  editFieldSpec |
  getKeys |
  helpText |
  isSearchableInCM |
  isSearchableInTE |
  localizedTitle |
  localizedHelpText |
  maxSize |
  minSize |
  name |
  setKeys |
  title |
  type |
  validEditFieldKeys |
  validEditFieldTypes |
  values |
  wantedTags)*
">
<!ELEMENT attribute-get %cm.attribute-get;>
```

Aufgabe: Liefert Parameterwerte für jedes der mit `attribute-where` ermittelten Felder.

Verwendung: `attribute-get` muss auf `attribute-where` folgen, da zunächst die Felder ermittelt werden müssen, auf die `attribute-get` angewendet werden soll.

Rückgabewert bei Erfolg: die Werte der angegebenen Parameter.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <attribute-where>
    <name>fruit</name>
  </attribute-where>
  <attribute-get>
    <type/>
  </attribute-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <attribute>
      <type>multienum</type>
    </attribute>
  </cm-code>
</cm-response>
```

7.3.9 <attribute-where> <attribute-removeEnumValues>

Definition:

```
<!ENTITY % cm.attribute-removeEnumValues "
  (listitem+)
">
<!ELEMENT attribute-removeEnumValues
          %cm.attribute-removeEnumValues;>
```

Aufgabe: Löscht bei einem `enum`- oder `multienum`-Feld die angegebenen Aufzählungswerte.

Verwendung: Zunächst wird mit `attribute-where` das gewünschte Feld ermittelt. Anschließend wird `attribute-removeEnumValues` verwendet, um Aufzählungswerte des Feldes zu löschen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalRTCEdit` haben.

Beispiel:

```
<cm-request...>
  <attribute-where>
    <name>fruit</name>
  </attribute-where>
  <attribute-removeEnumValues>
    <listitem>banana</listitem>
    <listitem>blueberry</listitem>
  </attribute-removeEnumValues>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
```

```
</cm-response>
```

7.3.10 <attribute-where> <attribute-set>

Definition:

```
<!ENTITY % cm.attribute-set "
  (callback |
  displayValueCallback |
  helpText |
  isSearchableInCM |
  isSearchableInTE |
  maxSize |
  minSize |
  title |
  type |
  values |
  wantedTags |
  editField)*
">
<!ELEMENT attribute-set %cm.attribute-set;>
```

Aufgabe: Setzt Feldparameter auf die angegebenen Werte für jedes mit `attribute-where` ermittelte Feld.

Verwendung: Zunächst wird mit `attribute-where` das Feld ermittelt, auf das anschließend `attribute-set` angewendet wird, um Parameterwerte zu setzen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalRTCEdit` haben.

Beispiel:

```
<cm-request...>
  <attribute-where>
    <name>fruit</name>
  </attribute-where>
  <attribute-set>
    <isSearchableInCM>1</isSearchableInCM>
    <isSearchableInTE>0</isSearchableInTE>
    <title>fruit juice</title>
    <type>enum</type>
  </attribute-set>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

8 Feldergruppen - attributeGroup

8.1 Definition

Der folgende Ausschnitt aus der DTD des Content Management Servers zeigt die Elemente, mit denen es möglich ist, auf Feldergruppen in Requests lesend und schreibend zuzugreifen:

```
<!ENTITY % cm.cm-request "
  ...
  (attributeGroup-create) |
  (attributeGroup-where+, attributeGroup-addAttributes)|
  (attributeGroup-where+, attributeGroup-delete) |
  (attributeGroup-where+, attributeGroup-description) |
  (attributeGroup-where+, attributeGroup-get) |
  (attributeGroup-where+, attributeGroup-moveAttribute) |
  (attributeGroup-where+, attributeGroup-moveToIndex) |
  (attributeGroup-where+, attributeGroup-removeAttributes) |
  (attributeGroup-where+, attributeGroup-set) |
  ...
">
```

Diese Elemente müssen unmittelbar unterhalb des `cm-request`-Elements angegeben werden. Sie werden im Abschnitt Funktionselemente für Feldergruppen erläutert.

8.2 Parameterelemente für Feldergruppen

Mit Hilfe von Funktionselementen kann auf Feldergruppen zugegriffen werden. So kann man beispielsweise mit dem `attributeGroup-get`-Element die Werte sämtlicher Feldergruppen-Parameter ermitteln. Um zu spezifizieren, welche Feldergruppeneigenschaften ausgelesen oder gesetzt werden sollen, verwendet man jedoch Parameterelemente. Es gibt folgende Parameterelemente für Feldergruppen:

`attributes`

Bedeutung: Die Namen der Felder, die die Feldergruppe enthält. Die Namen werden in der Reihenfolge zurückgegeben, in der die Felder der Gruppe zugeordnet sind.

Definition:

```
<!ENTITY % cm.atom "#PCDATA">
<!ELEMENT attributes ((attribute)* | (listitem)* | (dictitem)+)>
<!ELEMENT attribute (%cm.atom; | %cm.attribute-get;)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
```

```
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.attribute-get`: siehe <attribute-where> <attribute-get> oder CRUL als DTD.

displayTitle

Bedeutung: Der in der HTML-Benutzerschnittstelle angezeigte Titel der Feldergruppe (eine Kombination aus Titel und Namen).

Definition:

```
<!ELEMENT displayTitle (%cm.atom;)>
```

getKeys

Bedeutung: Liste der mit `attributeGroup-get` abfragbaren Parameter.

Definition:

```
<!ELEMENT getKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

index

Bedeutung: Der Index der Gruppe in der Feldgruppenliste der Dateivorlage. Die erste Gruppe hat den Index 0.

Definition:

```
<!ELEMENT index (%cm.atom;)>
```

isDefaultGroup

Bedeutung: Liefert `1`, wenn die Gruppe die Basisgruppe ist, ansonsten `0`.

Definition:

```
<!ELEMENT isDefaultGroup (%cm.atom;)>
```

isEmpty

Bedeutung: Liefert `1`, wenn die Gruppe keine Felder enthält, ansonsten `0`.

Definition:

```
<!ELEMENT isEmpty (%cm.atom;)>
```

localizedTitle

Bedeutung: Der lokalisierte Titel in der Sprache, die der authentifizierte Benutzer eingestellt hat. Ist dieser Titel nicht belegt, so wird `title` zurückgegeben, wenn `title` nicht ebenfalls leer ist. Andernfalls wird der Name der Feldergruppe zurückgegeben.

Definition:

```
<!ELEMENT localizedTitle (%cm.atom;)>
```

name

Bedeutung: Der Name der Feldergruppe.

Definition:

```
<!ELEMENT name (%cm.atom;)>
```

objClass

Bedeutung: Der Name der Vorlage, zu der die Feldgruppe gehört.

Definition:

```
<!ELEMENT objClass (%cm.atom; | %cm.objClass-get;)*>
```

`cm.objClass-get`: siehe <objClass-where> <objClass-get> oder CRUL als DTD.

setKeys

Bedeutung: Liste der mit `attributeGroup-set` setzbaren Parameter.

Definition:

```
<!ELEMENT setKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
```

```
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`title`

Bedeutung: Der Titel der Feldergruppe.

Definition:

```
<!ELEMENT title (%cm.atom;)>
<!ATTLIST title
          lang (en | de | it | fr | es) #IMPLIED
>
```

Bedeutung der Attribute:

- `lang`: Kennzeichnet die Sprache eines Feldergruppentitels. Zu Dimensionen von Werten im CMS siehe Dimensionen von Werten.

Beispiel:

```
<cm-request...>
  <attributeGroup-where>
    <name>furniture</name>
  </attributeGroup-where>
  <attributeGroup-get>
    <title lang="en"/>
  </attributeGroup-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <attributeGroup>
      <title lang="en">chairs and tables</title>
    </attributeGroup>
  </cm-code>
</cm-response>
```

8.3 Funktionselemente für Feldergruppen

8.3.1 <attributeGroup-create>

Definition:

```
<!ENTITY % cm.attributeGroup-create "
  (objClass,
  name,
  title?,
  index?)
">
<!ELEMENT attributeGroup-create %cm.attributeGroup-create;>
```

Aufgabe: Legt eine neue Feldergruppe an.

Rückgabewert bei Erfolg: der Name der neuen Gruppe.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalRTCEdit` haben.

Beispiel:

```
<cm-request...>
  <attributeGroup-create>
    <objClass>document</objClass>
    <name>furniture</name>
    <index>1</index>
  </attributeGroup-create>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <attributeGroup>
      <name>furniture</name>
    </attributeGroup>
  </cm-code>
</cm-response>
```

8.3.2 <attributeGroup-where>

Ausschnitt aus der DTD:

```
<!ENTITY % cm.cm-request "
  ...
  (attributeGroup-where+, attributeGroup-addAttributes)|
  (attributeGroup-where+, attributeGroup-delete) |
  (attributeGroup-where+, attributeGroup-get) |
  (attributeGroup-where+, attributeGroup-moveAttribute) |
  (attributeGroup-where+, attributeGroup-moveToIndex) |
  (attributeGroup-where+, attributeGroup-removeAttributes) |
  (attributeGroup-where+, attributeGroup-set) |
  ...
">
```

Definition:

```
<!ENTITY % cm.attributeGroup-where "
  (objClass,
  name?)
">
<!ELEMENT attributeGroup-where %cm.attributeGroup-where;>
```

Aufgabe: `attributeGroup-where` ermöglicht die Suche nach den Feldergruppen, bei denen der Wert der angegebenen Parameter den jeweils spezifizierten String enthält. Die Namen aller Feldergruppen werden zurückgegeben, wenn keine Parameter angegeben werden.

Verwendung: `attributeGroup-where` ist ein Funktionselement, das nicht ohne ein weiteres unmittelbar darauf folgendes Funktionselement verwendet werden kann. Mit `attributeGroup-where` erhält man die Liste der Feldergruppennamen, die dem Suchkriterium entsprechen. Erst durch das folgende Funktionselement wird jedoch bestimmt, wie diese Feldergruppennamen zu behandeln sind. Jedes der Elemente, die auf `attributeGroup-where` folgen können, entspricht

dabei einer Schablone, die nacheinander auf jeden der mit `attributeGroup-where` ermittelten Feldergruppennamen angewendet wird (siehe Requests als Schablonen und die Darstellung der Ergebnis-Daten).

Erforderliche Rechte: keine Einschränkungen.

8.3.3 <attributeGroup-where> <attributeGroup-addAttributes>

Definition:

```
<!ENTITY % cm.attributeGroup-addAttributes "
  (listitem)*
">
<!ELEMENT attributeGroup-addAttributes
          %cm.attributeGroup-addAttributes;>
```

Aufgabe: Mit attributeGroup-addAttributes können mehrere Felder in eine Feldergruppe aufgenommen werden.

Zusatzinformationen: Jedes der aufzunehmenden Felder muss der Vorlage zugewiesen sein, zu der die Feldergruppe gehört. Der Basisgruppe können keine Felder zugewiesen werden. Nur Felder, die sich in der Basisgruppe befinden, können einer anderen Gruppe zugewiesen werden. Die Felder werden in der angegebenen Reihenfolge an die Felderliste der Feldergruppe angehängt.

Spezielle Parameter dieses Funktionselements:

- `attribute`: spezifiziert den Namen eines Feldes, das der Gruppe zugewiesen werden soll.

Verwendung: Zunächst wird mit `attributeGroup-where` die gewünschte Feldergruppe ermittelt. Anschließend wird `attributeGroup-addAttributes` verwendet, um Felder in diese Gruppe aufzunehmen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalRTCEdit` haben.

Beispiel:

```
<cm-request...>
  <attributeGroup-where>
    <objClass>document</objClass>
    <name>furniture</name>
  </attributeGroup-where>
  <attributeGroup-addAttributes>
    <listitem>type</listitem>
    <listitem>price</listitem>
    <listitem>weight</listitem>
  </attributeGroup-addAttributes>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

8.3.4 <attributeGroup-where> <attributeGroup-delete>

Definition:

```
<!ENTITY % cm.attributeGroup-delete "EMPTY">
<!ELEMENT attributeGroup-delete %cm.attributeGroup-delete;>
```

Aufgabe: attributeGroup-delete löscht jede mit attributeGroup-where ermittelte Feldergruppe.

Zusatzinformationen: Die in den zu löschenden Gruppen enthaltenen Felder werden automatisch zur Basisgruppe hinzugefügt. Die Basisgruppe kann nicht gelöscht werden.

Verwendung: Zunächst werden mit `attributeGroup-where` Feldergruppen ermittelt. Anschließend werden diese Feldergruppen mit `attributeGroup-delete` gelöscht.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalRTCEdit` haben.

Beispiel:

```
<cm-request...>
  <attributeGroup-where>
    <objClass>document</objClass>
    <name>old_furniture</name>
  </attributeGroup-where>
  <attributeGroup-delete/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

8.3.5 <attributeGroup-where> <attributeGroup-description>

Definition:

```
<!ENTITY % cm.atom "#PCDATA">
<!ELEMENT attributeGroup-description %cm.atom;>
```

Aufgabe: `attributeGroup-description` liefert eine Zeichenketten-Repräsentation der wichtigsten Feldergruppen-Parameter.

Verwendung: Zunächst wird mit `attributeGroup-where` die Feldergruppe ermittelt, deren Zeichenketten-Repräsentation anschließend mit `attributeGroup-description` ausgelesen wird.

Rückgabewert bei Erfolg: die Zeichenketten-Repräsentation.

Erforderliche Rechte: keine.

Beispiel:

```
<cm-request...>
  <attributeGroup-where>
```

```
    <name>headlines</name>
  </attributeGroup-where>
  <attributeGroup-description/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok>
    {
      name = headlines;
      title = Schlagzeilen;
      title.german = "";
      title.english = "";
      index = 1;
      attributes = (head1, head2);
    }
  </cm-code>
</cm-response>
```

8.3.6 <attributeGroup-where> <attributeGroup-get>

Definition:

```
<!ENTITY % cm.attributeGroup-get "
  (attributes |
  displayTitle |
  getKeys |
  index |
  isDefaultGroup |
  isEmpty |
  localizedTitle |
  name |
  objClass |
  setKeys |
  title)*
">
<!ELEMENT attributeGroup-get %cm.attributeGroup-get;>
```

Aufgabe: Liefert Parameterwerte für jede der mit `attributeGroup-where` ermittelten Feldergruppen.

Verwendung: `attributeGroup-get` folgt immer auf `attributeGroup-where`, da zunächst die Feldergruppen ermittelt werden müssen, auf die `attributeGroup-get` angewendet werden soll.

Rückgabewert bei Erfolg: der Wert der angegebenen Parameter.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <attributeGroup-where>
    <objClass>document</objClass>
    <name>furniture</name>
  </attributeGroup-where>
  <attributeGroup-get>
    <attributes/>
  </attributeGroup-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <attributeGroup>
      <attributes>
```

```
        <listitem>type</listitem>
        <listitem>price</listitem>
        <listitem>weight</listitem>
      </attributes>
    </attributeGroup>
  </cm-code>
</cm-response>
```

8.3.7 <attributeGroup-where> <attributeGroup-moveAttribute>

Definition:

```
<!ENTITY % cm.attributeGroup-moveAttribute "
  (attribute,
  index)
">
<!ELEMENT attributeGroup-moveAttribute
          %cm.attributeGroup-moveAttribute;>
```

Aufgabe: Der Befehl verschiebt ein Attribut einer Feldergruppe an eine andere Stelle in der Liste der Felder dieser Gruppe.

Spezielle Parameter dieses Funktionselements:

- `attribute`: spezifiziert den Namen eines Feldes, das in der Felderliste der Gruppe verschoben werden soll.

Verwendung: Zunächst wird mit `attributeGroup-where` die gewünschte Feldergruppe ermittelt. Anschließend wird `attributeGroup-moveAttribute` verwendet, um ein Feld an eine andere Stelle dieser Feldergruppe zu verschieben.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalRTCEdit` haben.

Beispiel:

```
<cm-request...>
  <attributeGroup-where>
    <objClass>document</objClass>
    <name>furniture</name>
  </attributeGroup-where>
  <attributeGroup-moveAttribute>
    <attribute>price</attribute>
    <index>1</index>
  </attributeGroup-moveAttribute>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

8.3.8 <attributeGroup-where> <attributeGroup-moveToIndex>

Definition:

```
<!ENTITY % cm.attributeGroup-moveToIndex "
  (index)
">
<!ELEMENT attributeGroup-moveToIndex
          %cm.attributeGroup-moveToIndex;>
```

Aufgabe: `attributeGroup-moveToIndex` verschiebt eine Feldergruppe an eine andere Stelle in der Feldergruppenliste einer Vorlage.

Verwendung: Zunächst wird mit `attributeGroup-where` die gewünschte Feldergruppe ermittelt. Anschließend wird `attributeGroup-moveToIndex` verwendet, um die Feldergruppe an eine andere Stelle der Feldergruppenliste der Vorlage zu verschieben.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalRTCEdit` haben.

Beispiel:

```
<cm-request...>
  <attributeGroup-where>
    <objClass>document</objClass>
    <name>furniture</name>
  </attributeGroup-where>
  <attributeGroup-moveToIndex>
    <index>2</index>
  </attributeGroup-moveToIndex>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

8.3.9 <attributeGroup-where> <attributeGroup-removeAttributes>

Definition:

```
<!ENTITY % cm.attributeGroup-removeAttributes "
  (listitem+)
">
<!ELEMENT attributeGroup-removeAttributes
          %cm.attributeGroup-removeAttributes;>
```

Aufgabe: attributeGroup-removeAttributes löscht Felder aus einer Feldergruppe.

Zusatzinformationen: Aus der Basisgruppe können keine Felder gelöscht werden. Felder, die aus Feldergruppen gelöscht werden, werden automatisch zur Basisgruppe hinzugefügt.

Verwendung: Zunächst wird mit `attributeGroup-where` die gewünschte Feldergruppe ermittelt. Anschließend wird `attributeGroup-removeAttributes` verwendet, um Felder aus dieser Feldergruppe zu löschen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalRTCEdit` haben.

Beispiel:

```
<cm-request...>
  <attributeGroup-where>
    <objClass>document</objClass>
    <name>furniture</name>
  </attributeGroup-where>
  <attributeGroup-removeAttributes>
    <listitem>price</listitem>
    <listitem>weight</listitem>
  </attributeGroup-removeAttributes>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

8.3.10 <attributeGroup-where> <attributeGroup-set>

Definition:

```
<!ENTITY % cm.attributeGroup-set "
  (title)*
">
<!ELEMENT attributeGroup-set %cm.attributeGroup-set;>
```

Aufgabe: Setzt Parameter einer Feldergruppe auf die angegebenen Werte.

Verwendung: `attributeGroup-set` muss immer auf `attributeGroup-where` folgen, da zunächst die Feldergruppen ermittelt werden müssen, auf die `attributeGroup-set` angewendet werden soll.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalRTCEdit` haben.

Beispiel:

```
<cm-request...>
  <attributeGroup-where>
    <objClass>document</objClass>
    <name>furniture</name>
  </attributeGroup-where>
  <attributeGroup-set>
    <title>to buy furniture</title>
  </attributeGroup-set>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

9

9 Benutzer - user

Mit den im Folgenden aufgeführten Parameter- und Funktionselementen kann man auf die Daten zugreifen, die der in den Content Manager integrierte Benutzermanager verwaltet. Wenn Sie einen anderen, beispielsweise einen auf LDAP basierenden Benutzermanager einsetzen, so können die `user`-Elemente nicht sinnvoll verwendet werden. Wie Sie auf die mit einem externen Benutzermanager verwalteten Daten mit Requests an den CMS Content Management Server zugegreifen können, wird im Abschnitt Benutzermanager-Proxy - userProxy und groupProxy beschrieben.

9.1 Definition

Der folgende Ausschnitt aus der DTD zeigt die Elemente, mit denen in Requests an den Content Management Server der lesende und schreibende Zugriff auf die Daten für Benutzer möglich ist:

```
<!ENTITY % cm.cm-request "
  ...
  (user-create) |
  (user-where+, user-addToGroups) |
  (user-where+, user-delete) |
  (user-where+, user-description) |
  (user-where+, user-get) |
  (user-where+, user-grantGlobalPermissions) |
  (user-where+, user-removeFromGroups) |
  (user-where+, user-revokeGlobalPermissions) |
  (user-where+, user-set) |
  ...
">
<!ATTLIST user-where
  maxResults CDATA #IMPLIED
>
```

Diese Elemente - sie müssen sich unmittelbar unterhalb des `cm-request`-Elements befinden - werden im Abschnitt Funktionselemente für Benutzer erläutert.

Mit dem Attribut `maxResults`, dessen Wert eine ganze Zahl ist, kann die maximale Anzahl der Treffer festgelegt werden. Ist der Wert von `maxResults` kleiner oder gleich null, ist die Anzahl nicht begrenzt.

9.1.1 Rechte

In XML-Requests an den Content Management Server und in seinen Responses werden grundsätzlich die folgenden Bezeichner für globale Rechte verwendet.

Globale Rechte	Bedeutung
`permissionGlobalRoot`	Administrationsrecht
`permissionGlobalUserEdit`	Benutzer und Gruppen anlegen und bearbeiten
`permissionGlobalUserAttributeEdit`	Benutzerattribute anlegen und bearbeiten
`permissionGlobalRTCEdit`	Felder, Workflows und Vorlagen bearbeiten
`permissionGlobalExport`	Das Recht, Dateien zu exportieren

Globale Rechte können jedoch auch im Systemkonfigurationseintrag `content.globalPermissions` frei definiert werden. Dies bedeutet, dass Sie globale Rechte zum Content Management Server hinzufügen können, indem Sie die Liste `globalPermissions` erweitern.

9.2 Parameterelemente für Benutzer

Mit Funktionselementen kann man auf die Daten der Benutzer zugreifen. Mit dem `user-get`-Element können beispielsweise die Werte sämtlicher Benutzer-Parameter ermittelt werden. Möchte man spezifizieren, welche Benutzereigenschaften ausgelesen oder gesetzt werden sollen, so werden Parameterelemente verwendet. Im Folgenden werden die Benutzer-Parameter und die zugehörigen Parameterelemente aufgeführt.

`defaultGroup`

Bedeutung: Standardgruppe des Benutzers.

Definition:

```
<!ELEMENT defaultGroup (%cm.atom; | %cm.group-get;)*>
```

`cm.group-get`: siehe <group-where> <group-get> oder CRUL als DTD.

`displayTitle`

Bedeutung: Der in der HTML-Benutzerschnittstelle angezeigte Bezeichner des Benutzers (eine Kombination aus Login und vollem Namen).

Definition:

```
<!ELEMENT displayTitle (%cm.atom;)>
```

`encryptedPassword`

Bedeutung: Verschlüsseltes Passwort des Benutzers.

Definition:

```
```

```
<!ELEMENT encryptedPassword (%cm.atom;)>
```

email

Bedeutung: E-Mail-Adresse des Benutzers.

Definition:

```
<!ELEMENT email (%cm.atom;)>
```

externalUserAttrNames

Bedeutung: Liste der zusätzlichen Benutzerfelder des Benutzers.

Definition:

```
<!ELEMENT externalUserAttrNames ((userAttribute)* | (listitem)*)>
<!ELEMENT userAttribute (%cm.userAttribute-get;)>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.userAttribute-get`: siehe <userAttribute-where> <userAttribute-get> oder CRUL als DTD.

getKeys

Bedeutung: Liste der mit `user-get` abfragbaren Parameter.

Definition:

```
<!ELEMENT getKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

globalPermissions

Bedeutung: Liste der globalen Rechte.

Definition:

```
<!ELEMENT globalPermissions (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

hasGlobalPermission

Bedeutung: Überprüft, ob der Benutzer das spezifizierte globale Recht hat.

Definition:

```
<!ELEMENT hasGlobalPermission (%cm.atom;)>
<!ATTLIST hasGlobalPermission
          permission (permissionGlobalExport |
                      permissionGlobalRoot |
                      permissionGlobalRTCEdit |
                      permissionGlobalUserAttributeEdit |
                      permissionGlobalUserEdit) CDATA #REQUIRED
>
```

Bedeutung der Attribute:

- `permission`: spezifiziert ein globales Recht.

Beispiel:

```
<cm-request...>
  <user-where>
    <login>stan</login>
  </user-where>
  <user-get>
    <hasGlobalPermission permission="permissionGlobalRTCEdit"/>
  </user-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <user>
      <hasGlobalPermission permission="permissionGlobalRTCEdit">1
      </hasGlobalPermission>
    </user>
  </cm-code>
</cm-response>
```

groups

Bedeutung: Liste der Gruppen, in denen der Benutzer Mitglied ist.

Definition:

```
<!ELEMENT groups ((group)* | (listitem)*)>
<!ELEMENT group (%cm.atom; | %cm.group-get;)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.group-get`: siehe <group-where> <group-get> oder CRUL als DTD.

hasPassword

Bedeutung: Prüft, ob der Benutzer das angegebene Passwort hat.

Definition:

```
<!ELEMENT hasPassword (%cm.atom;)>
<!ATTLIST hasPassword
          password CDATA #REQUIRED
>
```

Bedeutung der Attribute:

- `password`: spezifiziert das Passwort (Klartext).

Beispiel:

```
<cm-request...>
  <user-where>
    <login>stan</login>
  </user-where>
  <user-get>
    <hasPassword password="xyz123"/>
  </user-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <user>
      <hasPassword password="xyz123">0</hasPassword>
    </user>
  </cm-code>
</cm-response>
```

isSuperUser

Bedeutung: Zeigt an, ob der Benutzer ein Superuser ist (`1`= ist Superuser; `0` = ist kein Superuser).

Definition:

```
<!ELEMENT isSuperUser (%cm.atom;)>
```

isOwnerOf

Bedeutung: Zeigt an, ob der Benutzer der Verwalter eines anderen Benutzers oder einer Benutzgruppe ist (`1`= ist Verwalter; `0`= ist nicht der Verwalter).

Definition:

```
<!ELEMENT isOwnerOf (%cm.atom;)>
<!ATTLIST isOwnerOf
          login CDATA #REQUIRED
>
```

Bedeutung der Attribute:

- `login`: spezifiziert einen Benutzer oder eine Benutzergruppe.

Beispiel:

```
<cm-request...>
  <user-where>
    <login>lisa</login>
  </user-where>
  <user-get>
    <isOwnerOf login="NewsEditors"/>
  </user-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <user>
      <isOwnerOf login="NewsEditors">1</isOwnerOf>
    </user>
  </cm-code>
</cm-response>
```

login

Bedeutung: Login des Benutzers.

Definition:

```
<!ELEMENT login (%cm.atom;)>
```

owner

Bedeutung: Der Verwalter des Benutzers.
Wenn ein Benutzer angelegt wird, wird der eingeloggte Benutzer automatisch zum direkten Verwalter (`owner`) des neuen Benutzers. Der Verwalter hat das Recht, Felder und Rechte des neuen Benutzers zu ändern. Das Login des direkten Verwalters ist im Parameter `owner` des Benutzers eingetragen.
Der Verwalter eines Benutzers hat auch die Möglichkeit, einen anderen Wert in den Parameter `owner` des Benutzers einzutragen. Dieser andere Wert kann entweder ein Login oder ein Gruppenname sein. Mit der Eintragung eines neuen Verwalters tritt der vorherige Verwalter seine Rechte an einen anderen Benutzer oder an eine Gruppe ab.
Ist im Parameter `owner` eines Benutzers eine Gruppe eingetragen, so gelten alle Mitglieder der Gruppe als indirekte Verwalter.
Um die Parameter eines Benutzers ändern zu können, muss man der direkte oder der indirekte Verwalter des Benutzers sein und das Recht `permissionGlobalUserEdit` haben. Ein Benutzer kann also seine Daten nicht selbst ändern, es sei denn, er ist sein eigener direkter oder indirekter Verwalter.
Die Wirkungsweise des Parameters `owner` gilt analog auch für Gruppen.

Definition:

```
<!ELEMENT owner (%cm.atom;)>
```

password

Bedeutung: Das Klartext-Passwort des Benutzers.

Definition:

```
<!ELEMENT password (%cm.atom;)>
```

realName

Bedeutung: Voller Name des Benutzers.

Definition:

```
<!ELEMENT realName (%cm.atom;)>
```

setKeys

Bedeutung: Liste der mit `user-set` setzbaren Parameter.

Definition:

```
<!ELEMENT setKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

userLocked

Bedeutung: Gibt an, ob der Benutzer gesperrt ist (`1`= ist gesperrt; `0`= ist nicht gesperrt).

Definition:

```
<!ELEMENT userLocked (%cm.atom;)>
```

9.3 Funktionselemente für Benutzer

9.3.1 <user-create>

Definition:

```
<!ENTITY % cm.user-create "
  (defaultGroup |
  encryptedPassword |
  email |
  globalPermissions |
  login |
  owner |
  password |
  realName |
  userLocked)*
">
<!ELEMENT user-create %cm.user-create;>
```

Aufgabe: `user-create` erzeugt einen neuen Benutzer mit den angegebenen Parameterwerten.

Zusatzinformationen: Wird ein Benutzer angelegt, muss der Parameter `login` gesetzt werden. Das Login kann nicht nachträglich geändert werden.
Der `owner` des Benutzers wird auf den authentifizierten Benutzer gesetzt, falls für `owner` nicht beim Aufruf von `user-create` ein Parameterwert angegeben ist.

Rückgabewert bei Erfolg: das Login des neuen Benutzers.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalUserEdit` haben. Er muss ferner der Verwalter der Standardgruppe des neuen Benutzers sein.

Beispiel:

```
<cm-request...>
  <user-create>
    <login>john</login>
    <realName>John Smith</realName>
    <email>smithj@up.com</email>
    <defaultGroup>NewsEditors</defaultGroup>
    <owner>NewsMasters</owner>
  </user-create>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <user>
      <login>john</login>
    </user>
  </cm-code>
</cm-response>
```

9.3.2 <user-where>

Ausschnitt aus der DTD:

```
<!ENTITY % cm.cm-request "
  ...
  (user-where+, user-addToGroups) |
```

```
  (user-where+, user-delete) |
  (user-where+, user-description) |
  (user-where+, user-get) |
  (user-where+, user-grantGlobalPermissions) |
  (user-where+, user-removeFromGroups) |
  (user-where+, user-revokeGlobalPermissions) |
  (user-where+, user-set) |
  ...
">
```

Definition:

```
<!ENTITY % cm.user-where "
  (login |
  userText)
">
<!ELEMENT user-where %cm.user-where;>
```

Aufgabe: `user-where` ermöglicht die Suche nach Benutzern, bei denen der Wert der angegebenen Parameter den jeweils spezifizierten String enthält. Werden keine Parameter angegeben, so werden die Logins aller Benutzer zurückgegeben.

Spezielle Parameter dieses Funktionselements:

- `userText`: bewirkt, dass die Logins der Benutzer zurückgegeben werden, bei denen die angegebene Zeichenkette im Login oder im vollständigen Namen des Benutzers enthalten ist.

Verwendung: `user-where` ist ein Funktionselement, das nicht ohne ein weiteres unmittelbar darauf folgendes Funktionselement verwendet werden kann. Mit `user-where` erhält man die Liste der Logins der Benutzer, die dem Suchkriterium entsprechen. Erst durch das folgende Funktionselement wird jedoch bestimmt, wie diese Benutzer zu behandeln sind. Jedes der Elemente, die auf `user-where` folgen können, entspricht dabei einer Schablone, die nacheinander auf jeden der mit `user-where` ermittelten Benutzer angewendet wird (siehe Requests als Schablonen und die Darstellung der Ergebnis-Daten).

Rückgabewert bei Erfolg: die Liste der Logins der Benutzer.

Erforderliche Rechte: keine Einschränkungen.

9.3.3 <user-where> <user-addToGroups>

Definition:

```
<!ENTITY % cm.user-addToGroups "
  (listitem+)
">
<!ELEMENT user-addToGroups %cm.user-addToGroups;>
```

Aufgabe: Die mit `user-where` ermittelten Benutzer werden in jede der angegebenen Gruppen aufgenommen, falls sie nicht schon Mitglied sind.

Zusatzinformationen: Durch diese Aktion wird nicht der Benutzer verändert, sondern die angegebenen Gruppen. Daher sind zur Ausführung des Kommandos die Rechte zur Modifikation der angegebenen Gruppen notwendig.

Verwendung: Zunächst werden mit `user-where` Benutzer ermittelt, bevor `user-addToGroups` verwendet wird, um diese Benutzer zu Gruppen zuzuordnen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss Superuser sein, oder er muss das Recht `permissionGlobalUserEdit` haben und direkter oder indirekter Verwalter (`owner`) jeder Gruppe sein, in die Benutzer aufgenommen werden sollen.

Beispiel:

```
<cm-request...>
  <user-where>
    <login>stan</login>
  </user-where>
  <user-addToGroups>
    <listitem>admins</listitem>
    <listitem>NewsMasters</listitem>
    <listitem>NewsEditors</listitem>
  <user-addToGroups>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

9.3.4 <user-where> <user-delete>

Definition:

```
<!ENTITY % cm.user-delete "EMPTY">
<!ELEMENT user-delete %cm.user-delete;>
```

Aufgabe: Mit user-delete werden Benutzer gelöscht.

Verwendung: Zunächst werden mit `user-where` Benutzer ermittelt. Anschließend werden diese Benutzer mit `user-delete` gelöscht.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss Superuser sein, oder er muss das Recht `permissionGlobalUserEdit` haben und direkter oder indirekter `owner` eines jeden mit `user-where` ermittelten Benutzers sein.

Beispiel:

```
<cm-request...>
  <user-where>
    <login>ida</login>
  </user-where>
  <user-delete/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

9.3.5 <user-where> <user-description>

Definition:

```
<!ENTITY % cm.atom "#PCDATA">
<!ELEMENT user-description %cm.atom;>
```

Aufgabe: `user-description` liefert eine Zeichenketten-Repräsentation der wichtigsten Benutzer-Parameter.

Verwendung: Zunächst wird mit `user-where` der Benutzer ermittelt, dessen Zeichenketten-Repräsentation anschließend mit `user-description` ausgelesen wird.

Rückgabewert bei Erfolg: die Zeichenketten-Repräsentation.

Erforderliche Rechte: keine.

Beispiel:

```
<cm-request...>
  <user-where>
    <login>katrin</login>
  </user-where>
  <user-description/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok>
    {
      defaultGroup = admins;
      globalPermissions = (
      );
      groups = (
        admins
      );
      login = katrin;
      owner = root;
      realName = "Katrin\ Landsberg";
      userLocked = 0;
    }
  </cm-code>
</cm-response>
```

9.3.6 <user-where> <user-get>

Definition:

```
<!ENTITY % cm.user-get "
  (defaultGroup |
  displayTitle |
  encryptedPassword |
  email |
  externalUserAttrNames |
  getKeys |
  globalPermissions |
  groups |
  hasGlobalPermission |
  isSuperUser |
  isOwnerOf |
  hasPassword |
```

```
  login |
  owner |
  realName |
  setKeys |
  userLocked)*
">
<!ELEMENT user-get %cm.user-get;>
```

Aufgabe: Liefert den Wert der angegebenen Parameter für jeden mit `user-where` ermittelten Benutzer.

Verwendung: Zunächst werden mit `user-where` Benutzer ermittelt. Anschließend wird `user-get` verwendet, um Parameterwerte dieser Benutzer auszulesen.

Rückgabewert bei Erfolg: die Werte der angegebenen Parameter.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <user-where>
    <login>mara</login>
  </user-where>
  <user-get>
    <defaultGroup/>
    <email/>
    <groups/>
    <realName/>
  </user-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <user>
      <defaultGroup>NewsMasters</defaultGroup>
      <email>maramc@up.com</email>
      <groups>
        <listitem>admins</listitem>
        <listitem>NewsMasters</listitem>
      </groups>
      <realName>Mara McLeod</realName>
    </user>
  </cm-code>
</cm-response>
```

9.3.7 <user-where> <user-grantGlobalPermissions>

Definition:

```
    <!ENTITY % cm.user-grantGlobalPermissions "
  (listitem+)
">
<!ELEMENT user-grantGlobalPermissions
          %cm.user-grantGlobalPermissions;>
```

Aufgabe: Erteilt den mit user-where ermittelten Benutzern die angegebenen Rechte.

Verwendung: Zunächst werden mit `user-where` Benutzer ermittelt. Anschließend wird `user-grantGlobalPermissions` verwendet, um diesen Benutzern die angegebenen globalen Rechte zu erteilen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss Superuser sein, oder er muss das Recht `permissionGlobalUserEdit` haben und direkter oder indirekter `owner` eines jeden mit `user-where` ermittelten Benutzers sein.

Beispiel:

```
<cm-request...>
  <user-where>
    <login>john</login>
  </user-where>
  <user-grantGlobalPermissions>
    <listitem>permissionGlobalRTCEdit</listitem>
    <listitem>permissionGlobalExport</listitem>
  </user-grantGlobalPermissions>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

9.3.8 <user-where> <user-removeFromGroups>

Definition:

```
<!ENTITY % cm.user-removeFromGroups "
  (listitem+)
">
<!ELEMENT user-removeFromGroups %cm.user-removeFromGroups;>
```

Aufgabe: Entfernt jeden mit `user-where` ermittelten Benutzer aus den spezifizierten Gruppen.

Zusatzinformationen: Ist eine der angegebenen Gruppen die Standardgruppe eines Benutzers, so kann der Benutzer nicht aus dieser Gruppe entfernt werden.

Verwendung: Zunächst werden mit `user-where` Benutzer ermittelt. Anschließend wird user-removeFromGroups verwendet, um diese Benutzer aus den spezifizierten Gruppen zu entfernen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss Superuser sein, oder er muss das Recht `permissionGlobalUserEdit` haben und direkter oder indirekter `owner` jeder Gruppe sein, aus der Benutzer entfernt werden soll.

Beispiel:

```
<cm-request...>
  <user-where>
    <userText>jo</userText>
  </user-where>
  <user-removeFromGroups>
    <listitem>admins</listitem>
    <listitem>NewsMasters</listitem>
```

```
  </user-removeFromGroups>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

9.3.9 <user-where> <user-revokeGlobalPermissions>

Definition:

```
<!ENTITY % cm.user-revokeGlobalPermissions "
  (listitem+)
">
<!ELEMENT user-revokeGlobalPermissions
          %cm.user-revokeGlobalPermissions;>
```

Aufgabe: Entzieht jedem mit `user-where` ermittelten Benutzer die angegebenen Rechte.

Verwendung: Zunächst werden mit `user-where` Benutzer ermittelt. Anschließend wird `user-revokeGlobalPermissions` verwendet, um diesen Benutzern die angegebenen globalen Rechte zu entziehen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss Superuser sein, oder er muss das Recht `permissionGlobalUserEdit` haben und direkter oder indirekter `owner` eines jeden mit `user-where` ermittelten Benutzers sein.

Beispiel:

```
<cm-request...>
  <user-where>
    <userText>sta</userText>
  </user-where>
  <user-revokeGlobalPermissions>
    <listitem>permissionGlobalRTCEdit</listitem>
  </user-revokeGlobalPermissions>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

9.3.10 <user-where> <user-set>

Definition:

```
<!ENTITY % cm.user-set "
  (defaultGroup |
  encryptedPassword |
  email |
  globalPermissions |
  owner |
  password |
  realName |
  userLocked)*
```

```
">
<!ELEMENT user-set %cm.user-set;>
```

Aufgabe: Setzt Parameter auf die angegebenen Werte für jeden der mit `user-where` ermittelten Benutzer.

Zusatzinformationen: Wird `owner` als leeres Element angegeben, also auf den Leerstring gesetzt, so wird der authentifizierte Benutzer als `owner` eingesetzt.

Verwendung: `user-set` muss immer auf `user-where` folgen, da zunächst die Benutzer ermittelt werden müssen, auf die `user-set` angewendet werden soll, um Parameterwerte zu setzen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss Superuser sein, oder er muss das Recht `permissionGlobalUserEdit` haben und direkter oder indirekter `owner` eines jeden mit `user-where` ermittelten Benutzers sein.

Beispiel:

```
<cm-request...>
  <user-where>
    <login>lisa</login>
  <user-where>
  <user-set>
    <defaultGroup>NewsMasters</defaultGroup>
    <email>lisam@up.com</email>
    <globalPermissions>
      <listitem>permissionGlobalRTCEdit</listitem>
      <listitem>permissionGlobalExport</listitem>
    </globalPermissions>
    <owner/>
    <realName>Lisa Monroe</realName>
  </user-set>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

10

10 Benutzerfelder - userAttribute

Mit den im Folgenden aufgeführten Parameter- und Funktionselementen kann man auf die Daten zugreifen, die der in den Content Manager integrierte Benutzermanager verwaltet. Wenn Sie einen anderen, beispielsweise einen auf LDAP basierenden Benutzermanager einsetzen, so können die `userAttribute`-Elemente nicht sinnvoll verwendet werden. Wie Sie auf die mit einem externen Benutzermanager verwalteten Daten mit Requests an den Content Management Server zugegreifen können, wird im Abschnitt Benutzermanager-Proxy - userProxy und groupProxy beschrieben.

10.1 Definition

Der folgende Ausschnitt aus der DTD des Content Management Servers zeigt die Elemente, mit denen in Requests schreibend und lesend auf Benutzerfelder zugegriffen werden kann:

```
<!ENTITY % cm.cm-request "
  ...
  (userAttribute-create) |
  (userAttribute-types) |
  (userAttribute-where+, userAttribute-addEnumValues) |
  (userAttribute-where+, userAttribute-delete) |
  (userAttribute-where+, userAttribute-description) |
  (userAttribute-where+, userAttribute-get) |
  (userAttribute-where+, userAttribute-removeEnumValues) |
  (userAttribute-where+, userAttribute-set) |
  ...
">
<!ATTLIST userAttribute-where
  maxResults CDATA #IMPLIED
>
```

Diese Elemente müssen sich unmittelbar unterhalb des `cm-request`-Elements befinden. Sie werden im Abschnitt Funktionselemente für Benutzerfelder erläutert.

Mit dem Attribut `maxResults`, dessen Wert eine ganze Zahl ist, kann die maximale Anzahl der Treffer festgelegt werden. Ist der Wert von `maxResults` kleiner oder gleich null, ist die Anzahl nicht begrenzt

10.1.1 Feldtypen

Benutzerfelder können im Content Management Server verschiedene Datentypen haben. Je nach Datentyp sind die Feldwerte größenbeschränkt. In der folgenden Tabelle sind die Datentypen, ihre Bedeutung sowie die maximale Größe eines Feldwerts für den jeweiligen Typ aufgeführt:

Datentyp	Größe (Bytes)	Inhalt

`date`	14	Datum
`enum`	250	Aufzählungswerte
`multienum`	250	Aufzählungswerte mit Mehrfachauswahl
`string`	max. 250	Zeichenkette (der voreingestellte Datentyp)

10.1.2 Eingabefelder

Eingabefelder für Benutzerfelder können folgende Datentypen haben:

Datentyp	Inhalt
`textfield`	Einzeiliges Textfeld
`textarea`	Mehrzeiliges Textfeld
`passwordfield`	Einzeiliges Textfeld mit verdeckter Eingabe
`multiselect`	Auswahlliste mit Möglichkeit der Mehrfachauswahl
`popup`	Aufklappmenü
`radio`	Ankreuzbares Feld

10.2 Parameterelemente für Benutzerfelder

Auf Benutzerfelder kann mit Hilfe von Funktionselementen zugegriffen werden. So kann man beispielsweise mit dem `userAttribute-get`-Element die Werte sämtlicher Benutzerfeld-Parameter ermitteln. Um zu spezifizieren, welche Eigenschaften von Benutzerfeldern ausgelesen oder gesetzt werden sollen, verwendet man Parameterelemente. Es gibt folgende Parameterelemente für Benutzerfelder:

`displayTitle`

Bedeutung: Der in der HTML-Benutzerschnittstelle angezeigte `displayTitle` des Benutzerfeldes - bei Benutzerfeldern wird der Name angezeigt.

Definition:

```
<!ELEMENT displayTitle (%cm.atom;)>
```

`getKeys`

Bedeutung: Liste der mit `userAttribute-get` abfragbaren Parameter.

Definition:

```
<!ELEMENT getKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

name

Bedeutung: Name des Benutzerfeldes.

Definition:

```
<!ELEMENT name (%cm.atom;)>
```

setKeys

Bedeutung: Liste der mit `userAttribute-set` setzbaren Parameter.

Definition:

```
<!ELEMENT setKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

type

Bedeutung: Typ des Benutzerfeldes (zulässig: `string` [default], `date`, `enum`, `multienum`).

Definition:

```
<!ELEMENT type (%cm.atom;)>
```

values

Bedeutung: Aufzählungswerte bei `enum-` und `multienum`-Benutzerfeldern.

Definition:

```
<!ELEMENT values (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

10.3 Funktionselemente für Benutzerfelder

10.3.1 <userAttribute-create>

Definition:

```
<!ENTITY % cm.userAttribute-create "
  (name |
  type)*
">
<!ELEMENT userAttribute-create %cm.userAttribute-create;>
```

Aufgabe: userAttribute-create erzeugt ein neues Benutzerfeld mit den angegebenen Parameterwerten.

Rückgabewert bei Erfolg: der Name des Benutzerfeldes.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalUserAttributeEdit` haben.

Beispiel:

```
<cm-request...>
  <userAttribute-create>
    <name>address</name>
    <type>string</type>
  </userAttribute-create>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <userAttribute>
      <name>address</name>
    </userAttribute>
  </cm-code>
</cm-response>
```

10.3.2 <userAttribute-types>

Definition:

```
<!ENTITY % cm.userAttribute-types "
  (listitem)*
">
<!ELEMENT userAttribute-types %cm.userAttribute-types;>
```

Aufgabe: Gibt die Liste der zulässigen Benutzerfeldtypen aus.

Rückgabewert bei Erfolg: die Liste der Benutzerfeldtypen.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <userAttribute-types/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <userAttribute-types>
      <listitem>date</listitem>
      <listitem>enum</listitem>
      <listitem>multienum</listitem>
      <listitem>string</listitem>
    </userAttribute-types>
  </cm-code>
</cm-response>
```

10.3.3 <userAttribute-where>

Ausschnitt aus der DTD:

```
<!ENTITY % cm.cm-request "
  ...
  (userAttribute-where+, userAttribute-addEnumValues) |
  (userAttribute-where+, userAttribute-delete) |
  (userAttribute-where+, userAttribute-description) |
  (userAttribute-where+, userAttribute-get) |
  (userAttribute-where+, userAttribute-removeEnumValues) |
  (userAttribute-where+, userAttribute-set) |
  ...
">
```

Definition:

```
<!ENTITY % cm.userAttribute-where "
  (name |
  (nameLike?,
  type?))?
">
<!ELEMENT userAttribute-where %cm.userAttribute-where;>
```

Aufgabe: `userAttribute-where` ermöglicht die Suche nach Benutzerfeldern, bei denen der Wert der angegebenen Parameter den jeweils spezifizierten String enthält. Werden keine Parameter angegeben, so werden die Namen sämtlicher Benutzerfelder zurückgegeben.

Spezielle Parameter dieses Funktionselements:

- `nameLike`: bewirkt, dass die Namen der Benutzerfelder zurückgegeben werden, bei denen die angegebene Zeichenkette im Namen enthalten ist.

Verwendung: `userAttribute-where` ist ein Funktionselement, das nicht ohne ein weiteres unmittelbar darauf folgendes Funktionselement verwendet werden kann. Mit `userAttribute-where` erhält man die Liste der Namen der Benutzerfelder, die dem Suchkriterium entsprechen. Erst durch das folgende Funktionselement wird jedoch bestimmt, wie diese Benutzerfelder zu behandeln sind. Jedes der Elemente, die auf `userAttribute-where` folgen können, entspricht dabei einer Schablone, die nacheinander auf jedes der mit `userAttribute-where` ermittelten Benutzerfelder angewendet wird (siehe Requests als Schablonen und die Darstellung der Ergebnis-Daten).

Rückgabewert bei Erfolg: die Liste der Namen der Benutzerfelder.

Erforderliche Rechte: keine Einschränkungen.

10.3.4 <userAttribute-where> <userAttribute-addEnumValues>

Definition:

```
<!ENTITY % cm.userAttribute-addEnumValues "
  (listitem+)
">
<!ELEMENT userAttribute-addEnumValues
          %cm.userAttribute-addEnumValues;>
```

Aufgabe: Zu einem mit `userAttribute-where` ermittelten `enum-` oder `multienum`-Benutzerfeld werden die angegebenen Aufzählungswerte hinzugefügt, falls diese nicht bereits existieren.

Verwendung: Zunächst wird das Benutzerfeld mit `userAttribute-where` ermittelt. Anschließend wird `userAttribute-addEnumValues` verwendet, um diesem Benutzerfeld die angegebenen Aufzählungswerte hinzuzufügen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalUserAttributeEdit` haben.

Beispiel:

```
<cm-request...>
  <userAttribute-where>
    <name>department</name>
  </userAttribute-where>
  <userAttribute-addEnumValues>
    <listitem>administration</listitem>
    <listitem>development</listitem>
    <listitem>marketing</listitem>
    <listitem>sales</listitem>
    <listitem>services</listitem>
  <userAttribute-addEnumValues>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

10.3.5 <userAttribute-where> <userAttribute-delete>

Definition:

```
<!ENTITY % cm.userAttribute-delete "EMPTY">
<!ELEMENT userAttribute-delete %cm.userAttribute-delete;>
```

Aufgabe: Mit `userAttribute-where` ermittelte Benutzerfelder werden gelöscht.

Verwendung: Zunächst werden mit `userAttribute-where` Benutzerfelder ermittelt, um diese anschließend mit `userAttribute-delete` zu löschen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalUserAttributeEdit` haben.

Beispiel:

```
<cm-request...>
  <userAttribute-where>
    <name>phone</name>
  </userAttribute-where>
  <userAttribute-delete/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

10.3.6 <userAttribute-where> <userAttribute-description>

Definition:

```
<!ENTITY % cm.atom "#PCDATA">
<!ELEMENT userAttribute-description %cm.atom;>
```

Aufgabe: `userAttribute-description` liefert eine Zeichenketten-Repräsentation der wichtigsten Benutzerfeld-Parameter.

Verwendung: Zunächst wird mit `userAttribute-where` das Benutzerfeld ermittelt, dessen Zeichenketten-Repräsentation anschließend mit `userAttribute-description` ausgelesen wird.

Rückgabewert bei Erfolg: die Zeichenketten-Repräsentation.

Erforderliche Rechte: keine.

Beispiel:

```
<cm-request...>
  <userAttribute-where>
    <name>department</name>
  </userAttribute-where>
  <userAttribute-description/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok>
    {
      name = department;
      type = multienum;
      values = (
        administration,
        development,
        marketing,
        services
      );
    }
  </cm-code>
</cm-response>
```

10.3.7 <userAttribute-where> <userAttribute-get>

Definition:

```
<!ENTITY % cm.userAttribute-get "
  (displayTitle |
  getKeys |
  name |
  setKeys |
  type |
  values)*
">
<!ELEMENT userAttribute-get %cm.userAttribute-get;>
```

Aufgabe: Liefert Parameterwerte für jedes mit `userAttribute-where` ermittelte Benutzerfeld.

Verwendung: Zunächst werden mit `userAttribute-where` Benutzerfelder ermittelt. Anschließend wird auf diese Benutzerfelder `userAttribute-get` angewendet, um Parameterwerte auszulesen.

Rückgabewert bei Erfolg: die Werte der angegebenen Benutzerfeldparameter.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <userAttribute-where>
    <name>department</name>
  </userAttribute-where>
  <userAttribute-get>
    <type/>
    <values/>
  </userAttribute-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <userAttribute>
      <type>multienum</type>
      <values>
        <listitem>administration</listitem>
        <listitem>development</listitem>
        <listitem>marketing</listitem>
        <listitem>sales</listitem>
        <listitem>services</listitem>
      </values>
    </userAttribute>
  </cm-code>
</cm-response>
```

10.3.8 <userAttribute-where> <userAttribute-removeEnumValues>

Definition:

```
<!ENTITY % cm.userAttribute-removeEnumValues "
  (listitem+)
">
<!ELEMENT userAttribute-removeEnumValues
          %cm.userAttribute-removeEnumValues;>
```

Aufgabe: Löscht bei einem mit `userAttribute-where` ermittelten `enum`- oder `multienum`-Feld die angegebenen Aufzählungswerte.

Verwendung: Zunächst wird mit `userAttribute-where` das gewünschte Benutzerfeld ermittelt. Anschließend wird `userAttribute-removeEnumValues` verwendet, um die angegebenen Werte des Benutzerfeldes zu löschen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalUserAttributeEdit` haben.

Beispiel:

```
<cm-request...>
  <userAttribute-where>
    <name>department</name>
  </userAttribute-where>
  <userAttribute-removeEnumValues>
    <listitem>administration</listitem>
    <listitem>sales</listitem>
  </userAttribute-removeEnumValues>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

10.3.9 <userAttribute-where> <userAttribute-set>

Definition:

```
<!ENTITY % cm.userAttribute-set "
  (type |
  values)*
">
<!ELEMENT userAttribute-set %cm.userAttribute-set;>
```

Aufgabe: Setzt Parameter für jedes der mit userAttribute-where ermittelten Benutzerfelder auf die angegebenen Werte.

Verwendung: `userAttribute-set` muss immer auf `userAttribute-where` folgen, da zunächst die Benutzerfelder ermittelt werden müssen, auf die `userAttribute-set` angewendet werden soll, um Parameterwerte zu setzen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalUserAttributeEdit` haben.

Beispiel:

```
<cm-request...>
  <userAttribute-where>
    <name>phone</name>
  </userAttribute-where>
  <userAttribute-set>
    <type>enum</type>
```

```
  </userAttribute-set>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

11

11 Benutzergruppen - group

Mit den im Folgenden aufgeführten Parameter- und Funktionselementen kann man auf die Daten zugreifen, die der in den Content Manager integrierte Benutzermanager verwaltet. Wenn Sie einen anderen, beispielsweise einen auf LDAP basierenden Benutzermanager einsetzen, so können die `group`-Elemente nicht sinnvoll verwendet werden. Wie Sie auf die mit einem externen Benutzermanager verwalteten Daten mit Requests an den Content Management Server zugegreifen können, wird im Abschnitt Benutzermanager-Proxy - userProxy und groupProxy beschrieben.

11.1 Definition

Der folgende Ausschnitt aus der DTD des Content Management Servers zeigt die Elemente, mit denen in Requests lesend und schreibend auf Benutzergruppen zugegriffen werden kann:

```
<!ENTITY % cm.cm-request "
  ...
  (group-create) |
  (group-where+, group-addUsers) |
  (group-where+, group-delete) |
  (group-where+, group-get) |
  (group-where+, group-grantGlobalPermissions) |
  (group-where+, group-removeUsers) |
  (group-where+, group-revokeGlobalPermissions) |
  (group-where+, group-set) |
  ...
">
<!ATTLIST group-where
  maxResults CDATA #IMPLIED
>
```

Diese Elemente, die sich unmittelbar unterhalb des `cm-request`-Elements befinden müssen, werden im Abschnitt Funktionselemente für Benutzergruppen erläutert.

Mit dem Attribut `maxResults`, dessen Wert eine ganze Zahl ist, kann die maximale Anzahl der Treffer festgelegt werden. Ist der Wert von `maxResults` kleiner oder gleich null, ist die Anzahl nicht begrenzt.

11.1.1 Rechte

In XML-Requests an den Content Management Server können Sie grundsätzlich die folgenden Bezeichner für globale Rechte verwenden.

Globale Rechte	Bedeutung

`permissionGlobalRoot`	Administrationsrecht
`permissionGlobalUserEdit`	Benutzer und Gruppen anlegen und bearbeiten
`permissionGlobalUserAttributeEdit`	Benutzerattribute anlegen und bearbeiten
`permissionGlobalRTCEdit`	Felder, Workflows und Vorlagen bearbeiten
`permissionGlobalExport`	Das Recht, Dateien zu exportieren

Globale Rechte können im Systemkonfigurationseintrag `content.globalPermissions` frei definiert werden. Dies bedeutet, dass Sie globale Rechte zum Content Management Server hinzufügen können, indem Sie die Liste `globalPermissions` erweitern. Auch solche benutzerdefinierten globalen Rechte können in Requests verwendet werden.

11.2 Parameterelemente für Benutzergruppen

Auf Benutzergruppen kann mit Hilfe von Funktionselementen zugegriffen werden. Mit dem `group-get`-Element kann man beispielsweise die Werte sämtlicher Benutzergruppen-Parameter ermitteln. Welche Benutzergruppeneigenschaften ausgelesen oder gesetzt werden sollen, spezifiziert man mit Parameterelementen. Im Folgenden werden die Parameterelemente für Benutzergruppen aufgeführt.

`displayTitle`

Bedeutung: Der in der HTML-Benutzerschnittstelle angezeigte Bezeichner der Gruppe. Dieser Titel ist eine Kombination aus dem Namen und dem vollen Namen.

Definition:

```
<!ELEMENT displayTitle (%cm.atom;)>
```

`getKeys`

Bedeutung: Die Liste der mit `group-get` abfragbaren Parameter.

Definition:

```
<!ELEMENT getKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`globalPermissions`

Bedeutung: Die Liste der globalen Rechte.

Definition:

```
<!ELEMENT globalPermissions (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

name

Bedeutung: Der Name der Gruppe.

Definition:

```
<!ELEMENT name (%cm.atom;)>
```

hasGlobalPermission

Bedeutung: Überprüft, ob die Gruppe das angegebene globale Recht hat.

Definition:

```
<!ELEMENT hasGlobalPermission (%cm.atom;)>
<!ATTLIST hasGlobalPermission
          permission (permissionGlobalExport |
                      permissionGlobalRoot |
                      permissionGlobalRTCEdit |
                      permissionGlobalUserAttributesEdit |
                      permissionGlobalUserEdit) CDATA #REQUIRED
>
```

Bedeutung der Attribute:

- `permission`: spezifiziert ein globales Recht.

Beispiel:

```
<cm-request...>
  <group-where>
    <name>NewsMasters</name>
  </group-where>
  <group-get>
    <hasGlobalPermission permission="permissionGlobalRTCEdit"/>
  </group-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <group>
      <hasGlobalPermission permission="permissionGlobalRTCEdit">0
      </hasGlobalPermission>
    </group>
  </cm-code>
</cm-response>
```

owner

Bedeutung: Der Verwalter der Benutzergruppe. Im Parameter `owner` einer Gruppe ist der Name des Verwalters der Gruppe (ein Benutzer- oder Gruppenname) gespeichert. Der Verwalter darf die Parameter und Rechte einer Gruppe ändern.

Definition:

```
<!ELEMENT owner (%cm.atom; | %cm.user-get;)*>
```

`cm.user-get`: siehe <user-where> <user-get> oder CRUL als DTD.

realName

Bedeutung: Der volle Name der Gruppe.

Definition:

```
<!ELEMENT realName (%cm.atom;)>
```

setKeys

Bedeutung: Die Liste der mit `group-set` setzbaren Parameter.

Definition:

```
<!ELEMENT setKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

users

Bedeutung: Die Liste der Benutzer, die Mitglieder der Gruppe sind.

Definition:

```
<!ELEMENT users ((user)* | (listitem)*)>
<!ELEMENT user (%cm.atom; | %cm.user-get;)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.user-get`: siehe <user-where> <user-get> oder CRUL als DTD.

11.3 Funktionselemente für Benutzergruppen

11.3.1 <group-create>

Definition:

```
<!ENTITY % cm.group-create "
  (globalPermissions |
  name |
  owner |
  realName)*
">
<!ELEMENT group-create %cm.group-create;>
```

Aufgabe: `group-create` erzeugt eine neuen Gruppe mit den angegebenen Parameterwerten.

Zusatzinformationen: Wird die Gruppe erzeugt, so wird der `owner` auf den authentifizierten Benutzer gesetzt, falls für `owner` beim Aufruf von `group-create` kein Parameterwert angegeben wurde. Der Name der Gruppe kann nachträglich nicht geändert werden.

Rückgabewert bei Erfolg: der Name der neuen Gruppe.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalUserEdit` haben.

Beispiel:

```
<cm-request...>
  <group-create>
    <name>NewsEditors</name>
    <owner>NewsMasters</owner>
  </group-create>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <group>
      <name>NewsEditors</name>
    </group>
  </cm-code>
</cm-response>
```

11.3.2 <group-where>

Ausschnitt aus der DTD:

```
<!ENTITY % cm.cm-request "
  ...
  (group-where+, group-addUsers) |
  (group-where+, group-delete) |
  (group-where+, group-description) |
  (group-where+, group-get) |
  (group-where+, group-grantGlobalPermissions) |
  (group-where+, group-removeUsers) |
  (group-where+, group-revokeGlobalPermissions) |
  (group-where+, group-set) |
  ...
">
```

Definition:

```
<!ENTITY % cm.group-where "
  (name |
  groupText)?
">
<!ELEMENT group-where %cm.group-where;>
```

Aufgabe: group-where ermöglicht die Suche nach den Namen der Benutzergruppen, bei denen der Wert der angegebenen Parameter den jeweils spezifizierten String enthält. Werden keine Parameterwerte angegeben, so werden die Namen aller vorhandenen Benutzergruppen zurückgegeben.

Spezielle Parameter dieses Funktionselements:

- `groupText`: bewirkt, dass die Namen der Benutzergruppen zurückgegeben werden, bei denen die angegebene Zeichenkette in `name` oder `realName` enthalten ist.

Verwendung: `group-where` ist ein Funktionselement, das nicht ohne ein weiteres unmittelbar darauf folgendes Funktionselement verwendet werden kann. Mit `group-where` erhält man die Liste der Gruppennamen, die dem Suchkriterium entsprechen. Erst durch das folgende Funktionselement wird jedoch bestimmt, wie die ermittelten Benutzergruppen zu behandeln sind. Jedes der Elemente, die auf `group-where` folgen können, entspricht dabei einer Schablone, die nacheinander auf jede der mit `group-where` ermittelten Gruppen angewendet wird (siehe Requests als Schablonen und die Darstellung der Ergebnis-Daten).

Erforderliche Rechte: keine Einschränkungen.

11.3.3 <group-where> <group-addUsers>

Definition:

```
<!ENTITY % cm.group-addUsers "
  (listitem+)
">
<!ELEMENT group-addUsers %cm.group-addUsers;>
```

Aufgabe: Mit `group-addUsers` werden Benutzer in eine Benutzergruppe aufgenommen.

Verwendung: Zunächst werden mit `group-where` Gruppen ermittelt. Anschließend wird `group-addUsers` verwendet, um Benutzer in diese Benutzergruppen aufzunehmen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss Superuser sein, oder er muss das Recht `permissionGlobalUserEdit` haben und direkter oder indirekter `owner` der mit `group-where` ermittelten Gruppen sein.

Beispiel:

```
<cm-request...>
  <group-where>
    <name>NewsEditors</name>
  </group-where>
  <group-addUsers>
    <listitem>ina</listitem>
```

```
    <listitem>lisa</listitem>
    <listitem>stan</listitem>
  </group-addUsers>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

11.3.4 <group-where> <group-delete>

Definition:

```
<!ENTITY % cm.group-delete "EMPTY">
<!ELEMENT group-delete %cm.group-delete;>
```

Aufgabe: Löscht jede mit `group-where` ermittelte Benutzergruppe.

Zusatzinformationen: Eine Gruppe kann solange nicht gelöscht werden, wie ihr Benutzer zugeordnet sind.

Verwendung: Zunächst werden mit `group-where` Benutzergruppen ermittelt. Anschließend werden diese Benutzergruppen mit `group-delete` gelöscht.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authetifizierte Benutzer muss Superuser sein, oder er muss das Recht `permissionGlobalUserEdit` haben und direkter oder indirekter `owner` der mit `group-where` ermittelten Gruppen sein.

Beispiel:

```
<cm-request...>
  <group-where>
    <groupText>Editors</groupText>
  </group-where>
  <group-delete/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

11.3.5 <group-where> <group-description>

Definition:

```
<!ENTITY % cm.atom "#PCDATA">
<!ELEMENT group-description %cm.atom;>
```

Aufgabe: `group-description` liefert eine Zeichenketten-Repräsentation der wichtigsten Benutzergruppen-Parameter.

Verwendung: Zunächst wird mit `group-where` die Benutzergruppe ermittelt, deren Zeichenketten-Repräsentation anschließend mit `group-description` ausgelesen wird.

Rückgabewert bei Erfolg: die Zeichenketten-Repräsentation.

Erforderliche Rechte: keine.

Beispiel:

```
<cm-request...>
  <group-where>
    <name>editors</name>
  </group-where>
  <group-description/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok>
    {
      globalPermissions = (
      );
      name = editors;
      owner = root;
      realName = Editoren;
      users = (
        katrin,
        volker
      );
    }
  </cm-code>
</cm-response>
```

11.3.6 <group-where> <group-get>

Definition:

```
<!ENTITY % cm.group-get "
  (displayTitle |
  getKeys |
  globalPermissions |
  hasGlobalPermission |
  users |
  name |
  owner |
  realName |
  setKeys)*
">
<!ELEMENT group-get %cm.group-get;>
```

Aufgabe: Liefert den Wert der angegebenen Parameter für jede mit `group-where` ermittelte Benutzergruppe.

Verwendung: `group-get` muss immer auf `group-where` folgen, da zunächst die Gruppen ermittelt werden müssen, auf die `group-get` angewendet wird, um Parameterwerte auszulesen.

Rückgabewert bei Erfolg: der Wert der angegebenen Parameter.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <group-where>
    <name>NewsEditors</name>
  </group-where>
  <group-get>
    <owner/>
  </group-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <group>
      <owner>NewsMasters</owner>
    </group>
  </cm-code>
</cm-response>
```

11.3.7 <group-where> <group-grantGlobalPermissions>

Definition:

```
<!ENTITY % cm.group-grantGlobalPermissions "
  (listitem+)
">
<!ELEMENT group-grantGlobalPermissions
          %cm.group-grantGlobalPermissions;>
```

Aufgabe: group-grantGlobalPermissions erteilt jeder mit `group-where` ermittelten Gruppe die angegebenen globalen Rechte.

Verwendung: Zunächst werden mit `group-where` die Benutzergruppen ermittelt, auf die anschließend `group-grantGlobalPermissions` angewendet wird, um ihnen die angegebenen Rechte zu erteilen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss Superuser sein, oder er muss das Recht `permissionGlobalUserEdit` haben und direkter oder indirekter `owner` der mit `group-where` ermittelten Gruppen sein.

Beispiel:

```
<cm-request...>
  <group-where>
    <name>NewsEditors</name>
  </group-where>
  <group-grantGlobalPermissions>
    <listitem>permissionGlobalRTCEdit</listitem>
    <listitem>permissionGlobalExport</listitem>
  </group-grantGlobalPermissions>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

11.3.8 <group-where> <group-removeUsers>

Definition:

```
<!ENTITY % cm.group-removeUsers "
  (listitem+)
">
<!ELEMENT group-removeUsers %cm.group-removeUsers;>
```

Aufgabe: Entfernt die angegebenen Benutzer aus den mit `group-where` ermittelten Gruppen.

Zusatzinformationen: Ist die Gruppe die Standardgruppe eines der angegebenen Benutzer, so kann der Benutzer nicht aus dieser Gruppe entfernt werden.

Verwendung: Zunächst werden mit `group-where` Benutzergruppen ermittelt. Anschließend wird `group-removeUsers` verwendet, um die angegebenen Benutzer aus diesen Gruppen auszutragen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss Superuser sein, oder er muss das Recht `permissionGlobalUserEdit` haben und direkter oder indirekter `owner` der mit `group-where` ermittelten Gruppen sein.

Beispiel:

```
<cm-request...>
  <group-where>
    <name>NewsEditors</name>
  </group-where>
  <group-removeUsers>
    <listitem>ina</listitem>
    <listitem>steve</listitem>
  </group-removeUsers>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

11.3.9 <group-where> <group-revokeGlobalPermissions>

Definition:

```
<!ENTITY % cm.group-revokeGlobalPermissions "
  (listitem+)
">
<!ELEMENT group-revokeGlobalPermissions
          %cm.group-revokeGlobalPermissions;>
```

Aufgabe: Entzieht den mit `group-where` ermittelten Gruppen die angegebenen globalen Rechte.

Verwendung: Zunächst werden mit `group-where` Benutzergruppen ermittelt. Anschließend wird `group-revokeGlobalPermissions` verwendet, um diesen Gruppen globale Rechte zu entziehen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss Superuser sein, oder er muss das Recht `permissionGlobalUserEdit` haben und direkter oder indirekter `owner` der mit `group-where` ermittelten Gruppen sein.

Beispiel:

```
<cm-request...>
  <group-where>
    <name>NewsEditors</name>
  </group-where>
  <group-revokeGlobalPermissions>
    <listitem>permissionGlobalRTCEdit</listitem>
    <listitem>permissionGlobalUserEdit</listitem>
  </group-revokeGlobalPermissions>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

11.3.10 <group-where> <group-set>

Definition:

```
<!ENTITY % cm.group-set "
  (globalPermissions |
  owner |
  realName)*
">
<!ELEMENT group-set %cm.group-set;>
```

Aufgabe: Setzt die spezifizierten Parameter einer Benutzergruppe auf die angegebenen Werte.

Verwendung: `group-set` muss immer auf `group-where` folgen, da zunächst die Benutzergruppen ermittelt werden müssen, auf die `group-set` angewendet wird, um Parameterwerte zu setzen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss Superuser sein, oder er muss das Recht `permissionGlobalUserEdit` haben und direkter oder indirekter `owner` der mit `group-where` ermittelten Gruppen sein.

Beispiel:

```
<cm-request...>
  <group-where>
    <name>NewsEditors</name>
  </group-where>
  <group-set>
    <globalPermissions>
      <listitem>permissionGlobalRTCEdit</listitem>
      <listitem>permissionGlobalUserEdit</listitem>
    </globalPermissions>
    <owner>NewsMasters</owner>
    <realName>Editors</realName>
  </group-set>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
```

```
</cm-response>
```

12

12 Benutzerkonfiguration - userConfig

12.1 Definition

Auf die Einstellungen des authentifizierten Benutzers kann über die XML-Schnittstelle des Content Management Servers lesend und schreibend zugegriffen werden. Hat eine Benutzereinstellung keinen Wert, so wird stattdessen der voreingestellte Wert aus der Systemkonfiguration (`userManagement.preferences`) geliefert. Dies gilt auch bei lesendem Zugriff auf Werte, die keine Benutzereinstellungen sind (also reine Systemkonfigurationswerte). Die benutzerspezifischen Werte können mit Hilfe der `userConfig-set...`-Elemente geändert werden. Diese Änderungen werden dauerhaft gespeichert.

Der folgende Ausschnitt aus der DTD des Content Managers zeigt die Elemente, mit denen der authentifizierte Benutzer in Requests schreibend und lesend auf seine persönlichen Einstellungen des Content Management Servers zugreifen kann:

```
<!ENTITY % cm.cm-request "
  ...
  (userConfig-getAll) |
  (userConfig-formatDate) |
  (userConfig-formatDateTime) |
  (userConfig-getAttributes) |
  (userConfig-getCounts) |
  (userConfig-getElements) |
  (userConfig-getKeys) |
  (userConfig-getTexts) |
  (userConfig-parseInputDate) |
  (userConfig-removeAttributes) |
  (userConfig-removeKeys) |
  (userConfig-setAttributes) |
  (userConfig-setElements) |
  (userConfig-setTexts) |
  ...
">
```

Diese Elemente müssen sich unmittelbar unterhalb des `cm-request`-Elements befinden. Sie werden im folgenden Abschnitt erläutert.

12.2 Funktionselemente für die Benutzerkonfiguration

12.2.1 <userConfig-formatDate>

Definition:

```
<!ELEMENT userConfig-formatDate (%cm.atom;)>
<!ATTLIST userConfig-formatDate
  login CDATA #IMPLIED
>
```

Aufgabe: Formatiert das angegebene Datum im Standard-Datumsformat. Das Datum wird in die Standardzeitzone des Servers konvertiert.

Bedeutung der Attribute:

- `login`: das Login eines Benutzers.

Zusatzinformationen: Als Zeitzone des Servers wird der benutzerspezifische Konfigurationswert `timeZone` verwendet. Als Ausgabeformat wird der Wert aus dem Dictionary `validDateOutputFormats` verwendet, dessen Name im benutzerspezifischen Konfigurationswert `dateOutputFormatName` steht.

Rückgabewert bei Erfolg: das formatierte Datum.

Erforderliche Rechte: Möchten Sie den benutzerspezifischen Konfigurationswert eines beliebigen Benutzers verwenden, so müssen Sie ein Superuser oder der Verwalter des betreffenden Benutzers sein. Soll der Konfigurationswert des authentifizierten Benutzers verwendet werden, so gibt es keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <userConfig-formatDate>20001215</userConfig-formatDate>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <userConfig-formatDate>15.12.2000</userConfig-formatDate>
  </cm-code>
</cm-response>
```

12.2.2 <userConfig-formatDateTime>

Definition:

```
<!ELEMENT userConfig-formatDateTime (%cm.atom;)>
<!ATTLIST userConfig-formatDateTime
  login CDATA #IMPLIED
>
```

Aufgabe: Formatiert das angegebene Datum im Standard-Zeitstempelformat. Datum und Uhrzeit werden in die Standardzeitzone des Servers konvertiert.

Bedeutung der Attribute:

- `login`: das Login eines Benutzers.

Zusatzinformationen: Als Zeitzone des Servers wird der benutzerspezifische Konfigurationswert `timeZone` verwendet. Als Ausgabeformat wird der Wert aus dem Dictionary `validDateOutputFormats` verwendet, dessen Name im benutzerspezifischen Konfigurationswert `dateOutputFormatName` steht.

Rückgabewert bei Erfolg: der formatierte Zeitstempel.

Erforderliche Rechte: Möchten Sie den benutzerspezifischen Konfigurationswert eines beliebigen Benutzers verwenden, so müssen Sie ein Superuser oder der Verwalter des betreffenden Benutzers sein. Soll der Konfigurationswert des authentifizierten Benutzers verwendet werden, so gibt es keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <userConfig-formatDateTime>20001215180001
  </userConfig-formatDateTime>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <userConfig-formatDateTime>15.12.2000 19:00 MET
    </userConfig-formatDateTime>
  </cm-code>
</cm-response>
```

12.2.3 <userConfig-getAll>

Definition:

```
<!ENTITY % cm.userConfig-getAll "
  (%cm.atom;)
">
<!ELEMENT userConfig-getAll %cm.userConfig-getAll;>
<!ATTLIST userConfig-getAll
  login CDATA #IMPLIED
>
```

Aufgabe: Gibt alle Schlüssel für die persönlichen Einstellungen eines Benutzers mit den dazu gehörenden Werten an.

Bedeutung der Attribute:

- `login`: das Login eines Benutzers.

Rückgabewert bei Erfolg: die Benutzerkonfiguration des authentifizierten Benutzers wird als Zeichenkette zurückgegeben. In dieser Zeichenkette sind die XML-spezifischen Zeichen wie '<' in die entsprechenden XML-Zeichen-Referenzen umgewandelt.

Erforderliche Rechte: Möchten Sie die Benutzerkonfiguration eines beliebigen Benutzers abfragen, so müssen Sie ein Superuser oder der Verwalter des betreffenden Benutzers sein. Soll die Konfiguration des authentifizierten Benutzers ermittelt werden, so gibt es keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <userConfig-getAll/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <userConfig-getAll>
      &lt;?xml version=&quot;1.0&quot;?&gt;
```

```
      &lt;configuration&gt;
        &lt;attributesToShow type=&quot;list&quot;&gt;
          &lt;attributeToShow&gt;name&lt;/attributeToShow&gt;
        &lt;/attributesToShow&gt;
        &lt;dateTimeInputFormatName&gt;dayMonthYear&lt;/dateTimeInputFormatName&gt;
        &lt;dateTimeOutputFormatName&gt;europeanShortDateTime&lt;/
dateTimeOutputFormatName&gt;
        &lt;hierarchyObjTypes type=&quot;list&quot;&gt;
          &lt;hierarchyObjType&gt;publication&lt;/hierarchyObjType&gt;
        &lt;/hierarchyObjTypes&gt;
        &lt;language&gt;europeanDate&lt;/language&gt;
        &lt;listDisplayRows&gt;20&lt;/listDisplayRows&gt;
        &lt;listDisplayRowsForSearch&gt;30&lt;/listDisplayRowsForSearch&gt;
        &lt;maxHierarchyDepth&gt;5&lt;/maxHierarchyDepth&gt;
        &lt;maxHierarchyLines&gt;100&lt;/maxHierarchyLines&gt;
        &lt;textAreaHeight&gt;15&lt;/textAreaHeight&gt;
        &lt;textAreaWidth&gt;65&lt;/textAreaWidth&gt;
        &lt;textFieldWidth&gt;65&lt;/textFieldWidth&gt;
        &lt;timeZone&gt;MET&lt;/timeZone&gt;
        &lt;wantActionIcons&gt;YES&lt;/wantActionIcons&gt;
        &lt;wantActionPopups&gt;YES&lt;/wantActionPopups&gt;
        &lt;wantIds&gt;YES&lt;/wantIds&gt;
        &lt;wantTextEditButton&gt;YES&lt;/wantTextEditButton&gt;
        &lt;wantTextMode&gt;NO&lt;/wantTextMode&gt;
      &lt;/configuration&gt;
    </userConfig-getAll>
  </cm-code>
</cm-response>
```

12.2.4 <userConfig-getAttributes>

Definition:

```
<!ENTITY % cm.userConfig-getAttributes "
  (key,
  attributeNames?)
">
<!ELEMENT userConfig-getAttributes
          %cm.userConfig-getAttributes;>
<!ATTLIST userConfig-getAttributes
  login CDATA #IMPLIED
>
```

Aufgabe: Liefert zum angegebenen Benutzerkonfigurationselement die spezifizierten Tag-Attribute und deren Werte.

Bedeutung der Attribute:

- `login`: das Login eines Benutzers.

Spezielle Parameter dieses Funktionselements:

- `key`: enthält den Namen des Konfigurationselements, dessen Tag-Attribute gesucht werden. Werden keine Tag-Attribute angegeben, so werden alle Tag-Attribute, die es für dieses Konfigurationselement gibt, mit ihren Werten geliefert.
- `attributeNames`: enthält den Namen eines Tag-Attributs, dessen Wert geliefert werden soll.

Rückgabewert bei Erfolg: die Liste der Tag-Attribute mit ihren Werten.

Erforderliche Rechte: Möchten Sie die Tag-Attribute und deren Werte eines Benutzerkonfigurationselements eines beliebigen Benutzers abfragen, so müssen Sie ein Superuser

oder der Verwalter des betreffenden Benutzers sein. Es gibt es keine Einschränkungen, wenn die Tag-Attribute von Konfigurationselementen des authentifizierten Benutzers ermittelt werden.

Beispiel:

```
<cm-request...>
  <userConfig-getAttributes>
    <key>dateTimeFormats</key>
  </userConfig-getAttributes>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <userConfig-getAttributes>
      <key>dateTimeFormats</key>
      <attributeNames>
        <listitem>countryKey</listitem>
        <listitem>timeZone</listitem>
      </attributeNames>
    </userConfig-getAttributes>
  </cm-code>
</cm-response>
```

12.2.5 <userConfig-getCounts>

Definition:

```
<!ENTITY % cm.userConfig-getCounts "
  (listitem+)
">
<!ELEMENT userConfig-getCounts %cm.userConfig-getCounts;>
<!ATTLIST userConfig-getCounts
  login CDATA #IMPLIED
>
```

Aufgabe: Liefert die Anzahl der Elemente, die unter dem spezifizierten Konfigurationselement gespeichert sind.

Bedeutung der Attribute:

- `login`: das Login eines Benutzers.

Rückgabewert bei Erfolg: eine Liste mit ebenso vielen Zahlen wie Elemente angegeben wurden. Jede Zahl bezeichnet die Anzahl Unterelemente des entsprechenden Konfigurationselements.

Erforderliche Rechte: Um die Anzahl der Elemente des Konfigurationselements eines beliebigen Benutzers zu ermitteln, müssen Sie ein Superuser oder der Verwalter des betreffenden Benutzers sein. Es gibt keine Einschränkungen, wenn die Anzahl der Elemente des Konfigurationselements des authentifizierten Benutzers ermittelt werden soll.

Beispiel:

```
<cm-request...>
  <userConfig-getCounts>
    <listitem>hierarchyObjTypes</listitem>
    <listitem>language</listitem>
  </userConfig-getCounts>
</cm-request>
```

```
<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <userConfig-getCounts>
      <listitem>1</listitem>
      <listitem>0</listitem>
    </userConfig-getCounts>
  </cm-code>
</cm-response>
```

12.2.6 <userConfig-getElements>

Definition:

```
<!ENTITY % cm.userConfig-getElements "
  (listitem+)
">
<!ELEMENT userConfig-getElements %cm.userConfig-getElements;>
<!ATTLIST userConfig-getElements
  login CDATA #IMPLIED
>
```

Aufgabe: Liefert die Werte der angegebenen Konfigurationselemente oder deren Unterelemente und deren Werte.

Bedeutung der Attribute:

- `login`: das Login eines Benutzers.

Rückgabewert bei Erfolg: die Liste der angegebenen Elemente. Für jedes angegebene Element werden die Werte oder die Namen der Unterelemente und deren Werte als Zeichenkette zurückgegeben. In dieser Zeichenkette sind die XML-spezifischen Zeichen wie '<' in die entsprechenden XML-Zeichen-Referenzen umgewandelt.

Erforderliche Rechte: Um die Werte oder Unterelemente der angegebenen Konfigurationselemente eines beliebigen Benutzers zu ermitteln, müssen Sie ein Superuser oder der Verwalter des betreffenden Benutzers sein. Sollen die Werte oder Unterelemente des authentifizierten Benutzers ermittelt werden, so gibt es keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <userConfig-getElements>
    <listitem>hierarchyObjTypes</listitem>
    <listitem>maxHierarchyLines</listitem>
  </userConfig-getElements>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <userConfig-getElements>
      <listitem>
        &lt;?xml version=&quot;1.0&quot;?&gt;
        &lt;hierarchyObjTypestype=&quot;list&quot;&gt
          ;&lt;hierarchyObjType&gt;publication&lt;/hierarchyObjType&gt;
        &lt;/hierarchyObjTypes&gt;
      </listitem>
      <listitem>
        &lt;?xml version=&quot;1.0&quot;?&gt;
        &lt;maxHierarchyLines&gt;100&lt;/maxHierarchyLines&gt;
      </listitem>
    </userConfig-getElements>
```

```
  </cm-code>
</cm-response>
```

12.2.7 <userConfig-getKeys>

Definition:

```
<!ENTITY % cm.userConfig-getKeys "
  (listitem+)
">
<!ELEMENT userConfig-getKeys %cm.userConfig-getKeys;>
<!ATTLIST userConfig-getKeys
  login CDATA #IMPLIED
>
```

Aufgabe: Liefert die Namen der Unterelemente der spezifizierten Benutzerkonfigurationselemente.

Bedeutung der Attribute:

- `login`: das Login eines Benutzers.

Rückgabewert bei Erfolg: Eine Liste, die für jedes angegebene Element die Liste der Namen der Unterelemente enthält.

Erforderliche Rechte: Wollen Sie die Namen der Unterelemente der angegebenen Konfigurationselemente eines beliebigen Benutzers ermitteln, so müssen Sie ein Superuser oder der Verwalter des betreffenden Benutzers sein. Es gibt jedoch keine Einschränkungen, wenn die Namen von Unterelementen der angegebenen Benutzerkonfigurationselemente des authentifizierten Benutzers ermittelt werden sollen.

Beispiel:

```
<cm-request...>
  <userConfig-getKeys>
    <listitem>dateOutputFormatName</listitem>
  </userConfig-getKeys>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <userConfig-getKeys>
      <listitem/>
    </userConfig-getKeys>
  </cm-code>
</cm-response>
```

12.2.8 <userConfig-getTexts>

Definition:

```
<!ENTITY % cm.userConfig-getTexts "
  (listitem+)
">
<!ELEMENT userConfig-getTexts %cm.userConfig-getTexts;>
<!ATTLIST userConfig-getTexts
  login CDATA #IMPLIED
```

```
>
```

Aufgabe: userConfig-getTexts liefert die Inhalte der spezifizierten Benutzerkonfigurationselemente.

Bedeutung der Attribute:

- `login`: das Login eines Benutzers.

Zusatzinformationen: Ein Konfigurationselement kann als Inhalt entweder Unterelemente oder einen Zeichenkettenwert enthalten, nicht jedoch beides. Daher liefert die Funktion eine leere Zeichenkette, wenn das auszulesende Konfigurationselement Unterelemente hat.

Rückgabewert bei Erfolg: die Liste der Inhalte der spezifizierten Konfigurationselemente.

Erforderliche Rechte: Wollen Sie die Inhalte der spezifizierten Konfigurationselemente eines beliebigen Benutzers ermitteln, so müssen Sie ein Superuser oder der Verwalter des betreffenden Benutzers sein. Es gibt jedoch keine Einschränkungen, wenn die Inhalte der angegebenen Benutzerkonfigurationselemente des authentifizierten Benutzers ermittelt werden sollen.

Beispiel:

```
<cm-request...>
  <userConfig-getTexts>
    <listitem>dateTimeInputFormatName</listitem>
    <listitem>dateTimeOutputFormatName</listitem>
  </userConfig-getTexts>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <userConfig-getTexts>
      <listitem>dayMonthYear</listitem>
      <listitem>europeanShortDateTime</listitem>
    </userConfig-getTexts>
  </cm-code>
</cm-response>
```

12.2.9 <userConfig-parseInputDate>

Definition:

```
<!ENTITY % cm.date "
 (%cm.atom; |
  isoDateTime |
  systemConfigFormattedTime |
  userConfigFormattedTime)*
">
<!ELEMENT userConfig-parseInputDate (%cm.date;)>
<!ATTLIST userConfig-parseInputDate
  login CDATA #IMPLIED
>
```

Aufgabe: Interpretiert den angegebenen String als Eingabewert für ein Datum und liefert die entsprechenden Datumsstrings.

Bedeutung der Attribute:

- `login`: das Login eines Benutzers.

Spezielle Parameter dieses Funktionselements:

- `isoDateTime`: Datum, als Zeitstempel in kanonischer Form formatiert.
- `systemConfigFormattedTime`: Datum, das entsprechend dem in der Systemkonfiguration angegebenen Wert für den Konfigurationseintrag `userManagement.preferences.dateTimeOutputFormatName` formatiert wird.
- userConfigFormattedTime: Datum, das entsprechend dem in den persönlichen Einstellungen des authentifizierten Benutzers angegebenen Wert für den Konfigurationseintrag dateTimeOutputFormatName formatiert wird.

Rückgabewert bei Erfolg: die Liste der Datumsstrings.

Erforderliche Rechte: Möchten Sie den benutzerspezifischen Konfigurationswert eines beliebigen Benutzers verwenden, so müssen Sie ein Superuser oder der Verwalter des betreffenden Benutzers sein. Soll der Konfigurationswert des authentifizierten Benutzers verwendet werden, so gibt es keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <userConfig-parseInputDate>15.12.2001 19:00 MET
  </userConfig-parseInputDate>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <userConfig-parseInputDate>
      <isoDateTime>20011215190001</isoDateTime>
      <systemConfigFormattedTime>15.12.2001 19:00
      </systemConfigFormattedTime>
      <userConfigFormattedTime>15.12.2001 15:00
      </userConfigFormattedTime>
    </userConfig-parseInputDate>
  </cm-code>
</cm-response>
```

12.2.10 <userConfig-removeAttributes>

Definition:

```
<!ENTITY % cm.userConfig-removeAttributes "
  (key,
  attributeNames?)
">
```

```
<!ELEMENT userConfig-removeAttributes
          %cm.userConfig-removeAttributes;>
<!ATTLIST userConfig-removeAttributes
  login CDATA #IMPLIED
>
```

Aufgabe: userConfig-removeAttributes löscht die angegebenen Tag-Attribute des spezifizierten Konfigurationselements.

Bedeutung der Attribute:

- `login`: das Login eines Benutzers.

Spezielle Parameter dieses Funktionselements:

- `key`: enthält den Namen des Konfigurationselements, dessen Tag-Attribute gelöscht werden sollen. Werden keine Attributnamen angegeben, so werden sämtliche Tag-Attribute des Konfigurationselements gelöscht.
- `attributeNames`: enthält den Namen eines Tag-Attributs, das gelöscht werden soll.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss ein Superuser sein.

Beispiel:

```
<cm-request...>
  <userConfig-removeAttributes>
      <key>keyName</key>
      <attributeNames>
        <listitem>tagAttribute</listitem>
        <listitem>anotherTagAttribute</listitem>
      </attributeNames>
  </userConfig-removeAttributes>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

12.2.11 <userConfig-removeKeys>

Definition:

```
<!ENTITY % cm.userConfig-removeKeys "
  (listitem*)
">
<!ELEMENT userConfig-removeKeys %cm.userConfig-removeKeys;>
<!ATTLIST userConfig-removeKeys
  login CDATA #IMPLIED
>
```

Aufgabe: Löscht die angegebenen Elemente aus der Benutzerkonfiguration.

Bedeutung der Attribute:

- `login`: das Login eines Benutzers.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss ein Superuser sein.

Beispiel:

```
<cm-request...>
  <userConfig-removeKeys>
    <listitem>anUnusedKey</listitem>
    <listitem>anotherUnusedKey</listitem>
  </userConfig-removeKeys>
```

```
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

12.2.12 <userConfig-setAttributes>

Definition:

```
<!ENTITY % cm.userConfig-setAttributes "
  (key,
  attributes*)
">
<!ELEMENT userConfig-setAttributes %cm.userConfig-setAttributes;>
<!ATTLIST userConfig-setAttributes
  login CDATA #IMPLIED
>
```

Aufgabe: userConfig-setAttributes ändert die Werte von Tag-Attributen oder fügt die als Name-Wert-Paare angegebenen Tag-Attribute zum spezifizierten Benutzerkonfigurationselement hinzu.

Bedeutung der Attribute:

- `login`: das Login eines Benutzers.

Spezielle Parameter dieses Funktionselements:

- `key`: enthält den Namen des Konfigurationselements, für welches die Werte seiner Tag-Attribute gesetzt werden sollen.
- `attribute`: enthält den Namen eines Tag-Attributs und die zu setzenden Werte.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss ein Superuser sein.

Beispiel:

```
<cm-request...>
  <userConfig-setAttributes>
    <key>dateTimeFormats</key>
    <attributes>
      <dictitem>
        <key>countryKey</key>
        <value>34</value>
      </dictitem>
      <dictitem>
        <key>timeZone</key>
        <value>GMT8</value>
      </dictitem>
    </attributes>
  </userConfig-setAttributes>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

12.2.13 <userConfig-setElements>

Definition:

```
<!ENTITY % cm.userConfig-setElements "
  (dictitem+)
">
<!ELEMENT userConfig-setElements %cm.userConfig-setElements;>
<!ATTLIST userConfig-setElements
  login CDATA #IMPLIED
>
```

Aufgabe: Ersetzt Benutzerkonfigurationselemente oder fügt Elemente zur Benutzerkonfiguration hinzu.

Bedeutung der Attribute:

- `login`: das Login eines Benutzers.

Zusatzinformationen: Bei den Werten der zu ersetzenden Elemente oder der neuen Elemente müssen die XML-spezifischen Zeichen-Referenzen für '<' und '>' verwendet werden.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss ein Superuser sein.

Beispiel:

```
<cm-request...>
  <userConfig-setElements>
    <dictitem>
      <key>dateOutputFormatName</key>
      <value>&lt;nameDateOutputFormat/&gt;</value>
    </dictitem>
    <dictitem>
      <key>dateTimeOutputFormatName</key>
      <value>&lt;nameDateTimeOutputFormat/&gt;</value>
    </dictitem>
  </userConfig-setElements>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

12.2.14 <userConfig-setTexts>

Definition:

```
<!ENTITY % cm.userConfig-setTexts "
  (dictitem+)
">
<!ELEMENT userConfig-setTexts %cm.userConfig-setTexts;>
<!ATTLIST userConfig-setTexts
  login CDATA #IMPLIED
>
```

Aufgabe: Ersetzt den Inhalt von Benutzerkonfigurationselementen.

Bedeutung der Attribute:

- `login`: das Login eines Benutzers.

Zusatzinformationen: Ein Konfigurationselement kann als Inhalt entweder Unterelemente oder einen Zeichenkettenwert enthalten, nicht jedoch beides. Daher werden alle Unterelemente eines Konfigurationselements gelöscht, wenn dessen Inhalt mit `setTexts` gesetzt wird.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss ein Superuser sein.

Beispiel:

```
<cm-request...>
  <userConfig-setTexts>
    <dictitem>
      <key>dateOutputFormatName</key>
      <value>europeanDate</value>
    </dictitem>
    <dictitem>
      <key>dateTimeOutputFormatName</key>
      <value>europeanShortDateTime</value>
    </dictitem>
  </userConfig-setTexts>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

13

13 Benutzermanager-Proxy - userProxy und groupProxy

Der Benutzermanager-Proxy ist ein integraler Bestandteil des Content Management Servers. Er dient zur Kommunikation zwischen dem Content Manager und dem integrierten oder einem externen Benutzermanager. In der Standardinstallation verwendet der Content Management Server diesen Proxy, um mit dem CMS-eigenen Benutzermanager zu kommunizieren.

Zu diesem Zweck ruft der Content Manager Tcl-Prozeduren auf, die er beim Start aus der Datei `usermanAPI.tcl` eingelesen hat. Die in dieser Datei definierten Funktionen stellen die Schnittstelle - das sogenannte Benutzermanager-API (`usermanAPI`) - im Benutzermanager-Proxy dar (siehe Handbuch zur *Systemadministration / Entwickulung*).

Wenn der Content Management Server mit einem externen Benutzermanager betrieben wird, so können die Funktionen des `usermanAPI` so umgeschrieben werden, dass sie ihren jeweiligen Rückgabewert berechnen, indem sie eine Prozedur dieses externen Benutzermanagers aufrufen. Auf diese Weise ist es möglich, lesend auf die von diesem verwalteten Benutzer- und Benutzergruppendaten zuzugreifen. Über den Benutzermanager-Proxy kann man nicht schreibend auf einen externen Benutzermanager zugreifen.

Daten, die mit einem externen Benutzermanager verwaltet werden, können auch in XML-Anfragen an den Content Management Server abgefragt werden, indem die `userProxy`- oder `groupProxy`-Elemente von CRUL verwendet werden. Solche Anfragen bewirken, dass sich die XML-Schnittstelle des Content Managers an den Benutzermanager-Proxy wendet. Dieser fordert die gewünschten Daten über das `usermanAPI` an. Die Ergebnisse werden in der Response zurückgegeben.

Die `userProxy`- oder `groupProxy`-Elemente sollten Sie nur mit einem externen Benutzermanager verwenden. Wird dagegen der interne Benutzermanager des Content Management Servers genutzt, so verwenden Sie bitte die in CRUL enthaltenen Elemente für Benutzer, Benutzergruppen und Benutzerfelder, um Daten auszulesen oder zu setzen (siehe Benutzer - user, Benutzerfelder - userAttribute und Benutzergruppen - group).

13.1 userProxy-Elemente

13.1.1 Definition

Der folgende Ausschnitt aus der DTD zeigt die Elemente, mit denen in Requests an den Content Management Server der lesende Zugriff auf Daten für Benutzer über den Benutzermanager-Proxy möglich ist:

```
<!ENTITY % cm.cm-request "
  ...
  (userProxy-where+, userProxy-get) |
```

```
  ...
">
```

Die Elemente `userProxy-where` und `userProxy-get` müssen sich unmittelbar unterhalb des `cm-request`-Elements befinden und werden im Abschnitt Funktionselemente für Requests mit userProxy-Elementen erläutert.

13.1.2 Parameterelemente für Requests mit userProxy-Elementen

Mit Hilfe von Funktionselementen kann man über den Benutzermanager-Proxy auf Benutzerdaten zugreifen. So lassen sich beispielsweise mit dem `userProxy-get`-Element die Werte sämtlicher Benutzer-Parameter ermitteln. Um zu spezifizieren, welche Benutzereigenschaften ausgelesen oder gesetzt werden sollen, verwendet man Parameterelemente. Welche Parameterelemente verwendet werden können, ist vom externen Benutzermanager abhängig. Im Folgenden werden die verfügbaren `userProxy`-Parameterelemente aufgeführt. Mit Ausnahme von `description` entsprechen sie den `user`-Parameterelementen (siehe Parameterelemente für Benutzer).

`defaultGroup`

Bedeutung: Die Standardgruppe des Benutzers.

Definition:

```
<!ELEMENT defaultGroup (%cm.atom; | %cm.group-get;)*>
```

`cm.group-get`: siehe <group-where> <group-get> oder CRUL als DTD.

`description`

Bedeutung: Liefert die Daten des angegebenen Benutzers.

Definition:

```
<!ELEMENT description (dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`displayTitle`

Bedeutung: Der in der HTML-Benutzerschnittstelle angezeigte Bezeichner des Benutzers (eine Kombination aus Login und vollem Namen).

Definition:

```
<!ELEMENT displayTitle (%cm.atom;)>
```

encryptedPassword

Bedeutung: Das verschlüsselte Passwort des Benutzers.

Definition:

```
<!ELEMENT encryptedPassword (%cm.atom;)>
```

email

Bedeutung: Die E-Mail-Adresse des Benutzers.

Definition:

```
<!ELEMENT email (%cm.atom;)>
```

externalUserAttrNames

Bedeutung: Liste der Benutzerfelder des Benutzers.

Definition:

```
<!ELEMENT externalUserAttrNames ((userAttribute)* | (listitem)*)>
<!ELEMENT userAttribute (%cm.userAttribute-get;)>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.userAttribute-get`: siehe <userAttribute-where> <userAttribute-get> oder CRUL als DTD.

getKeys

Bedeutung: Die Liste der mit `user-get` abfragbaren Parameter.

Definition:

```
<!ELEMENT getKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

globalPermissions

Bedeutung: Die Liste der globalen Rechte.

Definition:

```
<!ELEMENT globalPermissions (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

hasGlobalPermission

Bedeutung: Überprüft, ob der Benutzer das spezifizierte globale Recht hat.

Definition:

```
<!ELEMENT hasGlobalPermission (%cm.atom;)>
<!ATTLIST hasGlobalPermission
          permission (permissionGlobalExport |
                      permissionGlobalRoot |
                      permissionGlobalRTCEdit |
                      permissionGlobalUserAttributeEdit |
                      permissionGlobalUserEdit) CDATA #REQUIRED
>
```

Bedeutung der Attribute:

- `permission`: spezifiziert ein globales Recht.

Beispiel:

```
<cm-request...>
  <userProxy-where>
    <login>stan</login>
  </userProxy-where>
  <userProxy-get>
    <hasGlobalPermission permission="permissionGlobalRTCEdit"/>
  </userProxy-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <userProxy>
      <hasGlobalPermission permission="permissionGlobalRTCEdit">1
      </hasGlobalPermission>
    </userProxy>
  </cm-code>
</cm-response>
```

groups

Bedeutung: Die Liste der Gruppen, in denen der Benutzer Mitglied ist.

Definition:

```
<!ELEMENT groups ((group)* | (listitem)*)>
<!ELEMENT group (%cm.atom; | %cm.group-get;)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
```

```
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.group-get`: siehe <group-where> <group-get> oder CRUL als DTD.

hasPassword

Bedeutung: Überprüft, ob das angegebene Passwort das Passwort des Benutzers ist (`1`= ist das Passwort des Benutzers; `0`= ist nicht das Passwort des Benutzers).

Definition:

```
<!ELEMENT hasPassword (%cm.atom;)>
<!ATTLIST hasPassword
          password CDATA #REQUIRED
>
```

Bedeutung der Attribute:

- `password`: spezifiziert das Passwort (Klartext).

Beispiel:

```
<cm-request...>
  <userProxy-where>
    <login>stan</login>
  </userProxy-where>
  <userProxy-get>
    <hasPassword password="xyz123"/>
  </userProxy-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <userProxy>
      <hasPassword password="xyz123">0</hasPassword>
    </userProxy>
  </cm-code>
</cm-response>
```

isSuperUser

Bedeutung: Zeigt an, ob der Benutzer ein Superuser ist (`1` = ist Superuser; `0` = ist kein Superuser).

Definition:

```
<!ELEMENT isSuperUser (%cm.atom;)>
```

isOwnerOf

Bedeutung: Zeigt an, ob der Benutzer der Verwalter eines anderen Benutzers oder einer Benutzgruppe ist (`1` = ist der Verwalter; `0` = ist nicht der Verwalter).

Definition:

```
<!ELEMENT isOwnerOf (%cm.atom;)>
<!ATTLIST isOwnerOf
          user CDATA #IMPLIED
          group CDATA #IMPLIED
>
```

Bedeutung der Attribute:

- `user`: spezifiziert einen Benutzer (login).
- `group`: spezifiziert eine Benutzergruppe (name).

Beispiel:

```
<cm-request...>
  <userProxy-where>
    <login>lisa</login>
  </userProxy-where>
  <userProxy-get>
    <isOwnerOf group="NewsEditors"/>
  </userProxy-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <userProxy>
      <isOwnerOf group="NewsEditors">1</isOwnerOf>
    </userProxy>
  </cm-code>
</cm-response>
```

login

Bedeutung: Das Login des Benutzers.

Definition:

```
<!ELEMENT login (%cm.atom;)>
```

owner

Bedeutung: Der Verwalter des Benutzers.
Wenn ein Benutzer angelegt wird, wird der eingeloggte Benutzer automatisch zum direkten Verwalter (`owner`) des neuen Benutzers. Der Verwalter hat das Recht, Felder und Rechte des neuen Benutzers zu ändern. Das Login des direkten Verwalters ist im Parameter `owner` des Benutzers eingetragen.
Der Verwalter eines Benutzers hat auch die Möglichkeit, einen anderen Wert in den Parameter `owner` des Benutzers einzutragen. Dieser andere Wert kann entweder ein Login oder ein

Gruppenname sein. Mit der Eintragung eines neuen Verwalters tritt der vorherige Verwalter seine Rechte an einen anderen Benutzer oder an eine Gruppe ab.
Ist im Parameter `owner` eines Benutzers eine Gruppe eingetragen, so gelten alle Mitglieder der Gruppe als indirekte Verwalter.
Um die Parameter eines Benutzers ändern zu können, muss man der direkte oder der indirekte Verwalter des Benutzers sein und das Recht `permissionGlobalUserEdit` haben. Ein Benutzer kann also seine Daten nicht selbst ändern, es sei denn, er ist sein eigener direkter oder indirekter Verwalter.
Die Wirkungsweise des Parameters `owner` gilt analog auch für Gruppen.

Definition:

```
<!ELEMENT owner (%cm.atom;)>
```

password

Bedeutung: Das Klartext-Passwort des Benutzers.

Definition:

```
<!ELEMENT password (%cm.atom;)>
```

realName

Bedeutung: Der volle Name des Benutzers.

Definition:

```
<!ELEMENT realName (%cm.atom;)>
```

setKeys

Bedeutung: Die Liste der mit `user-set` setzbaren Parameter.

Definition:

```
<!ELEMENT setKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

userLocked

Bedeutung: Gibt an, ob der Benutzer gesperrt ist (`1` = ist gesperrt; `0` = ist nicht gesperrt).

Definition:

```
<!ELEMENT userLocked (%cm.atom;)>
```

13.1.3 Funktionselemente für Requests mit userProxy-Elementen

<userProxy-where>

Ausschnitt aus der DTD:

```
<!ENTITY % cm.cm-request "
  ...
  (userProxy-where+, userProxy-get) |
  ...
">
```

Definition:

```
<!ENTITY % cm.user-where "
  (login |
  userText)
">
<!ELEMENT userProxy-where %cm.user-where;>
```

Aufgabe: `userProxy-where` ermöglicht die Suche nach Benutzern über den Benutzermanager-Proxy. Wird kein Parameter angegeben, so werden die Logins aller Benutzer zurückgegeben.

Spezielle Parameter dieses Funktionselements:

- `userText`: bewirkt, dass die Logins der Benutzer zurückgegeben werden, bei denen die angegebene Zeichenkette im Login oder im vollständigen Namen der Benutzer enthalten ist.

Verwendung: `userProxy-where` ist ein Funktionselement, das nicht ohne ein weiteres unmittelbar darauf folgendes Funktionselement verwendet werden kann. Mit `userProxy-where` erhält man entsprechend dem Suchkriterium das Login eines Benutzers, die Liste der Logins der Benutzer, deren Login oder voller Name die angegebene Zeichenkette enthält, oder die Liste der Logins aller Benutzer. Erst durch das folgende Funktionselement userProxy-get wird jedoch bestimmt, wie der oder die Benutzer zu behandeln sind. userProxy-get entspricht dabei einer Schablone, die nacheinander auf jeden mit `userProxy-where` ermittelten Benutzer angewendet wird (siehe Requests als Schablonen und die Darstellung der Ergebnis-Daten).

Rückgabewert bei Erfolg: die Liste der Logins der Benutzer oder die Liste aller Anmeldenamen.

Erforderliche Rechte: keine Einschränkungen.

<userProxy-where> <userProxy-get>

Definition:

```
<!ENTITY % cm.user-get "
  (defaultGroup |
  displayTitle |
```

```
  encryptedPassword |
  email |
  externalUserAttrNames |
  getKeys |
  globalPermissions |
  groups |
  hasGlobalPermission |
  isSuperUser |
  isOwnerOf |
  hasPassword |
  login |
  owner |
  realName |
  setKeys |
  userLocked)*
">
<!ELEMENT userProxy-get %cm.user-get;>
```

Aufgabe: Liefert den Wert der angegebenen Parameter für jeden mit `userProxy-where` ermittelten Benutzer.

Verwendung: Zunächst werden mit `userProxy-where` Benutzer ermittelt. Anschließend wird `userProxy-get` verwendet, um Parameterwerte dieser Benutzer auszulesen.

Rückgabewert bei Erfolg: die Werte der angegebenen `userProxy`-Parameter.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <userProxy-where>
    <login>mara</login>
  </userProxy-where>
  <userProxy-get>
    <defaultGroup/>
    <email/>
    <isOwnerOf group="NewsEditors"/>
    <owner/>
    <realName/>
  </userProxy-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <userProxy>
      <defaultGroup>NewsMasters</defaultGroup>
      <email>maramc@up.com</email>
      <isOwnerOf group="NewsEditors">1</isOwnerOf>
      <owner>stan</owner>
      <realName>Mara McLeod</realName>
    </userProxy>
  </cm-code>
</cm-response>
```

13.2 groupProxy-Elemente

13.2.1 Definition

Der folgende Ausschnitt aus der DTD des Content Management Servers zeigt die Elemente, mit denen in Requests an den Content Manager über den Benutzermanager-Proxy lesend auf Benutzergruppendaten zugegriffen werden kann:

```
<!ENTITY % cm.cm-request "
  ...
  (groupProxy-where+, groupProxy-get) |
  ...
">
```

Die Elemente groupProxy-where und groupProxy-get, die sich unmittelbar unterhalb des `cm-request`-Elements befinden müssen, werden im Abschnitt Funktionselemente für Requests mit groupProxy-Elementen erläutert.

13.2.2 Parameterelemente für Requests mit groupProxy-Elementen

Auf Benutzergruppen kann man über den Benutzermanager-Proxy mit Hilfe von Funktionselementen zugreifen. Mit dem `groupProxy-get`-Element lassen sich beispielsweise die Werte sämtlicher Benutzergruppen-Parameter ermitteln. Die Benutzergruppeneigenschaften, die ausgelesen oder gesetzt werden sollen, spezifiziert man mit Parameterelementen. Welche Parameterelemente verwendet werden können, ist vom externen Benutzermanager abhängig. Im Folgenden werden die `groupProxy`-Parameterelemente aufgeführt. Mit Ausnahme von `description` sind sie sind mit den Parameterelementen für Benutzergruppen identisch (siehe Parameterelemente für Benutzergruppen).

`displayTitle`

Bedeutung: Der in der HTML-Benutzerschnittstelle angezeigte Bezeichner der Gruppe. Dieser Titel ist eine Kombination aus dem Namen und dem vollen Namen.

Definition:

```
<!ELEMENT displayTitle (%cm.atom;)>
```

`description`

Bedeutung: Liefert die Daten der angegebenen Benutzergruppe.

Definition:

```
<!ELEMENT description (dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

getKeys

Bedeutung: Die Liste der mit `group-get` abfragbaren Parameter.

Definition:

```
<!ELEMENT getKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

globalPermissions

Bedeutung: Die Liste der globalen Rechte.

Definition:

```
<!ELEMENT globalPermissions (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

name

Bedeutung: Der Name der Gruppe.

Definition:

```
<!ELEMENT name (%cm.atom;)>
```

hasGlobalPermission

Bedeutung: Überprüft, ob die Gruppe das angegebene globale Recht hat.

Definition:

```
<!ELEMENT hasGlobalPermission (%cm.atom;)>
<!ATTLIST hasGlobalPermission
          permission (permissionGlobalExport |
                      permissionGlobalRoot |
                      permissionGlobalRTCEdit |
                      permissionGlobalUserAttributesEdit |
                      permissionGlobalUserEdit) CDATA #REQUIRED
>
```

Bedeutung der Attribute:

- `permission`: spezifiziert ein globales Recht.

Beispiel:

```
<cm-request...>
  <groupProxy-where>
    <name>NewsMasters</name>
  </groupProxy-where>
  <groupProxy-get>
    <hasGlobalPermission permission="permissionGlobalRTCEdit"/>
  </groupProxy-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <groupProxy>
      <hasGlobalPermission permission="permissionGlobalRTCEdit">0
      </hasGlobalPermission>
    </groupProxy>
  </cm-code>
</cm-response>
```

owner

Bedeutung: Der Verwalter der Benutzergruppe. Im Parameter `owner` einer Gruppe ist der Name des Verwalters der Gruppe (ein Benutzer- oder Gruppenname) gespeichert. Der Verwalter darf die Parameter und Rechte einer Gruppe ändern.

Definition:

```
<!ELEMENT owner (%cm.atom; | %cm.user-get;)*>
```

`cm.user-get`: siehe <user-where> <user-get> oder CRUL als DTD.

realName

Bedeutung: Der volle Name der Gruppe.

Definition:

```
<!ELEMENT realName (%cm.atom;)>
```

setKeys

Bedeutung: Die Liste der mit `group-set` setzbaren Parameter.

Definition:

```
<!ELEMENT setKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
```

```
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

users

Bedeutung: Die Liste der Benutzer, die Mitglieder der Gruppe sind.

Definition:

```
<!ELEMENT users ((user)* | (listitem)*)>
<!ELEMENT user (%cm.atom; | %cm.user-get;)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.user-get`: siehe <user-where> <user-get> oder CRUL als DTD.

13.2.3 Funktionselemente für Requests mit groupProxy-Elementen

<groupProxy-where>

Ausschnitt aus der DTD:

```
<!ENTITY % cm.cm-request "
  ...
  (groupProxy-where+, groupProxy-get) |
  ...
">
```

Definition:

```
<!ENTITY % cm.group-where "
  (name |
  groupText)
">
<!ELEMENT groupProxy-where %cm.group-where;>
```

Aufgabe: groupProxy-where ermöglicht die Suche nach Benutzergruppennamen über den Benutzermanager-Proxy. Wird kein Parameterwert angegeben, so werden die Namen aller Benutzergruppen zurückgegeben.

Spezielle Parameter dieses Funktionselements:

- `groupText`: bewirkt, dass die Namen der Benutzergruppen zurückgegeben werden, bei denen die angegebene Zeichenkette in `name` oder `realName` enthalten ist.

Verwendung: `groupProxy-where` ist ein Funktionselement, das nicht ohne ein weiteres unmittelbar darauf folgendes Funktionselement verwendet werden kann. Mit `groupProxy-where` erhält man abhängig vom Suchkriterium einen oder mehrere Gruppennamen oder die Liste aller Gruppennamen. Erst durch das folgende Funktionselement `groupProxy-get` wird jedoch bestimmt, wie die

ermittelten Benutzergruppen zu behandeln sind. `groupProxy-get` entspricht dabei einer Schablone, die nacheinander auf jede der mit `groupProxy-where` ermittelten Gruppen angewendet wird (siehe Requests als Schablonen und die Darstellung der Ergebnis-Daten).

Erforderliche Rechte: keine Einschränkungen.

<groupProxy-where> <groupProxy-get>

Definition:

```
<!ENTITY % cm.group-get "
  (displayTitle |
  getKeys |
  globalPermissions |
  hasGlobalPermission |
  users |
  name |
  owner |
  realName |
  setKeys |
  description)*
">
<!ELEMENT groupProxy-get %cm.group-get;>
```

Aufgabe: Liefert den Wert der angegebenen Parameter.

Verwendung: `groupProxy-get` muss immer auf `groupProxy-where` folgen, da zunächst die Gruppen ermittelt werden müssen, auf die `groupProxy-get` angewendet wird, um Parameterwerte auszulesen.

Rückgabewert bei Erfolg: der Wert der angegebenen Parameter.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <groupProxy-where>
    <name>NewsEditors</name>
  </groupProxy-where>
  <groupProxy-get>
    <owner/>
  </groupProxy-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <groupProxy>
      <owner>NewsMasters</owner>
    </groupProxy>
  </cm-code>
</cm-response>
```

14

14 Channels - channel

14.1 Definition

Ein Channel ist ein Name, unter dem Nachrichten (News) thematisch zusammengefasst werden können. Channels lassen sich zwar mit dem Content Management Server definieren, sie haben für den Betrieb dieses Servers jedoch keine inhaltliche Bedeutung. Auf einem Live-System dagegen, das die Template Engine und den Portal Manager umfasst, können die definierten Channels verwendet werden, um den Website-Besuchern gegebenenfalls personalisierte Newslisten anzubieten. Der folgende CRUL-Ausschnitt zeigt die Elemente, mit denen Channels im Content Management Server administriert werden können:

```
<!ENTITY % cm.cm-request "
  ...
  (channel-create) |
  (channel-where+, channel-delete) |
  (channel-where+, channel-get) |
  (channel-where+, channel-description) |
  (channel-where+, channel-set) |
  ...
">
<!ATTLIST channel-where
  maxResults CDATA #IMPLIED
>
```

Diese Elemente müssen sich unmittelbar unterhalb des `cm-request`-Elements befinden. Sie werden im Abschnitt Funktionselemente für Channels erläutert.

Mit dem Attribut `maxResults`, dessen Wert eine ganze Zahl ist, kann die maximale Anzahl der Treffer festgelegt werden. Ist der Wert von `maxResults` kleiner oder gleich null, ist die Anzahl nicht begrenzt

14.2 Parameterelemente für Channels

Auf Channels kann man mit Hilfe von Funktionselementen zugreifen. So kann man beispielsweise mit dem `channel-get`-Element die Werte von Channel-Parametern ermitteln. Welche Channel-Eigenschaften ausgelesen werden sollen, spezifiziert man mit Parameterelementen. Im Folgenden werden die Channel-Parameterelemente aufgeführt.

`getKeys`

Bedeutung: Die Liste der mit `channel-get` abfragbaren Parameter.

Definition:

```
<!ELEMENT getKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

name

Bedeutung: Eindeutiger Name des Channels.

Definition:

```
<!ELEMENT name (%cm.atom;)>
```

setKeys

Bedeutung: Die Liste der mit `channel-set` setzbaren Channel-Felder.

Definition:

```
<!ELEMENT setKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

title

Bedeutung: Der Titel des Channels.

Definition:

```
<!ELEMENT title (%cm.atom;)>
<!ATTLIST title
          lang (en | de | it | fr | es) #IMPLIED
>
```

Bedeutung der Attribute:

- `lang`: Kennzeichnet die Sprache eines Channeltitels. Zu Dimensionen von Werten im CMS siehe Dimensionen von Werten.

Beispiel:

```
<cm-request...>
```

```
  <channel-where>
    <name>technews</name>
  </channel-where>
  <channel-get>
    <title lang="en"/>
  </channel-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <channel>
      <title lang="en">Technical Stuff</title>
    </channel>
  </cm-code>
</cm-response>
```

14.3 Funktionselemente für Channels

14.3.1 <channel-where>

Ausschnitt aus der DTD:

```
<!ENTITY % cm.cm-request "
  ...
  (channel-where+, channel-delete) |
  (channel-where+, channel-get) |
  (channel-where+, channel-description) |
  (channel-where+, channel-set) |
  ...
">
```

Definition:

```
<!ENTITY % cm.channel-where "
  (name |
  namePrefix)*
">
<!ELEMENT channel-where %cm.channel-where;>
```

Aufgabe: channel-where ermöglicht es, nach Channels zu suchen. Sämtliche Channelnamen werden zurückgegeben, wenn kein Parameterwert angegeben wird.

Spezielle Parameter dieses Funktionselements:

- `name`: bewirkt, dass der Channel mit dem angegebenen Namen zurückgegeben wird.
- `namePrefix`: bewirkt, dass die Channels zurückgegeben werden, deren Name mit der angegebenen Zeichenkette beginnt.

Verwendung: `channel-where` ist ein Funktionselement, das nicht ohne ein weiteres unmittelbar darauf folgendes Funktionselement verwendet werden kann. Mit `channel-where` erhält man den Channelnamen, der dem Suchkriterium entspricht oder die Liste aller Channelnamen. Erst durch das folgende Funktionselement wird jedoch bestimmt, wie diese Channelnamen zu behandeln sind. channel-get entspricht dabei einer Schablone, die nacheinander auf jeden der mit `channel-where` ermittelten Channelnamen angewendet wird (siehe Requests als Schablonen und die Darstellung der Ergebnis-Daten).

Erforderliche Rechte: keine Einschränkungen.

14.3.2 <channel-where> <channel-delete>

Definition:

```
<!ENTITY % cm.channel-delete "EMPTY">
<!ELEMENT channel-delete %cm.channel-delete;>
```

Aufgabe: channel-delete löscht einen Channel.

Verwendung: Zunächst wird der Channel mit `channel-where` ermittelt. Anschließend wird er mit `channel-delete` gelöscht.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalRTCEdit` haben.

Beispiel:

```
<cm-request...>
  <channel-where>
    <name>technews</name>
  </channel-where>
  <channel-delete/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

14.3.3 <channel-where> <channel-description>

Definition:

```
<!ELEMENT channel-description %cm.atom;>
<!ENTITY % cm.atom "#PCDATA">
```

Aufgabe: `channel-description` liefert eine Zeichenketten-Repräsentation der wichtigsten Channel-Parameter.

Verwendung: Zunächst wird mit `channel-where` der Channel ermittelt, dessen Zeichenketten-Repräsentation anschließend mit `channel-description` ausgelesen wird.

Rückgabewert bei Erfolg: die Zeichenketten-Repräsentation.

Erforderliche Rechte: keine.

Beispiel:

```
<cm-request...>
  <channel-where>
    <name>technews</name>
  </channel-where>
```

```
  <channel-description/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok>
    {
      name = technews;
      title = "Technical Stuff";
    }
  </cm-code>
</cm-response>
```

14.3.4 <channel-where> <channel-get>

Definition:

```
<!ENTITY % cm.channel-get "
  (name |
  title)*
">
<!ELEMENT channel-get %cm.channel-get;>
```

Aufgabe: channel-get liefert den Wert der angegebenen Parameterelemente für die mit `channel-where` spezifizierten Channels.

Verwendung: `channel-get` muss immer auf `channel-where` folgen, da zunächst die Channels ermittelt werden müssen, auf die `channel-get` angewendet werden soll, um Parameterwerte auszulesen.

Rückgabewert bei Erfolg: der Wert der angegebenen Parameter.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <channel-where>
    <name>technews</name>
  </channel-where>
  <channel-get>
    <title lang="en" />
  </channel-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <channel>
      <title lang="en">Technical Stuff</title>
    </channel>
  </cm-code>
</cm-response>
```

14.3.5 <channel-where> <channel-set>

Definition:

```
<!ENTITY % cm.channel-set "
```

```
   (name |
   title)*
">
<!ELEMENT channel-set %cm.channel-set;>
```

Aufgabe: Setzt die spezifizierten Parameter eines Channels auf die angegebenen Werte.

Verwendung: `channel-set` muss immer auf `channel-where` folgen, da zunächst die Channels ermittelt werden müssen, auf die `channel-set` angewendet wird, um Parameterwerte zu setzen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss Superuser sein, oder er muss das Recht `permissionGlobalRTCEdit` haben.

Beispiel:

```
<cm-request...>
  <channel-where>
    <name>technews</name>
  </channel-where>
  <channel-set>
    <title lang="en">Technical Stuff</title>
    <title lang="de">Technisches</title>
  </channel-set>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

15

15 Versionen - content

15.1 Definition

Auf Versionen kann man in Requests lesend und schreibend zugreifen. Der folgende DTD-Ausschnitt zeigt die Elemente, mit denen dies über die XML-Schnittstelle des Content Management Servers möglich ist:

```
<!ENTITY % cm.cm-request "
  ...
  (content-where+, content-addLinkTo) |
  (content-where+, content-debugExport) |
  (content-where+, content-delete) |
  (content-where+, content-description) |
  (content-where+, content-get) |
  (content-where+, content-load) |
  (content-where+, content-resolveRefs) |
  (content-where+, content-set) |
  ...
">
```

Diese Elemente müssen sich unmittelbar unterhalb des `cm-request`-Elements befinden. Im Abschnitt Funktionselemente für Versionen werden sie erläutert.

15.2 Parameterelemente für Versionen

Mit Hilfe von Funktionselementen kann man auf Versionen zugreifen. So können mit dem `content-get`-Element beispielsweise die Werte sämtlicher Versionsparameter ermittelt werden. Um zu spezifizieren, welche Versionsfelder ausgelesen oder gesetzt werden sollen, verwendet man jedoch Parameterelemente. Im Folgenden werden die Parameterelemente für Versionen aufgeführt.

anchors

Bedeutung: Liste der Ankernamen in der Version (nur bei Dateien vom Typ `publication` oder `document`).

Definition:

```
<!ELEMENT anchors (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

blob

Bedeutung: Der Inhalt zur Version. Je nach Datei, zu der die Version gehört, werden ggf. im Inhalt vorkommende URLs wie im Export zurückgegeben. Siehe `obj-create` auf <obj-create> zur Erläuterung der Werte des Attributs `encoding`.

Definition:

```
<!ELEMENT blob (%cm.atom;)>
<!ATTLIST blob
  encoding (plain | base64 | stream) #IMPLIED
>
```

blobLength

Bedeutung: Liefert bei Dateien vom Typ `image`(`Bild`) und `generic` (Ressource) die Länge der Binärdaten, bei Dateien der anderen Typen die Größe des Hauptinhalts in Bytes.

Definition:

```
<!ELEMENT blobLength (%cm.atom;)>
```

body

Bedeutung: Nur bei Layouts, Ordnern und Dokumenten: Vorverarbeitete Version des HTML-Codes der Version, die für interne Zwecke verwendet wird.

Definition:

```
<!ELEMENT body (%cm.atom;)>
```

channels

Bedeutung: Liste der Channels, zu denen die Version auf dem Live-Server gehört (nicht bei Dateien vom Typ *Layout*).

Definition:

```
<!ELEMENT channels ((channel)* | (listitem)*)>
<!ELEMENT channel (%cm.atom; | %cm.channel-get;)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

cm.channel-get: siehe <channel-where> <channel-get> oder CRUL als DTD.

codeForPreview

Bedeutung: Liefert den vom Content Manager berechneten Code für die Vorschauseite einer Version. In der gelieferten Zeichenkette sind die XML-spezifischen Zeichen wie '<' in die entsprechenden XML-Zeichen-Referenzen umgewandelt.

Definition:

```
<!ELEMENT codeForPreview (%cm.atom;)>
<!ATTLIST codeForPreview
    attributeName CDATA #REQUIRED
    previewPageUrlSuffix CDATA #REQUIRED
    frameName CDATA #IMPLIED
>
```

Bedeutung der Attribute:

- `attributeName`: der Name eines Feldes.
- `previewPageUrlSuffix`: die URL der Vorschauseite.
- `frameName`: der Name eines Frames, sofern die Version durch ein Framelayout bearbeitet wird.

Beispiel:

```
<cm-request...>
  <content-where>
    <objectId>43265</objectId>
    <state>edited</state>
  </content-where>
  <content-get>
    <codeForPreview attributeName="blob"
      previewPageUrlSuffix="http://my.server.de:3001/...html"/>
  </content-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <content>
      <codeForPreview attributeName="blob"
        previewPageUrlSuffix="http://my.server.de:3001/...html">
      der Code für die Vorschauseite (gequotet)
      </codeForPreview>
    </content>
  </cm-code>
</cm-response>
```

codeForSourceView

Bedeutung: Liefert den vom Content Manager berechneten Code für den Quelltext einer Version. In der gelieferten Zeichenkette sind die XML-spezifischen Zeichen wie '<' in die entsprechenden XML-Zeichen-Referenzen umgewandelt.

Definition:

```
<!ELEMENT codeForSourceView (%cm.atom;)>
<!ATTLIST codeForSourceView
    attributeName CDATA #REQUIRED
    objectPageUrl CDATA #REQUIRED
    linkPageUrl CDATA #REQUIRED
    linkIconUrl CDATA #REQUIRED
    useJavaScript CDATA #IMPLIED
>
```

Bedeutung der Attribute:

- `attributeName`: der Name eines Feldes.
- `objectPageUrl`: die URL der Content-Navigator-Seite der Datei.
- `linkPageUrl`: die URL für einen Link zur Link-Bearbeitungsseite.
- `linkIconUrl`: die URL für das Linkicon.
- `useJavaScript`: Wenn vorhanden und `true`, schreibt der Content Manager JavaScript-Code in die Links zur Dateibearbeitung.

Beispiel:

```
<cm-request...>
  <content-where>
    <objectId>5765</objectId>
    <state>edited</state>
  </content-where>
  <content-get>
    <codeForSourceView attributeName="body"
      objectPageUrl="http://my.server.de:...html?display=default&count=100"
      linkPageUrl="http://my.server.de:3001/CM/CMLinkEditPage/"
      linkIconUrl="/images/link.gif"/>
  </content-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <content>
      <codeForSourceView attributeName="body"
        objectPageUrl="http://my.server.de:...html?display=default&count=100"
        linkPageUrl="http://my.server.de:3001/CM/CMLinkEditPage/"
        linkIconUrl="/images/link.gif">
      der Code für die Anzeige des Quelltextes (gequotet)
      </codeForSourceView>
    </content>
  </cm-code>
</cm-response>
```

codeForThumbnail

Bedeutung: Liefert die URL, den alternativen Text sowie die Größe und Breite eines Vorschaubildes (Thumbnail).

Definition:

```
<!ELEMENT codeForThumbnail (%cm.atom;)>
<!ATTLIST codeForThumbnail
          thumbnailPageUrl CDATA #REQUIRED
```

```
          defaultImageUrl CDATA #REQUIRED
>
```

Bedeutung der Attribute:

- `thumbnailPageUrl`: die URL der Seite, die die URL des Thumbnails erzeugt.
- `defaultImageUrl`: die URL des Bildes, das immer dann angezeigt wird, wenn keine URL für das Vorschaubild erzeugt werden kann.

Beispiel:

```
<cm-request...>
  <content-where>
    <objectId>5765</objectId>
    <state>edited</state>
  </content-where>
  <content-get>
    <codeForThumbnail
        thumbnailPageUrl="http://my.server.de:3001/CM/CMObjectThumbnailPage.jsp"
        defaultImageUrl="http://my.server.de:3001/CM/Images/image.jpg"/>
  </content-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <content>
      <codeForThumbnail
          thumbnailPageUrl="http://my.server.de:3001/CM/CMObjectThumbnailPage.jsp"
          defaultImageUrl="http://my.server.de:3001/CM/Images/image.jpg">
        &lt;IMG BORDER=&quot;0&quot;
          SRC=&quot;http://...Servlet?
&amp;pageAction=defaultPageAction&objectId=54076&quot;
          ALT=&quot;Vorschau&quot; WIDTH=&quot;48&quot;
          HEIGHT=&quot;48&quot;&gt;
      </codeForThumbnail>
    </content>
  </cm-code>
</cm-response>
```

contentType

Bedeutung: Die Dateiendung des Inhalts, die der Version zugeordnet ist. Für Versionen von Dateien des Typs `document` und `publication` gilt: Nur wenn der zur Dateiendung gehörende MIME-Typ `text/html` ist, wird der Haupttext der Version vom Content Manager verarbeitet, werden also beispielsweise darin enthaltene Links mit der Linkverwaltung abgeglichen. Andernfalls wird der Haupttext als reiner Text behandelt. Siehe auch den Systemkonfigurationseintrag `content.mimeTypes` im Handbuch zur *Systemadministration / Entwicklung*.

Definition:

```
<!ELEMENT contentType (%cm.atom;)>
```

displayTitle

Bedeutung: Der in der HTML-Benutzerschnittstelle angezeigte Titel der Version (entspricht dem Wert von `title`).

Definition:

```
<!ELEMENT displayTitle (%cm.atom;)>
```

editor

Bedeutung: Der Anmeldename des Bearbeiters der Version.

Definition:

```
<!ELEMENT editor (%cm.atom; | %cm.user-get;)*>
```

`cm.user-get`: siehe <user-where> <user-get> oder CRUL als DTD.

exportBlob

Bedeutung: Der Inhalt zur Version. Je nach Datei, zu der die Version gehört, wird der Blob mittels der zuständigen Layouts in das endgültige Export-Format gebracht.

Definition:

```
<!ELEMENT exportBlob (%cm.atom;)>
<!ATTLIST exportBlob
  encoding (plain | base64 | stream) #IMPLIED
>
```

exportBlobForFrame

Bedeutung: String-Repräsentation des Blobs für einen bestimmten Frame. Je nach Datei, zu der die Version gehört, wird der Blob mit Hilfe der zuständigen Layouts in das endgültige Export-Format gebracht (für Dateien vom Typ *Ordner* oder *Dokument*).

Definition:

```
<!ELEMENT exportBlobForFrame (%cm.atom;)>
<!ATTLIST exportBlobForFrame
          frame CDATA #REQUIRED
>
```

Bedeutung der Attribute:

- `frame`: spezifiziert den Namen eines Frames.

Beispiel:

```
<cm-request...>
  <content-where>
    <id>563</id>
  </content-where>
  <content-get>
    <exportBlobForFrame frame="header"/>
  </content-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <content>
      <exportBlobForFrame frame="header">
        Die String-Repräsentation des Blobs
      </exportBlobForFrame>
    </content>
  </cm-code>
</cm-response>
```

exportFiles

Bedeutung: Eine Liste, die für jede beim Export der Version erzeugte Datei ein Wertepaar enthält. Der erste Wert des Paars ist der Dateiname (ohne Pfad), der zweite Wert ist der zu dieser Datei gehörende Inhalt.

Definition:

```
<!ELEMENT exportFiles (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

externalAttributes

Bedeutung: Die Liste der Namen und Werte aller zusaätzlichen (kundenspezifischen) Felder, die der Version zugeordnet sind.

Definition:

```
<!ELEMENT externalAttributes (dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

externalAttrNames

Bedeutung: Die Liste der Namen aller kundenspezifischen Felder, die der Version zugeordnet sind. Die Liste wird über die Vorlage der Datei ermittelt, der die Version zugeordnet ist.

Definition:

```
<!ELEMENT externalAttrNames ((attribute)* | (listitem)*)>
<!ELEMENT attribute (%cm.atom; | %cm.attribute-get;)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.attribute-get`: siehe <attribute-where> <attribute-get> oder CRUL als DTD.

`frameNames`

Bedeutung: Die Liste der Namen aller beim Export der Version entstehenden Frames.

Definition:

```
<!ELEMENT frameNames (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`freeLinks`

Bedeutung: Die Liste der IDs aller in der Version enthaltenen freien Links.

Definition:

```
<!ELEMENT freeLinks ((link)* | (listitem)*)>
<!ELEMENT link (%cm.link-get;)>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.link-get`: siehe <link-where> <link-get> oder CRUL als DTD.

`getKeys`

Bedeutung: Die Liste der mit `content-get` abfragbaren Versionsfelder.

Definition:

```
<!ELEMENT getKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

hasThumbnail

Bedeutung: Gibt an, ob die Version ein Thumbnail hat.

Definition:

```
<!ELEMENT hasThumbnail (%cm.atom;)>
```

height

Bedeutung: Die Höhe des Bildes bei Versionen, die zu Dateien vom Typ `image` gehören, sofern das Bild eines der unterstützen Formate (GIF, JPG, PNG) hat, andernfalls 0.

Definition:

```
<!ELEMENT height (%cm.atom;)>
```

id

Bedeutung: Die ID der Version.

Definition:

```
<!ELEMENT id (%cm.atom;)>
```

isCommitted

Bedeutung: Gibt an, ob es sich um eine eingereichte Version handelt.

Definition:

```
<!ELEMENT isCommitted (%cm.atom;)>
```

isComplete

Bedeutung: Gibt an, ob die Version vollständig ist.

Definition:

```
<!ELEMENT isComplete (%cm.atom;)>
```

isEdited

Bedeutung: Gibt an, ob es sich um eine Arbeitsversion handelt.

Definition:

```
<!ELEMENT isEdited (%cm.atom;)>
```

isReleased

Bedeutung: Gibt an, ob es sich um eine freigebene Version handelt.

Definition:

```
<!ELEMENT isReleased (%cm.atom;)>
```

lastChanged

Bedeutung: Das Datum der letzten Änderung der Version.

Definition:

```
<!ENTITY % cm.date " (%cm.atom; | isoDateTime | systemConfigFormattedTime |
  userConfigFormattedTime)*">
<!ELEMENT lastChanged (%cm.date;)>
<!ATTLIST lastChanged  type CDATA #IMPLIED>
```

Bedeutung der Attribute:

- `type`: Der Typ des Feldes `lastChanged`.

Beispiel:

```
<cm-request...>
  <content-where>
    <id>123</id>
  </content-where>
  <content-get>
    <lastChanged type="date"/>
  </content-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <content>
      <lastChanged type="date">
        <isoDateTime>20011212141420</isoDateTime>
        <systemConfigFormattedTime>12.12.2001 14:14 MET</systemConfigFormattedTime>
        <userConfigFormattedTime>12.12.2001 08:14 EST</userConfigFormattedTime>
      </lastChanged>
    </content>
  </cm-code>
</cm-response>
```

linkListAttributes

Bedeutung: Die Liste der Namen der Felder vom Typ `linklist`, die der Version zugeordnet sind.

Definition:

```
<!ELEMENT linkListAttributes ((attribute)* | (listitem)*)>
<!ELEMENT attribute (%cm.atom; | %cm.attribute-get;)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.attribute-get`: siehe <attribute-where> <attribute-get> oder CRUL als DTD.

mimeType

Bedeutung: Gibt an, welche Namenserweiterungen zulässig sind, wenn eine Datei als Haupttext einer Version importiert wird. Die erlaubten Dateiendungen sind vom Typ der Datei abhängig, zu der die Version gehört.

Definition:

```
<!ELEMENT mimeType (%cm.atom;)>
```

nextEditGroup

Bedeutung: Der Name der nächsten Gruppe im Bearbeitungsworkflow.

Definition:

```
<!ELEMENT nextEditGroup (%cm.atom; | %cm.group-get;)*>
```

`cm.group-get`: siehe <group-where> <group-get> oder CRUL als DTD.

nextSignGroup

Bedeutung: Der Name der nächsten Gruppe im Prüfungsworkflow.

Definition:

```
<!ELEMENT nextSignGroup (%cm.atom; | %cm.group-get;)*>
```

`cm.group-get`: siehe <group-where> <group-get> oder CRUL als DTD.

objectId

Bedeutung: Die ID der zur Version gehörenden Datei.

Definition:

```
<!ELEMENT objectId (%cm.atom; | %cm.obj-get;)*>
```

`cm.obj-get`: siehe <obj-where> <obj-get> oder CRUL als DTD.

reasonsForIncompleteState

Bedeutung: Die Liste der Gründe, aus denen eine Arbeitsversion unvollständig ist.

Definition:

```
<!ELEMENT reasonsForIncompleteState (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

setKeys

Bedeutung: Die Liste der mit `content-set` setzbaren Versionsfelder.

Definition:

```
<!ELEMENT setKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

signatureAttributes

Bedeutung: Die Liste der Namen und Werte der Unterschriftsfelder, die der Version zugeordnet sind. Die Liste wird aus dem Workflow der zur Version gehörenden Datei ermittelt.

Definition:

```
<!ELEMENT signatureAttributes (namevalue)*>
<!ELEMENT namevalue (name, value)>
<!ELEMENT name (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

signatureAttrNames

Bedeutung: Die Liste der Namen der Unterschriftenfelder, die der Version zugeordnet sind. Die Liste wird aus dem Workflow der zur Version gehörenden Datei ermittelt.

Definition:

```
<!ELEMENT signatureAttrNames ((attribute)* | (listitem)*)>
<!ELEMENT attribute (%cm.atom; | %cm.attribute-get;)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.attribute-get`: siehe <attribute-where> <attribute-get> oder CRUL als DTD.

sortKey1

Bedeutung: Erste Sortierschlüsselkomponente (nur bei Ordnern setzbar).

Definition:

```
<!ELEMENT sortKey1 (%cm.atom;)>
```

sortKey2

Bedeutung: Zweite Sortierschlüsselkomponente (nur bei Ordnern setzbar).

Definition:

```
<!ELEMENT sortKey2 (%cm.atom;)>
```

sortKey3

Bedeutung: Dritte Sortierschlüsselkomponente (nur bei Ordnern setzbar).

Definition:

```
<!ELEMENT sortKey3 (%cm.atom;)>
```

sortKeyLength1

Bedeutung: Anzahl der signifikanten Zeichen der ersten Sortierschlüsselkomponente (nur bei Ordnern setzbar).

Definition:

```
<!ELEMENT sortKeyLength1 (%cm.atom;)>
```

sortKeyLength2

Bedeutung: Anzahl der signifikanten Zeichen der zweiten Sortierschlüsselkomponente (nur bei Ordnern setzbar).

Definition:

```
<!ELEMENT sortKeyLength2 (%cm.atom;)>
```

sortKeyLength3

Bedeutung: Anzahl der signifikanten Zeichen der dritten Sortierschlüsselkomponente (nur bei Ordnern setzbar).

Definition:

```
<!ELEMENT sortKeyLength3 (%cm.atom;)>
```

sortOrder

Bedeutung: Sortierrichtung (nur bei Ordnern).

Definition:

```
<!ELEMENT sortOrder (%cm.atom;)>
```

sortType1

Bedeutung: Sortiermodus der ersten Sortierschlüsselkomponente (nur bei Ordnern).

Definition:

```
<!ELEMENT sortType1 (%cm.atom;)>
```

sortType2

Bedeutung: Sortiermodus der zweiten Sortierschlüsselkomponente (nur bei Ordnern).

Definition:

```
<!ELEMENT sortType2 (%cm.atom;)>
```

sortType3

Bedeutung: Sortiermodus der dritten Sortierschlüsselkomponente (nur bei Ordnern).

Definition:

```
<!ELEMENT sortType3 (%cm.atom;)>
```

subLinks

Bedeutung: Die Liste der IDs der in der Version vorkommenden Links.

Definition:

```
<!ELEMENT subLinks ((link)* | (listitem)*)>
<!ELEMENT link (%cm.link-get;)>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.link-get`: siehe <link-where> <link-get> oder CRUL als DTD.

thumbnail

Bedeutung: Ein base-64-kodiertes Vorschaubild im JPEG-Format (siehe auch das Handbuch zur *Systemadministration* / *Entwicklung*). Nur verfügbar bei Dateien vom Typ `image` und `generic`.

Definition:

```
<!ELEMENT thumbnail (%cm.atom;)>
```

title

Bedeutung: Der Titel der Version.

Definition:

```
<!ELEMENT title (%cm.atom;)>
<!ATTLIST title lang (en | de | it | fr | es) #IMPLIED>
```

Bedeutung der Attribute:

- `lang`: Kennzeichnet die Sprache eines Versionstitels. Zu Dimensionen von Werten im CMS siehe Dimensionen von Werten.

Beispiel:

```
<cm-request...>
  <content-where>
    <id>123</id>
  </content-where>
  <content-get>
    <title lang="en"/>
  </content-get></cm-request> <cm-response...>
  <cm-code numeric="0" phrase="ok">
    <content>
      <title lang="en">summer and winter</title>
    </content>
  </cm-code>
</cm-response>
```

validFrom

Bedeutung: Das Datum des Beginns der Gültigkeit der Version. Bei Layouts wird dieser Wert ignoriert.

Definition:

```
<!ENTITY % cm.date " (%cm.atom; | isoDateTime | systemConfigFormattedTime |
  userConfigFormattedTime)*">
  <!ELEMENT validFrom (%cm.date;)>
  <!ATTLIST validFrom type CDATA #IMPLIED>
```

Bedeutung der Attribute:

- `type`: Der Typ des Feldes validFrom.

Beispiel:

```
<cm-request...>
  <content-where>
    <objectId>32875</objectId>
    <state>edited</state>
  </content-where>
  <content-get>
    <validFrom type="date"/>
  </content-get></cm-request> <cm-response...>
  <cm-code numeric="0" phrase="ok">
    <content>
      <validFrom type="date">
        <isoDateTime>20010401121207</isoDateTime>
        <systemConfigFormattedTime>01.04.2001 12:12 MET</systemConfigFormattedTime>
        <userConfigFormattedTime>01.04.2001 06:12 EST</userConfigFormattedTime>
      </validFrom>
    </content>
  </cm-code>
</cm-response>
```

validSortKeys

Bedeutung: Die Liste aller gültigen Werte für die Felder `sortKeyn` .

Definition:

```
<!ELEMENT validSortKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

validSortOrders

Bedeutung: Die Liste aller gültigen Werte für das Feld `sortOrder`.

Definition:

```
<!ELEMENT validSortOrders (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

validSortTypes

Bedeutung: Die Liste aller gültigen Werte für die Felder `sortTypen` .

Definition:

```
<!ELEMENT validSortTypes (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

validUntil

Bedeutung: Das Datum des Endes der Gültigkeit der Version. Bei Layouts wird dieser Wert ignoriert.

Definition:

```
<!ENTITY % cm.date " (%cm.atom; | isoDateTime | systemConfigFormattedTime |
  userConfigFormattedTime)*">
<!ELEMENT validUntil (%cm.date;)>
<!ATTLIST validUntil  type CDATA #IMPLIED>
```

Bedeutung der Attribute:

- `type`: Der Typ des Feldes validUntil.

Beispiel:

```
<cm-request...>
  <content-where>
    <id>3287.5</id>
  </content-where>
  <content-get>
    <validUntil type="date"/>
  </content-get></cm-request> <cm-response...>
  <cm-code numeric="0" phrase="ok">
    <content>
      <validUntil type="date">
        <isoDateTime>20051231060002</isoDateTime>
        <systemConfigFormattedTime>31.12.2005 06:00 MET</systemConfigFormattedTime>
        <userConfigFormattedTime>30.12.2005 00:00 EST</userConfigFormattedTime>
      </validUntil>
    </content>
  </cm-code>
</cm-response>
```

width

Bedeutung: Die Breite des Bildes bei Versionen, die zu Dateien vom Typ `image` gehören, sofern das Bild eines der unterstützen Formate (GIF, JPG, PNG) hat, andernfalls 0.

Definition:

```
<!ELEMENT width (%cm.atom;)>
```

workflowComment

Bedeutung: Der Kommentar zu der letzten Workflow-Aktion, die auf diese Version angewendet wurde.

Definition:

```
<!ELEMENT workflowComment (%cm.atom;)>
```

xmlBlob

Bedeutung: String-Repräsentation des Blobs der Version als XML-Dokument (nicht verfügbar bei Layouts).

Definition:

```
<!ELEMENT xmlBlob (%cm.atom;)>
```

15.3 Funktionselemente für Versionen

15.3.1 <content-where>

Ausschnitt aus der DTD:

```
<!ENTITY % cm.cm-request "
  ...
  (content-where+, content-addLinkTo) |
  (content-where+, content-debugExport) |
  (content-where+, content-delete) |
  (content-where+, content-description) |
  (content-where+, content-get) |
  (content-where+, content-load) |
  (content-where+, content-resolveRefs) |
  (content-where+, content-set) |
  ...
">
```

Definition:

```
<!ENTITY % cm.content-where "
  (id |
  (objectId,
  state))
">
```

Aufgabe: content-where ermöglicht die Suche nach Versionen, bei denen der Wert der angegebenen Parameter den jeweils spezifizierten String enthält. Werden keine Parameterwerte angegeben, so werden die IDs aller Versionen zurückgegeben.

Verwendung: `content-where` ist ein Funktionselement, das nicht ohne ein weiteres unmittelbar darauf folgendes Funktionselement verwendet werden kann. Mit `content-where` erhält man die Liste der Versionen, die dem Suchkriterium entsprechen. Erst durch das folgende Funktionselement wird jedoch bestimmt, wie diese zu behandeln sind. Jedes der Elemente, die auf `content-where` folgen können, entspricht dabei einer Schablone, die nacheinander auf jeden der mit `content-where` ermittelten Versionen angewendet wird (siehe Requests als Schablonen und die Darstellung der Ergebnis-Daten).

Erforderliche Rechte: keine Einschränkungen.

15.3.2 <content-where> <content-addLinkTo>

Definition:

```
<!ENTITY % cm.content-addLinkTo "
  (attribute |
  destinationUrl |
  target |
  title |
  position)*
">
<!ELEMENT content-addLinkTo %cm.content-addLinkTo;>
```

Aufgabe: content-addLinkTo erzeugt einen freien Link und fügt ihn zum angegebenen Linklistfeld einer Arbeitsversion hinzu.

Spezielle Parameter dieses Funktionselements:

- `attribute`: spezifiziert den Namen des Linklistfeldes, zu dem der freie Link hinzugefügt werden soll. Das betreffende Feld muss vom Typ `linklist` sein.
- `destinationUrl`: enthält das Linkziel.
- `title`: enthält den Titel des Links.
- `target`: enthält den Zielrahmen.
- `position`: (ab Version 6.6.1) enthält den Sortierwert des Links.

Verwendung: Zunächst wird mit `content-where` die Version ermittelt. Anschließend wird `content-addLinkTo` verwendet, um einem Linklistfeld der Version einen freien Link hinzuzufügen.

Rückgabewert bei Erfolg: die ID des erzeugten Links.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionWrite` für die Datei haben, zu dem die Version gehört und Bearbeiter der Version sein.

Beispiel:

```
<cm-request...>
  <content-where>
    <objectId>116624</objectId>
    <state>edited</state>
  </content-where>
  <content-addLinkTo>
    <attribute>relatedLinks</attribute>
    <destinationUrl>http://www.flower.com</destinationUrl>
  </content-addLinkTo>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <link>
      <id>116624.54.6</id>
    </link>
  </cm-code>
</cm-response>
```

15.3.3 <content-where> <content-debugExport>

Definition:

```
<!ELEMENT content-debugExport ANY>
<!ATTLIST content-debugExport
          allowEditedContents (true | false)
          detailed (true | false) "false"
          quoteHtml (true | false) "false"
          htmlPrefix CDATA #IMPLIED
          htmlSuffix CDATA #IMPLIED
          infoPrefix CDATA #IMPLIED
          infoSuffix CDATA #IMPLIED
          errorPrefix CDATA #IMPLIED
          errorSuffix CDATA #IMPLIED
          preferEditedTemplates (true | false)
          templateName CDATA #IMPLIED
```

```
>
```

Aufgabe: content-debugExport simuliert den Export der Version mit der angegebenen ID zu dem Zweck, fehlerhaften Layout-Code ausfindig zu machen. Aus der Ausgabe ist ersichtlich, welche NPSOBJ-Tags aus welchen Quell-Layouts in welcher Reihenfolge ausgewertet werden und welche Fehler bei der Auswertung aufgetreten sind.

Spezielle Parameter dieses Funktionselements:

- `allowEditedContents`: Gibt an, ob bei Dateien, die in der betreffenden Version referenziert werden, auf Arbeitsversionen zurückgegriffen werden darf, wenn freigegebene nicht verfügbar sind.
- `detailed`: Gibt an, ob sich die Ausgabe nur auf Fehlermeldungen beschränken (`false`) oder ausführlich sein soll (`true`). Der voreingestellte Wert ist `false`.
- `errorPrefix`: Gibt eine Zeichenkette an, die vor jeder Fehlermeldung ausgegeben werden soll. Der voreingestellte Wert ist „*** ".
- `errorSuffix`: Gibt eine Zeichenkette an, die nach jeder Fehlermeldung ausgegeben werden soll (voreingestellt leer).
- `htmlPrefix`: Gibt eine Zeichenkette an, die vor HTML-Text ausgegeben werden soll. Der voreingestellte Wert ist „<pre>".
- `htmlSuffix`: Gibt eine Zeichenkette an, die nach HTML-Text ausgegeben werden soll. Der voreingestellte Wert ist „</pre>".
- `infoPrefix`: Gibt eine Zeichenkette an, die vor jeder NPSOBJ-Information ausgegeben werden soll (voreingestellt leer).
- `infoSuffix`: Gibt eine Zeichenkette an, die nach jeder NPSOBJ-Information ausgegeben werden soll (voreingestellt leer).
- `preferEditedTemplates`: Gibt an, ob die Arbeitsversionen der Layouts gegenüber den freigegebenen Versionen bevorzugt verwendet werden sollen. Der voreingestellte Wert entspricht dem gleichnamigen Wert aus den Benutzereinstellungen, der auch für die Vorschau verwendet wird.
- `quoteHtml`: Gibt an, ob die Zeichen <, > und & als HTML-Entities ausgegeben werden sollen (`true`) oder nicht (`false`). Die Präfix- und Suffix-Parameter sind hiervon nicht betroffen. Der voreingestellte Wert ist `false`.
- `templateName`: Gibt an, mit welchem initialen Layout (Basislayout) der Exporttest durchgeführt werden soll. Ist `templateName` nicht angegeben, wird das benutzerspezifische Standard-Layout verwendet (voreingestellt `mastertemplate`).

Verwendung: Zunächst wird mit `content-where` die Version ermittelt. Anschließend wird `content-debugExport` verwendet, um den Export-Bericht zu erzeugen.

Rückgabewert bei Erfolg: der formatierte Exportbericht (string).

Erforderliche Rechte: der Benutzer muss das Recht `permissionGlobalExport` oder `permissionRead` für die zur Version gehörende Datei haben.

Beispiel:

```
<cm-request...>
  <content-where>
    <objectId>116624</objectId>
    <state>edited</state>
  </content-where>
  <content-debugExport quoteHtml="true"
```

```
    htmlPrefix="&lt;pre&gt;"
    htmlSuffix="&lt;/pre&gt;"
    infoPrefix="&lt;font
    color=&quot;blue&quot;&gt;"
    infoSuffix="&lt;/font&gt;&lt;br&gt;"
    errorPrefix="&lt;font color=&quot;red&quot;&gt;"
    errorSuffix="&lt;/font&gt;&lt;br&gt;" />
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <content-debugExport>
      <dictitem>
        <key>hallo.html</key>
        <value>
          &lt;font color="red"&gt; FEHLER [140015]
          In der NPSOBJ-Anweisung 'modifyvar' fehlen die folgenden Attribute:
          varname&lt;/font&gt;&lt;br&gt;&lt;font
          color="red"&gt; FEHLER [140012] Die Export-Variable
          oder das Schlüsselwort 'contentTypeAndCharset' in der
          NPSOBJ-Anweisung 'insertvalue' ist unbekannt.
          &lt;/font&gt;&lt;br&gt;
        </value>
      </dictitem>
    </content-debugExport>
  </cm-code>
</cm-response>
```

15.3.4 <content-where> <content-delete>

Definition:

```
<!ENTITY % cm.content-delete "EMPTY">
<!ELEMENT content-delete %cm.content-delete;>
```

Aufgabe: content-delete löscht eine archivierte oder freigegebene Version. Verwenden Sie `obj-revert`, um eine Arbeitsversion zu löschen (siehe <obj-where> <obj-revert>).

Verwendung: Zunächst wird die Version mit `content-where` ermittelt. Anschließend wird sie mit `content-delete` gelöscht.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionRoot` für die Datei haben, zu der die Version gehört.

Beispiel:

```
<cm-request...>
  <content-where>
    <objectId>9248</objectId>
    <state>released</state>
  </content-where>
  <content-delete/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

15.3.5 <content-where> <content-description>

Definition:

```
<!ENTITY % cm.atom "#PCDATA">
<!ELEMENT content-description %cm.atom;>
```

Aufgabe: `content-description` liefert eine Zeichenketten-Repräsentation der wichtigsten Versionsparameter.

Verwendung: Zunächst wird mit `content-where` die Version ermittelt, deren Zeichenketten-Repräsentation anschließend mit `content-description` ausgelesen wird.

Rückgabewert bei Erfolg: die Zeichenketten-Repräsentation.

Erforderliche Rechte: keine.

Beispiel:

```
<cm-request...>
  <content-where>
    <id>87569.16</id>
  </content-where>
  <content-description/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok>
    {
      contentType = html;
      editor = katrin;
      sortKeyLength1 = 50;
      sortKeyLength2 = 50;
      sortKeyLength3 = 50;
      sortOrder = ascending;
      sortType1 = alphaNumeric;
      sortType2 = alphaNumeric;
      sortType3 = alphaNumeric;
      title = untitled;
      validFrom = 20011216230000;
    }
  </cm-code>
</cm-response>
```

15.3.6 <content-where> <content-get>

Definition:

```
<!ENTITY % cm.content-get "
  (anchors |
  blob |
  body |
  bodyTemplateName |
  channels |
  codeForActionPreview |
  codeForPreview |
  codeForSourceView |
  codeForThumbnail |
  contentType |
  displayTitle |
```

```
   editor |
   exportBlob |
   exportBlobForFrame |
   exportFiles |
   externalAttrNames |
   externalAttributes |
   frameNames |
   freeLinks |
   getKeys |
   height |
   hasThumbnail |
   id |
   isCommitted |
   isComplete |
   isEdited |
   isReleased |
   lastChanged |
   linkListAttributes |
   mimeType |
   nextEditGroup |
   nextSignGroup |
   objectId |
   reasonsForIncompleteState |
   setKeys |
   signatureAttrNames |
   signatureAttributes |
   sortKey1 |
   sortKey2 |
   sortKey3 |
   sortKeyLength1 |
   sortKeyLength2 |
   sortKeyLength3 |
   sortOrder |
   sortType1 |
   sortType2 |
   sortType3 |
   subLinks |
   thumbnail |
   thumbnailSize |
   title |
   validFrom |
   validSortKeys |
   validSortOrders |
   validSortTypes |
   validUntil |
   width |
   workflowComment |
   xmlBlob)*
">

<!ELEMENT content-get %cm.content-get;>
```

Aufgabe: Liefert den Wert der angegebenen Versionsparameter für jede mit `content-where` ermittelte Version.

Verwendung: `content-get` muss immer auf `content-where` folgen, da zunächst die Versionen ermittelt werden müssen, auf die `content-get` angewendet werden soll, um Parameterwerte auszulesen.

Zusatzinformationen: Anders als bei `content-set` können bei `content-get` die Art der Übertragung und das Speicherformat des Blobs nicht vorgegeben werden. Wie der Server den Blob zurückgibt, zeigt er in der Antwort mit Hilfe des Attributs `encoding` beim Element `blob` an. Zur Erläuterung des Attributs und seiner Werte siehe <obj-create>. Mit dem Systemkonfigurationseintrag `tuning.minStreamingDataLength` lässt sich die Mindestgröße in Kilobytes festlegen, die ein Blob haben muss, damit der Server ihn per Streaming überträgt.

Rückgabewert bei Erfolg: der Wert der angegebenen Parameter.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionRead` für die Datei haben, zu der die Versionen gehören. Für die Abfrage des Feldes `exportBlob` genügt auch das Recht `permissionGlobalExport`.

Beispiel:

```
<cm-request...>
  <content-where>
    <id>5628.14</id>
  </content-where>
  <content-get>
    <editor/>
  </content-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <content>
      <editor>mustermann</editor>
    </content>
  </cm-code>
</cm-response>
```

15.3.7 <content-where> <content-load>

Definition:

```
<!ENTITY % cm.content-load "
  (blob |
  contentType |
  filter)*
">
<!ELEMENT content-load %cm.content-load;>
```

Aufgabe: Lädt den in den Parameterelementen spezifizierten Blob in die mit `content-where` ermittelte Version.

Spezielle Parameter dieses Funktionselements:

- `filter`: der beim Laden des Blobs zu verwendende Konverter. Dieser Parameter ist aus Gründen der Kompatibilität noch vorhanden, wird jedoch ignoriert.
- `blob`: siehe <obj-create>.

Verwendung: Zunächst wird die gewünschte Version mit `content-where` ermittelt. Anschließend wird `content-load` verwendet, um den spezifizierten Blob in diese Version zu laden.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionWrite` für die Datei haben, zu der die Version. Er muss der Bearbeiter der Datei sein.

Beispiel:

```
<cm-request...>
  <content-where>
    <objectId>7558</objectId>
    <state>edited</state>
  </content-where>
```

```
  <content-load>
    <contentType>html</contentType>
    <blob>/Users/NPS/Upload/news.htm</blob>
  </content-load>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

15.3.8 <content-where> <content-resolveRefs>

Definition:

```
<!ENTITY % cm.content-resolveRefs "EMPTY">
<!ELEMENT content-resolveRefs %cm.content-resolveRefs;>
```

Aufgabe: Löst die im Blob und in HTML-Feldern der mit `content-where` ermittelten Versionen enthaltenen URLs auf und ordnet ihnen die entsprechenden Links zu.

Verwendung: Zunächst werden mit `content-where` Versionen ermittelt. Anschließend wird `content-resolveRefs` verwendet, um Links im Blob und den HTML-Feldern dieser Versionen aufzulösen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss Bearbeiter der Versionen sein und das Recht `permissionWrite` für die Datei haben, zu der die Versionen gehören.

Beispiel:

```
<cm-request...>
  <content-where>
    <id>7463.33</id>
  </content-where>
  <content-resolveRefs/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

15.3.9 <content-where> <content-set>

Definition:

```
<!ENTITY % cm.content-set "
  (blob |
  contentType |
  channels |
  sortKey1 |
  sortKey2 |
  sortKey3 |
  sortKeyLength1 |
  sortKeyLength2 |
  sortKeyLength3 |
  sortOrder |
```

```
  sortType1 |
  sortType2 |
  sortType3 |
  title |
  validFrom |
  validUntil)*
">
<!ELEMENT content-set %cm.content-set;>
```

Aufgabe: Setzt Parameter auf die angegebenen Werte für jede mit `content-where` ermittelte Version.

Verwendung: `content-set` muss immer auf `content-where` folgen, da zunächst die Versionen ermittelt werden müssen, auf die `content-set` angewendet werden soll, um Parameterwerte zu setzen. Bitte beachten Sie die Details zum Parameter `blob`, die im Abschnitt <obj-create> zu finden sind.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss Bearbeiter der Versionen sein und das Recht `permissionWrite` für die Datei haben, zu der die Versionen gehören.

Beispiel:

```
<cm-request...>
  <content-where>
    <id>2317.19</id>
  </content-where>
  <content-set>
    <contentType>htm</contentType>
  </content-set>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

16

16 Inkrementeller Export - incrExport

Der inkrementelle Export ist ein Verfahren, mit dem ein Webserver mit dem Content Manager automatisch synchronisiert werden kann. Zu diesem Zweck ist es erforderlich, auf der Seite des Webservers die Template Engine einzusetzen.

Der Content Management Server übergibt der Template Engine über das XML-Interface laufend aktualisierte Versionen und weist sie in regelmäßigen, konfigurierbaren Abständen an, diese zu exportieren und live zu schalten. Aktualisierte Daten wie auch Anweisungen werden in Form so genannter Update-Records vom Content Management Server zur Template Engine übertragen.

Um die `incrExport`-Funktionselemente in der XML-Schnittstelle verwenden zu können, muss man das Recht `permissionGlobalExport` haben oder ein Superuser sein.

16.1 Definition

Der inkrementelle Export kann mit Requests zurückgesetzt werden, und es ist möglich, unter anderem den Modus abzufragen:

```
<!ENTITY % cm.cm-request "
  ...
  (incrExport-get) |
  (incrExport-getUpdateData) |
  (incrExport-removeUpdateRecords) |
  (incrExport-reset) |
  ...
">
```

Diese Elemente müssen sich unmittelbar unterhalb des `cm-request`-Elements befinden. Sie werden im Abschnitt Funktionselemente für den inkrementellen Export erläutert.

16.2 Parameterelemente für den inkrementellen Export

Um zu spezifizieren welche Eigenschaften des inkrementellen Exports ausgelesen oder gesetzt werden sollen, verwendet man Parameterelemente. Im Folgenden werden die Parameterelemente für den inkrementellen Export aufgeführt.

`mode`

Bedeutung: Liefert den Modus des inkrementellen Exports, `on` oder `off`. `on`: sämtliche Änderungen werden zur Template Engine übertragen. `off`: Änderungen werden nicht protokolliert. Der Modus

kann mit dem Systemkonfigurationseintrag `export.incrementalUpdate.isActive` eingestellt werden.

Definition:

```
<!ELEMENT mode (%cm.atom;)>
```

getKeys

Bedeutung: Liste der mit `incrExport-get` abfragbaren Parameter.

Definition:

```
<!ELEMENT getKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

updateRecordCount

Bedeutung: Liefert im Modus `active` die Anzahl der Update-Records, die zur Übertragung zur Template Engine anstehen. Im Modus `suspended` liefert `updateRecordsCount` den Wert `0` und im Modus `off` wird eine Fehlermeldung erzeugt.

Definition:

```
<!ELEMENT updateRecordsCount (%cm.atom;)>
```

updateRecords

Bedeutung: Liefert im Modus (`mode`) `active` eine Liste mit Update-Records. Jeder Update-Record ist selbst eine Liste, die aus zwei Elementen besteht, `updateRecordId` und `updateType`. Im Modus `suspended` liefert `updateRecords` die leere Liste und im Modus `off` den Fehlertext "incremental Export is turned off". Die zurück gegebenen Update-Records werden nicht aus der Liste der Update-Records gestrichen. Dies muss explizit mit dem Befehl `incrExport removeUpdateRecords` ausgelöst werden.

Definition:

```
<!ELEMENT updateRecords (%cm.atom;)>
```

16.3 Funktionselemente für den inkrementellen Export

16.3.1 <incrExport-get>

Definition:

```
<!ENTITY % cm.incrExport-get "
  (getKeys |
  mode |
  updateRecords |
  updateRecordsCount)*
">
<!ELEMENT incrExport-get %cm.incrExport-get;>
```

Aufgabe: incrExport-get liefert den Wert der angegebenen `incrExport`-Parameter.

Rückgabewert bei Erfolg: der Wert der angegebenen Parameter.

Erforderliche Rechte: Der authentifizierte Benutzer muss Superuser sein oder das Recht `permissionGlobalExport` haben.

Beispiel:

```
<cm-request...>
  <incrExport-get>
    <mode/>
  </incrExport-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <incrExport>
      <mode>suspended</mode>
    </incrExport>
  </cm-code>
</cm-response>
```

16.3.2 <incrExport-getUpdateData>

Definition:

```
<!ELEMENT incrExport-getUpdateData %cm.incrExport-getUpdateData;>
<!ATTLIST incrExport-getUpdateData
          updateRecordId CDATA #REQUIRED
>
<!ENTITY % cm.incrExport-getUpdateData "
  (updateData?)
">
```

Aufgabe: Liefert alle für den Export notwendigen Daten der Datei, die zum Update-Record mit der angegebenen ID gehört. Die Daten werden im XML-Format so geliefert, dass sie von der Template Engine verarbeitet werden können.

Bedeutung der Attribute:

- `updateRecordId`: die ID des Update-Record.

Spezielle Parameter dieses Funktionselements:

- `updateData`: enthält die zum Update-Record gehörenden Daten der Datei.

Rückgabewert bei Erfolg: Das entsprechende XML-Dokument, das die Daten enthält. Das Wurzelelement des Dokuments ist `updateData`.

Erforderliche Rechte: Der authentifizierte Benutzer muss Superuser sein oder das Recht `permissionGlobalExport` haben.

Beispiel:

```
<cm-request...>
  <incrExport-getUpdateData updateRecordId="2144"/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <incrExport-getUpdateData>
      <updateData>
        <updateRecordId>2144</updateRecordId>
        <updateType>4</updateType>
        <content>
          <contentType>html</contentType>
          <mimeType>text/html</mimeType>
          <lastChanged>20010208173015</lastChanged>
          <title>untitled</title>
          <validFrom>20010207230000</validFrom>
          <validUntil/>
          <encodedBlob>RGFzIGlzdCB...Y2hhZnRlbi4=</encodedBlob>
          <externalAttributes>
            <dictitem>
              <key>relatedLinks</key>
              <value/>
            </dictitem>
            <dictitem>
              <key>linklistattr</key>
              <value/>
            </dictitem>
            <dictitem>
              <key>attr</key>
              <value/>
            </dictitem>
            <dictitem>
              <key>htmlattr</key>
              <value/>
            </dictitem>
            <dictitem>
              <key>attrhtml</key>
              <value/>
            </dictitem>
          </externalAttributes>
          <signatureAttributes>
            <dictitem>
              <key>signattr</key>
              <value>
                <dictitem>
                  <key>timeStamp</key>
                  <value>20010208173015</value>
                </dictitem>
                <dictitem>
                  <key>login</key>
                  <value>root</value>
                </dictitem>
              </value>
            </dictitem>
          </signatureAttributes>
          <externalAttrTypeDict>
            <dictitem>
```

```
              <key>relatedLinks</key>
              <value>linklist</value>
            </dictitem>
            <dictitem>
              <key>linklistattr</key>
              <value>linklist</value>
            </dictitem>
            <dictitem>
              <key>attr</key>
              <value>string</value>
            </dictitem>
            <dictitem>
              <key>htmlattr</key>
              <value>html</value>
            </dictitem>
            <dictitem>
              <key>attrhtml</key>
              <value>html</value>
            </dictitem>
          </externalAttrTypeDict>
          <indexAttributeNames/>
        </content>
        <path>/test_document</path>
      </updateData>
    </incrExport-getUpdateData>
  </cm-code>
</cm-response>
```

16.3.3 <incrExport-removeUpdateRecords>

Definition:

```
<!ELEMENT incrExport-removeUpdateRecords
          %cm.incrExport-removeUpdateRecords;>
<!ENTITY % cm.incrExport-removeUpdateRecords "
  (updateRecordId+ |
  all)
">
```

Aufgabe: Löscht die Update-Records mit den angegebenen IDs sowie alle Detail-Daten wie die zu den Update-Records gehörenden XML-Dateien.

Spezielle Parameter dieses Funktionselements:

- `updateRecordId`: enthält die Id eines Update-Record.
- `all`: alle Update-Records.

Rückgabewert bei Erfolg: die Anzahl der gelöschten Update-Records.

Erforderliche Rechte: Der authentifizierte Benutzer muss Superuser sein oder das Recht `permissionGlobalExport` haben.

Beispiel:

```
<cm-request...>
  <incrExport-removeUpdateRecords>
    <updateRecordId>3461</updateRecordId>
    <updateRecordId>3462</updateRecordId>
    <updateRecordId>8867</updateRecordId>
  </incrExport-removeUpdateRecords>
</cm-request>
```

```
<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <incrExport-removeUpdateRecords>3
    </incrExport-removeUpdateRecords>
  </cm-code>
</cm-response>
```

16.3.4 <incrExport-reset>

Definition:

```
<!ENTITY % cm.incrExport-reset "EMPTY">
<!ELEMENT incrExport-reset %cm.incrExport-reset;>
```

Aufgabe: Löscht die interne Tabelle der Update-Records und erzeugt sie neu. Die Zeit, die der Content Management Server benötigt, um den Befehl abzuarbeiten, ist proportional zur Anzahl der Dateien.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss Superuser sein oder das Recht `permissionGlobalExport` haben.

Beispiel:

```
<cm-request...>
  <incrExport-reset/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

17

17 Jobs - job

Mit dem Job-Befehl lassen sich kundenspezifische Tcl-Skripte einmalig oder wiederholt ausführen. Der Content Management Server unterscheidet zwischen System-Jobs und Benutzer-Jobs. System-Jobs sind vordefinierte Routinen, etwa zur regelmäßigen Übertragung von Update-Records zur Template Engine im Rahmen des inkrementellen Live-Server-Updates. Sie können weder gelöscht noch geändert werden. Benutzer-Jobs dagegen können bedarfsweise angelegt, modifiziert und auch wieder gelöscht werden. Beide Job-Kategorien teilen sich einen Namensraum, d. h. zwei Jobs können auch dann nicht den gleichen Namen haben, wenn sie zu unterschiedlichen Kategorien gehören.

Die Jobs einer Kategorie werden zum Ausführungszeitpunkt in die Warteschlange dieser Kategorie aufgenommen, sofern sie nicht bereits aufgenommen wurden. Dadurch ist sicher gestellt, dass ein häufiger auszuführender Job nicht mehrmals in die Warteschlange gestellt wird, wenn ein anderer oder auch dieser Job gerade abgearbeitet wird.

Alle Jobs einer Kategorie werden sequenziell abgearbeitet, da sie sich eine Warteschlange teilen. Die Jobs unterschiedlicher Kategorien werden parallel abgearbeitet.

Die Ausführung von Jobs wird protokolliert. Je Job legt das System maximal so viele Protokolleinträge an, wie es der Wert des Systemkonfigurationseintrags `tuning.jobMaxLogLength` festgelegt.

17.1 Der Ausführungsplan

Jedem Job ist ein Ausführungsplan (*engl.* schedule = Terminplan) zugeordnet. In diesem Plan sind die (gegebenenfalls wiederkehrenden) Ausführungszeitpunkte des Jobs festgelegt. Jeder Ausführungszeitpunkt besteht aus Angaben über die Jahre (`years`), Monate (`months`), Tage (`days`) oder Wochentage (`weekdays`), die Stunden (`hours`) und die Minuten (`minutes`). Die Angaben müssen nicht vollständig sein, d. h. Bestandteile können weggelassen werden.

Der Ausführungsplan ist eine Liste von Einträgen, von denen jeder wiederum eine Liste von Dictionary-Einträgen (Name-Wert-Paaren) ist (siehe das Beispiel zum Parameterelement `schedule`).

Für die Logik der Wiederholung ist es entscheidend, dass ein fehlender Bestandteil eines Eintrags so interpretiert wird, als wären alle möglichen Werte für diesen Bestandteil angegeben. Führt man beispielsweise nicht die Tage oder Wochentage auf, in denen der Job ausgeführt werden soll, so wird er täglich ausgeführt. Ein leerer Eintrag für einen Ausführungszeitpunkt bewirkt also, dass der Job jede Minute ausgeführt wird, und zwar so lange, bis der Eintrag gelöscht oder geändert wird. Ein einmaliger Ausführungszeitpunkt (ohne Wiederholung) lässt sich definieren, indem für jeden Bestandteil außer `weekdays` genau ein Wert angegeben wird.

Die Stunden werden als Zahlen von 0 bis 23, die Wochentage mit den Zahlen 1 bis 7 angegeben, wobei 1 für Montag, 2 für Dienstag usw. steht.

Alle Einträge in Ausführungsplänen beziehen sich auf die Rechner-Zeit mit ihrer spezifischen Zeitzone.

17.2 Definition

Auf Jobs kann mit Requests an den Content Manager lesend und schreibend zugegriffen werden. Der folgende Ausschnitt aus der DTD des Content Management Servers zeigt die Elemente, mit denen dies möglich ist:

```
<!ENTITY % cm.cm-request "
  ...
  (job-create) |
  (job-listQueue) |
  (job-where+, job-cancel) |
  (job-where+, job-delete) |
  (job-where+, job-description) |
  (job-where+, job-exec) |
  (job-where+, job-get) |
  (job-where+, job-getLogEntry) |
  (job-where+, job-getOutput) |
  (job-where+, job-set) |
  ...
">
<!ATTLIST job-where
  maxResults CDATA #IMPLIED
>
```

Diese Elemente, die sich unmittelbar unterhalb des `cm-request`-Elements befinden müssen, werden im Abschnitt Funktionselemente für Jobs erläutert.

Mit dem Attribut `maxResults`, dessen Wert eine ganze Zahl ist, kann die maximale Anzahl der Treffer festgelegt werden. Ist der Wert von `maxResults` kleiner oder gleich null, ist die Anzahl nicht begrenzt.

17.3 Parameterelemente für Jobs

Mit Hilfe von Funktionselementen kann man auf Jobs zugreifen. So können beispielsweise mit dem `job-get`-Element die Werte sämtlicher Job-Parameter ermittelt werden. Welche Jobeigenschaften ausgelesen oder gesetzt werden sollen, wird durch Parameterelemente spezifiziert. Im Folgenden werden die Job-Parameter und die zugehörigen Parameterelemente aufgeführt.

`category`

Bedeutung: Die Job-Kategorie (`user` oder `system`).

Definition:

```
<!ELEMENT category (%cm.atom;)>
```

`comment`

Bedeutung: Die Beschreibung des Jobs.

Definition:

```
<!ELEMENT comment (%cm.atom;)>
```

displayTitle

Bedeutung: Der in der HTML-Benutzerschnittstelle angezeigte Titel des Jobs (eine Kombination aus Titel und Namen).

Definition:

```
<!ELEMENT displayTitle (%cm.atom;)>
```

execLogin

Bedeutung: Das Login, unter dem das Skript ausgeführt werden kann.

Definition:

```
<!ELEMENT execLogin (%cm.atom;)>
```

execPerm

Bedeutung: Das globale Recht, das ein Benutzer benötigt, um den Job auszuführen, d. h. in die Warteschlange zu stellen. Ist kein Recht angegeben, so dürfen alle Benutzer den Job ausführen.

Definition:

```
<!ELEMENT execPerm (%cm.atom;)>
```

getKeys

Bedeutung: Die Liste der mit `job-get` abfragbaren Parameter.

Definition:

```
<!ELEMENT getKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

id

Bedeutung: Die ID des Jobs.

Definition:

```
<!ELEMENT id (%cm.atom;)>
```

isActive

Bedeutung: Gibt an, ob ein Job aktiv (`1`) oder nicht aktiv (`0`) ist. Nur aktive Jobs können ausgeführt werden.

Definition:

```
<!ELEMENT isActive (%cm.atom;)>
```

lastExecEnd

Bedeutung: Der Zeitpunkt, an dem die letzte Ausführung des Jobs beendet wurde.

Definition:

```
<!ENTITY % cm.date " (%cm.atom; | isoDateTime | systemConfigFormattedTime |
  userConfigFormattedTime)*">
<!ELEMENT lastExecEnd (%cm.date;)>
<!ATTLIST lastExecEnd  type CDATA #IMPLIED>
```

Bedeutung der Attribute:

- `type`: Der Typ des Feldes `lastExecEnd`.

Beispiel:

```
<cm-request...>
  <job-where>
    <execLogin>stan</execLogin>
  </job-where>
  <job-get>
    <lastExecEnd type="date"/>
  </job-get></cm-request> <cm-response...>
  <cm-code numeric="0" phrase="ok">
    <job>
      <lastExecStart type="date">
        <isoDateTime>20010401164700</isoDateTime>
        <systemConfigFormattedTime>01.04.2001 08:47 MET</systemConfigFormattedTime>
        <userConfigFormattedTime>01.04.2001 16:47 EST</userConfigFormattedTime>
      </lastExecStart>
    </job>
  </cm-code>
</cm-response>
```

lastExecResult

Bedeutung: Das Ergebnis der letzten Ausführung des Skripts.

Definition:

```
<!ELEMENT lastExecResult (%cm.atom;)>
```

lastExecStart

Bedeutung: Der Zeitpunkt, an dem die letzte Ausführung des Jobs begonnen wurde.

Definition:

```
<!ENTITY % cm.date " (%cm.atom; | isoDateTime | systemConfigFormattedTime |
  userConfigFormattedTime)*">
<!ELEMENT lastExecStart (%cm.date;)>
<!ATTLIST lastExecStart type CDATA #IMPLIED>
```

Bedeutung der Attribute:

- `type`: Der Typ des Feldes `lastExecStart`.

Beispiel:

```
<cm-request...>
  <job-where>
    <id>5423</id>
  </job-where>
  <job-get>
    <lastExecStart type="date"/>
  </job-get></cm-request> <cm-response...>
  <cm-code numeric="0" phrase="ok">
    <job>
      <lastExecStart type="date">
        <isoDateTime>20010613153502</isoDateTime>
        <systemConfigFormattedTime>13.06.2001 15:35 MET</systemConfigFormattedTime>
        <userConfigFormattedTime>13.06.2001 07:35 EST</userConfigFormattedTime>
      </lastExecStart>
    </job>
  </cm-code>
</cm-response>
```

lastLogEntry

Bedeutung: Der letzte (aktuelle) Protokolleintrag des Jobs.

Definition:

```
<!ENTITY % cm.logEntry "  (logEntryId,  execResult,  execStart,  execEnd)">
<!ELEMENT lastLogEntry %cm.logEntry;>
```

Beispiel:

```
<cm-request...>
  <job-where>
    <id>5423</id>
  </job-where>
  <job-get>
    <lastLogEntry/>
  </job-get></cm-request> <cm-response...>
  <cm-code numeric="0" phrase="ok">
    <job>
      <lastLogEntry>
        <logEntryId>69811.61</logEntryId>
        <execResult />
        <execStart type="date">
          <isoDateTime>20020716130256</isoDateTime>
          <systemConfigFormattedTime>16.07.2002 15:02</systemConfigFormattedTime>
          <userConfigFormattedTime>16.07.2002 15:02</userConfigFormattedTime>
        </execStart>
        <execEnd type="date">
          <isoDateTime>20020716130256</isoDateTime>
          <systemConfigFormattedTime>16.07.2002 15:02</systemConfigFormattedTime>
          <userConfigFormattedTime>16.07.2002 15:02</userConfigFormattedTime>
        </execEnd>
      </lastLogEntry>
    </job>
  </cm-code>
</cm-response>
```

`lastOutput`

Bedeutung: Die letzte vom Job produzierte Ausgabe.

Definition:

```
<!ELEMENT lastOutput (%cm.atom;)>
```

Beispiel:

```
<cm-request...>
  <job-where>
    <name>myJob</name>
  </job-where>
  <job-get>
    <lastOutput/>
  </job-get></cm-request> <cm-response...>
  <cm-code numeric="0" phrase="ok">
    <job>
      <lastOutput><![CDATA[Die Job-Ausgabe.]]></lastOutput>
    </job>
  </cm-code>
</cm-response>
```

`log`

Bedeutung: Die Liste der IDs der letzten Protokolleinträge des Jobs. Die maximale Anzahl der Einträge ist im Systemkonfigurationseintrag `tuning.jobMaxLogLength` definiert.

Definition:

```
<!ELEMENT log (%cm.atom;)>
```

logEntries

Bedeutung: Die letzten Protokolleinträge des Jobs, von denen jeder wiederum eine Liste ist.

Definition:

```
<!ELEMENT logEntries (%cm.listitem;)>
```

Beispiel:

```
<cm-request...>
  <job-where>
    <id>5423</id>
  </job-where>
  <job-get>
    <logEntries/>
  </job-get></cm-request> <cm-response...>
  <cm-code numeric="0" phrase="ok">
    <job>
      <logEntries>
        <listitem>
          <logEntryId>67921.31</logEntryId>
          <execResult />
          <execStart type="date">
            <isoDateTime>20020716130256</isoDateTime>
            <systemConfigFormattedTime>16.07.2002 15:02</systemConfigFormattedTime>
            <userConfigFormattedTime>16.07.2002 15:02</userConfigFormattedTime>
          </execStart>
          <execEnd type="date">
            <isoDateTime>20020716130256</isoDateTime>
            <systemConfigFormattedTime>16.07.2002 15:02</systemConfigFormattedTime>
            <userConfigFormattedTime>16.07.2002 15:02</userConfigFormattedTime>
          </execEnd>
        </listitem>
      </logEntries>
    </job>
  </cm-code>
</cm-response>
```

name

Bedeutung: Der Name des Jobs. Namen von Jobs der Kategorie `user` dürfen nicht mit `system` oder einem Unterstrich beginnen.

Definition:

```
<!ELEMENT script (%cm.atom;)>
```

nextExecStart

Bedeutung: Zeitpunkt, an dem der Job das nächste Mal ausgeführt wird (leer, wenn er gerade läuft oder es keinen nächsten Termin gibt).

Definition:

```
<!ENTITY % cm.date " (%cm.atom; | isoDateTime | systemConfigFormattedTime |
  userConfigFormattedTime)*">
<!ELEMENT nextExecStart (%cm.date;)>
<!ATTLIST nextExecStart type CDATA #IMPLIED>
```

Bedeutung der Attribute:

- `type`: Der Typ des Feldes `nextExecStart`.

Beispiel: siehe `lastExecStart`.

queuePos

Bedeutung: Wenn der Wert größer 0 ist, gibt er die Position des Jobs in der Warteschlange an, ist er 0, so wird der Job gerade ausgeführt, andernfalls ist der Job nicht in der Warteschlange.

Definition:

```
<!ELEMENT queuePos (%cm.atom;)>
```

schedule

Bedeutung: Der Ausführungsplan.

Definition:

```
<!ELEMENT schedule (%cm.atom;)>
```

Beispiel:

```
<cm-request...>
  <job-where>
    <id>5423</id>
  </job-where>
  <job-get>
    <schedule/>
  </job-get></cm-request> <cm-response...>
  <cm-code numeric="0" phrase="ok">
    <job>
      <schedule>
        <listitem>
          <dictitem>
```

```
            <key>years</key>
            <value>
              <listitem>2007</listitem>
              <listitem>2008</listitem>
            </value>
          </dictitem>
          <dictitem>
            <key>minutes</key>
            <value>
              <listitem>3</listitem>
              <listitem>5</listitem>
              <listitem>6</listitem>
              <listitem>7</listitem>
            </value>
          </dictitem>
          ...
        </listitem>
      </schedule>
    </job>
  </cm-code>
</cm-response>
```

script

Bedeutung: Das Tcl-Skript, das abgearbeitet werden soll, wenn der Job ausgeführt wird.

Definition:

```
<!ELEMENT script (%cm.atom;)>
```

setKeys

Bedeutung: Die Liste der mit `job-set` setzbaren Parameter.

Definition:

```
<!ELEMENT setKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

title

Bedeutung: Der Titel des Jobs.

Definition:

```
<!ELEMENT title (%cm.atom;)>
```

17.4 Funktionselemente für Jobs

17.4.1 <job-create>

Definition:

```
<!ENTITY % cm.job-create "
  (name,
  comment?,
  execLogin?,
  execPerm?,
  isActive?,
  schedule?,
  script?,
  title?)
">
<!ELEMENT job-create %cm.create>
```

Aufgabe: Erzeugt einen neuen Job mit den angegebenen Parameterwerten.

Zusatzinformationen: Es ist empfehlenswert, mindestens ein globales Recht (beispielsweise `permissionGlobalJobExec`) zu definieren und neuen Jobs als erforderliches Ausführungsrecht zuzuweisen. Andernfalls können die Jobs von allen Benutzern ausgeführt werden.
Wird der Parameter `execLogin` nicht angegeben, so wird als dessen Wert das Login des authentifizierten Benutzers eingetragen.
Der Name des neuen Jobs muss angegeben werden. Von Benutzern können nur Jobs in der Kategorie `user` angelegt werden. Dies ist die voreingestellte Kategorie, so dass das Element `category` nicht angegeben werden muss.

Rückgabewert bei Erfolg: der Name des neuen Jobs.

Erforderliche Rechte: Der authentifizierte Benutzer muss Superuser sein.

Beispiel:

```
<cm-request...>
  <job-create>
    <name>myJob</name>
    <execLogin>stan</execLogin>
    <execPerm>permissionGlobalJobExec</execPerm>
    <script>obj root exportSubtree filePrefix /tmp</script>
  </job-create>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <job>
      <name>myJob</name>
    </job>
  </cm-code>
</cm-response>
```

17.4.2 <job-listQueue>

Definition:

```
<!ENTITY % cm.job-listQueue "EMPTY">
```

```
<!ELEMENT job-listQueue %cm.listQueue>
<!ATTLIST job-listQueue
  category CDATA #IMPLIED
>
```

Aufgabe: Liefert die Namen aller Jobs in der Ausführungswarteschlange, aufsteigend sortiert nach dem Job-Parameter `queuePos`.

Spezielle Parameter dieses Funktionselements:

- `category`: Es werden nur die Jobs der angegebenen Kategorie zurückgegeben (Voreinstellung: `user`).

Rückgabewert bei Erfolg: die Liste der Namen der Jobs in der betreffenden Warteschlange.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <job-listQueue/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <job-listQueue>
      <name>myJob</name>
      <name>yourJob</name>
      <name>ourJob</name>
    </job-listQueue>
  <cm-code>
</cm-response>
```

17.4.3 <job-where>

Ausschnitt aus der DTD:

```
<!ENTITY % cm.cm-request "
  ...
  (job-where+, job-cancel) |
  (job-where+, job-delete) |
  (job-where+, job-description) |
  (job-where+, job-exec) |
  (job-where+, job-get) |
  (job-where+, job-getLogEntry) |
  (job-where+, job-getOutput) |
  (job-where+, job-set) |
  ...
">
    Definition:
<!ENTITY % cm.job-where "
  (category |
  comment |
  execLogin |
  id |
  isActive |
  isQueued |
  name |
  title)*
">
<!ELEMENT job-where %cm.job-where>
```

Aufgabe: Sucht nach Jobs, bei denen alle durch die Werte der angegebenen Parameterelementen spezifizierten Bedingungen erfüllt sind. Wird kein Parameter angegeben, so wird die Liste der Namen aller Jobs geliefert.

Spezielle Parameter dieses Funktionselements:

- `category`, `id`: Die Jobs, bei denen der Parameter dem angegebenen Wert entspricht, werden zurückgegeben.
- `comment`, `name`, `title`: Die Jobs, bei denen der Parameter den angegebenen Wert enthält, werden zurückgegeben.
- `execLogin`: Liefert die Jobs, bei denen das Skript unter dem angegebenen Login ausgeführt wird.
- `isActive`: Liefert die Jobs, bei denen der Parameter `isActive` den angegebenen Wert enthält.
- `isQueued`: Ist ein Wert ungleich 0 angegeben, so werden alle Jobs geliefert, bei denen `queuePos` größer oder gleich null ist. Andernfalls werden die Jobs geliefert, bei denen `queuePos` kleiner null ist.

Verwendung: `job-where` ist ein Funktionselement, das nicht ohne ein weiteres unmittelbar darauf folgendes Funktionselement verwendet werden kann. Mit `job-where` erhält man die Liste der Namen der Jobs, die dem Suchkriterium entsprechen. Erst durch das folgende Funktionselement wird bestimmt, wie die ermittelten Jobs zu behandeln sind. Jedes der Elemente, die auf `job-where` folgen können, entspricht dabei einer Schablone, die nacheinander auf jeden der mit `job-where` ermittelten Jobs angewendet wird (siehe Requests als Schablonen und die Darstellung der Ergebnis-Daten).

Rückgabewert bei Erfolg: die Liste der Namen der passenden Job-Einträge.

Erforderliche Rechte: keine Einschränkungen.

17.4.4 <job-where> <job-cancel>

Definition:

```
<!ENTITY % cm.job-cancel "EMPTY">
<!ELEMENT job-cancel %cm.job-cancel>
```

Aufgabe: job-cancel nimmt jeden mit job-where ermittelten Job aus der Warteschlange.

Zusatzinformationen: Bei den gewünschten Jobs wird `queuePos` auf `-1` gesetzt. Der gerade ausgeführte Job (`queuePos = 0`) kann nicht zurückgezogen werden.

Verwendung: Zunächst werden mit `job-where` Jobs ermittelt. Anschließend wird `job-cancel` verwendet, um diese Jobs abzubrechen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss Superuser sein, das als `execLogin` spezifizierte Login oder das als Wert von `execPerm` angegebene Recht für jeden mit `job-where` ermittelten Job haben.

Beispiel:

```
<cm-request...>
  <job-where>
    <name>myJob</name>
  </job-where>
```

```
  <job-cancel/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

17.4.5 <job-where> <job-delete>

Definition:

```
<!ENTITY % cm.job-delete "EMPTY">
<!ELEMENT job-delete %cm.job-delete>
```

Aufgabe: Löscht jeden mit `job-where` ermittelten Job aus der Jobliste.

Verwendung: Zunächst werden mit `job-where` Jobs ermittelt, um anschließend diese Jobs mit `job-delete` zu löschen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss Superuser sein.

Beispiel:

```
<cm-request...>
  <job-where>
    <name>myJob</name>
  </job-where>
  <job-delete/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

17.4.6 <job-where> <job-description>

Definition:

```
<!ENTITY % cm.atom "#PCDATA">
<!ELEMENT job-description %cm.atom;>
```

Aufgabe: `job-description` liefert eine Zeichenketten-Repräsentation der wichtigsten Job-Parameter.

Verwendung: Zunächst wird mit `job-where` der Job ermittelt, dessen Zeichenketten-Repräsentation anschließend mit `job-description` ausgelesen wird.

Rückgabewert bei Erfolg: die Zeichenketten-Repräsentation.

Erforderliche Rechte: keine.

Beispiel:

```
<cm-request...>
  <job-where>
    <name>someJob</name>
  </job-where>
  <job-description/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok>
    {
      id = 3245;
      name = someJob;
      queuePos = -1;
      title = "Rootpub exportieren";
      execLogin = root;
      isActive = 1;
      ...
    }
  </cm-code>
</cm-response>
```

17.4.7 <job-where> <job-exec>

Definition:

```
<!ENTITY % cm.job-exec "EMPTY">
<!ELEMENT job-exec %cm.job-exec>
```

Aufgabe: Mit job-exec wird jeder mit `job-where` ermittelte Job in die Ausführungswarteschlange gestellt.

Zusatzinformationen: Befindet sich ein Job bereits in der Ausführungswarteschlange, so wird er nicht ein zweites Mal in die Warteschlange gestellt.

Verwendung: Zunächst müssen mit `job-where` Jobs ermittelt werden. Anschließend wird `job-exec` verwendet, um diese Jobs in die Ausführungswarteschlange zu stellen.

Rückgabewert bei Erfolg: Der neue Wert für `queuePos`. Befinden sich Jobs bereits in der Warteschlange, so wird jeweils der aktuelle Wert von `queuePos` zurückgegeben.

Erforderliche Rechte: Der authentifizierte Benutzer muss ein Superuser sein, das als `execLogin` spezifizierte Login oder das als Wert von `execPerm` angegebene Recht für jeden der mit `job-where` ermittelten Jobs haben.

Beispiel:

```
<cm-request...>
  <job-where>
    <name>rootJob</name>
  </job-where>
  <job-exec/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <job>
      <queuePos>5</queuePos>
    </job>
  </cm-code>
</cm-response>
```

17.4.8 <job-where> <job-get>

Definition:

```
<!ENTITY % cm.job-get "
  (category |
  comment |
  displayTitle |
  execLogin |
  execPerm |
  getKeys |
  id |
  isActive |
  lastExecEnd |
  lastExecResult |
  lastExecStart |
  lastLogEntry |
  lastOutput |
  logEntries |
  name |
  nextExecStart |
  queuePos |
  schedule |
  script |
  setKeys |
  title)*
">
<!ELEMENT job-get %cm.job-get>
```

Aufgabe: Liefert zu jedem mit `job-where` ermittelten Job den Wert der angegebenen Parameterelemente.

Verwendung: Zunächst werden mit `job-where` Jobs ermittelt, um anschließend mit `job-get` ihre Parameterwerte auszulesen.

Rückgabewert bei Erfolg: der Wert der angegebenen Job-Parameter.

Erforderliche Rechte: keine Einschränkungen

Beispiel:

```
<cm-request...>
  <job-where>
    <id>4762</id>
  </job-where>
  <job-get>
    <title/>
  </job-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <job>
      <title>Rootpub exportieren</title>
    </job>
  </cm-code>
</cm-response>
```

17.4.9 <job-where> <job-getLogEntry>

Definition:

```
<!ENTITY % cm.logEntry "
  (logEntryId,
  execResult,
  execStart,
  execEnd)
">
<!ENTITY % cm.job-getLogEntry "%cm.logEntry;">
<!ELEMENT job-getLogEntry %cm.job-getLogEntry>
<!ATTLIST job-getLogEntry
  id CDATA #REQUIRED
>
<!ELEMENT logEntryId (%cm.atom;)>
<!ELEMENT execResult (%cm.atom;)>
<!ELEMENT execStart %cm.date;>
<!ATTLIST execStart
  type CDATA #IMPLIED
>
<!ELEMENT execEnd %cm.date;>
<!ATTLIST execEnd
  type CDATA #IMPLIED
>
```

Aufgabe: Der Befehl gibt einen Protokolleintrag eines Jobs aus.

Spezielle Parameter dieses Funktionselements:

- `id`: Die ID des Protokolleintrags.

Verwendung: Zunächst wird mit `job-where` ein Job ermittelt. Anschließend ermittelt ein Client die Liste der Protokolleinträge zu diesem Job, indem er den Wert des Parameters `log` abfragt. Er kann nun mit `job-getLogEntry` je ID die im Protokoll gespeicherten Daten auslesen.

Rückgabewert bei Erfolg: ein Protokolleintrag.

Erforderliche Rechte: keine Einschränkungen

Beispiel:

```
<cm-request...>
  <job-where>
    <name>myJob</name>
  </job-where>
  <job-getLogEntry id="2060"/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <job>
      <logEntry>
        <logEntryId>76321.26</logEntryId>
        <execResult />
        <execStart type="date">
          <isoDateTime>20020716130246</isoDateTime>
          <systemConfigFormattedTime>16.07.2002 15:02
          </systemConfigFormattedTime>
          <userConfigFormattedTime>16.07.2002 15:02
          </userConfigFormattedTime>
        </execStart>
        <execEnd type="date">
          <isoDateTime>20020716130246</isoDateTime>
          <systemConfigFormattedTime>16.07.2002 15:02
```

```
          </systemConfigFormattedTime>
          <userConfigFormattedTime>16.07.2002 15:02
          </userConfigFormattedTime>
        </execEnd>
      </logEntry>
    </job>
  </cm-code>
</cm-response>
```

17.4.10 <job-where> <job-getOutput>

Definition:

```
<!ENTITY % cm.job-getOutput "EMPTY">
<!ELEMENT job-getOutput %cm.job-getOutput>
<!ATTLIST job-getOutput
  id CDATA #REQUIRED
>
```

Aufgabe: Der Befehl gibt zu einem Job-Protokolleintrag die Ausgabe eines Jobs aus.

Verwendung: job-getOutput wird analog zu job-getLogEntry verwendet.

Rückgabewert bei Erfolg: ein Protokolleintrag.

Erforderliche Rechte: keine Einschränkungen

Beispiel:

```
<cm-request...>
  <job-where>
    <name>myJob</name>
  </job-where>
  <job-getOutput id="2060"/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <job>
      <job-getOutput id="666"><![CDATA[Die Job-Ausgabe.]]>
      </job-getOutput>
    </job>
  </cm-code>
</cm-response>
```

17.4.11 <job-where> <job-set>

Definition:

```
<!ENTITY % cm.job-set "
  (comment |
  execLogin |
  execPerm |
  isActive |
  schedule |
  script |
  title)*
">
```

```
<!ELEMENT job-set %cm.job-set>
```

Aufgabe: Setzt bei den mit `job-where` ermittelten Jobs die Werte der angegebenen Parameter auf die spezifizierten Werte.

Verwendung: Zunächst werden mit `job-where` Jobs ermittelt. Anschließend wird `job-set` verwendet, um Parameterwerte für diese Jobs zu setzen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss Superuser sein oder das zu diesem Zweck im Systemkonfigurationseintrag `content.globalPermissions` definierte Recht haben. Nur ein Superuser darf `execLogin` auf ein anderes Login als sein eigenes setzen.

Beispiel:

```
<cm-request...>
  <job-where>
    <name>myJob</name>
  </job-where>
  <job-set>
    <execLogin>Mustermann</execLogin>
    <isActive>1</isActive>
  </job-set>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

18

18 Kundenspezifische Menübefehle - customCommand

18.1 Definition

Über Requests kann man die Ausführung kundenspezifischer Menübefehle (Custom Commands) steuern. Der folgende Ausschnitt aus der DTD des Content Managers zeigt das Element, das dies ermöglicht:

```
<!ENTITY % cm.cm-request "
  ...
  (customCommand-execute) |
  ...
">
```

Dieses Element muss sich unmittelbar unterhalb des `cm-request`-Elements befinden. Es wird im Abschnitt Funktionselement für kundenspezifische Menübefehle erläutert.

18.2 Parameterelemente für kundenspezifische Menübefehle

Um zu spezifizieren, welcher Menübefehl ausgeführt werden soll, werden die folgenden Parameterelemente verwendet:

`arguments`

Bedeutung: Liefert die Argumente des angegebenen Custom Commands.

Definition:

```
<!ELEMENT arguments (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`command`

Bedeutung: Der Menübefehl, der ausgeführt werden soll.

Definition:

```
<!ELEMENT command (%cm.atom;)>
```

18.3 Funktionselement für kundenspezifische Menübefehle

18.3.1 <customCommand-execute>

Definition:

```
<!ELEMENT customCommand-execute ((command, arguments?)| %cm.atom)>
```

Aufgabe: Führt einen kundenspezifischen Menübefehl aus.

Zusatzinformationen: Bei Custom Commands vom Typ *Tcl-Prozedur* wird eine URL zurückgegeben, unter der man sich per HTTP-Request die Ausgabe des Custom Commands abholen kann.
Wie Die unterschiedlichen Typen von Custom Commands funktionieren, ist im Handbuch zur *Systemadministration* / *Entwicklung* beschrieben.

Rückgabewert bei Erfolg: URL.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

Request bei einem kundenspezifischen Befehl vom Typ *Tcl-Prozedur*:

```
<cm-request...>
  <customCommand-execute>
    <command>shuffle</command>
    <arguments>
      <listitem>a1</listitem>
    </arguments>
  </customCommand-execute>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <customCommand-execute>/stream/ICAyMjIy</customCommand-execute>
  </cm-code>
</cm-response>
```

Request bei einem kundenspezifischen Befehl vom Typ *Wizard* (die zweite Response zeigt die Antwort beim letzten Dialog):

```
<cm-request...>
  <customCommand-execute>
    <isSynchronous />
    <command>infonicaNewsWizard</command>
    <arguments>
      <listitem>
        <dictitem>
          <key>longDescriptionGerman</key>
          <value></value>
        </dictitem>
        <dictitem>
```

```
          <key>page</key>
          <value>1</value>
        </dictitem>
        <dictitem>
          <key>subTitle</key>
          <value></value>
        </dictitem>
        <dictitem>
          <key>name</key>
          <value>abcd</value>
        </dictitem>
        <dictitem>
          <key>language</key>
          <value>de</value>
        </dictitem>
        <dictitem>
          <key>shortDescriptionGerman</key>
          <value></value>
        </dictitem>
        <dictitem>
          <key>button</key>
          <value>cancel</value>
        </dictitem>
      </listitem>
      <listitem>
        <listitem>24855</listitem>
        <listitem>24546</listitem>
        <listitem>24666</listitem>
        <listitem>24528</listitem>
        <listitem>24058</listitem>
        <listitem>24066</listitem>
        <listitem>24520</listitem>
      </listitem>
    </arguments>
  </customCommand-execute>
</cm-request>

<cm-response...>
  <cm-code numeric="200" phrase="OK">
    <customCommand-execute>
      &lt;npsform page='1' title='Wizard'&gt;
      &lt;h1&gt;Pressemitteilung anlegen&lt;/h1&gt;
      ...
    </customCommand-execute>
  </cm-code>
</cm-response>

<cm-response...>
  <cm-code numeric="200" phrase="OK">
    <customCommand-execute/>
  </cm-code>
</cm-response>
```

19

19 Links - link

19.1 Definition

Der folgende Ausschnitt aus der DTD des Content Management Servers zeigt die Elemente, mit denen in Requests lesend und schreibend auf Links zugegriffen werden kann:

```
<!ENTITY % cm.cm-request "
  ...
  (link-create) |
  (link-where+, link-delete) |
  (link-where+, link-description) |
  (link-where+, link-get) |
  (link-where+, link-set) |
  ...
">
```

Diese Elemente, die sich unmittelbar unterhalb des `cm-request`-Elements befinden müssen, werden im Abschnitt Funktionselemente für Links erläutert.

19.2 Parameterelemente für Links

Auf Links kann mit Hilfe von Funktionselementen zugegriffen werden. So kann man beispielsweise mit dem `link-get`-Element die Werte sämtlicher Link-Parameter ermitteln. Welche Linkeigenschaften ausgelesen oder gesetzt werden sollen, wird über Parameterelemente spezifiziert. Im Folgenden werden die Parameterelemente für Links aufgeführt.

`attributeName`

Bedeutung: Der Name des Versionsfeldes, in dem der Link steht (nur bei freien Links).

Definition:

```
<!ELEMENT attributeName (%cm.atom; | %cm.attribute-get;)*>
```

`cm.attribute-get`: siehe <attribute-where> <attribute-get> oder CRUL als DTD.

`canHaveAnchor`

Bedeutung: Gibt an, ob der Link ein Einsprungziel in der Zieldatei haben kann.

Definition:

```
<!ELEMENT canHaveAnchor (%cm.atom;)>
```

canHaveTarget

Bedeutung: Gibt an, ob der Link ein Frame-Target haben kann, in dem die Zieldatei dargestellt wird.

Definition:

```
<!ELEMENT canHaveTarget (%cm.atom;)>
```

destination

Bedeutung: ID der Zieldatei (bei internen Links).

Definition:

```
<!ELEMENT destination (%cm.atom; | %cm.obj-get;)*>
```

`cm.obj-get`: siehe <obj-where> <obj-get> oder CRUL als DTD.

destinationUrl

Bedeutung: Die URL, die der Link repräsentiert.

Definition:

```
<!ELEMENT destinationUrl (%cm.atom;)>
```

displayTitle

Bedeutung: Der in der HTML-Benutzerschnittstelle angezeigte Titel des Links. Hat der Link keinen Titel, so wird bei internen Links der Titel der Zieldatei geliefert.

Definition:

```
<!ELEMENT displayTitle (%cm.atom;)>
```

expectedPath

Bedeutung: Der Pfad zur referenzierten Datei in der Ordnerhierarchie (bei internen Links).

Definition:

```
<!ELEMENT expectedPath (%cm.atom;)>
```

getKeys

Bedeutung: Liste der mit `link-get` abfragbaren Parameter.

Definition:

```
<!ELEMENT getKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

id

Bedeutung: Die ID des Links.

Definition:

```
<!ELEMENT id (%cm.atom;)>
```

isComplete

Bedeutung: Gibt an, ob der Link aufgelöst (resolved) ist.

Definition:

```
<!ELEMENT isComplete (%cm.atom;)>
```

isContextLink

Bedeutung: Gibt an, ob der Link ein Kontextlink ist.

Definition:

```
<!ELEMENT isContextLink (%cm.atom;)>
```

isExternalLink

Bedeutung: Gibt an, ob der Link ein externer Link ist.

Definition:

```
<!ELEMENT isExternalLink (%cm.atom;)>
```

isFreeLink

Bedeutung: Gibt an, ob der Link ein freier Link, d. h. einem linklist-Feld zugeordnet ist.

Definition:

```
<!ELEMENT isFreeLink (%cm.atom;)>
```

isIncludeLink

Bedeutung: Gibt an, ob der Hauptinhalt des Linkziels in die Quelldatei eingefügt werden soll (`<NPSOBJ includetext="...">`).

Definition:

```
<!ELEMENT isIncludeLink (%cm.atom;)>
```

isInlineReferenceLink

Bedeutung: Gibt an, ob es sich um einen Inline Reference Link handelt.

Definition:

```
<!ELEMENT isInlineReferenceLink (%cm.atom;)>
```

isLinkFromCommittedContent

Bedeutung: Gibt an, ob sich der Link in der eingereichten Version befindet.

Definition:

```
<!ELEMENT isLinkFromCommittedContent (%cm.atom;)>
```

isLinkFromEditedContent

Bedeutung: Gibt an, ob sich der Link in der Arbeitsversion befindet.

Definition:

```
<!ELEMENT isLinkFromEditedContent (%cm.atom;)>
```

isLinkFromReleasedContent

Bedeutung: Gibt an, ob sich der Link in der freigegebenen Version befindet.

Definition:

```
<!ELEMENT isLinkFromReleasedContent (%cm.atom;)>
```

isRelatedLink

Bedeutung: Gibt an, ob der Link ein freier Link, d. h. einem linklist-Feld zugeordnet ist (gleiches Ergebnis wie `isFreeLink`). Dieser Parameter ist obsolet.

Definition:

```
<!ELEMENT isRelatedLink (%cm.atom;)>
```

isWritable

Bedeutung: Gibt an, ob der authentifizierte Benutzer den Link bearbeiten kann. Dies trifft zu, wenn sich der Link in der Arbeitsversion befindet und der Benutzer das Datei-Schreibrecht hat.

Definition:

```
<!ELEMENT isWritable (%cm.atom;)>
```

lastChecked

Bedeutung: Gibt das Datum der letzten Prüfung eines externen Links durch den Linkchecker an.

Definition:

```
<!ELEMENT lastChecked (%cm.atom;)>
```

position

Bedeutung: Wenn der Link in einer Linkliste enthalten ist, fungiert `position` als Sortierwert, der die Position des Links in der Linkliste bestimmt. Dieser Wert sollte bei einem Link stets unter Berücksichtigung der `position`-Werte aller anderen Links in der Linkliste gesetzt werden.

Definition:

```
<!ELEMENT position (%cm.atom;)>
```

setKeys

Bedeutung: Liste der mit `link-set` setzbaren Linkparameter.

Definition:

```
<!ELEMENT setKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

source

Bedeutung: Die ID der Datei, die den Link enthält.

Definition:

```
<!ELEMENT source (%cm.atom; | %cm.obj-get;)*>
```

`cm.obj-get`: siehe <obj-where> <obj-get> oder CRUL als DTD.

sourceContent

Bedeutung: Die ID der Dateiversion, die den Link enthält.

Definition:

```
<!ELEMENT sourceContent (%cm.atom; | %cm.content-get;)*>
```

`cm.content-get`: siehe <content-where> <content-get> oder CRUL als DTD.

sourcetagAttribute

Bedeutung: Der Name des HTML-Tag-Attributs, in dem der Link definiert ist.

Definition:

```
<!ELEMENT sourcetagAttribute (%cm.atom;)>
```

sourcetagName

Bedeutung: Der Name des HTML-Tags, in dem der Link definiert ist.

Definition:

```
<!ELEMENT sourcetagName (%cm.atom;)>
```

target

Bedeutung: Das Ziel-Frame oder -Fenster.

Definition:

```
<!ELEMENT target (%cm.atom;)>
```

title

Bedeutung: Der Titel des Links.

Definition:

```
<!ELEMENT title (%cm.atom;)>
```

19.3 Funktionselemente für Links

19.3.1 <link-create>

Definition:

```
<!ENTITY % cm.link-create "
  (attributeName |
  destinationUrl |
  sourceContent |
  position)*
">
<!ELEMENT link-create %cm.link-create;>
```

Aufgabe: link-create erzeugt einen neuen Link mit den angegebenen Parameterwerten.

Rückgabewert bei Erfolg: die ID des neuen Links.

Erforderliche Rechte: Der authentifizierte Benutzer muss der Bearbeiter der Datei sein, in deren Arbeitsversion sich der Link befindet. Dies setzt voraus, dass er für die Datei das Recht `permissionWrite` hat.

Beispiel:

```
<cm-request...>
  <link-create>
    <attributName>imageLinks</attributeName>
    <destinationUrl>http://www.flowers.com</destinationUrl>
    <sourceContent>5641.24</sourceContent>
  </link-create>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <link>
      <id>5641.24.36</id>
    </link>
  </cm-code>
</cm-response>
```

19.3.2 <link-where>

Ausschnitt aus der DTD:

```
<!ENTITY % cm.cm-request "
  ...
  (link-where+, link-delete) |
  (link-where+, link-description) |
  (link-where+, link-get) |
  (link-where+, link-set) |
  ...
">
```

Definition:

```
<!ENTITY % cm.link-where "
  (id?)
">
<!ELEMENT link-where %cm.link-where;>
```

Aufgabe: `link-where` ermöglicht die Suche nach einem Link, bei dem der Wert des angegebenen Parameters dem spezifizierten String entspricht. Wird kein Parameterwert angegeben, so werden alle Links ermittelt.

Verwendung: `link-where` ist ein Funktionselement, das nicht ohne ein weiteres unmittelbar darauf folgendes Funktionselement verwendet werden kann. Mit `link-where` erhält man die ID des Links, der dem Suchkriterium entspricht, oder die Liste aller Link-IDs. Erst durch das folgende Funktionselement wird bestimmt, wie die Link-ID oder die Link-IDs zu behandeln sind. Jedes der Elemente, die auf `link-where` folgen können, entspricht dabei einer Schablone, die nacheinander auf jeden der mit `link-where` ermittelten Links angewendet wird (siehe Requests als Schablonen und die Darstellung der Ergebnis-Daten).

Erforderliche Rechte: keine Einschränkungen.

19.3.3 <link-where> <link-delete>

Definition:

```
<!ENTITY % cm.link-delete "EMPTY">
<!ELEMENT link-delete %cm.link-delete;>
```

Aufgabe: Mit dieser Anfrage wird ein Link gelöscht, wenn es ein freier Link ist, der sich in einer Arbeitsversion befindet.

Verwendung: Zunächst wird der Link mit `link-where` ermittelt, um ihn anschließend mit `link-delete` zu löschen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss der Bearbeiter der Datei sein, in deren Arbeitsversion sich der Link befindet. Dies setzt voraus, dass er für die Datei das Recht `permissionWrite` hat.

Beispiel:

```
<cm-request...>
  <link-where>
    <id>4756.44.39</id>
  </link-where>
  <link-delete/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

19.3.4 <link-where> <link-description>

Definition:

```
<!ENTITY % cm.atom "#PCDATA">
<!ELEMENT link-description %cm.atom;>
```

Aufgabe: `link-description` liefert eine Zeichenketten-Repräsentation der wichtigsten Link-Parameter.

Verwendung: Zunächst wird mit `link-where` der Link ermittelt, dessen Zeichenketten-Repräsentation anschließend mit `link-description` ausgelesen wird.

Rückgabewert bei Erfolg: die Zeichenketten-Repräsentation.

Erforderliche Rechte: keine.

Beispiel:

```
<cm-request...>
  <link-where>
    <id>4273.9.16</id>
  </link-where>
```

```
  <link-description/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok>
    {
      destination = 6518;
      destinationUrl = /index.html;
      displayTitle = untitled;
      id = 4273.9.16;
      isComplete = 1;
      isContextLink = 0;
      isExternalLink = 0;
      isIncludeLink = 0;
      isInlineReferenceLink = 0;
      isFreeLink = 1;
      source = 2012;
      sourceContent = 2012.7;
      target = "_new";
    }
  </cm-code>
</cm-response>
```

19.3.5 <link-where> <link-get>

Definition:

```
<!ENTITY % cm.link-get "
  (attributeName |
  canHaveAnchor |
  canHaveTarget |
  destination |
  destinationUrl |
  displayTitle |
  expectedPath |
  getKeys |
  id |
  isComplete |
  isContextLink |
  isFreeLink |
  isIncludeLink |
  isInlineReferenceLink |
  isLinkFromCommittedContent |
  isLinkFromEditedContent |
  isLinkFromReleasedContent |
  isRelatedLink |
  isWritable |
  position |
  setKeys |
  source |
  sourceContent |
  sourcetagAttribute |
  sourcetagName |
  target |
  title)*
">
<!ELEMENT link-get %cm.link-get;>
```

Aufgabe: Liefert den Wert der angegebenen Parameter für jeden mit `link-where` ermittelten Link.

Verwendung: `link-get` muss immer auf `link-where` folgen, da zunächst die Links ermittelt werden müssen, deren Parameterwerte mit Hilfe von `link-get` ausgelesen werden.

Rückgabewert bei Erfolg: der jeweilige Wert der angegebenen Parameter.

Erforderliche Rechte: Der authentifizierte Benutzer muss jeweils das Recht `permissionRead` für die Datei haben, in der der Link definiert ist.

Beispiel:

```
<cm-request...>
  <link-where>
    <id>576</name>
  </link-where>
  <link-get>
    <attributeName/>
  </link-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <link>
      <attributeName>imageLinks</attributeName>
    </link>
  </cm-code>
</cm-response>
```

19.3.6 <link-where> <link-set>

Definition:

```
<!ENTITY % cm.link-set "
  (destinationUrl |
  target |
  title |
  position)*
">
<!ELEMENT link-set %cm.link-set;>
```

Aufgabe: Setzt bei einem Link die spezifizierten Linkparameter auf die angegebenen Werte. Der Link muss sich in einer Arbeitsversion befinden.

Verwendung: `link-set` muss immer auf `link-where` folgen, da zunächst Links ermittelt werden müssen, auf die `link-set` angewendet wird, um Parameterwerte zu setzen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss jeweils der Bearbeiter der Datei sein, in deren Arbeitsversion sich der Link befindet. Dies setzt voraus, dass er für die Datei das Recht `permissionWrite` hat.

Beispiel:

```
<cm-request...>
  <link-where>
    <id>4573.136.21</id>
  </link-where>
  <link-set>
    <destinationUrl>www.weather.com</destinationUrl>
  </link-set>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

20

20 Lizenzmanager - licenseManager

20.1 Definition

Auf den Lizenzmanager kann man in Requests an den Content Manager lesend und schreibend zugreifen. Der folgende Ausschnitt aus der DTD zeigt die Elemente, mit denen dies möglich ist:

```
<!ENTITY % cm.cm-request "
  ...
  (licenseManager-license) |
  (licenseManager-licenseExpirationDate) |
  (licenseManager-licenseType) |
  (licenseManager-loginCount) |
  (licenseManager-logins) |
  (licenseManager-logout) |
  (licenseManager-maxConcurrentUsers) |
  ...
">
```

Diese Elemente, die sich unmittelbar unterhalb des `cm-request`-Elements befinden müssen, werden im folgenden Abschnitt erläutert.

20.2 Funktionselemente für den Lizenzmanager

20.2.1 <licenseManager-checkLicense>

Definition:

```
<!ENTITY % cm.licenseManager-checkLicense "EMPTY">
<!ELEMENT licenseManager-checkLicense
          %cm.licenseManager-checkLicense;>
<!ATTLIST licenseManager-checkLicense
          appName CDATA #REQUIRED
          version CDATA #REQUIRED
>
```

Aufgabe: `licenseManager-checkLicense` prüft, ob für die angegebene Applikation in der angegebenen Version eine Lizenz vorhanden ist.

Bedeutung der Attribute:

- `appName`: spezifiziert den Namen der Applikation, deren Vorhandensein in der Lizenzdatei `license.xml` geprüft werden soll. Der Name muss genau so angegeben werden wie in der Lizenzdatei.

- `version`: Die Versionszeichenkette, die mit der Versionsnummer in der Lizenzdatei verglichen wird.

Rückgabewert bei Erfolg: Der leere Wert. Ist die Applikation `appName` nicht in der Lizenzdatei aufgeführt oder beginnt die Versionsnummer in der Lizenzdatei nicht mit `version`, so wird ein Fehler gemeldet.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <licenseManager-checkLicense appName="CM" version="5.0"/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

20.2.2 <licenseManager-license ExpirationDate>

Definition:

```
<!ENTITY % cm.licenseManager-licenseExpirationDate "
  (%cm.atom;)
">
<!ELEMENT licenseManager-licenseExpirationDate
          %cm.licenseManager-licenseExpirationDate;>
```

Aufgabe: licenseManager-licenseExpirationDate gibt das Datum und Uhrzeit zurück, an dem die Lizenz ausläuft.

Rückgabewert bei Erfolg: das Datum, an dem die Lizenz ausläuft.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <licenseManager-licenseExpirationDate/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <licenseManager-licenseExpirationDate>20011231235959
    </licenseManager-licenseExpirationDate>
  </cm-code>
</cm-response>
```

20.2.3 <licenseManager-license>

Definition:

```
<!ENTITY % cm.licenseManager-license "
  (%cm.atom;)
```

```
">
<!ELEMENT licenseManager-license %cm.licenseManager-license;>
```

Aufgabe: `licenseManager-license` gibt sämtliche Lizenzierungsdaten des CMS zurück.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <licenseManager-license/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <licenseManager-license>
      <applications>
        <CM>
          <version>5</version>
          <concurrentUsers>5</concurrentUsers>
          <applicationId>CM-Infopark-DEV-0</applicationId>
        </CM>
        <TP>
          <version>1</version>
          <applicationId>TP-Infopark-DEV-0</applicationId>
        </TP>
        <SES>
          <version>1</version>
          <applicationId>SES-Infopark-DEV-0</applicationId>
        </SES>
      </applications>
      <type>demo</type>
      <expirationDate>20071231235959</expirationDate>
      <owner>Infopark AG Development</owner>
    </licenseManager-license>
  </cm-code>
</cm-response>
```

20.2.4 <licenseManager-licenseType>

Definition:

```
<!ENTITY % cm.licenseManager-licenseType "
  (%cm.atom;)
">
<!ELEMENT licenseManager-licenseType
          %cm.licenseManager-licenseType;>
```

Aufgabe: licenseManager-licenseType gibt je nach Art der installierten Lizenz `demo` oder `full` zurück.

Rückgabewert bei Erfolg: die Art der installierten Lizenz.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <licenseManager-licenseType/>
</cm-request>

<cm-response...>
```

```
  <cm-code numeric="0" phrase="ok">
    <licenseManager-licenseType>demo</licenseManager-licenseType>
  </cm-code>
</cm-response>
```

20.2.5 <licenseManager-loginCount>

Definition:

```
<!ENTITY % cm.licenseManager-loginCount "
  (%cm.atom;)
">
<!ELEMENT licenseManager-loginCount %cm.licenseManager-loginCount;>
```

Aufgabe: `licenseManager-loginCount` gibt die Anzahl der Benutzer zurück, die gegenwärtig am System angemeldet sind.

Rückgabewert bei Erfolg: die Anzahl der gegenwärtig am System angemeldeten Benutzer.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <licenseManager-loginCount/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <licenseManager-loginCount>3</licenseManager-loginCount>
  </cm-code>
</cm-response>
```

20.2.6 <licenseManager-logins>

Definition:

```
<!ENTITY % cm.licenseManager-logins "
  (listitem*)
">
<!ELEMENT licenseManager-logins %cm.licenseManager-logins;>
```

Aufgabe: `licenseManager-logins` gibt eine Liste von Dictionaries zurück. Jedes Dictionary bezieht sich auf einen angemeldeten Benutzer und enthält zwei Werte, `login` und `timeStamp`. `login` gibt das verwendete Interface, die Session und das Login an (beispielsweise `T(1069544507) root` wobei T=tcl, X=XML, J=Jobs). Der Wert von `timeStamp` spezifiziert Datum und Uhrzeit des letzten Zugriffs auf das System durch den Benutzer.

Rückgabewert bei Erfolg: die Liste der gegenwärtig am System angemeldeten Benutzer mit ihrem Login und der Zeit ihrer Anmeldung am System.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <licenseManager-logins/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <licenseManager-logins>
      <listitem>{login = &amp;quot;T(1069544507) root&amp;quot;;timestamp = 20040505152718;}</
listitem>
      <listitem>{login = &amp;quot;X(3xipmoh001) root&amp;quot;;timestamp = 20040505152725;}
      </listitem>
    </licenseManager-logins>
  </cm-code>
</cm-response>
```

20.2.7 <licenseManager-logout>

Definition:

```
<!ENTITY % cm.licenseManager-logout "EMPTY">
<!ELEMENT licenseManager-logout %cm.licenseManager-logout;>
<!ATTLIST licenseManager-logout
          login CDATA #IMPLIED
>
```

Aufgabe: `licenseManager-logout` beendet eine Verbindung mit dem Content Manager.

Bedeutung der Attribute:

- `login`: ist der Anmeldename des Benutzers, dessen Verbindung beendet werden soll. Man muss ein Superuser oder der Verwalter des auszuloggenden Benutzers sein, um einen anderen Benutzer als sich selbst ausloggen zu können. Ist das Tag-Attribut nicht angegeben, so wird die aktuelle Verbindung getrennt.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der Benutzer muss ein Superuser sein, um eine andere als die eigene Verbindung trennen zu können.

Beispiel:

```
<cm-request...>
  <licenseManager-logout login="stan"/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

20.2.8 <licenseManager-maxConcurrentUsers>

Definition:

```
<!ENTITY % cm.licenseManager-maxConcurrentUsers "
  (%cm.atom;)
">
```

```
<!ELEMENT licenseManager-maxConcurrentUsers
          %cm.licenseManager-maxConcurrentUsers;>
```

Aufgabe: `licenseManager-maxConcurrentUsers` gibt die maximale Anzahl der Benutzer aus, die gleichzeitig am System angemeldet sein dürfen.

Rückgabewert bei Erfolg: die maximale Anzahl der Benutzer.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <licenseManager-maxConcurrentUsers/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <maxConcurrentUsers>15</maxConcurrentUsers>
  </cm-code>
</cm-response>
```

21

21 Log-Einträge - logEntry

21.1 Definition

Auf Log-Einträge kann in Requests an den Content Management Server lesend und schreibend zugegriffen werden. Der folgende DTD-Ausschnitt zeigt die Elemente, die dies ermöglichen:

```
<!ENTITY % cm.cm-request "
  ...
  (logEntry-where+, logEntry-delete) |
  (logEntry-where+, logEntry-get) |
  ...
">
```

Diese Elemente müssen sich unmittelbar unterhalb des `cm-request`-Elements befinden. Sie werden im Abschnitt Funktionselemente für Log-Einträge erläutert.

21.2 Parameterelemente für Log-Einträge

Mit Hilfe von Funktionselementen kann auf Log-Einträge zugegriffen werden. So kann man beispielsweise mit dem `logEntry-get`-Element die Werte sämtlicher Parameter für Log-Einträge ermitteln. Um zu spezifizieren, welche Eigenschaften von Log-Einträgen ausgelesen oder welche Log-Einträge gelöscht werden sollen, verwendet man Parameterelemente. Im Folgenden werden die `logEntry`-Parameter und die dazu gehörenden Parameterelemente aufgeführt.

`fromDate`

Bedeutung: Das Anfangsdatum des Zeitraums, in dem Log-Einträge gesucht werden (kann nur im Element `<logEntry-where>` verwendet werden).

Definition:

```
<!ELEMENT fromDate (%cm.atom;)>
```

`logText`

Bedeutung: Der im Kommentar eines Log-Eintrags enthaltene Text.

Definition:

```
<!ELEMENT logText (%cm.atom;)>
```

logTime

Bedeutung: Zeitpunkt, zu dem der Log-Eintrag erstellt wurde.

Definition:

```
<!ELEMENT logTime (%cm.atom;)>
```

logTypes

Bedeutung: Die gesuchten Log-Typen (siehe Log-Typen).

Definition:

```
<!ELEMENT logTypes (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

objectId

Bedeutung: Die ID der Datei, zu der Log-Einträge gesucht werden.

Definition:

```
<!ELEMENT objectId (%cm.atom; | %cm.obj-get;)*>
```

`cm.obj-get`: siehe <obj-where> <obj-get> oder CRUL als DTD.

receiver

Bedeutung: Der Name der Gruppe, die eine Datei durch eine Workflow-Aktion erhalten hat.

Definition:

```
<!ELEMENT receiver (%cm.atom;)>
```

untilDate

Bedeutung: Das Enddatum des Zeitraums, zu dem Log-Einträge gesucht werden (kann nur im Element `<logEntry-where>` verwendet werden).

Definition:

```
<!ELEMENT untilDate (%cm.atom;)>
```

`userLogin`

Bedeutung: Der Benutzer, zu dem Log-Einträge gesucht werden.

Definition:

```
<!ELEMENT userLogin (%cm.atom; | %cm.user-get;)*>
```

`cm.user-get`: siehe <user-where> <user-get> oder CRUL als DTD.

21.3 Log-Typen

Log-Typ	Auslösende Aktion
`action_create_subobj`	Erzeugung einer Datei. Als `objectId` wird im Eintrag die Datei-ID des Ordners festgehalten, in dem die Datei angelegt wurde.
`action_delete_subobj`	Löschen einer Datei.
`action_admin_obj`	Verschieben einer Datei oder Änderung von Dateifeldern.
`action_edit_obj`	Workflow-Aktion `edit`.
`action_give_obj`	Workflow-Aktion `give`.
`action_take_obj`	Workflow-Aktion `take`.
`action_forward_obj`	Workflow-Aktion `forward`.
`action_commit_obj`	Workflow-Aktion `commit`.
`action_reject_obj`	Workflow-Aktion `reject`.
`action_revert_obj`	Workflow-Aktion `revert`.
`action_sign_obj`	Workflow-Aktion `sign`.
`action_release_obj`	Workflow-Aktion `release`.
`action_unrelease_obj`	Workflow-Aktion `unrelease`.

21.4 Funktionselemente für Log-Einträge

21.4.1 <logEntry-where>

Ausschnitt aus der DTD:

```
<!ENTITY % cm.cm-request "
  ...
  (logEntry-where+, logEntry-delete) |
  (logEntry-where+, logEntry-get) |
  ...
">
```

Definition:

```
<!ENTITY % cm.logEntry-where "
  (fromDate?,
  logText?,
  logTypes?,
  objectId?,
  receiver?,
  untilDate?,
  userLogin?)?
">
<!ELEMENT logEntry-where %cm.logEntry-where;>
```

Aufgabe: `logEntry-where` ermöglicht die Suche nach Log-Einträgen, bei denen der Wert der angegebenen Parameter den jeweils spezifizierten String enthält. Eine Liste aller vorhandenen Log-Einträge wird zurückgegeben, wenn keine Parameterwerte angegeben werden.

Verwendung: `logEntry-where` ist ein Funktionselement, das nicht ohne ein weiteres unmittelbar darauf folgendes Funktionselement verwendet werden kann. Mit `logEntry-where` erhält man die Liste der Log-Einträge, die dem Suchkriterium entsprechen. Erst durch das folgende Funktionselement wird jedoch bestimmt, wie diese zu behandeln sind. Jedes der Elemente, die auf `logEntry-where` folgen können, entspricht dabei einer Schablone, die nacheinander auf jeden der mit `logEntry-where` ermittelten Log-Einträge angewendet wird (siehe Requests als Schablonen und die Darstellung der Ergebnis-Daten).

Erforderliche Rechte: keine Einschränkungen.

21.4.2 <logEntry-where> <logEntry-delete>

Definition:

```
<!ENTITY % cm.logEntry-delete "EMPTY">
<!ELEMENT logEntry-delete %cm.logEntry-delete;>
```

Aufgabe: Löscht jeden mit `logEntry-where` ermittelten Log-Eintrag.

Verwendung: Zunächst werden mit `logEntry-where` Log-Einträge ermittelt, um diese anschließend mit `logEntry-delete` zu löschen.

Rückgabewert bei Erfolg: die Anzahl der gelöschten Einträge.

Erforderliche Rechte: Der authentifizierte Benutzer muss ein Superuser sein.

Beispiel:

```
<cm-request...>
  <logEntry-where>
    <objectId>1756</objectId>
  </logEntry-where>
  <logEntry-delete/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <deleteLogEntriesCount>5</deleteLogEntriesCount>
  </cm-code>
</cm-response>
```

21.4.3 <logEntry-where> <logEntry-get>

Definition:

```
<!ENTITY % cm.logEntry-get "
  (logTime |
  logText |
  logType |
  objectId |
  receiver |
  userLogin)*
">
<!ELEMENT logEntry-get %cm.logEntry-get;>
```

Aufgabe: logEntry-get liefert den Wert der angegebenen Parameter für jeden der mit `logEntry-where` ermittelten Log-Einträge.

Verwendung: `logEntry-get` muss immer auf `logEntry-where` folgen, da zunächst die Log-Einträge ermittelt werden müssen, auf die anschließend `logEntry-get` angewendet wird, um Parameterwerte auszulesen.

Rückgabewert bei Erfolg: der Wert der angegebenen Parameter.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <logEntry-where>
    <objectId>12759</objectId>
  </logEntry-where>
  <logEntry-get>
    <logType/>
    <userLogin/>
  </logEntry-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <logEntry>
      <logType>action_release_obj</logType>
      <userLogin>stan</userLogin>
    </logEntry>
  </cm-code>
</cm-response>
```

22

22 Dateien - obj

22.1 Definition

Der folgende Ausschnitt aus der DTD des Content Management Server zeigt die Elemente, mit denen in Requests an den Content Manager schreibend und lesend auf Dateien zugegriffen werden kann:

```
<!ENTITY % cm.cm-request "
  ...
  (obj-contentTypesForObjType) |
  (obj-create) |
  (obj-generatePreview) |
  (obj-search+, obj-addComment) |
  (obj-search+, obj-commit) |
  (obj-search+, obj-copy) |
  (obj-search+, obj-delete) |
  (obj-search+, obj-description) |
  (obj-search+, obj-edit) |
  (obj-search+, obj-exportSubtree) |
  (obj-search+, obj-forward) |
  (obj-search+, obj-get) |
  (obj-search+, obj-give) |
  (obj-search+, obj-permissionGrantTo) |
  (obj-search+, obj-permissionRevokeFrom) |
  (obj-search+, obj-reject) |
  (obj-search+, obj-release) |
  (obj-search+, obj-removeActiveContents) |
  (obj-search+, obj-removeArchivedContents) |
  (obj-search+, obj-revert) |
  (obj-search+, obj-set) |
  (obj-search+, obj-sign) |
  (obj-search+, obj-take) |
  (obj-search+, obj-unrelease) |
  (obj-types) |
  (obj-where+, obj-commit) |
  (obj-where+, obj-copy) |
  (obj-where+, obj-delete) |
  (obj-where+, obj-description) |
  (obj-where+, obj-edit) |
  (obj-where+, obj-exportSubtree) |
  (obj-where+, obj-forward) |
  (obj-where+, obj-get) |
  (obj-where+, obj-give) |
  (obj-where+, obj-permissionGrantTo) |
  (obj-where+, obj-permissionRevokeFrom) |
  (obj-where+, obj-reject) |
  (obj-where+, obj-release) |
  (obj-where+, obj-removeActiveContents) |
  (obj-where+, obj-removeArchivedContents) |
  (obj-where+, obj-revert) |
  (obj-where+, obj-set) |
  (obj-where+, obj-sign) |
  (obj-where+, obj-take) |
  (obj-where+, obj-unrelease) |
```

```
  ...
">
<!ATTLIST obj-where
    maxResults CDATA #IMPLIED
>
```

Diese Elemente müssen sich unmittelbar unterhalb des `cm-request`-Elements befinden. Sie werden im Abschnitt Funktionselemente für Dateien erläutert.

22.1.1 Rechte

In Requests an und in Responses vom Content Management Server können die folgenden Bezeichner für dateibezogene Rechte enthalten sein. Die dateibezogenen Rechte des Content Managers sind nicht erweiterbar.

Dateispezifische Rechte	Bedeutung
`permissionRead`	Leserecht
`permissionWrite`	Schreibrecht
`permissionCreateChildren`	Recht, in einem Ordner Dateien anzulegen
`permissionRoot`	Dateiadministrationsrecht
`permissionLiveServerRead`	Leserecht auf dem Liveserver

22.2 Parameterelemente für Dateien

Auf Dateien kann man mit Funktionselementen zugreifen. Mit dem `obj-get`-Element können beispielsweise die Werte sämtlicher Dateiparameter ermittelt werden. Welche Parameter ausgelesen oder gesetzt werden sollen, spezifiziert man mit Hilfe von Parameterelementen. Im Folgenden werden die Parameterelemente für Dateien aufgeführt.

`archivedContents`

Bedeutung: Liefert die Liste der IDs der archivierten Versionen der Datei.

Definition:

```
<!ELEMENT archivedContents ((content)* | (listitem)*)>
<!ELEMENT content (%cm.content-get;)>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.content-get`: siehe <content-where> <content-get> oder CRUL als DTD.

`children`

Bedeutung: Liste der IDs der Dateien in einem Ordner.

Definition:

```
<!ELEMENT children ((obj)* | (listitem)*)>
<!ELEMENT obj (%cm.obj-get;)>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.obj-get`: siehe <obj-where> <obj-get> oder CRUL als DTD.

committedContentId

Bedeutung: Liefert die ID der eingereichten Version einer Datei, sofern sie eine solche Version hat.

Definition:

```
<!ELEMENT committedContentId (%cm.atom; | %cm.content-get;)*>
```

`cm.content-get`: siehe <content-where> <content-get> oder CRUL als DTD.

contentIds

Bedeutung: Liefert die Liste der IDs aller Versionen der Datei.

Definition:

```
<!ELEMENT contentIds ((content)* | (listitem)*)>
<!ELEMENT content (%cm.content-get;)>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.content-get`: siehe <content-where> <content-get> oder CRUL als DTD.

contents

Bedeutung: Liefert die Liste der IDs aller Versionen der Datei.

Definition:

```
<!ELEMENT contents ((content)* | (listitem)*)>
<!ELEMENT content (%cm.content-get;)>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.content-get`: siehe <content-where> <content-get> oder CRUL als DTD.

exportContent

Bedeutung: Liefert die ID der Exportversion einer Datei. Wird als Attributwert von `wantReleased` der Wert `1` angegeben, so liefert `exportContent` die ID der freigegebenen Version. Gibt es keine freigegebene Version, so wird die ID der eingereichten Version zurückgegeben. Existiert auch diese nicht, wird die ID der Arbeitsversion geliefert. Wird der Attributwert `0` oder kein Attributwert angegeben, so wird die ID der Arbeitsversion geliefert. Falls diese nicht existiert, wird die ID der eingereichten Version und wenn auch diese nicht vorhanden ist, die ID der freigegebenen Version zurückgegeben.
`exportContent` kann ferner verwendet werden, um Eigenschaften der Exportversion abzufragen.

Definition:

```
<!ELEMENT exportContent (%cm.atom; | %cm.content-get;)*>
<!ATTLIST exportContent
          wantReleased (0 | 1) #IMPLIED
>
```

`cm.content-get`: siehe <content-where> <content-get> oder CRUL als DTD.

Bedeutung der Attribute:

- `wantReleased`: gibt an, ob die ID der freigegebenen Version ermittelt werden soll.

Beispiel:

```
<cm-request...>
  <obj-where>
    <id>21463</id>
  </obj-where>
  <obj-get>
    <exportContent wantReleased="1"/>
  </obj-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <obj>
      <exportContent wantReleased="1">21463.35</exportContent>
    </obj>
  </cm-code>
</cm-response>
```

editedContent

Bedeutung: Liefert die ID der Arbeitsversion einer Datei, sofern sie eine solche Version hat.

Definition:

```
<!ELEMENT editedContent (%cm.atom; | %cm.content-get;)*>
```

`cm.content-get`: siehe <content-where> <content-get> oder CRUL als DTD.

editedContentId

Bedeutung: Liefert die ID der Arbeitsversion einer Datei, sofern sie eine solche Version hat.

Definition:

```
<!ELEMENT editedContentId (%cm.atom; | %cm.content-get;)*>
```

`cm.content-get`: siehe <content-where> <content-get> oder CRUL als DTD.

exportMimeType

Bedeutung: Der Mime-Typ zur Dateiendung der freigegebenen (oder, falls diese nicht existiert, der Arbeits-) Version.

Definition:

```
<!ELEMENT exportMimeType (%cm.atom;)>
```

getKeys

Bedeutung: Die Liste der mit `obj-get` abfragbaren Parameter.

Definition:

```
<!ELEMENT getKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

hasChildren

Bedeutung: Gibt an, ob die Datei ein Ordner ist, der Dateien enthält.

Definition:

```
<!ELEMENT hasChildren (%cm.atom;)>
```

hasSuperLinks

Bedeutung: Gibt an, ob auf die Datei mit Links verwiesen wird, die in der Linkverwaltung erfasst sind.

Definition:

```
<!ELEMENT hasSuperLinks (%cm.atom;)>
```

hierarchy

Bedeutung: Liefert die unterhalb der Datei liegende Dateihierarchie zurück.

Definition:

```
<!ELEMENT hierarchy (%cm.listitem;)*>
<!ATTLIST hierarchy
          maxDepth CDATA #IMPLIED
          maxLines CDATA #IMPLIED
          document (0 | 1) #IMPLIED
          generic (0 | 1) #IMPLIED
          image (0 | 1) #IMPLIED
          publication (0 | 1) #IMPLIED
          template (0 | 1) #IMPLIED>
```

Bedeutung der Attribute:

- `maxDepth`: gibt die maximale Tiefe an, die die ermittelte Hierarchie aufweisen darf.
- `maxLines`: gibt die Anzahl der Dateien an, die maximal in der Hierarchie enthalten sein dürfen.
- `document`: gibt an, ob die Hierachie Dateien vom Typ *Dokument* enthalten soll oder nicht.
- `generic`: gibt an, ob die Hierachie Dateien vom Typ *Ressource* enthalten soll oder nicht.
- `image`: gibt an, ob die Hierachie Dateien vom Typ *Bild* enthalten soll oder nicht.
- `publication`: gibt an, ob die Hierachie Dateien vom Typ `publication` enthalten soll oder nicht. Fehlt der Typ `publication`, so ergibt sich keine Hierarchie.
- `template`: gibt an, ob die Hierachie Dateien vom Typ `template` enthalten soll oder nicht.

Beispiel:

```
<cm-request...>
  <obj-where>
    <id>7657</id>
  </obj-where>
  <obj-get>
    <hierarchy maxDepth="2" document="1" publication="1" template="0"/>
  </obj-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <obj>
      <hierarchy maxDepth="2">
        <listitem>0</listitem>
        <listitem>
          <listitem>87243</listitem>
        </listitem>
        <listitem>
          <listitem>
            <listitem>12654</listitem>
          </listitem>
        </listitem>
      </hierarchy>
    </obj>
  </cm-code>
```

```
</cm-response>
```

id

Bedeutung: Die Datei-ID. Der Wert dieses Feldes kann auch ohne Dateileserecht abgefragt werden.

Definition:

```
<!ELEMENT id (%cm.atom;)>
```

isCommitted

Bedeutung: Gibt an, ob die Datei eine eingereichte Version hat.

Definition:

```
<!ELEMENT isCommitted (%cm.atom;)>
```

isEdited

Bedeutung: Gibt an, ob die Datei eine Arbeitsversion hat.

Definition:

```
<!ELEMENT isEdited (%cm.atom;)>
```

isExportable

Bedeutung: Gibt an, ob die Datei eine freigegebene und zeitlich gültige Version hat (prüft `validFrom` und `validUntil`).

Definition:

```
<!ELEMENT isExportable (%cm.atom;)>
```

isGoodDestination

Bedeutung: Liefert `1`, wenn die Datei ein zulässiges Ziel des Links mit der als Argument angegebenen ID ist. Andernfalls liefert sie `0`.

Definition:

```
<!ELEMENT isGoodDestination (%cm.atom;)>
<!ATTLIST isGoodDestination
          linkId CDATA #REQUIRED
>
```

Bedeutung der Attribute:

- `linkId`: spezifiziert einen Link.

Beispiel:

```
<cm-request...>
  <obj-where>
    <id>56432</id>
  </obj-where>
  <obj-get>
    <isGoodDestination linkId="66311.35.4"/>
  </obj-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <obj>
      <isGoodDestination linkId="66311.35.4"/>1</isGoodDestination>
    </obj>
  </cm-code>
</cm-response>
```

`isGoodParent`

Bedeutung: Liefert `1`, wenn die Vorlage der Datei eine der erlaubten Vorlagen für Dateien in dem Ordner mit der angegebenen ID ist. Andernfalls ist der Rückgabewert `0`.

Definition:

```
<!ELEMENT isGoodParent (%cm.atom;)>
<!ATTLIST isGoodParent
          objectId CDATA #REQUIRED
          operation (move | copy) #REQUIRED
>
```

Bedeutung der Attribute:

- `objectId`: spezifiziert eine Datei.
- operation: spezifiziert die Operation (kopieren oder verschieben), die für die Datei ausgeführt werden soll.

Beispiel:

```
<cm-request...>
  <obj-where>
    <nameLike>Art</nameLike>
  </obj-where>
  <obj-get>
```

```
    <isGoodParent objectId="76576" operation="move"/>
  </obj-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <obj>
      <isGoodParent objectId="76576" operation="move"/>1</isGoodParent>
    </obj>
  </cm-code>
</cm-response>
```

isReleased

Bedeutung: Gibt an, ob die Datei eine freigegebene Version hat.

Definition:

```
<!ELEMENT isReleased (%cm.atom;)>
```

isRoot

Bedeutung: Gibt an, ob die Datei der Basisordner ist.

Definition:

```
<!ELEMENT isRoot (%cm.atom;)>
```

name

Bedeutung: Der Name der Datei. Der Wert dieses Feldes kann auch ohne Dateileserecht abgefragt werden.

Definition:

```
<!ELEMENT name (%cm.atom;)>
```

next

Bedeutung: Die ID der Datei, die in dem darüber liegenden Ordner der Nachfolger der spezifizierten Datei ist.

Definition:

```
<!ELEMENT next (%cm.atom;)>
```

objClass

Bedeutung: Der Name der Vorlage der Datei.

Definition:

```
<!ELEMENT objClass (%cm.atom; | %cm.objClass-get;)*>
```

`cm.objClass-get`: siehe <objClass-where> <objClass-get> oder CRUL als DTD.

objType

Bedeutung: Der Dateityp.

Definition:

```
<!ELEMENT objType (%cm.atom;)>
```

objectsToRoot

Bedeutung: Die Liste der IDs der Dateien, die auf dem Pfad von der Datei zum Basisordner liegen (jeweils einschließlich).

Definition:

```
<!ELEMENT objectsToRoot ((obj)* | (listitem)*)>
<!ELEMENT obj (%cm.obj-get;)>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.obj-get`: siehe <obj-where> <obj-get> oder CRUL als DTD.

parent

Bedeutung: Die ID der Datei, in der die spezifizierte Datei direkt enthalten ist.

Definition:

```
<!ELEMENT parent (%cm.atom; | %cm.obj-get;)*>
```

`cm.obj-get`: siehe <obj-where> <obj-get> oder CRUL als DTD.

path

Bedeutung: Der Pfad der Datei (gebildet aus den Namen der darüber liegenden Dateien). Der Wert dieses Feldes kann auch ohne Dateileserecht abgefragt werden.

Definition:

```
<!ELEMENT path (%cm.atom;)>
```

permission

Bedeutung: Liefert die Liste der Gruppen, die das angegebene Recht für den Zugriff auf die Datei haben.

Definition:

```
<!ELEMENT permission ((group)* | (%cm.listitem;)*)>
<!ATTLIST permission
              permission (permissionCreateChildren |
                          permissionWrite |
                          permissionRead |
                          permissionRoot |
                          permissionLiverServerRead) #REQUIRED
>
<!ELEMENT group (%cm.group-get;)>
```

`cm.group-get`: siehe <group-where> <group-get> oder CRUL als DTD.

Bedeutung der Attribute:

- `permission`: spezifiziert ein dateispezifisches Recht.

Beispiel:

```
<cm-request...>
  <obj-where>
    <title>newsArticlexyz</title>
  </obj-where>
  <obj-get>
    <permission permission="permissionWrite"/>
  </obj-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <obj>
      <permission permission="permissionWrite">
        <group>
          <name>NewsMasters</name>
        </group>
      </permission>
    </obj>
  </cm-code>
</cm-response>
```

permissionGrantedTo

Bedeutung: Prüft, ob ein Benutzer oder eine Gruppe das angegebene Recht für den Zugriff auf die Datei hat (`1` = hat das Recht; `0` = hat das Recht nicht).

Definition:

```
<!ELEMENT permissionGrantedTo (%cm.atom;)>
<!ATTLIST permissionGrantedTo
          permission (permissionCreateChildren |
                      permissionWrite |
                      permissionRead |
                      permissionRoot |
                      permissionLiverServerRead) #REQUIRED
          user CDATA #IMPLIED
          group CDATA #IMPLIED
>
```

Bedeutung der Attribute:

- `permission`: spezifiziert ein dateispezifisches Recht.
- `user`: spezifiziert den Anmeldenamen eines Benutzers.
- `group`: spezifiziert den Namen einer Benutzergruppe.

Beispiel:

```
<cm-request...>
  <obj-where>
    <id>853875</id>
  </obj-where>
  <obj-get>
    <permissionGrantedTo permission="permissionWrite" user="elton"/>
  </obj-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <obj>
      <permissionGrantedTo permission="permissionWrite" user="elton">1
      </permissionGrantedTo>
    </obj>
  </cm-code>
</cm-response>
```

prefixPath

Bedeutung: Der Pfad der Datei, bei Ordnern mit einem „/" am Ende.

Definition:

```
<!ELEMENT prefixPath (%cm.atom;)>
```

previous

Bedeutung: Die ID der Datei, die in dem darüber liegenden Ordner der Vorgänger der spezifizierten Datei ist.

Definition:

```
<!ELEMENT previous (%cm.atom; | %cm.obj-get;)*>
```

`cm.obj-get`: siehe <obj-where> <obj-get> oder CRUL als DTD.

`releasedContent`

Bedeutung: Liefert die ID der freigegebenen Version einer Datei, sofern sie eine solche Version hat.

Definition:

```
<!ELEMENT releasedContent (%cm.atom; | %cm.content-get;)*>
```

`cm.content-get`: siehe <content-where> <content-get> oder CRUL als DTD.

`releasedContentId`

Bedeutung: Liefert die ID der freigegebenen Version einer Datei, sofern sie eine solche Version hat.

Definition:

```
<!ELEMENT releasedContentId (%cm.atom; | %cm.content-get;)*>
```

`cm.content-get`: siehe <content-where> <content-get> oder CRUL als DTD.

`releasedVersions`

Bedeutung: Die Liste der IDs aller bereits freigegebenen Versionen der Datei (aktuelle freigegebene Version und alle archivierten Versionen).

Definition:

```
<!ELEMENT releasedVersions ((content)* | (listitem)*)>
<!ELEMENT content (%cm.content-get;)>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.content-get`: siehe <content-where> <content-get> oder CRUL als DTD.

`rootPermissionFor`

Bedeutung: Prüft, ob ein Benutzer oder eine Gruppe das Administrationsrecht für die Datei hat oder Superuser ist. Bei Erfolg wird `1` zurückgegeben; andernfalls `0`.

Definition:

```
<!ELEMENT rootPermissionFor (%cm.atom;)>
<!ATTLIST rootPermissionFor user CDATA #IMPLIED group CDATA #IMPLIED>
```

Bedeutung der Attribute:

- `user`: spezifiziert den Anmeldenamen eines Benutzers.
- `group`: spezifiziert den Namen einer Benutzergruppe.

Beispiel:

```
<cm-request...>
  <obj-where>
    <id>7345</id>
  </obj-where>
  <obj-get>
    <rootPermissionFor user="sam"/>
  </obj-get>
;</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <obj>
      <rootPermissionFor user="sam">1</rootPermissionFor>
    </obj>
  </cm-code>
</cm-response>
```

`setKeys`

Bedeutung: Die Liste der mit `obj-set` setzbaren Parameter.

Definition:

```
<!ELEMENT setKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`sortValue`

Bedeutung: Der Wert, nach dem die Dateien in dem Ordner, zu dem sie gehören, sortiert werden. Wird durch den Sortierschlüssel des darüber liegenden Ordners bestimmt.

Definition:

```
<!ELEMENT sortValue (%cm.atom;)>
```

superLinks

Bedeutung: Die Liste aller IDs der Links, die auf die Datei verweisen.

Definition:

```
<!ELEMENT superLinks ((link)* | (listitem)*)>
<!ELEMENT link (%cm.link-get;)>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.link-get`: siehe <link-where> <link-get> oder CRUL als DTD.

superObjects

Bedeutung: Die Liste aller IDs der Dateien, die Links auf die Datei enthalten.

Definition:

```
<!ELEMENT superObjects ((obj)* | (listitem)*)>
<!ELEMENT obj (%cm.obj-get;)>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.obj-get`: siehe <obj-where> <obj-get> oder CRUL als DTD.

suppressExport

Bedeutung: Gibt an, ob die Datei exportiert werden kann (`1` = Export elaubt; `0` = Datei darf nicht exportiert werden).

Definition:

```
<!ELEMENT suppressExport (%cm.atom;)>
```

toclist

Bedeutung: Nur für Ordner verfügbar! Die Liste der IDs der Dateien in einem Ordner, die in einer toclist (`<NPSOBJ list="toclist"></NPSOBJ>`) erscheinen (alle zeitlich gültigen Ordner und Dokumente mit freigegebener Version, bei denen `suppressExport` nicht gesetzt ist).

Definition:

```
<!ELEMENT toclist ((obj)* | (listitem)*)>
<!ELEMENT obj (%cm.obj-get;)>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
```

```
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.obj-get`: siehe <obj-where> <obj-get> oder CRUL als DTD.

`validControlActionKeys`

Bedeutung: Die Namen der Aktionen, die für die Datei und den jeweiligen Benutzer ausführbar sind - Untermenge aus {`edit`, `commit`, `release`, `unrelease`, `revert`, `reject`}.

Definition:

```
<!ELEMENT validControlActionKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`validCreateObjClasses`

Bedeutung: Die Liste der Namen der Vorlagen, die der Benutzer beim Anlegen von Dateien im aktuellen Ordner verwenden darf. (Dies entspricht der Liste `validSubObjClasses`, die in der Vorlage der Datei spezifiziert sind, für die der Benutzer die entsprechenden Dateierzeugungsrechte hat.).

Definition:

```
<!ELEMENT validCreateObjClasses ((objClass)* | (listitem)*)>
<!ELEMENT objClass (%cm.atom; | %cm.objClass-get;)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.objClass-get`: siehe <objClass-where> <objClass-get> oder CRUL als DTD.

`validObjClasses`

Bedeutung: Die Liste der Namen der Vorlagen, die der Benutzer der aktuellen Datei zuweisen darf. (Dies entspricht den `validSubObjClasses` des darüber liegenden Ordners, die den gleichen Dateityp haben).

Definition:

```
<!ELEMENT validObjClasses ((objClass)* | (%cm.listitem;)*)>
<!ELEMENT objClass (%cm.atom; | %cm.objClass-get;)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.objClass-get`: siehe <objClass-where> <objClass-get> oder CRUL als DTD.

validPermissions

Bedeutung: Die Liste aller für die Datei verfügbaren Rechte.

Definition:

```
<!ELEMENT validPermissions (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

version

Bedeutung: Die Revision der Datei.

Definition:

```
<!ELEMENT version (%cm.atom;)>
```

visibleExportTemplates

Bedeutung: Die Liste der IDs der Layouts, die von der Datei aus sichtbar sind und für einen Export in Frage kommen.

Definition:

```
<!ELEMENT visibleExportTemplates ((obj)* | (listitem)*)>
<!ELEMENT obj (%cm.obj-get;)>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.obj-get`: siehe <obj-where> <obj-get> oder CRUL als DTD.

visibleName

Bedeutung: Der Name der Datei, ergänzt um die Dateiendung als Extension (außer bei Ordnern). Der Wert dieses Feldes kann auch ohne Dateileserecht abgefragt werden.

Definition:

```
<!ELEMENT visibleName (%cm.atom;)>
```

visiblePath

Bedeutung: Der Pfad der Datei, ergänzt um die Dateiendung als Extension (bei Ordnern wird "/index.<contentType>" angehängt). Der Wert dieses Feldes kann auch ohne Dateileserecht abgefragt werden.

Definition:

```
<!ELEMENT visiblePath (%cm.atom;)>
```

workflowName

Bedeutung: Der Name des Workflows, der der Datei zugeordnet ist.

Definition:

```
<!ELEMENT workflowName (%cm.atom; | %cm.workflow-get;)*>
```

`cm.workflow-get`: siehe <workflow-where> <workflow-get> oder CRUL als DTD.

22.3 Funktionselemente für Dateien

22.3.1 <obj-contentTypesForObjType>

Definition:

```
<!ENTITY % cm.obj-contentTypesForObjType "
 (listitem*)
">
<!ELEMENT obj-contentTypesForObjType
          %cm.obj-contentTypesForObjType;>
<!ATTLIST obj-contentTypesForObjType
          objType CDATA #REQUIRED
>
```

Aufgabe: obj-contentTypesForObjType liefert die Liste aller Dateiendungen, die vom angegebenen Dateityp unterstützt werden.

Rückgabewert bei Erfolg: die Liste der Dateiendungen.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <obj-contentTypesForObjType objType="image"/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <obj-contentTypesForObjType>
      <listitem>jpeg</listitem>
      <listitem>jpg</listitem>
      <listitem>gif</listitem>
```

```
    </obj-contentTypesForObjType>
  </cm-code>
</cm-response>
```

22.3.2 <obj-create>

Definition:

```
<!ENTITY % cm.obj-create "
  (name |
  objClass |
  suppressExport |
  contentType |
  blob |
  filter |
  file |
  xmlBlob)*
">
<!ELEMENT obj-create %cm.obj-create;>
```

Aufgabe: Erzeugt eine neue Datei mit den angegebenen Parameterwerten.

Spezielle Parameter dieses Funktionselements:

- `blob`: der in der Arbeitsversion der Datei zu speichernde Blob. Mit dem Attribut `encoding` und dessen Werten `plain`, `base64` und `stream` werden Ursprung und Speicherformat des Blobs festgelegt:
  - `plain`: der Blob ist als Inhalt des Elements `blob` angegeben.
  - `base64`: der Blob ist als base64-kodierter Inhalt des Elements `blob` angegeben.
  - `stream`: der Inhalt des Elements `blob` ist ein Streaming-Ticket, unter dem der Blob zu finden ist. Zuvor wurde der Blob über das Streaming Interface zum Server übertragen, der das Ticket zurückgegeben hat (zum Streaming-Interface siehe auch das *Handbuch zur Systemadministration / Entwicklung*).
  - `contentType`: die Dateiendung der Arbeitsversion der Datei.
- `file`: ein relativer, aus höchstens zwei Komponenten bestehender Pfad zu der richtig kodierten zu importierenden Datei. Der Pfad bezieht sich auf das temporäre Verzeichnis des Benutzers und darf nicht mit dem übergeordneten Verzeichnis beginnen. `file` und `blob` schließen sich gegenseitig aus.
- `filter`: der Konverter, der beim Laden verwendet werden soll. Dieser Parameter ist aus Gründen der Kompatibilität noch vorhanden, wird jedoch ignoriert.

Rückgabewert bei Erfolg: die ID der neuen Datei.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionCreateChildren` in dem darüber liegenden Ordner der späteren Datei und das in der Vorlage der späteren Datei definierte globale Dateierzeugungsrecht haben, um es anlegen zu dürfen.

Beispiel:

```
<cm-request...>
  <obj-where>
    <id>2001</id>
  </obj-where>
  <obj-create>
    <objClass>news</objClass>
```

```
    <name>2007-03-26</name>
    <contentType>html</contentType>
    <blob>Ein Hauptinhalt</blob>
  </obj-create>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <obj>
      <id>2001</id>
    </obj>
  </cm-code>
</cm-response>
```

22.3.3 <obj-generatePreview>

Definition:

```
<!ELEMENT obj-generatePreview %cm.obj-generatePreview;>
<!ATTLIST obj-generatePreview
  editor CDATA #REQUIRED
  fsPrefix CDATA #REQUIRED
  masterTemplate CDATA #REQUIRED
  mode (0|1|2|3|4|5|6|7) #REQUIRED
  path CDATA #REQUIRED
  urlPrefix CDATA #REQUIRED
>
```

Aufgabe: Das Element holt eine Vorschauseite.

Spezielle Parameter dieses Funktionselements:

- `editor`: Login des Benutzers, für den die Vorschau erstellt werden soll.
- `fsPrefix`: das Zielverzeichnis für dynamisch inkludierte Dateien. Siehe den Systemkonfigurationswert `dynamicPreviewDirectory` in der Konfigurationsdatei `guiConfig.xml`.
- `masterTemplate`: der Pfad des für den Export der Vorschauseite zu verwendenden Basislayouts.
- `mode`: eine Bitmaske. Die folgenden Werte können nach Bedarf addiert werden, um die jeweils angegebenen Effekte zu erzielen:
  - 1: Arbeitsversionen bevorzugen (ansonsten 0, freigegebene Versionen verwenden)
  - 2: Arbeitsversionen von Layouts verwenden (ansonsten 0, freigegebene Versionen von Layouts verwenden).
  - 4: Reserviert.
- `path`: der Pfad der betreffenden Datei, einschließlich Dateiendung, bei Ordnern einschließlich `/index.html`.
- `urlPrefix`: Teil-URL, die der berechneten URL vorangestellt werden soll.

Rückgabewert bei Erfolg: der HTML-Text der Vorschauseite.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionRead` für alle referenzierten Dateien haben.

Beispiel:

```
<obj-generatePreview
```

```
  editor="chiefeditor"
  fsPrefix="/"
  masterTemplate="mastertemplate"
  mode="0"
  path="/index.html"
  urlPrefix="/NPS/preview/mastertemplate/0"
/>
```

22.3.4 <obj-search>

Ausschnitt aus der DTD:

```
<!ENTITY % cm.cm-request "
  ...
  (obj-search+, obj-commit) |
  (obj-search+, obj-copy) |
  (obj-search+, obj-delete) |
  (obj-search+, obj-description) |
  (obj-search+, obj-edit) |
  (obj-search+, obj-exportSubtree) |
  (obj-search+, obj-forward) |
  (obj-search+, obj-get) |
  (obj-search+, obj-give) |
  (obj-search+, obj-permissionGrantTo) |
  (obj-search+, obj-permissionRevokeFrom) |
  (obj-search+, obj-reject) |
  (obj-search+, obj-release) |
  (obj-search+, obj-removeActiveContents) |
  (obj-search+, obj-removeArchivedContents) |
  (obj-search+, obj-revert) |
  (obj-search+, obj-set) |
  (obj-search+, obj-sign) |
  (obj-search+, obj-take) |
  (obj-search+, obj-unrelease) |
  ...
">
```

Definition:

```
<!ENTITY % cm.obj-search "
 (query |
  minRelevance |
  maxDocs |
  offsetStart |
  offsetLength |
  parser |
  collections)
">
<!ELEMENT obj-search %cm.obj-search;>
```

Aufgabe: `obj-search` sucht nach Dateien, die auf die übergebene Suchanfrage passen.

Spezielle Parameter dieses Funktionselements:

- `query`: als Inhalt dieses Elements wird der Suchanfragetext übergeben.
- `minRelevance`: eine ganze Zahl zwischen 0 und 100 (jeweils einschließlich), die die Relevanz angibt, die die Treffer mindestens haben müssen, um ins Suchergebnis aufgenommen zu werden.
- `offsetStart`: der Index des ersten zurückzugebenden Dokuments im Gesamtsuchergebnis. Das erste Dokument hat den Index 1.
- `offsetLength`: die Anzahl der ab `offsetStart` zurückzugebenden Dokumente.

- `parser`: Der bei der Suche zu verwendende Parser (siehe das Handbuch *Die Infopark Search Cartridge*). Der Inhalt des `parser`-Elements muss eine der drei folgenden Zeichenketten sein: `simple`, `explicit`, `freetext`.
- `collections`: Die zu durchsuchenden Collections, jeweils als `listitem`-Unterelemente. Sind keine Collections angegeben, so werden alle Collections durchsucht. Die verfügbaren Collections sind nicht mit CRUL abfragbar (nur im SES-single-Modus mit dem Kommando `listCollections`). Die bei der Indizierung anzuwendenden Regeln zur Auswahl einer Collection sind im Systemkonfigurationseintrag `indexing.advancedSearch.collectionSelection` definierbar.

Verwendung: Wie `obj-where` ist `obj-search` ein Funktionselement, das nicht ohne ein weiteres unmittelbar darauf folgendes Funktionselement verwendet werden kann. Mit `obj-search` erhält man die Liste der Dateien, die auf die Suchanfrage im `query`-Element passen. Erst durch das auf `obj-search` folgende Funktionselement wird jedoch bestimmt, wie diese Dateien zu behandeln sind. Jedes der Elemente, die auf `obj-search` folgen können, entspricht dabei einer Schablone, die nacheinander auf jedes der mit `obj-search` ermittelten Dateien angewendet wird (siehe Requests als Schablonen und die Darstellung der Ergebnis-Daten). Auf `obj-search` können die gleichen Elemente folgen wie auf `obj-where` (siehe <obj-where>).

Rückgabewert bei Erfolg: die Liste der IDs der passenden Dateien, auf denen der Benutzer das Leserecht hat.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <obj-search>
    <query>sportsnews &lt;#IN&gt; objclass</query>
    <minRelevance>100</minRelevance>
  </obj-search>
  <obj-copy>
    <parent>/news/archiv</parent>
  </obj-copy>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <obj>
      <id>4563</id>
    <obj>...
  <cm-code/>
</cm-response>
```

22.3.5 <obj-types>

Definition:

```
<!ENTITY % cm.obj-types "
  (listitem)*
">
<!ELEMENT obj-types %cm.obj-types;>
```

Aufgabe: Listet die zulässigen Dateitypen auf.

Rückgabewert bei Erfolg: die Liste der Dateitypen.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <obj-types/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <obj-types>
      <listitem>document</listitem>
      <listitem>generic</listitem>
      <listitem>image</listitem>
      <listitem>publication</listitem>
      <listitem>template</listitem>
    </obj-types>
  </cm-code>
</cm-response>
```

22.3.6 <obj-where>

Ausschnitt aus der DTD:

```
<!ENTITY % cm.cm-request "
  ...
  (obj-where+, obj-addComment) |
  (obj-where+, obj-commit) |
  (obj-where+, obj-copy) |
  (obj-where+, obj-delete) |
  (obj-where+, obj-description) |
  (obj-where+, obj-edit) |
  (obj-where+, obj-exportSubtree) |
  (obj-where+, obj-forward) |
  (obj-where+, obj-get) |
  (obj-where+, obj-give) |
  (obj-where+, obj-permissionGrantTo) |
  (obj-where+, obj-permissionRevokeFrom) |
  (obj-where+, obj-reject) |
  (obj-where+, obj-release) |
  (obj-where+, obj-removeActiveContents) |
  (obj-where+, obj-removeArchivedContents) |
  (obj-where+, obj-revert) |
  (obj-where+, obj-set) |
  (obj-where+, obj-sign) |
  (obj-where+, obj-take) |
  (obj-where+, obj-unrelease) |
  ...
">
<!ATTLIST obj-where
    maxResults CDATA #IMPLIED
>
    Definition:
<!ENTITY % cm.obj-where "
  (id |
  ids |
  path |
  parent |
  condition*
">

<!ELEMENT obj-where %cm.obj-where;>
<!ELEMENT condition (%cm.atom; | %cm.listitem;)*>
<!ATTLIST condition
  subject CDATA #REQUIRED
  verb CDATA #REQUIRED
```

```
>
```

Aufgabe: `obj-where` ermöglicht die Suche nach Dateien, wobei die angegebenen Parameter die Trefferliste einschränken. Werden keine Parameterwerte angegeben, so wird eine Liste der IDs aller Dateien zurückgegeben. Die Höchstzahl der zu liefernden Treffer kann mit dem Tag-Attribut `maxResults` angegeben werden.

Spezielle Parameter dieses Funktionselements:

- `ids`: der Inhalt dieses Elements ist eine Liste von Datei-IDs, bei der die einzelnen IDs durch Leerzeichen voneinander separiert sind. Alternativ kann jede ID in einem separaten `listitem`-Element angegeben werden. Die Suche wird durch `ids` auf die Dateien mit den angegebenen IDs beschränkt.
- `id`: der Inhalt dieses Elements ist eine Datei-ID. Die Suche wird auf die Datei mit dieser ID beschränkt.
- `condition`: Mit `condition` kann in Verbindung mit `ids` eine Bedingung angegeben werden, die eine Datei erfüllen muss, um in die Ergebnisliste aufgenommen zu werden. Die Bedingung besteht aus drei Teilen, einer Eigenschaft, einem Vergleichsoperator und einem Vergleichswert, wobei die Eigenschaft und der Vergleichsoperator als die Tag-Attribute `subject` und `verb` und der Vergleichswert als Inhalt des `condition`-Elements angegeben werden. Es gibt die folgenden Kombinationen dieser drei Bestandteile einer Bedingung:

Eigenschaft	Operator	Vergleichswert

`name`
`title`
`nameOrTitle`
`objClass`
	`is`
	`contains`
	`startsWith`
		Zeichenkette

`objType`	`is`
	`isOneOf`
		`document`
		`publication`
		`generic`
		`image`
		`template`
		Liste mit Dateitypen (s. `is`)

`state`	`is`
	`isNot`
	`isOneOf`
		`edited`
		`committed`
		`released` Liste mit Workflowzuständen (s. `is`)

Es können mehrere Bedingungen angegeben werden, die dann implizit und-verknüpft werden. Beispiel:

```
<obj-where maxResults="20">
  <condition subject="nameOrTitle" verb="contains">foo</condition>
  <condition subject="objType" verb="isOneOf">
    <listitem>publication</listitem>
    <listitem>generic</listitem>
  </condition>
</obj-where>
```

Wird bei `objType isOneOf` oder `state isOneOf` eine leere Liste angegeben, so ist die Treffermenge leer.

`state isNot` ist die einzige Negation. Diese ist erforderlich, weil Workflowzustände nicht exklusiv sind (eine Datei kann eine Arbeitsversion im Zustand `edited` oder `committed` sowie zusätzlich eine Version im Zustand `released` haben.

Die `obj where`-Bedingungen können auch in der Template Engine verwendet werden. Allerdings gibt es dort nur freigegebene Dateien, weshalb `state` dort das Folgende liefert:

`state is` / `isOneOf`...
`released`: die Bedingung wird ignoriert
`committed` / `edited`: die Treffermenge ist immer leer

`state isNot`...
`released`: die Treffermenge ist immer leer
`committed`/ `edited`: die Bedingung wird ignoriert

Verwendung: `obj-where` ist ein Funktionselement, das nicht ohne ein weiteres unmittelbar darauf folgendes Funktionselement verwendet werden kann. Mit `obj-where` erhält man die Liste der IDs der passenden Dateien, die dem Suchkriterium entsprechen. Erst durch das folgende Funktionselement wird jedoch bestimmt, wie diese Dateien zu behandeln sind. Jedes der Elemente, die auf `obj-where` folgen können, entspricht dabei einer Schablone, die nacheinander auf jedes der mit `obj-where` ermittelten Dateien angewendet wird (siehe Requests als Schablonen und die Darstellung der Ergebnis-Daten).

Erforderliche Rechte: keine Einschränkungen.

22.3.7 <obj-where> <obj-addComment>

Definition:

```
<!ELEMENT obj-addComment %cm.atom;>
```

Aufgabe: Zu den mit `obj-where` ermittelten Dateien wird ein Kommentar hinzugefügt. Der Kommentar wird ins Protokoll geschrieben.

Verwendung: Zunächst werden mit `obj-where` Dateien ermittelt. Anschließend wird `obj-addComment` verwendet, um für diese Dateien den Kommentar zu setzen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionWrite` für jedes der mit `obj-where` ermittelten Dateien haben, oder er muss Administrator der Dateien sein.

Beispiel:

```
<cm-request...>
  <obj-where>
    <id>31477</id>
  </obj-where>
  <obj-addComment>
    Hauptinhalt um externe Links erweitert.
  </obj-addComment>
```

```
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

22.3.8 <obj-where> <obj-commit>

Definition:

```
<!ENTITY % cm.obj-commit "
  (comment)
">
<!ELEMENT obj-commit %cm.obj-commit;>
```

Aufgabe: Für die mit `obj-where` ermittelten Dateien wird die Workflow-Aktion *Einreichen* ausgeführt, wobei jeweils die Arbeitsversion in eine eingereichte Version umgewandelt wird, sofern ihr Workflow eine Unterschrift vorsieht. Andernfalls wird die Datei freigegeben.

Spezielle Parameter dieses Funktionselements:

- `comment`: ist der Kommentar, der für die nächste Gruppe, die die Datei übernimmt, eingetragen wird.

Verwendung: Zunächst werden mit `obj-where` Dateien ermittelt. Anschließend wird `obj-commit` verwendet, um für diese Dateien die Workflow-Aktion *Einreichen* auszuführen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionWrite` für jedes der mit `obj-where` ermittelten Dateien haben und ihr Bearbeiter sein, oder er muss Administrator der Dateien sein.

Beispiel:

```
<cm-request...>
  <obj-where>
    <id>31477</id>
  </obj-where>
  <obj-commit>
    <comment/>
  </obj-commit>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

22.3.9 <obj-where> <obj-copy>

Definition:

```
<!ENTITY % cm.obj-copy "
 ((parent | name)+, recursive?)
">
```

```
<!ELEMENT obj-copy %cm.obj-copy;>
<!ELEMENT recursive (%cm.atom;)>
```

Aufgabe: Die mit `obj-where` ermittelten Dateien werden kopiert und erhalten jeweils eine Arbeitsversion.

Spezielle Parameter dieses Funktionselements:

- `parent`: spezifiziert die ID des Zielordners – fehlt der Parameter, wird als Ziel der Ordner angenommen, in der sich die Datei befindet.
- `name`: gibt an, dass `value` den Namen enthält, den die Dateikopie erhalten soll. Fehlt der Name, wird der Name der zu kopierenden Datei verwendet. Gibt es diesen in dem Zielordner bereits, wird ein neuer Name nach dem Muster `Name`*X* berechnet, wobei *X* eine Zahl ist.
- `recursive`: wenn die betreffende Datei ein Ordner ist, gibt `value` an, ob die darin enthaltenen Dateien ebenfalls (rekursiv) kopiert werden sollen (`YES`, `TRUE` oder `1`) oder nicht (`NO`, `FALSE` oder `0`). Links auf Dateien innerhalb des kopierten Baumes verweisen auf die Kopien, während Links auf Dateien außerhalb des Baumes auf die bisherigen Linkziele zeigen. Voreinstellung: `FALSE`.

Zusatzinformationen: Haben die Quelldateien eine freigegebene Version, so wird diese jeweils in einen neue Arbeitsversion kopiert. Ist keine freigegebene Version vorhanden, so wird versucht, die eingereichte Version zu verwenden. Ist auch diese nicht vorhanden, wird die Arbeitsversion verwendet. Gibt es diese nicht, so erhalten die neuen Dateien eine leere Arbeitsversion.

Verwendung: Zunächst werden mit `obj-where` Dateien ermittelt, bevor `obj-copy` verwendet wird, um diese Dateien zu kopieren.

Rückgabewert bei Erfolg: die IDs der neuen Dateien.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionRead` für die Quelldateien haben; der Benutzer muss ebenfalls das Recht `permissionCreateChildren` in dem Zielordner und das in der Vorlage der Zieldateien definierte globale Dateianlegerecht haben.

Beispiel:

```
<cm-request...>
  <obj-where>
    <id>2315</id>
  </obj-where>
  <obj-copy>
    <parent>/news/archiv</parent>
  </obj-copy>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <obj>
      <id>11387</id>
    </obj>
  <cm-code/>
</cm-response>
```

22.3.10 <obj-where> <obj-delete>

Definition:

```
<!ENTITY % cm.obj-delete "EMPTY">
```

```
<!ELEMENT obj-delete %cm.obj-delete;>
```

Aufgabe: Die mit `obj-where` ermittelten Dateien werden gelöscht.

Zusatzinformationen: Eine Datei kann nur gelöscht werden, wenn es nicht der Basisordnerist, sie keine Dateien enthält hat und nicht von Links referenziert wird. Links in archivierten Versionen verhindern jedoch nicht, dass eine Version gelöscht wird. In solchen Links werden alle Informationen über die Zieldatei bis auf `expectedPath` entfernt, und `expectedPath` wird auf den Pfad gesetzt, den die Datei zum Zeitpunkt der Löschung hat.

Verwendung: Zunächst werden mit `obj-where` Dateien ermittelt. Anschließend werden diese Dateien mit `obj-delete` gelöscht.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss Superuser sein oder das Recht `permissionRoot` für die zu löschenden Dateien haben.

Beispiel:

```
<cm-request...>
  <obj-where>
    <objClass>newsPublication</objClass>
    <objType>publication</objType>
    <nameLike>January</nameLike>
    <state>archived</state>
  </obj-where>
  <obj-delete/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

22.3.11 <obj-where> <obj-description>

Definition:

```
<!ENTITY % cm.atom "#PCDATA">
<!ELEMENT obj-description %cm.atom;>
```

Aufgabe: `obj-description` liefert eine Zeichenketten-Repräsentation der wichtigsten Datei-Parameter.

Verwendung: Zunächst wird mit `obj-where` die Datei ermittelt, deren Zeichenketten-Repräsentation anschließend mit `obj-description` ausgelesen wird.

Rückgabewert bei Erfolg: die Zeichenketten-Repräsentation.

Erforderliche Rechte: keine.

Beispiel:

```
<cm-request...>
  <obj-where>
    <id>2001</id>
```

```
  </obj-where>
  <obj-description/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok>
    {
      hasChildren = 1;
      hasSuperLinks = 1;
      id = 2001;
      name = ROOTPUB;
      objType = publication;
      path = "/";
      suppressExport = 0;
    }
  </cm-code>
</cm-response>
```

22.3.12 <obj-where> <obj-edit>

Definition:

```
<!ENTITY % cm.obj-edit "
  (comment)
">
<!ELEMENT obj-edit %cm.obj-edit;>
```

Aufgabe: Für die mit `obj-where` ermittelten Dateien wird jeweils eine Arbeitsversion erzeugt.

Spezielle Parameter dieses Funktionselements:

- `comment`: ist der Kommentar, der für die nächste Gruppe, die die Datei übernimmt, eingetragen wird.

Verwendung: Zunächst werden mit `obj-where` Dateien ermittelt. Anschließend wird `obj-edit` verwendet, um für diese Dateien jeweils eine Arbeitsversion zu erzeugen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionWrite` für die mit `obj-where` ermittelten Dateien haben.

Beispiel:

```
<cm-request...>
  <obj-where>
    <id>5534</id>
  </obj-where>
  <obj-edit>
    <comment/>
  </obj-edit>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

22.3.13 <obj-where> <obj-exportSubtree>

Definition:

```
<!ENTITY % cm.obj-exportSubtree "
  (absoluteFsPrefix |
  absoluteUrlPrefix |
  exportCharset |
  filePrefix |
  templateName |
  purgeCollections)*
">
<!ELEMENT obj-exportSubtree %cm.obj-exportSubtree;>
```

Aufgabe: Die mit `obj-where` ermittelten Ordner und alle darin enthaltenen Dateien werden exportiert.

Zusatzinformationen: `obj-exportSubtree` kann nur auf Ordner angewendet werden. Beim Export wird eine der Ordnerhierarchie entsprechende Verzeichnisstruktur erzeugt, in der für jeden Ordner ein eigenes Verzeichnis erstellt wird. In diesem Verzeichnis wird der Inhalt der Ordner als `index.html` abgelegt. Dokumente werden in dieses Verzeichnis als *`dokumentname`*`.html` exportiert, Bilder und Ressourcen erhalten die Dateinamenserweiterung ihres Feldes `contentType`.

Spezielle Parameter dieses Funktionselements:

- `absoluteFsPathPrefix`: gibt an, dass als Dateipfade exportierte Links als absolute Pfade exportiert werden sollen der Wert dieses Elements als deren Präfix verwendet werden soll. Dieser Parameter hebt im Rahmen dieses Befehls die Werte der entsprechenden Parameter im Systemkonfigurationseintrag `export` auf.
- `absoluteUrlPrefix`: wie `absoluteFsPathPrefix`, jedoch für die Pfade in URLs.
- `exportCharset`: gibt den beim Export für generierte Dokumente zu verwendenden Zeichensatz an. Die unterstützten Werte sind im Systemkonfigurationseintrag `export.charsetMap` aufgeführt.
- `filePrefix`: gibt den Prefix an, der den Namen der Exportdateien voranzustellen ist.
- `templateName`: gibt den Namen eines Layouts an, das (statt des Basislayouts) beim Export verwendet werden soll.
- `purgeCollections`: gibt an, ob die Collections geleert werden sollen, bevor Inhalte exportiert werden (`1`, Voreinstellung) oder nicht (`0`).

Verwendung: Zunächst werden mit `obj-where` Dateien ermittelt. Anschließend wird `obj-exportSubtree`verwendet, um diese Dateien und sämtliche darin enthaltenen Dateien zu exportieren.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalExport` für die mit `obj-where` ermittelten Dateien haben.

Beispiel:

```
<cm-request...>
  <obj-where>
    <objClass>newsPublication</objClass>
    <title>sales_week</title>
  </obj-where>
  <obj-exportSubtree>
```

```
    <filePrefix>/news</filePrefix>
    <templateName>localtemplate</templateName>
  </obj-exportSubtree>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

22.3.14 <obj-where> <obj-forward>

Definition:

```
<!ENTITY % cm.obj-forward "
  (comment)
">
<!ELEMENT obj-forward %cm.obj-forward;>
```

Aufgabe: Reicht die mit `obj-where` ermittelten Dateien an die nächste Gruppe im Bearbeitungsworkflow weiter.

Spezielle Parameter dieses Funktionselements:

- `comment`: ist der Kommentar, der für die nächste Gruppe, die die Datei übernimmt, eingetragen wird.

Verwendung: Zunächst werden mit `obj-where` Dateien ermittelt. Anschließend wird `obj-forward` verwendet, um diese Dateien weiterzuleiten.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionWrite` für die mit `obj-where` ermittelten Dateien haben und der Bearbeiter der Dateien sein.

Beispiel:

```
<cm-request...>
  <obj-where>
    <id>34511</id>
  </obj-where>
  <obj-forward>
    <comment/>
  </obj-forward>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

22.3.15 <obj-where> <obj-get>

Definition:

```
<!ENTITY % cm.obj-get "
  (archivedContents |
  children |
```

```
  committedContentId |
  contentIds |
  contents |
  editedContentId |
  editedContent |
  exportContent |
  exportMimeType |
  getKeys |
  hasChildren |
  hasSuperLinks |
  hierarchy |
  id |
  isCommitted |
  isEdited |
  isExportable |
  isReleased |
  isRoot |
  isGoodDestination |
  isGoodParent |
  name |
  next |
  objClass |
  objType |
  objectsToRoot |
  parent |
  path |
  permission |
  permissionGrantedTo |
  prefixPath |
  previous |
  releasedContentId |
  releasedContent |
  releasedVersions |
  rootPermissionFor |
  setKeys |
  sortValue |
  superLinks |
  superObjects |
  suppressExport |
  toclist |
  validControlActionKeys |
  validCreateObjClasses |
  validObjClasses |
  validPermissions |
  version |
  visibleExportTemplates |
  visibleName |
  visiblePath |
  workflowName)*
">
<!ELEMENT obj-get %cm.obj-get;>
```

Aufgabe: Liefert den Wert der angegebenen Parameter für jede mit `obj-where` ermittelte Datei.

Verwendung: `obj-get` muss immer auf `obj-where` folgen, da zunächst die Dateien mit `obj-where` ermittelt werden müssen, auf die `obj-get` angewendet werden soll, um Parameterwerte auszulesen.

Rückgabewert bei Erfolg: die Werte der angegebenen Parameter.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionRead` für die mit `obj-where` ermittelten Dateien haben. Die Dateiattribute `id`, `name`, `visibleName`, `path`, `visiblePath` und `objType` können auch ohne `permissionRead` ausgelesen werden.

Beispiel:

```
<cm-request...>
  <obj-where>
    <id>3452</id>
```

```
  </obj-where>
  <obj-get>
    <objClass/>
  </obj-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <obj>
      <objClass>newsArticle</objClass>
    </obj>
  </cm-code>
</cm-response>
```

22.3.16 <obj-where> <obj-give>

Definition:

```
<!ENTITY % cm.obj-give "
  (comment,
  (groupName |
  userLogin))
">
<!ELEMENT obj-give %cm.obj-give;>
```

Aufgabe: Reicht die mit `obj-where` ermittelten Dateien zur Bearbeitung an eine andere Gruppe oder einen anderen Benutzer weiter.

Spezielle Parameter dieses Funktionselements:

- `comment`: ist der Kommentar, der für die nächste Gruppe, die die Datei übernimmt, eingetragen wird.
- `groupName`: gibt die Gruppe an, der die Datei übergeben wird.
- `userLogin`: gibt den Benutzer an, dem die Datei übergeben wird.

Verwendung: Zunächst werden mit `obj-where` Dateien ermittelt. Anschließend wird `obj-give` verwendet, um diese Dateien weiterzuleiten.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionWrite` für die Dateien haben.

Beispiel:

```
<cm-request...>
  <obj-where>
    <id>1311</id>
  </obj-where>
  <obj-give>
    <groupName>NewsMasters</groupName>
  </obj-give>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

22.3.17 <obj-where> <obj-mirror>

Definition:

```
<!ENTITY % cm.obj-mirror "
 (parent |
  name)*
">
<!ELEMENT obj-mirror %cm.obj-mirror;>
```

Aufgabe: Von den mit `obj-where` ermittelten Dateien wird je eine Spiegeldatei angelegt.

Spezielle Parameter dieses Funktionselements:

- `parent`: spezifiziert die ID des Zielordners – fehlt der Parameter, wird als Ziel der Ordner angenommen, in dem sich die jeweils zu spiegelnde Datei befindet. Der Zielordner darf keine Spiegeldatei sein.
- `name`: gibt den Namen der jeweiligen neuen Spiegeldatei an. Fehlt der Name, wird der Name der jeweils zu spiegelnden Datei verwendet. Gibt es diesen in dem Zielordner bereits, wird ein neuer Name nach dem Muster `Name`*`X`* berechnet, wobei *`X`* eine Zahl ist.

Verwendung: Zunächst werden mit `obj-where` Dateien ermittelt, bevor `obj-mirror` verwendet wird, um diese Dateien zu spiegeln.

Rückgabewert bei Erfolg: die IDs der neuen Dateien.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionRead` für die Quelldateien haben; der Benutzer muss ebenfalls das Recht `permissionCreateChildren` in dem Zielordner und das in der Vorlage der Zieldateien definierte globale Dateianlegerecht haben.

Beispiel:

```
<cm-request ...>
  <obj-where>
    <id type="string">44127</id>
  </obj-where>
  <obj-mirror>
    <parent type="string">93687</parent>
    <name type="string">ommit_me</name>
  </obj-mirror>
</cm-request>

<cm-response ...>
  <cm-code numeric="200" phrase="OK">
    <obj>
      <id>98617</id>
    </obj>
  </cm-code>
</cm-response>
```

22.3.18 <obj-where> <obj-permissionGrantTo>

Definition:

```
<!ENTITY % cm.obj-permissionGrantTo "
  (group+)
">
```

```
<!ELEMENT obj-permissionGrantTo %cm.obj-permissionGrantTo;>
<!ATTLIST obj-permissionGrantTo
         permission (permissionCreateChildren |
                     permissionWrite |
                     permissionRead |
                     permissionRoot |
                     permissionLiveServerRead) #REQUIRED
>
```

Aufgabe: Den angegebenen Gruppen wird das angegebene dateispezifische Recht für die mit `obj-where` ermittelten Dateien erteilt. Andere Rechte der Gruppen bleiben unberührt.

Bedeutung der Attribute:

- `permission`: spezifiziert ein dateispezifisches Recht.

Spezielle Parameter dieses Funktionselements:

- `group`: enthält den Namen der Gruppe, der das gewünschte dateispezifische Recht zugewiesen werden soll.

Verwendung: Zunächst werden mit `obj-where` Dateien ermittelt. Anschließend wird `obj-permissionGrantTo` verwendet, um Gruppen ein dateispezifisches Recht für diese Dateien zu erteilen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionRoot` für die Dateien haben.

Beispiel:

```
<cm-request...>
  <obj-where>
    <objClass>newsPublication</objClass>
    <nameLike>new</nameLike>
  </obj-where>
  <obj-permissionGrantTo permission="permissionWrite">
    <group>NewsMasters</group>
  </obj-permissionGrantTo
>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

22.3.19 <obj-where> <obj-permissionRevokeFrom>

Definition:

```
<!ENTITY % cm.obj-permissionRevokeFrom "
  (group+)
">
<!ELEMENT obj-permissionRevokeFrom %cm.obj-permissionRevokeFrom;>
<!ATTLIST obj-permissionRevokeFrom
         permission (permissionCreateChildren |
                     permissionWrite |
                     permissionRead |
                     permissionRoot |
```

```
                    permissionLiverServerRead) #REQUIRED
>
```

Aufgabe: Den angegebenen Gruppen wird das spezifizierte Recht für die mit `obj-where` ermittelten Dateien entzogen.

Bedeutung der Attribute:

- `permission`: spezifiziert ein dateispezifisches Recht.

Spezielle Parameter dieses Funktionselements:

- `group`: enthält den Namen einer Gruppe, der das gewünschte dateispezifische Recht entzogen werden soll.

Verwendung: Zunächst werden mit `obj-where` Dateien ermittelt. Anschließend wird `obj-permissionRevokeFrom` verwendet, um Gruppen das angegebene dateispezifische Recht für diese Dateien zu entziehen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionRoot` für die mit `obj-where` ermittelten Dateien haben.

Beispiel:

```
<cm-request...>
  <obj-where>
    <id>3543</id>
  </obj-where>
  <obj-permissionRevokeFrom permission="permissionWrite">
    <group>NewsMasters</group>
  </obj-permissionRevokeFrom
>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

22.3.20 <obj-where> <obj-reject>

Definition:

```
<!ENTITY % cm.obj-reject "
  (comment)
">
<!ELEMENT obj-reject %cm.obj-reject;>
```

Aufgabe: Bricht für die mit `obj-where` ermittelten Dateien den aktuellen Workflow ab und beginnt mit einem neuen Workflow (Workflow-Aktion *Ablehnen*).

Zusatzinformationen: Falls die Datei eine eingereichte Version hat, wird sie in eine Arbeitsversion umgewandelt.

Spezielle Parameter dieses Funktionselements:

- `comment`: ist der Kommentar, der für die nächste Gruppe, die die Datei übernimmt, eingetragen wird.

Verwendung: Zunächst werden mit `obj-where` Dateien ermittelt. Anschließend wird `obj-reject` verwendet, um den Workflow für diese Dateien abzubrechen und wieder neu zu beginnen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Wenn es eine Arbeitsversion gibt, muss der authentifizierte Benutzer das Recht `permissionWrite` für jede Datei haben. Gibt es eine eingereichte Version, muss der Benutzer das Recht `permissionRead` für die Dateien haben und Mitglied in einer der verbleibenden unterschriftsberechtigten Gruppen sein oder das Recht `permissionRoot` für die mit `obj-where` ermittelten Dateien haben.

Beispiel:

```
<cm-request...>
  <obj-where>
    <id>6634</id>
  </obj-where>
  <obj-reject>
    <comment/>
  </obj-reject>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

22.3.21 <obj-where> <obj-release>

Definition:

```
<!ENTITY % cm.obj-release "
  (comment)
">
<!ELEMENT obj-release %cm.obj-release;>
```

Aufgabe: Gibt die mit `obj-where` ermittelten Dateien frei.

Zusatzinformationen: Sollte die Datei noch nicht eingereicht sein, so wird sie vorher eingereicht (d. h. einer `commit`-Operation unterzogen). Ferner werden noch fehlende Unterschriften erzeugt.

Spezielle Parameter dieses Funktionselements:

- `comment`: ist der Kommentar, der für die nächste Gruppe, die die Datei übernimmt, eingetragen wird.

Verwendung: Zunächst werden mit `obj-where` Dateien ermittelt. Anschließend wird `obj-release` verwendet, um diese Dateien freizugeben.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Wenn es eine Arbeitsversion für eine Datei gibt, muss der authentifizierte Benutzer der Bearbeiter sein und das Schreibrecht haben (`permissionWrite`). Ferner muss die Freigabe der letzte Schritt im Workflow der Versionen sein, es sei denn, der Benutzer hat das

Dateiadministrationsrecht (`permissionRoot`). In diesem Fall kann er die Version unabhängig vom Workflow freigeben, wenn er der Bearbeiter ist.
Wenn es eine eingereichte Version gibt und nur eine Unterschrift fehlt, muss der Benutzer Mitglied in einer Gruppe sein, die zu dieser Unterschrift berechtigt ist, und das Leserecht haben.

Beispiel:

```
<cm-request...>
  <obj-where>
    <id>9785</id>
  </obj-where>
  <obj-release>
    <comment/>
  </obj-release>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

22.3.22 <obj-where> <obj-removeActiveContents>

Definition:

```
<!ENTITY % cm.obj-removeActiveContents "EMPTY">
<!ELEMENT obj-removeActiveContents %cm.obj-removeActiveContents;>
```

Aufgabe: Löscht alle aktiven Contents der Dateien, die mit obj-where ermittelt wurden. Das bedeutet, der aktuelle Arbeitscontent oder der eingereichte Content und der aktuelle freigegebene Content werden gelöscht. Archivierte Contents werden nicht gelöscht (siehe dazu <obj-where> <obj-removeArchivedContents> und <content-where> <content-delete>).

Zusatzinformationen: Falls ein freigegebener Content existiert, so wird er nur gelöscht, wenn die Archivierung ausgeschaltet ist. Ansonsten wird er in einen archivierten Content umgewandelt.

Verwendung: Zunächst werden mit `obj-where` Dateien ermittelt, bevor `obj-removeActiveContents` verwendet wird, um die aktiven Contents dieser Dateien zu löschen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss der Administrator der Dateien sein, zu denen die Versionen gehören.

Beispiel:

```
<cm-request...>
  <obj-where>
    <id>2441</id>
  </obj-where>
  <obj-removeActiveContents/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

22.3.23 <obj-where> <obj-removeArchivedContents>

Definition:

```
<!ENTITY % cm.obj-removeArchivedContents "EMPTY">
<!ELEMENT obj-removeArchivedContents
  %cm.obj-removeArchivedContents;>
```

Aufgabe: Löscht alle archivierten Versionen der Dateien, die mit obj-where ermittelt wurden. Das bedeutet, dass alle Versionen bis auf die aktuelle Arbeitsversion oder eingereichte Version und die aktuellen freigegebene Version gelöscht werden.

Verwendung: Zunächst werden mit `obj-where` Dateien ermittelt, bevor `obj-removeArchivedContents` verwendet wird, um die archivierten Versionen dieser Dateien zu löschen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss der Administrator der Dateien sein, zu denen die Versionen gehören.

Beispiel:

```
<cm-request...>
  <obj-where>
    <id>2441</id>
  </obj-where>
  <obj-removeArchivedContents/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

22.3.24 <obj-where> <obj-revert>

Definition:

```
<!ENTITY % cm.obj-revert "
  (comment)
">
<!ELEMENT obj-revert %cm.obj-revert;>
```

Aufgabe: Löscht die Arbeitsversionen der Dateien, die mit `obj-where` ermittelt wurden.

Spezielle Parameter dieses Funktionselements:

- `comment`: ist der Kommentar, der für die nächste Gruppe, die die Datei übernimmt, eingetragen wird.

Verwendung: Zunächst werden mit `obj-where` Dateien ermittelt. Anschließend wird `obj-revert` verwendet, um die Arbeitsversionen dieser Dateien zu löschen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionWrite` für die mit `obj-where` ermittelten Dateien haben.

Beispiel:

```
<cm-request...>
  <obj-where>
    <id>378</id>
  </obj-where>
  <obj-revert>
    <comment/>
  </obj-revert>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

22.3.25 <obj-where> <obj-set>

Definition:

```
<!ENTITY % cm.obj-set "
  (suppressExport |
  name |
  objClass |
  parent |
  permission |
  workflowName)*
">
<!ELEMENT obj-set %cm.obj-set;>
```

Aufgabe: Setzt bei den mit `obj-where` ermittelten Dateien die spezifizierten Dateiparameter auf die angegebenen Werte.

Verwendung: Zunächst werden mit `obj-where` Dateien ermittelt. Anschließend wird `obj-set` verwendet, um Parameterwerte dieser Dateien zu setzen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionWrite` für die Dateien haben.

Beispiel:

```
<cm-request...>
  <obj-where>
    <id>8812</id>
  </obj-where>
  <obj-set>
    <permission permission="permissionRoot" type="list">
      <listitem>NewsEditors</listItem>
      <listitem>NewsMasters</listItem>
    </permission>
  </obj-set>
</cm-request

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

22.3.26 <obj-where> <obj-sign>

Definition:

```
<!ENTITY % cm.obj-sign "
  (comment)
">
<!ELEMENT obj-sign %cm.obj-sign;>
```

Aufgabe: Versieht die eingereichte Version einer Datei mit der Signatur des authentifizierten Benutzers und erzeugt jeweils einen neuen Task.

Zusatzinformationen: Die letzte Unterschrift in einem Workflow kann nur geleistet werden, indem die Datei freigegeben wird.

Spezielle Parameter dieses Funktionselements:

- `comment`: ist der Kommentar, der für die nächste Gruppe, die die Datei übernimmt, eingetragen wird.

Verwendung: Zunächst werden mit `obj-where` Dateien ermittelt, bevor anschließend `obj-sign` verwendet wird.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionRead` für die mit `obj-where` ermittelten Dateien haben und Mitglied einer Gruppe sein, die berechtigt ist, die Unterschrift zu leisten.

Beispiel:

```
<cm-request...>
  <obj-where>
    <id>5322</id>
  </obj-where>
  <obj-sign>
    <comment/>
  </obj-sign>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

22.3.27 <obj-where> <obj-take>

Definition:

```
<!ENTITY % cm.obj-take "
  (comment)
">
<!ELEMENT obj-take %cm.obj-take;>
```

Aufgabe: Macht den authentifizierten Benutzer zum Bearbeiter einer Datei und erzeugt jeweils einen neuen Task.

Spezielle Parameter dieses Funktionselements:

- `comment`: ist der Kommentar, der für die nächste Gruppe, die die Datei übernimmt, eingetragen wird.

Verwendung: Zunächst werden mit `obj-where` Dateien ermittelt. Anschließend wird `obj-take` verwendet.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionWrite` für die mit `obj-where` ermittelten Dateien haben.

Beispiel:

```
<cm-request...>
  <obj-where>
    <id>7523</id>
  </obj-where>
  <obj-take>
    <comment/>
  </obj-take>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

22.3.28 <obj-where> <obj-touch>

Definition:

```
<!ENTITY % cm.obj-touch "EMPTY">
<!ELEMENT obj-touch %cm.obj-touch;>
```

Aufgabe: Der ermittelten Dateien werden durch diesen Befehl beim nächsten Export mit der Template Engine neu exportiert. Dieser Befehl ist zu Testzwecken verfügbar. Es ist normalerweise nicht erforderlich, ihn zu verwenden.

Verwendung: Zunächst werden mit `obj-where` Dateien ermittelt. Anschließend wird `obj-touch` verwendet, um diese Dateien für den erneuten Export zu markieren.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionRoot` für die Dateien haben.

Beispiel:

```
<cm-request...>
  <obj-where>
    <objClass>newsPublication</objClass>
    <nameLike>new</nameLike>
  </obj-where>
```

```
  <obj-touch />
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

22.3.29 <obj-where> <obj-unrelease>

Definition:

```
<!ENTITY % cm.obj-unrelease "
  (comment)
">
<!ELEMENT obj-unrelease %cm.obj-unrelease;>
```

Aufgabe: Die Freigabe der mit `obj-where` ermittelten Dateien wird zurückgenommen.

Zusatzinformationen: Die aktuelle freigegebene Version einer Datei wird archiviert, wenn die Archivierung eingeschaltet ist. Wenn es keine Arbeitsversion gibt, so wird die freigegebene Version in eine neue Arbeitsversion übernommen. Die Datei verfügt anschließend über keine freigegebene Version mehr.

Spezielle Parameter dieses Funktionselements:

- `comment`: ist der Kommentar, der für die nächste Gruppe, die die Datei übernimmt, eingetragen wird.

Verwendung: Zunächst werden mit `obj-where` Dateien ermittelt. Anschließend wird `obj-unrelease` verwendet, um die Freigabe dieser Dateien zurückzunehmen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionRoot` für die Dateien haben.

Beispiel:

```
<cm-request...>
  <obj-where>
    <objClass>newsPublicatio</objClass>
    <nameLike>new</nameLike>
  </obj-where>
  <obj-unrelease>
    <comment/>
  </obj-unrelease>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

23

23 Vorlagen - objClass

23.1 Definition

Mit Requests an den Content Manager kann man lesend und schreibend auf Vorlagen zugreifen. Der folgende DTD-Ausschnitt zeigt die Elemente, mit denen dies über die XML-Schnittstelle des Content Management Servers möglich ist:

```
<!ENTITY % cm.cm-request "
  ...
  (objClass-create) |
  (objClass-goodAvailableBlobEditorsForObjType) |
  (objClass-validAttributes) |
  (objClass-where+, objClass-delete) |
  (objClass-where+, objClass-description) |
  (objClass-where+, objClass-get) |
  (objClass-where+, objClass-set) |
  ...
">
<!ATTLIST objClass-where
  maxResults CDATA #IMPLIED
>
```

Diese Elemente müssen sich unmittelbar unterhalb des `cm-request`-Elements befinden. Sie werden im Abschnitt Funktionselemente für Vorlagen erläutert.

Mit dem Attribut `maxResults`, dessen Wert eine ganze Zahl ist, kann die maximale Anzahl der Treffer festgelegt werden. Ist der Wert von `maxResults` kleiner oder gleich null, ist die Anzahl nicht begrenzt.

23.1.1 Rechte

In Requests, mit denen auf Vorlagen zugegriffen wird, und in den Responses des Content Managers, können grundsätzlich die folgenden Bezeichner für globale Rechte verwendet werden.

Globale Rechte	Bedeutung
`permissionGlobalRoot`	Administrationsrecht
`permissionGlobalUserEdit`	Benutzer und Gruppen anlegen und bearbeiten
`permissionGlobalUserAttributeEdit`	Benutzerattribute anlegen und bearbeiten
`permissionGlobalRTCEdit`	Attribute, Workflows und Vorlagen bearbeiten
`permissionGlobalExport`	Das Recht, Dateien zu exportieren

Globale Rechte können im Systemkonfigurationseintrag `content.globalPermissions` frei definiert werden. Dies bedeutet, dass Sie globale Rechte zum Content Management Server hinzufügen können, indem Sie die Liste `globalPermissions` erweitern.

23.2 Parameterelemente für Vorlagen

Mit Funktionselementen kann man auf Vorlagen zugreifen. So lassen sich beispielsweise mit dem `objClass-get`-Element die Werte sämtlicher Vorlagen-Parameter ermitteln. Um zu spezifizieren, welche Vorlageneigenschaften ausgelesen oder gesetzt werden sollen, verwendet man Parameterelemente. Im Folgenden werden die Parameterelemente für Vorlagen aufgeführt.

attributeGroups

Bedeutung: Die Namen der Feldgruppen der Vorlage. Die Reihenfolge der zurückgegebenen Namen entspricht der Reihenfolge der Gruppen.

Definition:

```
<!ELEMENT attributeGroups ((attributeGroup)* | (listitem)*)>
<!ELEMENT attributeGroup (%cm.attributeGroup-get;)>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.attributeGroup-get`: siehe <attributeGroup-where> <attributeGroup-get> oder CRUL als DTD.

attributes

Bedeutung: Die Liste der Felder, die in der Vorlage verwendet werden.

Definition:

```
<!ELEMENT attributes ((attribute)* | (listitem)* | (dictitem)+)>
<!ELEMENT attribute (%cm.atom; | %cm.attribute-get;)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.attribute-get`: siehe <attribute-where> <attribute-get> oder CRUL als DTD.

availableBlobEditors

Bedeutung: Die Liste der Editoren, mit denen der Haupttext der Arbeitsversion bearbeitet werden kann. Die Liste kann bis zu vier der folgenden Elemente enthalten: `internalEditor`, `externalEditor`, `htmlEditor`, `tinymceEditor`.

Definition:

```
<!ELEMENT availableBlobEditors (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
```

```
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

bodyTemplateName

Bedeutung: Der Name des Layouts, mit dem die jeweiligen Hauptinhalte aller Dateien exportiert werden, die auf der Vorlage basieren.

Definition:

```
<!ELEMENT bodyTemplateName (%cm.atom;)>
```

canCreateNewsItems

Bedeutung: Gibt an, ob für Dateien mit dieser Vorlage beim Freigeben News-Einträge erzeugt werden sollen.

Definition:

```
<!ELEMENT canCreateNewsItems (%cm.atom;)>
```

completionCheck

Bedeutung: Benutzerdefinierbares Tcl-Skript, mit dem zusätzliche Vollständigkeitsüberprüfungen vorgenommen werden können. Das Skript wird immer dann aufgerufen, wenn eine Version eingereicht wird. Eine Version kann nur eingereicht werden, wenn alle Links aufgelöst, die obligatorischen Felder mit gültigen Werten belegt sind und der String `completionCheck` leer ist oder ein Skript enthält, das als Ergebnis 1 zurückliefert.

Definition:

```
<!ELEMENT completionCheck (%cm.atom;)>
```

contentTypes

Bedeutung: Die Liste der für die Version einer Datei mit dieser Vorlage zulässigen Dateiendungen. Die Liste ergibt sich aus `validContentTypes` unter Berücksichtigung von `obj-contentTypesForObjType`.

Definition:

```
<!ELEMENT contentTypes (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
```

```
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

createPermission

Bedeutung: Das für die Erzeugung einer Datei mit dieser Vorlage erforderliche Recht.

Definition:

```
<!ELEMENT createPermission (%cm.atom;)>
```

customBlobEditorUrl

Bedeutung: Die URL, an die ein Request gesendet wird, wenn der Hauptinhalt einer Arbeitsversion mit dem kundenspezifischen Editor bearbeitet werden soll (siehe auch availableBlobEditors).

Definition:

```
<!ELEMENT customBlobEditorUrl (%cm.atom;)>
```

defaultAttributeGroup

Bedeutung: Der Name der Basisfeldergruppe (`defaultGroup`).

Definition:

```
<!ELEMENT defaultAttributeGroup (%cm.atom; | attributeGroup)*
<!ELEMENT attributeGroup (%cm.attributeGroup-get;)>
```

`cm.attributeGroup-get`: siehe <attributeGroup-where> <attributeGroup-get> oder CRUL als DTD

displayTitle

Bedeutung: Der in der HTML-Benutzerschnittstelle angezeigte Titel der Vorlage (eine Kombination aus Titel und Namen).

Definition:

```
<!ELEMENT displayTitle (%cm.atom;)>
```

emptyAttributeGroups

Bedeutung: Die Liste der Feldergruppen, denen keine Felder zugewiesen wurden.

Definition:

```
<!ELEMENT emptyAttributeGroups ((attributeGroup)* | (listitem)*)>
<!ELEMENT attributeGroup (%cm.attributeGroup-get;)>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.attributeGroup-get`: siehe <attributeGroup-where> <attributeGroup-get> oder CRUL als DTD.

getKeys

Bedeutung: Liste der mit `objClass-get` abfragbaren Parameter.

Definition:

```
                <!ELEMENT getKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

goodAttributeGroupAttributes

Bedeutung: Die Liste der Felder, die in Feldergruppen der Vorlage aufgenommen werden können.

Definition:

```
<!ELEMENT goodAttributeGroupAttributes ((attribute)* | (listitem)*)>
<!ELEMENT attribute (%cm.atom; | %cm.attribute-get;)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.attribute-get`: siehe <attribute-where> <attribute-get> oder CRUL als DTD.

goodAttributes

Bedeutung: Die Liste der Felder, die in die Vorlage aufgenommen werden können.

Definition:

```
<!ELEMENT goodAttributes ((attribute)* | (listitem)*)>
<!ELEMENT attribute (%cm.atom; | %cm.attribute-get;)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
```

```
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.attribute-get`: siehe <attribute-where> <attribute-get> oder CRUL als DTD.

goodMandatoryAttributes

Bedeutung: Die Liste der Felder, die in die Liste der `mandatoryAttributes` der Vorlage aufgenommen werden können.

Definition:

```
<!ELEMENT goodMandatoryAttributes ((attribute)* | (listitem)*)>
<!ELEMENT attribute (%cm.atom; | %cm.attribute-get;)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.attribute-get`: siehe <attribute-where> <attribute-get> oder CRUL als DTD.

goodPresetAttributes

Bedeutung: Die Liste der Felder, die in die Liste der `presetAttributes` der Vorlage aufgenommen werden können.

Definition:

```
<!ELEMENT goodPresetAttributes ((attribute)* | (listitem)*)>
<!ELEMENT attribute (%cm.atom; | %cm.attribute-get;)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.attribute-get`: siehe <attribute-where> <attribute-get> oder CRUL als DTD.

goodPresetFromParentAttributes

Bedeutung: Die Liste der Felder, die in die Liste der `presetFromParentAttributes` der Vorlage aufgenommen werden können.

Definition:

```
<!ELEMENT goodPresetFromParentAttributes ((attribute)* | (listitem)*)>
<!ELEMENT attribute (%cm.atom; | %cm.attribute-get;)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.attribute-get`: siehe <attribute-where> <attribute-get> oder CRUL als DTD.

isEnabled

Bedeutung: Gibt an, ob die Vorlage Dateien zugewiesen werden darf.

Definition:

```
<!ELEMENT isEnabled (%cm.atom;)>
```

localizedTitle

Bedeutung: Der Titel in der Sprache, die der authentifizierte Benutzer eingestellt hat. Ist dieser leer, wird `title` zurückgegeben. Ist auch dieser Titel leer, wird `name` zurückgegeben.

Definition:

```
<!ELEMENT localizedTitle (%cm.atom;)></
```

mandatoryAttributes

Bedeutung: Die Liste der obligatorischen Felder einer Datei, die auf dieser Vorlage beruht.

Definition:

```
<!ELEMENT mandatoryAttributes ((attribute)* | (listitem)*)>
<!ELEMENT attribute (%cm.atom; | %cm.attribute-get;)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.attribute-get`: siehe <attribute-where> <attribute-get> oder CRUL als DTD.

name

Bedeutung: Der Name der Vorlage.

Definition:

```
<!ELEMENT name (%cm.atom;)>
```

objType

Bedeutung: Der Typ einer Datei mit dieser Vorlage (`document`, `publication`, `template`, `image`, `generic`).

Definition:

```
<!ELEMENT objType (%cm.atom;)>
```

resetAttributes

Bedeutung: Die Liste der Felder, die bei der Dateierzeugung mit vordefinierten Werten aus der Vorlage belegt werden. Die Liste enthält paarweise die Feldnamen und die Werte (zulässig sind alle in der `attributes`-Liste aufgeführten und die vordefinierten Felder).

Definition:

```
<!ELEMENT presetAttributes (namevalue)*>
<!ELEMENT namevalue (name, value)>
<!ELEMENT name (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

presetFromParentAttributes

Bedeutung: Liste der Felder, deren Werte aus dem übergeordneten Ordner einer Datei mit dieser Vorlage übernommen werden (zulässig sind alle in der `attributes`-Liste aufgeführten und die vordefinierten Felder).

Definition:

```
<!ELEMENT presetFromParentAttributes ((attribute)* | (listitem)*)>
<!ELEMENT attribute (%cm.atom; | %cm.attribute-get;)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.attribute-get`: siehe <attribute-where> <attribute-get> oder CRUL als DTD.

recordSetCallback

Bedeutung: Benutzerdefinierter Tcl-Code, der aufgerufen wird, wenn einer Arbeitsversion einer Datei mit dieser Vorlage Feldwerte zugewiesen werden).

Definition:

```
<!ELEMENT recordSetCallback (%cm.atom;)>
```

setKeys

Bedeutung: Die Liste der mit `objClass-set` setzbaren Parameter.

Definition:

```
<!ELEMENT setKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`title`

Bedeutung: Der Titel der Vorlage.

Definition:

```
<!ELEMENT title (%cm.atom;)>
<!ATTLIST title
          lang (en | de | it | fr | es) #IMPLIED
>
```

Bedeutung der Attribute:

- `lang`: Kennzeichnet die Sprache eines Vorlagentitels. Zu Dimensionen von Werten im CMS siehe Dimensionen von Werten.

Beispiel:

```
<cm-request...>
  <objClass-where>
    <name>newsArticle</name>
  </objClass-where>
  <objClass-get>
    <title lang="en"/>
  </objClass-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <objClass>
      <title lang="en">This week in News and International</title>
    </objClass>
  </cm-code>
</cm-response>
```

`validContentTypes`

Bedeutung: Die Liste der für die Version einer Datei mit dieser Vorlage zulässigen Content-Typen. Ist die Liste leer, so sind alle Dateiendungen erlaubt, die `obj-contentTypesForObjType` liefert.

Definition:

```
<!ELEMENT validContentTypes (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

validSubObjClassCheck

Bedeutung: Tcl-Code, der aufgerufen wird, wenn eine Datei in einem Ordner angelegt werden soll, der auf dieser Vorlage beruht.

Definition:

```
<!ELEMENT validSubObjClassCheck (%cm.atom;)>
```

validSubObjClasses

Bedeutung: Die Liste der Namen der Vorlagen, die für Dateien eines Ordners mit dieser Vorlage zulässig sind. Diese Liste darf nur in Vorlagen belegt sein, mit denen Dateien vom Typ *Ordner* angelegt werden.

Definition:

```
<!ELEMENT validSubObjClasses ((objClass)* | (listitem)*)>
<!ELEMENT objClass (%cm.atom; | %cm.objClass-get;)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.objClass-get`: siehe <objClass-where> <objClass-get> oder CRUL als DTD.

workflowModification

Bedeutung: Tcl-Code, der aufgerufen wird, bevor der Arbeitsversion einer Datei, die auf dieser Vorlage beruht, ein Workflow zugewiesen wird.

Definition:

```
<!ELEMENT workflowModification (%cm.atom;)>
```

xmldtd

Bedeutung: Die zur Vorlage gehörende XML-DTD.

Definition:

```
<!ELEMENT xmldtd (%cm.atom;)>
```

23.3 Funktionselemente für Vorlagen

23.3.1 <objClass-create>

Definition:

```
<!ENTITY % cm.objClass-create "
  (attributes |
  bodyTemplateName |
  completionCheck |
  createPermission |
  isEnabled |
  mandatoryAttributes |
  name |
  objType |
  presetAttributes |
  presetFromParentAttributes |
  recordSetCallback |
  title |
  validContentTypes |
  validSubObjClassCheck |
  validSubObjClasses |
  workflowModification)*
">
<!ELEMENT objClass-create %cm.objClass-create;>
```

Aufgabe: Erzeugt eine neue Vorlage mit den angegebenen Parameterwerten. Die Runtime-Konfiguration wird neu geschrieben, nachdem `objClass-create` ausgeführt wurde.

Rückgabewert bei Erfolg: der Name der Vorlage.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalRTCEdit` haben.

Beispiel:

```
<cm-request...>
  <objClass-create>
    <attributes>
      <listitem>abstract</listitem>
      <listitem>author</listitem>
      <listitem>keywords</listitem>
      <listitem>newsSign</listitem>
    </attributes>
    <mandatoryAttributes>
      <listitem>abstract</listitem>
      <listitem>author</listitem>
      <listitem>keywords</listitem>
    </mandatoryAttributes>
    <name>newsArticle</name>
    <objType>document</objType>
    <presetAttributes>
      <namevalue>
        <name>keywords</name>
        <value>
          <listitem>business</listitem>
          <listitem>comment</listitem>
          <listitem>special</listitem>
          <listitem>sport</listitem>
          <listitem>world news</listitem>
        </value>
      </namevalue>
    </presetAttributes>
  </objClass-create>
```

```
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <objClass>
      <name>newsArticle</name>
    </objClass>
  </cm-code>
</cm-response>
```

23.3.2 <objClass-goodAvailableBlobEditorsForObjType>

Definition:

```
<!ENTITY % cm.objClass-goodAvailableBlobEditorsForObjType "
  (listitem*)
">
<!ELEMENT objClass-goodAvailableBlobEditorsForObjType
          %cm.objClass-goodAvailableBlobEditorsForObjType;>
<!ATTLIST objClass-goodAvailableBlobEditorsForObjType
          objType CDATA #REQUIRED
>
```

Aufgabe: Erzeugt eine Liste der Editoren, mit denen der Hauptinhalt der Arbeitsversion von Dateien des angegebenen Typs bearbeitbar ist.

Rückgabewert bei Erfolg: die Liste der verfügbaren Editoren.

Erforderliche Rechte: Keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <objClass-goodAvailableBlobEditorsForObjType objType="document"/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <objClass-goodAvailableBlobEditorsForObjType objType="document">
      <listitem>internalEditor</listitem>
      <listitem>externalEditor</listitem>
      <listitem>customEditor</listitem>
      <listitem>htmlEditor</listitem>
    </objClass-goodAvailableBlobEditorsForObjType>
  </cm-code>
</cm-response>
```

23.3.3 <objClass-validAttributes>

Definition:

```
<!ENTITY % cm.objClass-validAttributes "
  (listitem)*
">
<!ELEMENT objClass-validAttributes %cm.objClass-validAttributes;>
```

Aufgabe: Gibt die Liste sämtlicher Felder zurück, die in jeder Vorlage gesetzt werden können.

Rückgabewert bei Erfolg: die Liste der Feldnamen.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <objClass-validAttributes/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <objClass-validAttributes>
      <listitem>abstract</listitem>
      <listitem>author</listitem>
      <listitem>keywords</listitem>
      <listitem>newsSign</listitem>
      <listitem>title</listitem>
      <listitem>validFrom</listitem>
      <listitem>validUntil</listitem>
    </objClass-validAttributes>
  </cm-code>
</cm-response>
```

23.3.4 <objClass-where>

Ausschnitt aus der DTD:

```
<!ENTITY % cm.cm-request "
  ...
  (objClass-where+, objClass-delete) |
  (objClass-where+, objClass-description) |
  (objClass-where+, objClass-get) |
  (objClass-where+, objClass-set) |
  ...
">
```

Definition:

```
<!ENTITY % cm.objClass-where "
  (name |
  (nameLike?,
  isEnabled?,
  objType?))?
">
<!ELEMENT objClass-where %cm.objClass-where;>
```

Aufgabe: `objClass-where` ermöglicht die Suche nach Vorlagen, bei denen der Wert der angegebenen Parameter den jeweils spezifizierten String enthält. Werden keine Parameterwerte angegeben, so werden die Namen aller Vorlagen zurückgegeben.

Spezielle Parameter dieses Funktionselements:

- `nameLike`: bewirkt, dass die Namen der Vorlagen zurückgegeben werden, bei denen die angegebene Zeichenkette im Namen der Vorlagen enthalten ist.

Verwendung: `objClass-where` ist ein Funktionselement, das nicht ohne ein weiteres unmittelbar darauf folgendes Funktionselement verwendet werden kann. Mit `objClass-where` erhält man die Liste der Namen der Vorlagen, die dem Suchkriterium entsprechen. Erst durch das folgende

Funktionselement wird jedoch bestimmt, wie diese Vorlagen zu behandeln sind. Jedes der Elemente, die auf `objClass-where` folgen können, entspricht dabei einer Schablone, die nacheinander auf jede der mit `objClass-where` ermittelten vorlagen angewendet wird (siehe Requests als Schablonen und die Darstellung der Ergebnis-Daten).

Erforderliche Rechte: keine Einschränkungen.

23.3.5 <objClass-where> <objClass-delete>

Definition:

```
<!ENTITY % cm.objClass-delete "EMPTY">
<!ELEMENT objClass-delete %cm.objClass-delete;>
```

Aufgabe: Die mit `objClass-where` ermittelten Vorlagen werden gelöscht.

Zusatzinformationen: Eine Vorlage kann nicht gelöscht werden, wenn sie noch von einer Klassendefinition eines Ordners referenziert wird, bei der also die im Ordner enthaltenen Dateien auf der zu löschenden Vorlage basieren. Es ist jedoch durchaus möglich, eine Vorlage zu löschen, auf der Dateien basieren.

Verwendung: Die Vorlagen werden zunächst mit `objClass-where` ermittelt. Anschließend werden diese Vorlagen mit `objClass-delete` gelöscht.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalRTCEdit` haben.

Beispiel:

```
<cm-request...>
  <objClass-where>
    <name>newsPublication</name>
  </objClass-where>
  <objClass-delete/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

23.3.6 <objClass-where> <objClass-description>

Definition:

```
<!ENTITY % cm.atom "#PCDATA">
<!ELEMENT objClass-description %cm.atom;>
```

Aufgabe: `objClass-description` liefert eine Zeichenketten-Repräsentation der wichtigsten Vorlagenparameter.

Verwendung: Zunächst wird mit `objClass-where` die Vorlage ermittelt, deren Zeichenketten-Repräsentation anschließend mit `objClass-description` ausgelesen wird.

Rückgabewert bei Erfolg: die Zeichenketten-Repräsentation.

Erforderliche Rechte: keine.

Beispiel:

```
<cm-request...>
  <objClass-where>
    <name>newslist</name>
  </objClass-where>
  <objClass-description/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok>
    {
      attributes = (
        abstract,
        source
      );
      isEnabled = 1;
      name = newslist;
      objType = publication;
      presetAttributes = {};
      title = "News-Liste";
      validSubObjClasses = (
        news
      );
    }
  </cm-code>
</cm-response>
```

23.3.7 <objClass-where> <objClass-get>

Definition:

```
<!ENTITY % cm.objClass-get "
  (attributeGroups |
  attributes |
  availableBlobEditors |
  bodyTemplateName |
  completionCheck |
  createPermission |
  customBlobEditorUrl |
  defaultAttributeGroup |
  displayTitle |
  emptyAttributeGroups |
  getKeys |
  goodAttributeGroupAttributes |
  goodAttributes |
  goodAvailableBlobEditors |
  goodMandatoryAttributes |
  goodPresetAttributes |
  goodPresetFromParentAttributes |
  isEnabled |
  localizedTitle |
  mandatoryAttributes |
  name |
  objType |
  presetAttributes |
  presetFromParentAttributes |
  recordSetCallback |
  setKeys |
  title |
  validContentTypes |
```

```
  validSubObjClassCheck |
  validSubObjClasses |
  workflowModification |
  xmldtd)*
">
<!ELEMENT objClass-get %cm.objClass-get;>
```

Aufgabe: Liefert die Werte der angegebenen Parameter für jede der mit `objClass-where` ermittelten Vorlagen.

Verwendung: Zunächst werden mit `objClass-where` Vorlagen ermittelt. Anschließend wird `objClass-get` verwendet, um Parameterwerte dieser Vorlagen auszulesen.

Rückgabewert bei Erfolg: die Werte der angegebenen Parameter.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <objClass-where>
    <name>newsArticle</name>
  </objClass-where>
  <objClass-get>
    <attributes/>
    <mandatoryAttributes/>
    <objType/>
    <presetAttributes/>
  </objClass-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <objClass>
      <attributes>
        <listitem>abstract</listitem>
        <listitem>author</listitem>
        <listitem>keywords</listitem>
        <listitem>newsSign</listitem>
        <listitem>title</listitem>
        <listitem>validFrom</listitem>
        <listitem>validUntil</listitem>
      </attributes>
      <mandatoryAttributes>
        <listitem>abstract</listitem>
        <listitem>author</listitem>
        <listitem>keywords</listitem>
      </mandatoryAttributes>
      <name>newsArticle</name>
      <objType>document</objType>
      <presetAttributes>
        <namevalue>
          <name>keywords</name>
          <value>
            <listitem>business</listitem>
            <listitem>comment</listitem>
            <listitem>special</listitem>
            <listitem>sport</listitem>
            <listitem>world news</listitem>
          </value>
        </namevalue>
      </presetAttributes>
    </objClass>
  </cm-code>
</cm-response>
```

23.3.8 <objClass-where> <objClass-set>

Definition:

```
<!ENTITY % cm.objClass-set "
  (attributes |
  availableBlobEditors |
  bodyTemplateName |
  completionCheck |
  createPermission |
  customBlobEditorUrl |
  isEnabled |
  mandatoryAttributes |
  name |
  presetAttributes |
  presetFromParentAttributes |
  recordSetCallback |
  title |
  validContentTypes |
  validSubObjClassCheck |
  validSubObjClasses |
  workflowModification)*
">
<!ELEMENT objClass-set %cm.objClass-set;>
```

Aufgabe: Setzt für die mit `objClass-where` ermittelten Vorlagen die spezifizierten Parameter auf die angegebenen Werte.

Verwendung: Die Vorlagen werden zunächst mit `objClass-where` ermittelt, bevor `objClass-set` verwendet werden kann, um Parameterwerte dieser Vorlagen zu setzen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalRTCEdit` haben.

Beispiel:

```
<cm-request...>
  <objClass-where>
    <name>newsArticle</name>
  </objClass-where>
  <objClass-set>
    <attributes>
      <attribute>header1</attribute>
      <attribute>header2</attribute>
    </attributes>
    <isEnabled>1</isEnabled>
  </objClass-set>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

24

24 Systemkonfiguration - systemConfig

Die `systemConfig`-Elemente ermöglichen den lesenden Zugriff auf die Systemkonfiguration, die der Content Management Server beim Programmstart aus seinen Initialisierungsdateien einliest. Über die XML-Schnittstelle des Content Managers können bis auf den Konfigurationsbereich `userManagement.preferences` (siehe Benutzerkonfiguration - userConfig) keine Änderungen an der Systemkonfiguration vorgenommen werden.

Das XML-Format, in dem die in der Systemkonfiguration enthaltenen Daten abgelegt sind und die verfügbaren Datenstrukturen werden im Handbuch zur *Systemadministration / Entwicklung* erläutert.

24.1 Definition

Der folgende Ausschnitt aus der DTD des Content Management Servers zeigt die Elemente, mit denen auf die Systemkonfiguration in Requests an den Content Manager lesend zugegriffen werden kann:

```
<!ENTITY % cm.cm-request "
  ...
  (systemConfig-formatDate) |
  (systemConfig-formatDateTime) |
  (systemConfig-getAttributes) |
  (systemConfig-getCounts) |
  (systemConfig-getElements) |
  (systemConfig-getKeys) |
  (systemConfig-getTexts) |
  (systemConfig-installedLanguages) |
  (systemConfig-parseInputDate) |
  (systemConfig-validInputCharsets) |
  (systemConfig-validTimeZones) |
  ...
">
```

Diese Elemente, die sich unmittelbar unterhalb des `cm-request`-Elements befinden müssen, werden im folgenden Abschnitt erläutert.

24.2 Funktionselemente für die Systemkonfiguration

24.2.1 <systemConfig-formatDate>

Definition:

```
<!ENTITY % cm.atom "#PCDATA">
<!ELEMENT userConfig-formatDate (%cm.atom;)>
```

Aufgabe: Formatiert das angegebene Datum im Standard-Datumsformat. Das Datum wird in die Standardzeitzone des Servers konvertiert.

Zusatzinformationen: Als Zeitzone des Servers wird der benutzerspezifische Konfigurationswert `timeZone` verwendet. Als Ausgabeformat wird der Wert aus dem Dictionary `export.validDateTimeOutputFormats` verwendet, dessen Name im benutzerspezifischen Konfigurationswert `dateTimeOutputFormatName` steht.

Rückgabewert bei Erfolg: das formatierte Datum.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <systemConfig-formatDate>20010102</systemConfig-formatDate>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <systemConfig-formatDate>02.01.2001</systemConfig-formatDate>
  </cm-code>
</cm-response>
```

24.2.2 <systemConfig-formatDateTime>

Definition:

```
<!ENTITY % cm.atom "#PCDATA">
<!ELEMENT systemConfig-formatDateTime (%cm.atom;)>
```

Aufgabe: Formatiert das angegebene Datum im Standard-Zeitstempelformat. Datum und Uhrzeit werden in die Standardzeitzone des Servers konvertiert.

Zusatzinformationen: Als Zeitzone des Servers wird der benutzerspezifische Konfigurationswert `timeZone` verwendet. Als Ausgabeformat wird der Wert aus dem Dictionary `export.validDateOutputFormats` verwendet, dessen Name im benutzerspezifischen Konfigurationswert `dateTimeOutputFormatName` steht.

Rückgabewert bei Erfolg: der formatierte Zeitstempel.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <systemConfig-formatDateTime>20010102180001
  <systemConfig-formatDateTime>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <systemConfig-formatDateTime>02.01.2001 19:00 MET
    </systemConfig-formatDateTime>
  </cm-code>
</cm-response>
```

24.2.3 <systemConfig-getAttributes>

Definition:

```
<!ENTITY % cm.systemConfig-getAttributes "
  (key,
  attributeNames?)
">
<!ELEMENT systemConfig-getAttributes
          %cm.systemConfig-getAttributes;>
```

Aufgabe: Liefert zum angegebenen Systemkonfigurationselement die spezifizierten Tag-Attribute und deren Werte.

Spezielle Parameter dieses Funktionselements:

- `key`: der Name des Konfigurationselements, dessen Tag-Attribute gesucht werden. Werden keine Tag-Attribute angegeben, so werden sämtliche Tag-Attribute mit ihren Werten geliefert.
- `attributeNames`: enthält die Namen eines Tag-Attributs, dessen Wert zurückgegeben werden soll.

Rückgabewert bei Erfolg: die Liste der Tag-Attribute mit ihren Werten.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <systemConfig-getAttributes>
    <key>server</key>
  </systemConfig-getAttributes>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <systemConfig-getAttributes>
      <dictitem>
        <key>fileName</key>
        <value>server.xml</value>
      </dictitem>
    </systemConfig-getAttributes>
  </cm-code>
</cm-response>
```

24.2.4 <systemConfig-getCounts>

Definition:

```
<!ENTITY % cm.systemConfig-getCounts "
  (listitem+)
">
<!ELEMENT systemConfig-getCounts %cm.systemConfig-getCounts;>
```

Aufgabe: Liefert die Anzahl der Elemente, die unter dem spezifizierten Konfigurationselement gespeichert sind.

Rückgabewert bei Erfolg: eine Liste mit ebenso vielen Zahlen wie Elemente angegeben wurden. Jede Zahl bezeichnet die Anzahl Unterelemente des entsprechenden Konfigurationselements.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <systemConfig-getCounts>
    <listitem>export.globalPermissions</listitem>
    <listitem>tuning.slave</listitem>
  </systemConfig-getCounts>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <systemConfig-getCounts>
      <listitem>5</listitem>
      <listitem>2</listitem>
    </systemConfig-getCounts>
  </cm-code>
</cm-response>
```

24.2.5 <systemConfig-getElements>

Definition:

```
<!ENTITY % cm.systemConfig-getElements "
  (listitem+)
">
<!ELEMENT systemConfig-getElements %cm.systemConfig-getElements;>
```

Aufgabe: Liefert Systemkonfigurationselemente mit ihren Unterelementen und deren Werten.

Rückgabewert bei Erfolg: eine Liste der angegebenen Elemente. Für jedes angegebene Element werden die Namen der Unterelemente und deren Werte als Zeichenkette zurückgegeben. In dieser Zeichenkette sind die XML-spezifischen Zeichen wie '<' in die entsprechenden XML-Zeichen-Referenzen umgewandelt.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <systemConfig-getElements>
    <listitem>tuning.master</listitem>
    <listitem>tuning.slave</listitem>
  </systemConfig-getElements>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <systemConfig-getElements>
      <listitem>
        &lt;?xml version=&quot;1.0&quot;?&gt;
        &lt;master fileName=&quot;master.xml&quot;&gt;
          &lt;maxSlaves&gt;1&lt;/maxSlaves&gt;
```

```
          &lt;minIdleSlaves&gt;1&lt;/minIdleSlaves&gt;
          &lt;slaveShutdownTimeout&gt;600&lt;/slaveShutdownTimeout&gt;
        &lt;/master&gt;
      </listitem>
      <listitem>
        &lt;?xml version=&quot;1.0&quot;?&gt;
        &lt;slave fileName=&quot;slave.xml&quot;&gt;
          &lt;maxNumberOfRequests&gt;400&lt;/maxNumberOfRequests&gt;
          &lt;requestTimeout&gt;700&lt;/requestTimeout&gt;
        &lt;/slave&gt;
      </listitem>
    </systemConfig-getElements>
  </cm-code>
</cm-response>
```

24.2.6 <systemConfig-getKeys>

Definition:

```
<!ENTITY % cm.systemConfig-getKeys "
  (listitem+)
">
<!ELEMENT systemConfig-getKeys %cm.systemConfig-getKeys;>
```

Aufgabe: Liefert die Namen der Unterelemente der spezifizierten Systemkonfigurationselemente.

Rückgabewert bei Erfolg: eine Liste, die für jedes angegebene Element die Liste der Namen der Unterelemente enthält.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <systemConfig-getKeys>
    <listitem>tuning.master</listitem>
    <listitem>tuning.slave</listitem>
  </systemConfig-getKeys>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <systemConfig-getKeys>
      <listitem>
        <listitem>maxSlaves</listitem>
        <listitem>minIdleSlaves</listitem>
        <listitem>slaveShutdownTimeout</listitem>
      </listitem>
      <listitem>
        <listitem>maxNumberOfRequests</listitem>
        <listitem>requestTimeout</listitem>
      </listitem>
    </systemConfig-getKeys>
  </cm-code>
</cm-response>
```

24.2.7 <systemConfig-getTexts>

Definition:

```
<!ENTITY % cm.systemConfig-getTexts "
  (listitem+)
">
<!ELEMENT systemConfig-getTexts %cm.systemConfig-getTexts;>
```

Aufgabe: Die Funktion liefert die Inhalte der spezifizierten Systemkonfigurationselemente.

Zusatzinformationen: Ein Konfigurationselement kann als Inhalt entweder Unterelemente oder einen Zeichenkettenwert enthalten, nicht jedoch beides. Daher liefert die Funktion eine leere Zeichenkette, wenn das auszulesende Konfigurationselement Unterelemente hat.

Rückgabewert bei Erfolg: die Liste der Inhalte der spezifizierten Konfigurationselemente.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <systemConfig-getTexts>
    <listitem>tuning.slave</listitem>
    <listitem>content.sortKey</listitem>
  </systemConfig-getTexts>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <systemConfig-getTexts>
      <slave/>
      <sortKey>name</sortKey>
    </systemConfig-getTexts>
  </cm-code>
</cm-response>
```

24.2.8 <systemConfig-installedLanguages>

Definition:

```
<!ENTITY % cm.systemConfig-installedLanguages "
  (listitem+)
">
<!ELEMENT systemConfig-installedLanguages
          %cm.systemConfig-installedLanguages>
```

Aufgabe: Liefert die installierten Sprachen des Content Management Servers. Diese Sprachen entsprechen denen, für die es Localizer gibt.

Rückgabewert bei Erfolg: die Liste der installierten Sprachen.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <systemConfig-installedLanguages/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
```

```
    <systemConfig-installedLanguages>
      <listitem>german</listitem>
      <listitem>english</listitem>
      <listitem>french</listitem>
      <listitem>spanish</listitem>
      <listitem>italian</listitem>
    </systemConfig-parseInputDate>
  </cm-code>
</cm-response>
```

24.2.9 <systemConfig-parseInputDate>

Definition:

```
<!ENTITY % cm.date "
 (%cm.atom; |
  isoDateTime |
  systemConfigFormattedTime |
  userConfigFormattedTime)*
">
<!ELEMENT systemConfig-parseInputDate (%cm.date;)>
```

Aufgabe: Interpretiert den angegebenen String als Eingabewert für ein Datum und liefert die entsprechenden Datumsstrings.

Spezielle Parameter dieses Funktionselements:

- `isoDateTime`: Datum, als Zeitstempel in kanonischer Form formatiert.
- `systemConfigFormattedTime`: Datum, das entsprechend dem in der Systemkonfiguration angegebenen Wert für den Konfigurationseintrag `preferences.dateTimeOutputFormatName` formatiert wird.
- userConfigFormattedTime: Datum, das entsprechend dem in den persönlichen Einstellungen des authentifizierten Benutzers angegebenen Wert für den Konfigurationseintrag dateTimeOutputFormatName formatiert wird.

Rückgabewert bei Erfolg: die Liste der Datumsstrings.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <systemConfig-parseInputDate>02.06.2001 19:00 MET
  </systemConfig-parseInputDate>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <systemConfig-parseInputDate>
      <isoDateTime>20010602180001</isoDateTime>
      <systemConfigFormattedTime>02.06.2001 17:00
      </systemConfigFormattedTime>
      <userConfigFormattedTime>02.06.2001 19:00
      </userConfigFormattedTime>
    </systemConfig-parseInputDate>
  </cm-code>
</cm-response>
```

24.2.10 <systemConfig-validInputCharsets>

Definition:

```
<!ENTITY % cm.systemConfig-validInputCharsets "
  (listitem+)
">
<!ELEMENT systemConfig-validInputCharsets
          %cm.systemConfig-validInputCharsets;>
```

Aufgabe: Liefert die verfügbaren Zeichensätze des Content Management Servers. Einer dieser Zeichensätze kann beim Dateiimport als Zeichensatz des zu importierenden Dokuments angegeben werden.

Rückgabewert bei Erfolg: die Liste der verfügbaren Zeichensätze.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <systemConfig-validInputCharsets/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <systemConfig-validInputCharsets>
      <listitem>iso8859-1</listitem>
      ...
      <listitem>ucs2</listitem>
    </systemConfig-validInputCharsets>
  </cm-code>
</cm-response>
```

24.2.11 <systemConfig-validTimeZones>

Definition:

```
<!ENTITY % cm.systemConfig-validTimeZones "
  (listitem+)
">
<!ELEMENT systemConfig-validTimeZones
          %cm.systemConfig-validTimeZones>
```

Aufgabe: Liefert die verfügbaren Zeitzonen des Content Management Servers.

Rückgabewert bei Erfolg: die Liste der verfügbaren Zeitzonen.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <systemConfig-validTimeZones/>
</cm-request>

<cm-response...>
```

```
  <cm-code numeric="0" phrase="ok">
    <systemConfig-validTimeZones>
      <listitem>GMT-12</listitem>
      ...
      <listitem>GMT+13</listitem>
    </systemConfig-validTimeZones>
  </cm-code>
</cm-response>
```

25

25 Tasks - task

25.1 Definition

Der folgende DTD-Ausschnitt zeigt die Elemente, mit denen auf Tasks in Requests an den Content Management Server zugegriffen werden kann:

```
<!ENTITY % cm.cm-request "
  ...
  (task-where+, task-delete) |
  (task-where+, task-description) |
  (task-where+, task-get) |
  ...
">
<!ATTLIST task-where
  maxResults CDATA #IMPLIED
>
```

Diese Elemente, die sich unmittelbar unterhalb des `cm-request`-Elements befinden müssen, werden im Abschnitt Funktionselemente für Tasks erläutert.

Mit dem Attribut `maxResults`, dessen Wert eine ganze Zahl ist, kann die maximale Anzahl der Treffer festgelegt werden. Ist der Wert von `maxResults` kleiner oder gleich null, ist die Anzahl nicht begrenzt

25.2 Parameterelemente für Tasks

Auf Tasks kann mit Hilfe von Funktionselementen zugegriffen werden. Beispielsweise können mit dem `task-get`-Element die Werte sämtlicher Task-Parameter ermittelt werden. Welche Taskeigenschaften ausgelesen oder welche Tasks gelöscht werden sollen, spezifiziert man mit Parameterelementen. Im Folgenden werden die Task-Parameter und die zugehörigen Parameterelemente aufgeführt.

`comment`

Bedeutung: Kommentar der auslösenden Workflow-Aktion.

Definition:

```
<!ELEMENT comment (%cm.atom;)>
```

displayTitle

Bedeutung: Der in der HTML-Benutzerschnittstelle angezeigte Titel der Aufgabe.

Definition:

```
<!ELEMENT displayTitle (%cm.atom;)>
```

getKeys

Bedeutung: Die Liste der mit `task-get` abfragbaren Parameter.

Definition:

```
<!ELEMENT getKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

groupName

Bedeutung: Der Name der Gruppe, für deren Mitglieder der Eintrag gilt.

Definition:

```
<!ELEMENT groupName (%cm.atom; | %cm.group-get;)*>
```

`cm.group-get`: siehe <group-where> <group-get> oder CRUL als DTD.

objectId

Bedeutung: Die ID der Datei, auf die sich der Eintrag bezieht.

Definition:

```
<!ELEMENT objectId (%cm.atom; | %cm.obj-get;)*>
```

`cm.obj-get`: siehe <obj-where> <obj-get> oder CRUL als DTD.

taskType

Bedeutung: Die Art der durchzuführenden Aufgabe. Kann `edit` oder `sign` sein.

Definition:

```
<!ELEMENT taskType (%cm.atom;)>
```

timeStamp

Bedeutung: Das Datum, an dem die Aufgabe erzeugt wurde.

Definition:

```
<!ELEMENT timeStamp (%cm.atom;)>
```

title

Bedeutung: Der Titel des tasks.

Definition:

```
<!ELEMENT title (%cm.atom;)>
```

userLogin

Bedeutung: Der Name des Benutzers, für den der Eintrag gilt.

Definition:

```
<!ELEMENT userLogin (%cm.atom; | %cm.user-get;)*>
```

`cm.user-get`: siehe <user-where> <user-get> oder CRUL als DTD.

25.3 Funktionselemente für Tasks

25.3.1 <task-where>

Ausschnitt aus der DTD:

```
<!ENTITY % cm.cm-request "
  ...
  (task-where+, task-delete) |
  (task-where+, task-description) |
  (task-where+, task-get) |
  ...
">
```

Definition:

```
<!ENTITY % cm.task-where "
```

```
  (id |
  (objectId,
  userLogin?,
  groupNames?,
  taskType?) |
  taskText)?
">
<!ELEMENT task-where %cm.task-where;>
```

Aufgabe: `task-where` ermöglicht die Suche nach Tasks, bei denen der Wert der angegebenen Parameter den jeweils spezifizierten String enthält. Eine Liste der IDs aller Tasks wird zurückgegeben, wenn keine Parameterwerte angegeben werden.

Spezielle Parameter dieses Funktionselements:

- `id`: die ID des gesuchten Tasks.
- `groupNames`: Es werden nur die Tasks zurückgegeben, die den unter `groupNames` angegebenen Gruppen zugeordnet sind.
- `taskText`: Text, der im Kommentar (`comment`) oder Titel (`title`) eines Tasks vorkommen muss.

Verwendung: `task-where` ist ein Funktionselement, das nicht ohne ein weiteres unmittelbar darauf folgendes Funktionselement verwendet werden kann. Mit `task-where` erhält man die Liste der IDs der Tasks, die dem Suchkriterium entsprechen. Erst durch das folgende Funktionselement wird jedoch bestimmt, wie diese Tasks zu behandeln sind. Jedes der Elemente, die auf `task-where` folgen können, entspricht dabei einer Schablone, die nacheinander auf jeden der mit `task-where` ermittelten Tasks angewendet wird (siehe Requests als Schablonen und die Darstellung der Ergebnis-Daten).

Rückgabewert bei Erfolg: die Liste der IDs der Tasks.

Erforderliche Rechte: keine Einschränkungen.

25.3.2 <task-where> <task-delete>

Definition:

```
<!ENTITY % cm.task-delete "EMPTY">
<!ELEMENT task-delete %cm.task-delete;>
```

Aufgabe: Die mit `task-where` ermittelten Tasks werden gelöscht.

Verwendung: Zunächst werden mit `task-where` Tasks ermittelt. Anschließend werden diese Tasks mit `task-delete` gelöscht.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss Superuser sein.

Beispiel:

```
<cm-request...>
  <task-where>
    <groupNames>
      <listitem>NewsEditors</listitem>
      <listitem>NewsMasters</listitem>
    </groupNames>
  </task-where>
  <task-delete/>
```

```
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

25.3.3 <task-where> <task-description>

Definition:

```
<!ENTITY % cm.atom "#PCDATA">
<!ELEMENT task-description %cm.atom;>
```

Aufgabe: `task-description` liefert eine Zeichenketten-Repräsentation der wichtigsten Task-Parameter.

Verwendung: Zunächst wird mit `task-where` der Task ermittelt, dessen Zeichenketten-Repräsentation anschließend mit `task-description` ausgelesen wird.

Rückgabewert bei Erfolg: die Zeichenketten-Repräsentation.

Erforderliche Rechte: keine.

Beispiel:

```
<cm-request...>
  <task-where>
    <id>20188.50</id>
  </task-where>
  <task-description/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok>
    {
      objId = 2001;
      taskType = edit;
      timeStamp = 20011129221803;
      title = "new\ content\ created";
      userLogin = root;
    }
  </cm-code>
</cm-response>
```

25.3.4 <task-where> <task-get>

Definition:

```
<!ENTITY % cm.task-get "
  (comment |
  displayTitle |
  getKeys |
  groupName |
  objectId |
  taskType |
  timeStamp |
  title |
  userLogin)*
```

```
">
<!ELEMENT task-get %cm.task-get;>
```

Aufgabe: Liefert den Wert der spezifizierten Parameter für die mit `task-where` ermittelten Tasks.

Verwendung: Die Tasks werden zunächst mit `task-where` ermittelt, bevor anschließend `task-get` verwendet werden kann, um Parameterwerte dieser Tasks auszulesen.

Rückgabewert bei Erfolg: die Werte der angegebenen Parameter.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <task-where>
    <id>4226.52</id>
  </task-where>
  <task-get>
    <userLogin/>
  </task-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <task>
      <userLogin>john</userLogin>
    </task>
  </cm-code>
</cm-response>
```

26

26 Workflows - workflow

26.1 Definition

In Requests an den Content Management Server kann man lesend und schreibend auf Workflows zugreifen. Der folgende Ausschnitt aus der DTD des Content Managers zeigt die Elemente, mit denen dies möglich ist:

```
<!ENTITY % cm.cm-request "
  ...
  (workflow-create) |
  (workflow-where+, workflow-delete) |
  (workflow-where+, workflow-description) |
  (workflow-where+, workflow-get) |
  (workflow-where+, workflow-set)
  ...
">
<!ATTLIST workflow-where
  maxResults CDATA #IMPLIED
>
```

Diese Elemente, die sich unmittelbar unterhalb des `cm-request`-Elements befinden müssen, werden im Abschnitt Funktionselemente für Workflows erläutert.

Mit dem Attribut `maxResults`, dessen Wert eine ganze Zahl ist, kann die maximale Anzahl der Treffer festgelegt werden. Ist der Wert von `maxResults` kleiner oder gleich null, ist die Anzahl nicht begrenzt.

26.2 Parameterelemente für Workflows

Mit Hilfe von Funktionselementen kann man auf Workflows zugreifen. Beispielsweise können mit dem `workflow-get`-Element die Werte sämtlicher Workflow-Parameter ermittelt werden. Um zu spezifizieren, welche Workfloweigenschaften ausgelesen oder gesetzt werden sollen, verwendet man Parameterelemente. Im Folgenden werden die Parameterelemente für Workflows aufgeführt.

`allowsMultipleSignatures`

Bedeutung: Gibt an, ob ein und dieselbe Person mehrere Unterschriften unter eine Version setzen darf.

Definition:

```
<!ELEMENT allowsMultipleSignatures (%cm.atom;)>
```

displayTitle

Bedeutung: Der in der HTML-Benutzerschnittstelle angezeigte Titel des Workflows (eine Kombination aus Titel und Namen).

Definition:

```
<!ELEMENT displayTitle (%cm.atom;)>
```

editGroups

Bedeutung: Gibt die Gruppen im Edit-Workflow, also die Bearbeitergruppen an.

Definition:

```
<!ELEMENT editGroups ((group)* | (listitem)*)>
<!ELEMENT group (%cm.atom; | %cm.group-get;)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

`cm.group-get`: siehe <group-where> <group-get> oder CRUL als DTD.

getKeys

Bedeutung: Die Liste der mit `workflow-get` abfragbaren Parameter.

Definition:

```
<!ELEMENT getKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

isEnabled

Bedeutung: Gibt an, ob neue Worflows dieses Typs begonnen werden dürfen.

Definition:

```
<!ELEMENT isEnabled (%cm.atom;)>
```

name

Bedeutung: Der Name des Workflows.

Definition:

```
<!ELEMENT name (%cm.atom;)>
```

title

Bedeutung: Der Titel des Workflows.

Definition:

```
<!ELEMENT title (%cm.atom;)>
<!ATTLIST title
          lang (en | de | it | fr | es) #IMPLIED
>
```

Bedeutung der Attribute:

- `lang`: Kennzeichnet die Sprache eines Workflowtitels. Zu Dimensionen von Werten im CMS siehe Dimensionen von Werten.

Beispiel:

```
<cm-request...>
  <workflow-where>
    <name>newsSport</name>
  </workflow-where>
  <workflow-get>
    <title lang="en"/>
  </workflow-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <workflow>
      <title lang="en">sport:footballArticleWorkflow</title>
    </workflow>
  </cm-code>
</cm-response>
```

setKeys

Bedeutung: Die Liste der mit `workflow-set` setzbaren Parameter.

Definition:

```
<!ELEMENT setKeys (listitem)*>
<!ELEMENT listitem (%cm.atom; | listitem | dictitem)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
```

```
<!ELEMENT value (%cm.atom; | listitem | dictitem)*>
```

signatureDefs

Bedeutung: Die Unterschriften und unterschriftsberechtigten Gruppen im Signature-Workflow.

Definition:

```
<!ELEMENT signatureDefs (signature)*>
<!ELEMENT signature (attribute, group)>
<!ELEMENT attribute (%cm.atom; | %cm.attribute-get;)*>
<!ELEMENT group (%cm.atom; | %cm.group-get;)*>
```

`cm.attribute-get`: siehe <attribute-where> <attribute-get>
`cm.group-get`: siehe <group-where> <group-get>.

26.3 Funktionselemente für Workflows

26.3.1 <workflow-create>

Definition:

```
<!ENTITY % cm.workflow-create "
  (name,
  editGroups?,
  signatureDefs?,
  allowsMultipleSignatures?,
  isEnabled?,
  title?)
">
<!ELEMENT workflow-create %cm.workflow-create;>
```

Aufgabe: workflow-create erzeugt einen neuen Workflow mit den angegebenen Parameterwerten. Die Runtime-Konfiguration wird neu geschrieben.

Rückgabewert bei Erfolg: der Name des Workflows.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalRTCEdit` haben.

Beispiel:

```
<cm-request...>
  <workflow-create>
    <name>wf_signNews</name>
    <editGroups>
      <group>NewsEditors</group>
      <group>NemsMasters</group>
    </editGroups>
    <signatureDefs>
      <signature>
        <attribute>signNews</attribute>
        <group>NewsMasters</group>
```

```
      </signature>
      <signature>
        <attribute>signNews2</attribute>
        <group>NewsAdmins</group>
      </signature>
    </signatureDefs>
  </workflow-create>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <workflow>
      <name>wf_signNews</name>
    </workflow>
  </cm-code>
</cm-response>
```

26.3.2 <workflow-where>

Ausschnitt aus der DTD:

```
<!ENTITY % cm.cm-request "
  ...
  (workflow-where+, workflow-delete) |
  (workflow-where+, workflow-description) |
  (workflow-where+, workflow-get) |
  (workflow-where+, workflow-set)
">
```

Definition:

```
<!ENTITY % cm.workflow-where "
  (name |
  nameLike)*
">
<!ELEMENT workflow-where %cm.workflow-where;>
```

Aufgabe: `workflow-where` ermöglicht die Suche nach Workflows, bei denen der Wert des angegebenen Parameterelements den jeweils spezifizierten String enthält. Werden keine Parameterwerte angegeben, so werden die Namen sämtlicher Workflows zurückgegeben.

Spezielle Parameter dieses Funktionselements:

- `nameLike`: bewirkt, dass die Namen der Workflows zurückgegeben werden, bei denen die im `nameLike`-Element angegebene Zeichenkette im Namen der Workflows enthalten ist.

Verwendung: `workflow-where` ist ein Funktionselement, das nicht ohne ein weiteres unmittelbar darauf folgendes Funktionselement verwendet werden kann. Mit `workflow-where` erhält man die Liste der Workflownamen, die dem Suchkriterium entsprechen. Erst durch das folgende Funktionselement wird jedoch bestimmt, wie diese Workflows zu behandeln sind. Jedes der Elemente, die auf `workflow-where` folgen können, entspricht dabei einer Schablone, die nacheinander auf jeden der mit `workflow-where` ermittelten Workflows angewendet wird (siehe Requests als Schablonen und die Darstellung der Ergebnis-Daten).

Rückgabewert bei Erfolg: die Liste der Namen der Workflows.

Erforderliche Rechte: keine Einschränkungen.

26.3.3 <workflow-where> <workflow-delete>

Definition:

```
<!ENTITY % cm.workflow-delete "EMPTY">
<!ELEMENT workflow-delete %cm.workflow-delete;>
```

Aufgabe: Die mit `workflow-where` ermittelten Workflows werden gelöscht.

Verwendung: Zunächst werden mit `workflow-where` Workflows ermittelt. Anschließend werden diese Workflows mit `workflow-delete` gelöscht.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalRTCEdit` haben.

Beispiel:

```
<cm-request...>
  <workflow-where>
    <name>wf_article</name>
  </workflow-where>
  <workflow-delete/>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

26.3.4 <workflow-where> <workflow-description>

Definition:

```
<!ENTITY % cm.atom "#PCDATA">
<!ELEMENT workflow-description %cm.atom;>
```

Aufgabe: `workflow-description` liefert eine Zeichenketten-Repräsentation der wichtigsten Workflow-Parameter.

Verwendung: Zunächst wird mit `workflow-where` der Workflow ermittelt, dessen Zeichenketten-Repräsentation anschließend mit `workflow-description` ausgelesen wird.

Rückgabewert bei Erfolg: die Zeichenketten-Repräsentation.

Erforderliche Rechte: keine.

Beispiel:

```
<cm-request...>
  <workflow-where>
    <name>wf_sig_news</name>
  </workflow-where>
  <workflow-description/>
</cm-request>
```

```
<cm-response...>
  <cm-code numeric="0" phrase="ok>
    {
      allowsMultipleSignatures = 1;
      editGroups = (
        editors
      );
      isEnabled = 1;
      signatureDefs = (
        (
          "sig_news",
          newsadmins
        )
      );
    }
  </cm-code>
</cm-response>
```

26.3.5 <workflow-where> <workflow-get>

Definition:

```
<!ENTITY % cm.workflow-get "
  (displayTitle |
  name |
  editGroups |
  signatureDefs |
  allowsMultipleSignatures |
  getKeys |
  isEnabled |
  setKeys |
  title)*
">
<!ELEMENT workflow-get %cm.workflow-get;>
```

Aufgabe: Liefert den Wert der angegebenen Parameter für die mit `workflow-where` ermittelten Workfows.

Verwendung: Zunächst werden mit workflow-where Workflows ermittelt. Anschließend wird `workflow-get` verwendet, um Parameterwerte dieser Workflows auszulesen.

Rückgabewert bei Erfolg: die Werte der angegebenen Workflowparameter.

Erforderliche Rechte: keine Einschränkungen.

Beispiel:

```
<cm-request...>
  <workflow-where>
    <name>wf_signNews</name>
  </workflow-where>
  <workflow-get>
    <editGroups/>
    <signatureDefs/>
    <isEnabled/>
  </workflow-get>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok">
    <workflow>
      <editGroups>
        <group>NewsEditors</group>
```

```
        <group>NewsMasters</group>
      </editGroups>
      <signatureDefs>
        <signature>
          <attribute>signNews</attribute>
          <group>NewsAdmins</group>
        </signature>
      </signatureDefs>
      <isEnabled>1</isEnabled>
    </workflow>
  </cm-code>
</cm-response>
```

26.3.6 <workflow-where> <workflow-set>

Definition:

```
<!ENTITY % cm.workflow-set "
  (editGroups |
  signatureDefs |
  allowsMultipleSignatures |
  isEnabled |
  title)*
">
<!ELEMENT workflow-set %cm.workflow-set;>
```

Aufgabe: Setzt die Parameter der mit workflow-where ermittelten Workflows auf die angegebenen Werte.

Verwendung: Zunächst werden mit `workflow-where` Workflows ermittelt. Anschließend wird `workflow-set` verwendet, um Parameterwerte dieser Workflows zu setzen.

Rückgabewert bei Erfolg: keiner.

Erforderliche Rechte: Der authentifizierte Benutzer muss das Recht `permissionGlobalRTCEdit` haben.

Beispiel:

```
<cm-request...>
  <workflow-where>
    <name>wf_signNews</name>
  </workflow-where>
  <workflow-set>
    <isEnabled>0</isEnabled>
  </workflow-set>
</cm-request>

<cm-response...>
  <cm-code numeric="0" phrase="ok"/>
</cm-response>
```

27

27 Die CRUL-DTD

Dieses Dokument enthält die Definition von CRUL (*Content Retrieval and Update Language*) als DTD (*Document Type Definition*). Wenn Sie wissen möchten, welche Elemente mit welchen Attributen CRUL unterstützt, so können Sie sie in diesem Dokument finden. Änderungen sind vorbehalten.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT cm-payload (cm-header, (cm-request+ | cm-response+))>
<!ATTLIST cm-payload
        payload-id CDATA #REQUIRED
        timestamp CDATA #REQUIRED
        cm.version CDATA #REQUIRED
>
<!ENTITY % cm.atom "#PCDATA">
<!ENTITY % cm.dictitem "dictitem">
<!ENTITY % cm.listitem "listitem">
<!ELEMENT listitem (%cm.atom; | %cm.listitem; | %cm.dictitem;)*>
<!ELEMENT dictitem (key, value)>
<!ELEMENT key (%cm.atom;)>
<!ELEMENT value (%cm.atom; | %cm.listitem; | %cm.dictitem;)*>
<!ENTITY % cm.date "
 (%cm.atom; |
  isoDateTime |
   systemConfigFormattedTime |
  userConfigFormattedTime)*
">
<!ENTITY % cm.logEntry "
 (logEntryId,
  execResult,
  execStart,
  execEnd)
">
<!ELEMENT cm-header (cm-sender, cm-receiver?, cm-authentication?)>
<!ELEMENT cm-sender EMPTY>
<!ATTLIST cm-sender
        sender-id CDATA #REQUIRED
        name CDATA #REQUIRED
>
<!ELEMENT cm-receiver EMPTY>
<!ATTLIST cm-receiver
        receiver-id CDATA #REQUIRED
        name CDATA #REQUIRED
>
<!ELEMENT cm-authentication EMPTY>
<!ATTLIST cm-authentication
        login CDATA #REQUIRED
        password CDATA #REQUIRED
        session CDATA #REQUIRED
>

<!ELEMENT cm-response (cm-code*)>
<!ATTLIST cm-response
        response-id CDATA #REQUIRED
        request-id CDATA #IMPLIED
        payload-id CDATA #IMPLIED
```

```
        success (true | false) #REQUIRED
>
<!ELEMENT cm-code ANY>
<!ATTLIST cm-code
        numeric CDATA #REQUIRED
        phrase CDATA #REQUIRED
>
<!ENTITY % cm.cm-request "
 (app-get) |
 (app-execute) |
 (attribute-create) |
 (attribute-types) |
 (attribute-getValidEditFieldKeys) |
 (attribute-where+, attribute-addEnumValues) |
 (attribute-where+, attribute-delete) |
 (attribute-where+, attribute-get) |
 (attribute-where+, attribute-description) |
 (attribute-where+, attribute-removeEnumValues) |
 (attribute-where+, attribute-set) |
 (attributeGroup-create) |
 (attributeGroup-where+, attributeGroup-addAttributes)|
 (attributeGroup-where+, attributeGroup-delete) |
 (attributeGroup-where+, attributeGroup-get) |
 (attributeGroup-where+, attributeGroup-description) |
 (attributeGroup-where+, attributeGroup-moveAttribute) |
 (attributeGroup-where+, attributeGroup-moveToIndex) |
 (attributeGroup-where+, attributeGroup-removeAttributes) |
 (attributeGroup-where+, attributeGroup-set) |
 (channel-create) |
 (channel-where+, channel-delete) |
 (channel-where+, channel-description) |
 (channel-where+, channel-get) |
 (channel-where+, channel-set) |
 (content-where+, content-addLinkTo) |
 (content-where+, content-delete) |
 (content-where+, content-get) |
 (content-where+, content-description) |
 (content-where+, content-load) |
 (content-where+, content-resolveRefs) |
 (content-where+, content-set) |
 (content-where+, content-debugExport) |
 (contentService-editedOverview) |
 (contentService-releasedOverview) |
 (customCommand-execute) |
 (group-create) |
 (group-where+, group-addUsers) |
 (group-where+, group-delete) |
 (group-where+, group-get) |
 (group-where+, group-description) |
 (group-where+, group-grantGlobalPermissions) |
 (group-where+, group-removeUsers) |
 (group-where+, group-revokeGlobalPermissions) |
 (group-where+, group-set) |
 (groupProxy-where+, groupProxy-get) |
 (incrExport-get) |
 (incrExport-getUpdateData) |
 (incrExport-removeUpdateRecords) |
 (incrExport-reset) |
 (job-create) |
 (job-listQueue) |
 (job-where+, job-cancel) |
 (job-where+, job-delete) |
 (job-where+, job-exec) |
 (job-where+, job-get) |
 (job-where+, job-getLogEntry) |
 (job-where+, job-getOutput) |
 (job-where+, job-description) |
 (job-where+, job-set) |
 (licenseManager-license) |
 (licenseManager-licenseExpirationDate) |
 (licenseManager-licenseType) |
 (licenseManager-loginCount) |
```

```
  (licenseManager-logins) |
  (licenseManager-logout) |
  (licenseManager-maxConcurrentUsers) |
  (link-create) |
  (link-where+, link-delete) |
  (link-where+, link-get) |
  (link-where+, link-description) |
  (link-where+, link-set) |
  (logEntry-where+, logEntry-delete) |
  (logEntry-where+, logEntry-get) |
  (obj-contentTypesForObjType) |
  (obj-generatePreview) |
  (obj-search+, obj-addComment) |
  (obj-search+, obj-commit) |
  (obj-search+, obj-copy) |
  (obj-search+, obj-create) |
  (obj-search+, obj-delete) |
  (obj-search+, obj-edit) |
  (obj-search+, obj-exportSubtree) |
  (obj-search+, obj-forward) |
  (obj-search+, obj-get) |
  (obj-search+, obj-description) |
  (obj-search+, obj-give) |
  (obj-search+, obj-mirror) |
  (obj-search+, obj-permissionGrantTo) |
  (obj-search+, obj-permissionRevokeFrom) |
  (obj-search+, obj-reject) |
  (obj-search+, obj-release) |
  (obj-search+, obj-removeActiveContents) |
  (obj-search+, obj-removeArchivedContents) |
  (obj-search+, obj-revert) |
  (obj-search+, obj-set) |
  (obj-search+, obj-sign) |
  (obj-search+, obj-take) |
  (obj-search+, obj-unrelease) |
  (obj-types) |
  (obj-where+, obj-addComment) |
  (obj-where+, obj-commit) |
  (obj-where+, obj-copy) |
  (obj-where+, obj-create) |
  (obj-where+, obj-delete) |
  (obj-where+, obj-edit) |
  (obj-where+, obj-exportSubtree) |
  (obj-where+, obj-forward) |
  (obj-where+, obj-get) |
  (obj-where+, obj-description) |
  (obj-where+, obj-give) |
  (obj-where+, obj-mirror) |
  (obj-where+, obj-permissionGrantTo) |
  (obj-where+, obj-permissionRevokeFrom) |
  (obj-where+, obj-reject) |
  (obj-where+, obj-release) |
  (obj-where+, obj-removeActiveContents) |
  (obj-where+, obj-removeArchivedContents) |
  (obj-where+, obj-revert) |
  (obj-where+, obj-set) |
  (obj-where+, obj-sign) |
  (obj-where+, obj-take) |
  (obj-where+, obj-unrelease) |
  (objClass-create) |
  (objClass-goodAvailableBlobEditorsForObjType) |
  (objClass-validAttributes) |
  (objClass-where+, objClass-delete) |
  (objClass-where+, objClass-get) |
  (objClass-where+, objClass-description) |
  (objClass-where+, objClass-set) |
  (systemConfig-formatDate) |
  (systemConfig-formatDateTime) |
  (systemConfig-getAttributes) |
  (systemConfig-getCounts) |
  (systemConfig-getElements) |
  (systemConfig-getKeys) |
```

```
  (systemConfig-getTexts) |
  (systemConfig-parseInputDate) |
  (systemConfig-installedLanguages) |
  (systemConfig-validInputCharsets) |
  (systemConfig-validTimeZones) |
  (task-where+, task-get) |
  (task-where+, task-description) |
  (user-create) |
  (user-where+, user-addToGroups) |
  (user-where+, user-delete) |
  (user-where+, user-get) |
  (user-where+, user-description) |
  (user-where+, user-grantGlobalPermissions) |
  (user-where+, user-removeFromGroups) |
  (user-where+, user-revokeGlobalPermissions) |
  (user-where+, user-set) |
  (user-writeUserFile) |
  (userAttribute-create) |
  (userAttribute-types) |
  (userAttribute-where+, userAttribute-addEnumValues) |
  (userAttribute-where+, userAttribute-delete) |
  (userAttribute-where+, userAttribute-get) |
  (userAttribute-where+, userAttribute-description) |
  (userAttribute-where+, userAttribute-removeEnumValues) |
  (userAttribute-where+, userAttribute-set) |
  (userConfig-getAll) |
  (userConfig-formatDate) |
  (userConfig-formatDateTime) |
  (userConfig-getAttributes) |
  (userConfig-getCounts) |
  (userConfig-getElements) |
  (userConfig-getKeys) |
  (userConfig-getTexts) |
  (userConfig-parseInputDate) |
  (userConfig-removeAttributes) |
  (userConfig-removeKeys) |
  (userConfig-setAttributes) |
  (userConfig-setElements) |
  (userConfig-setTexts) |
  (userProxy-where+, userProxy-get) |
  (workflow-create) |
  (workflow-where+, workflow-delete) |
  (workflow-where+, workflow-get) |
  (workflow-where+, workflow-description) |
  (workflow-where+, workflow-set)
">

<!ELEMENT cm-request (%cm.cm-request;)>
<!ATTLIST cm-request
        request-id CDATA #REQUIRED
        preclusive (true | false) "false"
>

<!ENTITY % cm.app-get "
  (appName |
   baseDir |
   binDir |
   configDir |
   dataDir |
   debugChannels |
   debugLevel |
   getKeys |
   instanceDir |
   libDir |
   logDir |
   now |
   rootConfigPath |
   scriptDir |
   shareDir |
   timeZone |
   tmpDir |
   today |
```

```
  version)*
">
<!ENTITY % cm.app-execute "
 (command,
  arguments)
">
<!ENTITY % cm.attribute-addEnumValues "
 (listitem+)
">
<!ENTITY % cm.attribute-create "
 (callback |
  helpText |
  isSearchableInCM |
  isSearchableInTE |
  maxSize |
  minSize |
  name |
  title |
  type |
  values |
  wantedTags |
  editField)*
">
<!ENTITY % cm.attribute-delete "EMPTY">
<!ENTITY % cm.attribute-get "
 callback |
  displayTitle |
  displayValueCallback |
  editField |
  editFieldSpec |
  getKeys |
  helpText |
  isSearchableInCM |
  isSearchableInTE |
  localizedTitle |
  localizedHelpText |
  maxSize |
  minSize |
  name |
  setKeys |
  title |
  type |
  validEditFieldKeys |
  validEditFieldTypes |
  values |
  wantedTags
">
<!ENTITY % cm.attribute-getValidEditFieldKeys "(listitem)*">
<!ENTITY % cm.attribute-removeEnumValues "
 (listitem+)
">
<!ENTITY % cm.attribute-set "
 (callback |
  displayValueCallback |
  helpText |
  isSearchableInCM |
  isSearchableInTE |
  maxSize |
  minSize |
  title |
  type |
  values |
  wantedTags |
  editField)*
">
<!ENTITY % cm.attribute-types "(listitem)*">
<!ENTITY % cm.attribute-where "
 (name |
 (nameLike?,
  type?))?
">
<!ENTITY % cm.attributeGroup-addAttributes "(listitem)*">
```

```
<!ENTITY % cm.attributeGroup-create "
 (objClass,
  name,
  title?,
  index?)
">
<!ENTITY % cm.attributeGroup-delete "EMPTY">
<!ENTITY % cm.attributeGroup-get "
 attributes |
  displayTitle |
  localizedTitle |
  getKeys |
  index |
  isDefaultGroup |
  isEmpty |
  name |
  objClass |
  setKeys |
  title
">
<!ENTITY % cm.attributeGroup-moveAttribute "
 (attribute, index)
">
<!ENTITY % cm.attributeGroup-moveToIndex "
 (index)
">
<!ENTITY % cm.attributeGroup-removeAttributes "
 (listitem+)
">
<!ENTITY % cm.attributeGroup-set "
 (title)*
">
<!ENTITY % cm.attributeGroup-where "
 (objClass)
">
<!ENTITY % cm.channel-create "
 (name,
  title*)
">
<!ENTITY % cm.channel-delete "EMPTY">
<!ENTITY % cm.channel-get "
 (name |
  title)*
">
<!ENTITY % cm.channel-set "
 (name |
  title)*
">
<!ENTITY % cm.channel-where "
 (name |
  namePrefix)*
">
<!ENTITY % cm.content-addLinkTo "
 (attribute,
  destinationUrl)*
">
<!ENTITY % cm.content-delete "EMPTY">
<!ENTITY % cm.content-get "
  anchors |
  blob |
  blobLength |
  body |
  bodyTemplateName |
  codeForSourceView |
  contentType |
  displayTitle |
  editor |
  exportBlob |
  exportFiles |
  externalAttrNames |
  externalAttributes |
  frameNames |
```

```
   freeLinks |
   getKeys |
   id |
   isActive |
   isCommitted |
   isComplete |
   isEdited |
   isReleased |
   lastChanged |
   linkListAttributes |
   mimeType |
   nextEditGroup |
   nextSignGroup |
   objectId |
   reasonsForIncompleteState |
   relatedLinks |
   setKeys |
   signatureAttrNames |
   signatureAttributes |
   sortKey1 |
   sortKey2 |
   sortKey3 |
   sortKeyLength1 |
   sortKeyLength2 |
   sortKeyLength3 |
   sortOrder |
   sortType1 |
   sortType2 |
   sortType3 |
   subLinks |
   textLinks |
   thumbnail |
   title |
   validFrom |
   validSortKeys |
   validSortOrders |
   validSortTypes |
   validUntil |
   workflowComment |
   xmlBlob
">
<!ENTITY % cm.content-load "
 (blob |
  contentType |
  filter)*
">
<!ENTITY % cm.content-resolveRefs "EMPTY">
<!ENTITY % cm.content-set "
 (contentType |
  sortKey1 |
  sortKey2 |
  sortKey3 |
  sortKeyLength1 |
  sortKeyLength2 |
  sortKeyLength3 |
  sortOrder |
  sortType1 |
  sortType2 |
  sortType3 |
  title |
  validFrom |
  validUntil)*
">
<!ENTITY % cm.content-where "
 (id |
 (objectId , state))
">
<!ENTITY % cm.group-addUsers "
 (listitem+)
">
<!ENTITY % cm.group-create "
 (globalPermissions |
```

```
  name |
  owner |
  realName)*
">
<!ENTITY % cm.group-delete "EMPTY">
<!ENTITY % cm.group-get "
 displayTitle |
  getKeys |
  globalPermissions |
  hasGlobalPermission |
  users |
  name |
  owner |
  realName |
  setKeys
">
<!ENTITY % cm.group-grantGlobalPermissions "
 (listitem+)
">
<!ENTITY % cm.group-removeUsers "
 (listitem+)
">
<!ENTITY % cm.group-revokeGlobalPermissions "
 (listitem+)
">
<!ENTITY % cm.group-set "
 (globalPermissions |
  owner |
  realName |
  users)*
">
<!ENTITY % cm.group-where "
 (name |
  groupText)?
">
<!ENTITY % cm.incrExport-get "
 (getKeys |
  mode |
  updateRecords |
  updateRecordCount)*
">
<!ENTITY % cm.incrExport-getUpdateData "
 (updateData?)
">
<!ENTITY % cm.incrExport-removeUpdateRecords "
 (updateRecordId+ |
  all)
">
<!ENTITY % cm.incrExport-reset "EMPTY">
<!ENTITY % cm.job-cancel "EMPTY">
<!ENTITY % cm.job-create "
 (name,
  comment?,
  execLogin?,
  execPerm?,
  schedule?,
  script?,
  title?)
">
<!ENTITY % cm.job-delete "EMPTY">
<!ENTITY % cm.job-exec "EMPTY">
<!ENTITY % cm.job-get "
 (category |
  comment |
  displayTitle |
  execLogin |
  execPerm |
  getKeys |
  id |
  isActive |
  lastExecEnd |
  lastExecResult |
```

```
  lastExecStart |
  lastLogEntry |
  lastOutput |
  log |
  logEntries |
  name |
  nextExecStart |
  queuePos |
  schedule |
  script |
  setKeys |
  title)*
">
<!ENTITY % cm.job-getLogEntry "%cm.logEntry;">
<!ATTLIST job-getLogEntry
  id CDATA #REQUIRED
>
<!ENTITY % cm.job-getOutput "(%cm.atom;)">
<!ENTITY % cm.job-listQueue "(%cm.listitem;)*">
<!ATTLIST job-listQueue
  category CDATA #IMPLIED
>

<!ENTITY % cm.job-set "
 (comment |
  execLogin |
  execPerm |
  isActive |
  schedule |
  script |
  title)*
">
<!ENTITY % cm.job-where "
 (category |
  comment |
  execLogin |
  id |
  isActive |
  isQueued |
  name |
  title)*
">
<!ENTITY % cm.licenseManager-license "(%cm.atom;)">
<!ENTITY % cm.licenseManager-licenseExpirationDate "(%cm.atom;)">
<!ENTITY % cm.licenseManager-licenseType "(%cm.atom;)">
<!ENTITY % cm.licenseManager-loginCount "(%cm.atom;)">
<!ENTITY % cm.licenseManager-logins "(listitem*)">
<!ENTITY % cm.licenseManager-logout "EMPTY">
<!ENTITY % cm.licenseManager-maxConcurrentUsers "(%cm.atom;)">
<!ENTITY % cm.link-create "
 (attributeName |
  destinationUrl |
  sourceContent)*
">
<!ENTITY % cm.link-delete "EMPTY">
<!ENTITY % cm.link-get "
 (attributeName |
  canHaveAnchor |
  canHaveTarget |
  destination |
  destinationUrl |
  displayTitle |
  expectedPath |
  getKeys |
  id |
  isComplete |
  isExternalLink |
  isFreeLink |
  isIncludeLink |
  isInlineReferenceLink |
  isLinkFromCommittedContent |
  isLinkFromEditedContent |
```

```
   isLinkFromReleasedContent |
   isRelatedLink |
   isWritable |
   setKeys |
   source |
   sourceContent |
   sourceTagAttribute |
   sourceTagName |
   target |
   title)*
">
<!ENTITY % cm.link-set "
 (destinationUrl | attributeName |
   target |
   title)*
">
<!ENTITY % cm.link-where "
 (id?)
">
<!ENTITY % cm.logEntry-delete "(deleteLogEntriesCount?)">
<!ENTITY % cm.logEntry-get "
 (logTime |
   logText |
   logType |
   objectId |
   receiver |
   userLogin)*
">
<!ENTITY % cm.logEntry-where "
 (fromDate?,
   logText?,
   logTypes?,
   objectId?,
   receiver?,
   untilDate?,
   userLogin?)
">
<!ENTITY % cm.obj-commit "
 (comment)
">
<!ENTITY % cm.obj-contentTypesForObjType "(listitem*)">
<!ENTITY % cm.obj-copy "
 ((parent | name)+,
   recursive?)
">
<!ENTITY % cm.obj-create "
 (blob |
   contentType |
   file |
   filter |
   name |
   objClass |
   suppressExport |
   xmlBlob)*
">
<!ENTITY % cm.obj-deactivate "EMPTY">
<!ENTITY % cm.obj-delete "EMPTY">
<!ENTITY % cm.obj-edit "
 (comment,
   contentId?)
">
<!ENTITY % cm.obj-exportSubtree "
 (absoluteFsPrefix |
 absoluteUrlPrefix |
 exportCharset |
 filePrefix |
 templateName |
 purgeCollections)*
">
<!ENTITY % cm.obj-forward "
 (comment)
">
```

```
<!ENTITY % cm.obj-generatePreview "EMPTY">
<!ENTITY % cm.obj-get "
  archivedContents |
  children |
  committedContentId |
  contentIds |
  contents |
  editedContentId |
  editedContent |
  exportMimeType |
  getKeys |
  hasChildren |
  hasMirrors |
  hasSuperLinks |
  hierarchy |
  id |
  isCommitted |
  isEdited |
  isExportable |
  isMirror |
  isReleased |
  isRoot |
  isGoodDestination |
  isGoodParent |
  name |
  next |
  objClass |
  objType |
  objectsToRoot |
  parent |
  path |
  permission |
  permissionGrantedTo |
  prefixPath |
  previous |
  releasedContentId |
  releasedContent |
  releasedVersions |
  rootPermissionFor |
  setKeys |
  sortValue |
  superLinks |
  superObjects |
  suppressExport |
  toclist |
  validControlActionKeys |
  validCreateObjClasses |
  validObjClasses |
  validPermissions |
  version |
  visibleExportTemplates |
  visibleName |
  visiblePath |
  workflowName
">
<!ENTITY % cm.obj-give "(comment, (groupName | userLogin))">
<!ENTITY % cm.obj-mirror "
 (parent |
  name)*
">
<!ENTITY % cm.obj-permissionGrantTo "
 (group+)
">
<!ENTITY % cm.obj-permissionRevokeFrom "
 (group+)
">
<!ENTITY % cm.obj-reject "
 (comment)
">
<!ENTITY % cm.obj-release "
 (comment)
">
```

```
<!ENTITY % cm.obj-removeActiveContents "EMPTY">
<!ENTITY % cm.obj-removeArchivedContents "EMPTY">
<!ENTITY % cm.obj-revert "
 (comment)
">
<!ENTITY % cm.obj-search "
 (query |
  minRelevance |
  maxDocs |
  offsetStart |
  offsetLength |
  parser |
  collections)*
">
<!ENTITY % cm.obj-set "
 (suppressExport |
  name |
  objClass |
  parent |
  permission |
  workflowName)*
">
<!ENTITY % cm.obj-sign "
 (comment)
">
<!ENTITY % cm.obj-take "
 (comment)
">
<!ENTITY % cm.obj-types "(listitem)*">
<!ENTITY % cm.obj-unrelease "
 (comment)
">
<!ENTITY % cm.obj-where "
 (id |
  ids |
  path |
  parent |
  condition)*
">
<!ENTITY % cm.objClass-create "
 (attributes |
  bodyTemplateName |
  completionCheck |
  createPermission |
  externalEditExtension |
  externalEditMimeType |
  isEnabled |
  mandatoryAttributes |
  name |
  objType |
  presetAttributes |
  presetFromParentAttributes |
  recordSetCallback |
  title |
  validContentTypes |
  validSortKeys |
  validSortOrders |
  validSortTypes |
  validSubObjClassCheck |
  validSubObjClasses |
  workflowModification)*
">
<!ENTITY % cm.objClass-delete "EMPTY">
<!ENTITY % cm.objClass-get "
 attributeGroups |
  attributes |
  availableBlobEditors |
  bodyTemplateName |
  completionCheck |
  createPermission |
  customBlobEditorUrl |
  defaultAttributeGroup |
```

```
   displayTitle |
   emptyAttributeGroups |
   externalEditExtension |
   externalEditMimeType |
   getKeys |
   goodAttributeGroupAttributes |
   goodAttributes |
   goodMandatoryAttributes |
   goodPresetAttributes |
   goodPresetFromParentAttributes |
   isEnabled |
   localizedTitle |
   mandatoryAttributes |
   name |
   objType |
   presetAttributes |
   presetFromParentAttributes |
   recordSetCallback |
   setKeys |
   title |
   validContentTypes |
   validSubObjClassCheck |
   validSubObjClasses |
   workflowModification |
   xmldtd
">
<!ENTITY % cm.objClass-goodAvailableBlobEditorsForObjType "(listitem*)">
<!ENTITY % cm.objClass-set "
 (attributes |
  availableBlobEditors |
  bodyTemplateName |
  completionCheck |
  createPermission |
  customBlobEditorUrl |
  externalEditExtension |
  externalEditMimeType |
  isEnabled |
  mandatoryAttributes |
  presetAttributes |
  presetFromParentAttributes |
  recordSetCallback |
  title |
  validContentTypes |
  validSubObjClassCheck |
  validSubObjClasses |
  workflowModification)*
">
<!ENTITY % cm.objClass-validAttributes "(listitem)*">
<!ENTITY % cm.objClass-where "
 (name |
 (nameLike?,
  isEnabled?,
  objType?))?
">
<!ENTITY % cm.systemConfig-getAttributes "(key, attributeNames?)">
<!ENTITY % cm.systemConfig-getCounts "(listitem+)">
<!ENTITY % cm.systemConfig-getElements "(listitem+)">
<!ENTITY % cm.systemConfig-getKeys "(listitem+)">
<!ENTITY % cm.systemConfig-getTexts "(listitem+)">
<!ENTITY % cm.systemConfig-installedLanguages "(listitem+)">
<!ENTITY % cm.systemConfig-validTimeZones "(listitem+)">
<!ENTITY % cm.systemConfig-validInputCharsets "(listitem+)">
<!ENTITY % cm.task-delete "EMPTY">
<!ENTITY % cm.task-get "
 (comment |
  displayTitle |
  getKeys |
  groupName |
  objectId |
  taskType |
  timeStamp |
  title |
```

```
  userLogin)*
">
<!ENTITY % cm.task-where "
 (id |
 (objectId,
  userLogin?,
  groupNames?,
  taskText?,
  taskType?))?
">
<!ENTITY % cm.user-addToGroups "
 (listitem+)
">
<!ENTITY % cm.user-create "
 (defaultGroup |
  encryptedPassword |
  email |
  globalPermissions |
  login |
  groups |
  owner |
  password |
  realName |
  userLocked)*
">
<!ENTITY % cm.user-delete "EMPTY">
<!ENTITY % cm.user-get "
 defaultGroup |
  displayTitle |
  encryptedPassword |
  email |
  externalUserAttrNames |
  getKeys |
  globalPermissions |
  groups |
  hasGlobalPermission |
  isSuperUser |
  isOwnerOf |
  hasPassword |
  login |
  owner |
  realName |
  setKeys |
  userLocked |
  externalAttrNames
">
<!ENTITY % cm.user-grantGlobalPermissions "
 (listitem+)
">
<!ENTITY % cm.user-removeFromGroups "
 (listitem+)
">
<!ENTITY % cm.user-revokeGlobalPermissions "
 (listitem+)
">
<!ENTITY % cm.user-set "
 (defaultGroup |
  encryptedPassword |
  email |
  globalPermissions |
  groups |
  owner |
  password |
  realName |
  userLocked)*
">
<!ENTITY % cm.user-where "
 (login |
  userText)
">
<!ENTITY % cm.user-writeUserFile "EMPTY">
<!ENTITY % cm.userAttribute-addEnumValues "
```

```
  (listitem+)
">
<!ENTITY % cm.userAttribute-create "
  (name |
   type)*
">
<!ENTITY % cm.userAttribute-delete "EMPTY">
<!ENTITY % cm.userAttribute-get "
  (displayTitle |
   getKeys |
   name |
   setKeys        |
   type |
   values)*
">
<!ENTITY % cm.userAttribute-removeEnumValues "
  (listitem+)
">
<!ENTITY % cm.userAttribute-set "
  (type |
   values)*
">
<!ENTITY % cm.userAttribute-types "(listitem)*">
<!ENTITY % cm.userAttribute-where "
  (name |
  (nameLike?,
   type?))?
">
<!ENTITY % cm.userConfig-getAll "(%cm.atom;)">
<!ENTITY % cm.userConfig-getAttributes "(key, attributeNames?)">
<!ENTITY % cm.userConfig-getCounts "(listitem+)">
<!ENTITY % cm.userConfig-getElements "(listitem+)">
<!ENTITY % cm.userConfig-getKeys "(listitem+)">
<!ENTITY % cm.userConfig-getTexts "(listitem+)">
<!ENTITY % cm.userConfig-removeAttributes "(key, attributeNames?)">
<!ENTITY % cm.userConfig-removeKeys "(listitem*)">
<!ENTITY % cm.userConfig-setAttributes "(key, attributes?)">
<!ENTITY % cm.userConfig-setElements "(dictitem+)">
<!ENTITY % cm.userConfig-setTexts "(dictitem+)">
<!ENTITY % cm.workflow-create "
  (name,
   editGroups?,
   signatureDefs?,
   allowsMultipleSignatures?,
   isEnabled?,
   title?)
">
<!ENTITY % cm.workflow-delete "EMPTY">
<!ENTITY % cm.workflow-get "
 name |
   displayTitle |
   editGroups |
   signatureDefs |
   allowsMultipleSignatures |
   getKeys |
   isEnabled |
   setKeys |
   title
">
<!ENTITY % cm.workflow-set "
  (editGroups |
   signatureDefs |
   allowsMultipleSignatures |
   getKeys |
   isEnabled |
   setKeys |
   title)*
">
<!ENTITY % cm.workflow-where "
  (name |
   nameLike)*
">
```

```
<!ELEMENT app-get %cm.app-get;>
<!ELEMENT app-execute %cm.app-execute;>
<!ELEMENT attribute-addEnumValues %cm.attribute-addEnumValues;>
<!ELEMENT attribute-create %cm.attribute-create;>
<!ELEMENT attribute-delete %cm.attribute-delete;>
<!ELEMENT attribute-get (%cm.attribute-get;)*>
<!ELEMENT attribute-description (%cm.atom;)>
<!ELEMENT attribute-getValidEditFieldKeys %cm.attribute-getValidEditFieldKeys;>
<!ATTLIST attribute-getValidEditFieldKeys
        type CDATA #REQUIRED
>
<!ELEMENT attribute-removeEnumValues %cm.attribute-removeEnumValues;>
<!ELEMENT attribute-set %cm.attribute-set;>
<!ELEMENT attribute-types %cm.attribute-types;>
<!ELEMENT attribute-where %cm.attribute-where;>
<!ATTLIST attribute-where
    maxResults CDATA #IMPLIED
>
<!ELEMENT attributeGroup-addAttributes %cm.attributeGroup-addAttributes;>
<!ELEMENT attributeGroup-create %cm.attributeGroup-create;>
<!ELEMENT attributeGroup-delete %cm.attributeGroup-delete;>
<!ELEMENT attributeGroup-get (%cm.attributeGroup-get;)*>
<!ELEMENT attributeGroup-description (%cm.atom;)>
<!ELEMENT attributeGroup-moveAttribute %cm.attributeGroup-moveAttribute;>
<!ELEMENT attributeGroup-moveToIndex %cm.attributeGroup-moveToIndex;>
<!ELEMENT attributeGroup-removeAttributes %cm.attributeGroup-removeAttributes;>
<!ELEMENT attributeGroup-set %cm.attributeGroup-set;>
<!ELEMENT attributeGroup-where %cm.attributeGroup-where;>
<!ELEMENT channel-create %cm.channel-create;>
<!ELEMENT channel-delete %cm.channel-delete;>
<!ELEMENT channel-description (%cm.atom;)>
<!ELEMENT channel-get %cm.channel-get;>
<!ELEMENT channel-set %cm.channel-set;>
<!ELEMENT channel-where %cm.channel-where;>
<!ATTLIST channel-where
    maxResults CDATA #IMPLIED
>
<!ELEMENT content-addLinkTo %cm.content-addLinkTo;>
<!ELEMENT content-delete %cm.content-delete;>
<!ELEMENT content-get (%cm.content-get;)+>
<!ELEMENT content-description (%cm.atom;)>
<!ELEMENT content-load %cm.content-load;>
<!ELEMENT content-resolveRefs %cm.content-resolveRefs;>
<!ELEMENT content-set %cm.content-set;>
<!ELEMENT content-where %cm.content-where;>
<!ELEMENT contentService-editedOverview (%cm.atom;)>
<!ATTLIST contentService-editedOverview
    encoding (plain | base64 | stream) #IMPLIED
>
<!ELEMENT contentService-releasedOverview (%cm.atom;)>
<!ATTLIST contentService-releasedOverview
    encoding (plain | base64 | stream) #IMPLIED
>
<!ELEMENT customCommand-execute (command, arguments?)>
<!ELEMENT content-debugExport ANY>
<!ATTLIST content-debugExport
  quoteHtml (true | false) "false"
  htmlPrefix CDATA #IMPLIED
  htmlSuffix CDATA #IMPLIED
  infoPrefix CDATA #IMPLIED
  infoSuffix CDATA #IMPLIED
  errorPrefix CDATA #IMPLIED
  errorSuffix CDATA #IMPLIED
  detailed (true | false) "false"
  preferEditedTemplates (true | false) #IMPLIED
  allowEditedContents (true | false) #IMPLIED
  templateName CDATA #IMPLIED
>
<!ELEMENT group-addUsers %cm.group-addUsers;>
<!ELEMENT group-create %cm.group-create;>
<!ELEMENT group-delete %cm.group-delete;>
```

```
<!ELEMENT group-get (%cm.group-get;)>
<!ELEMENT group-description (%cm.atom;)>
<!ELEMENT group-grantGlobalPermissions %cm.group-grantGlobalPermissions;>
<!ELEMENT group-removeUsers %cm.group-removeUsers;>
<!ELEMENT group-revokeGlobalPermissions %cm.group-revokeGlobalPermissions;>
<!ELEMENT group-set %cm.group-set;>
<!ELEMENT group-where %cm.group-where;>
<!ATTLIST group-where
    maxResults CDATA #IMPLIED
>
<!ELEMENT groupProxy-get (%cm.group-get;)>
<!ELEMENT groupProxy-where %cm.group-where;>
<!ELEMENT incrExport-get %cm.incrExport-get;>
<!ELEMENT incrExport-getUpdateData %cm.incrExport-getUpdateData;>
<!ATTLIST incrExport-getUpdateData
        updateRecordId CDATA #REQUIRED
>
<!ELEMENT incrExport-removeUpdateRecords %cm.incrExport-removeUpdateRecords;>
<!ELEMENT incrExport-reset %cm.incrExport-reset;>
<!ELEMENT job-cancel %cm.job-cancel;>
<!ELEMENT job-create %cm.job-create;>
<!ELEMENT job-delete %cm.job-delete;>
<!ELEMENT job-exec %cm.job-exec;>
<!ELEMENT job-get %cm.job-get;>
<!ELEMENT job-getLogEntry %cm.job-getLogEntry;>
<!ELEMENT job-getOutput %cm.job-getOutput;>
<!ELEMENT job-description (%cm.atom;)>
<!ELEMENT job-listQueue %cm.job-listQueue;>
<!ELEMENT job-set %cm.job-set;>
<!ELEMENT licenseManager-license %cm.licenseManager-license;>
<!ELEMENT licenseManager-licenseExpirationDate %cm.licenseManager-licenseExpirationDate;>
<!ELEMENT licenseManager-licenseType %cm.licenseManager-licenseType;>
<!ELEMENT licenseManager-loginCount %cm.licenseManager-loginCount;>
<!ELEMENT licenseManager-logins %cm.licenseManager-logins;>
<!ELEMENT licenseManager-logout %cm.licenseManager-logout;>
<!ATTLIST licenseManager-logout
        login CDATA #IMPLIED
>
<!ELEMENT licenseManager-maxConcurrentUsers %cm.licenseManager-maxConcurrentUsers;>
<!ELEMENT job-where %cm.job-where;>
<!ATTLIST job-where
    maxResults CDATA #IMPLIED
>
<!ELEMENT link-create %cm.link-create;>
<!ELEMENT link-delete %cm.link-delete;>
<!ELEMENT link-get %cm.link-get;>
<!ELEMENT link-description (%cm.atom;)>
<!ELEMENT link-set %cm.link-set;>
<!ELEMENT link-where %cm.link-where;>
<!ELEMENT logEntry-delete %cm.logEntry-delete;>
<!ELEMENT logEntry-get %cm.logEntry-get;>
<!ELEMENT logEntry-where %cm.logEntry-where;>
<!ELEMENT obj-addComment (%cm.atom;)>
<!ELEMENT obj-commit %cm.obj-commit;>
<!ELEMENT obj-contentTypesForObjType %cm.obj-contentTypesForObjType;>
<!ATTLIST obj-contentTypesForObjType
        objType CDATA #IMPLIED
>
<!ELEMENT obj-copy %cm.obj-copy;>
<!ELEMENT obj-create %cm.obj-create;>
<!ELEMENT obj-delete %cm.obj-delete;>
<!ELEMENT obj-edit %cm.obj-edit;>
<!ELEMENT obj-exportSubtree %cm.obj-exportSubtree;>
<!ELEMENT obj-forward %cm.obj-forward;>
<!ELEMENT obj-generatePreview %cm.obj-generatePreview;>
<!ATTLIST obj-generatePreview
        fsPrefix CDATA #REQUIRED
        masterTemplate CDATA #REQUIRED
        mode (0|1|2|3|4|5|6|7) #REQUIRED
        path CDATA #REQUIRED
        urlPrefix CDATA #REQUIRED
    editor CDATA #REQUIRED>
```

```
<!ELEMENT obj-get (%cm.obj-get;)*>
<!ELEMENT obj-description (%cm.atom;)>
<!ELEMENT obj-give %cm.obj-give;>
<!ELEMENT obj-mirror %cm.obj-mirror;>
<!ELEMENT obj-permissionGrantTo %cm.obj-permissionGrantTo;>
<!ATTLIST obj-permissionGrantTo
        permission (permissionCreateChildren | permissionWrite | permissionRead |
 permissionRoot) #REQUIRED
>
<!ELEMENT obj-permissionRevokeFrom %cm.obj-permissionRevokeFrom;>
<!ATTLIST obj-permissionRevokeFrom
        permission (permissionCreateChildren | permissionWrite | permissionRead |
 permissionRoot) #REQUIRED
>
<!ELEMENT obj-reject %cm.obj-reject;>
<!ELEMENT obj-release %cm.obj-release;>
<!ELEMENT obj-removeActiveContents %cm.obj-removeActiveContents;>
<!ELEMENT obj-removeArchivedContents %cm.obj-removeArchivedContents;>
<!ELEMENT obj-revert %cm.obj-revert;>
<!ELEMENT obj-search %cm.obj-search;>
<!ELEMENT obj-set %cm.obj-set;>
<!ELEMENT obj-sign %cm.obj-sign;>
<!ELEMENT obj-take %cm.obj-take;>
<!ELEMENT obj-types %cm.obj-types;>
<!ELEMENT obj-unrelease %cm.obj-unrelease;>
<!ELEMENT obj-where %cm.obj-where;>
<!ATTLIST obj-where
    maxResults CDATA #IMPLIED
>


<!ELEMENT objClass-create %cm.objClass-create;>
<!ELEMENT objClass-delete %cm.objClass-delete;>
<!ELEMENT objClass-get (%cm.objClass-get;)*>
<!ELEMENT objClass-description (%cm.atom;)>
<!ELEMENT objClass-goodAvailableBlobEditorsForObjType %cm.objClass-
goodAvailableBlobEditorsForObjType;>
<!ATTLIST objClass-goodAvailableBlobEditorsForObjType
        objType CDATA #REQUIRED
>
<!ELEMENT objClass-set %cm.objClass-set;>
<!ELEMENT objClass-validAttributes %cm.objClass-validAttributes;>
<!ELEMENT objClass-where %cm.objClass-where;>
<!ATTLIST objClass-where
    maxResults CDATA #IMPLIED
>
<!ELEMENT systemConfig-formatDate (%cm.atom;)>
<!ELEMENT systemConfig-formatDateTime (%cm.atom;)>
<!ELEMENT systemConfig-getAttributes %cm.systemConfig-getAttributes;>
<!ELEMENT systemConfig-getCounts %cm.systemConfig-getCounts;>
<!ELEMENT systemConfig-getElements %cm.systemConfig-getElements;>
<!ELEMENT systemConfig-getKeys %cm.systemConfig-getKeys;>
<!ELEMENT systemConfig-getTexts %cm.systemConfig-getTexts;>
<!ELEMENT systemConfig-parseInputDate %cm.date;>
<!ELEMENT systemConfig-installedLanguages %cm.systemConfig-installedLanguages;>
<!ELEMENT systemConfig-validTimeZones %cm.systemConfig-validTimeZones;>
<!ELEMENT systemConfig-validInputCharsets %cm.systemConfig-validInputCharsets;>
<!ELEMENT task-delete %cm.task-delete;>
<!ELEMENT task-get %cm.task-get;>
<!ELEMENT task-description (%cm.atom;)>
<!ELEMENT task-where %cm.task-where;>
<!ATTLIST task-where
    maxResults CDATA #IMPLIED
>
<!ELEMENT user-addToGroups %cm.user-addToGroups;>
<!ELEMENT user-create %cm.user-create;>
<!ELEMENT user-delete %cm.user-delete;>
<!ELEMENT user-get (%cm.user-get;)*>
<!ELEMENT user-description (%cm.atom;)>
<!ELEMENT user-grantGlobalPermissions %cm.user-grantGlobalPermissions;>
<!ELEMENT user-removeFromGroups %cm.user-removeFromGroups;>
<!ELEMENT user-revokeGlobalPermissions %cm.user-revokeGlobalPermissions;>
```

```
<!ELEMENT user-set %cm.user-set;>
<!ELEMENT user-where %cm.user-where;>
<!ATTLIST user-where
    maxResults CDATA #IMPLIED
>
<!ELEMENT user-writeUserFile %cm.user-writeUserFile;>
<!ELEMENT userAttribute-addEnumValues %cm.userAttribute-addEnumValues;>
<!ELEMENT userAttribute-create %cm.userAttribute-create;>
<!ELEMENT userAttribute-delete %cm.userAttribute-delete;>
<!ELEMENT userAttribute-get %cm.userAttribute-get;>
<!ELEMENT userAttribute-description (%cm.atom;)>
<!ELEMENT userAttribute-removeEnumValues %cm.userAttribute-removeEnumValues;>
<!ELEMENT userAttribute-set %cm.userAttribute-set;>
<!ELEMENT userAttribute-where %cm.userAttribute-where;>
<!ATTLIST userAttribute-where
    maxResults CDATA #IMPLIED
>
<!ELEMENT userAttribute-types %cm.userAttribute-types;>
<!ELEMENT userConfig-getAll %cm.userConfig-getAll;>
<!ATTLIST userConfig-getAll
        login CDATA #IMPLIED
>
<!ELEMENT userConfig-formatDate (%cm.atom;)>
<!ATTLIST userConfig-formatDate
        login CDATA #IMPLIED
>
<!ELEMENT userConfig-formatDateTime (%cm.atom;)>
<!ATTLIST userConfig-formatDateTime
        login CDATA #IMPLIED
>
<!ELEMENT userConfig-getAttributes %cm.userConfig-getAttributes;>
<!ATTLIST userConfig-getAttributes
        login CDATA #IMPLIED
>
<!ELEMENT userConfig-getCounts %cm.userConfig-getCounts;>
<!ATTLIST userConfig-getCounts
        login CDATA #IMPLIED
>
<!ELEMENT userConfig-getElements %cm.userConfig-getElements;>
<!ATTLIST userConfig-getElements
        login CDATA #IMPLIED
>
<!ELEMENT userConfig-getKeys %cm.userConfig-getKeys;>
<!ATTLIST userConfig-getKeys
        login CDATA #IMPLIED
>
<!ELEMENT userConfig-getTexts %cm.userConfig-getTexts;>
<!ATTLIST userConfig-getTexts
        login CDATA #IMPLIED
>
<!ELEMENT userConfig-initFromFile (%cm.atom;)>
<!ATTLIST userConfig-initFromFile
        login CDATA #IMPLIED
>
<!ELEMENT userConfig-parseInputDate %cm.date;>
<!ATTLIST userConfig-parseInputDate
        login CDATA #IMPLIED
>
<!ELEMENT userConfig-removeAttributes %cm.userConfig-removeAttributes;>
<!ATTLIST userConfig-removeAttributes
        login CDATA #IMPLIED
>
<!ELEMENT userConfig-removeKeys %cm.userConfig-removeKeys;>
<!ATTLIST userConfig-removeKeys
        login CDATA #IMPLIED
>
<!ELEMENT userConfig-setAttributes %cm.userConfig-setAttributes;>
<!ATTLIST userConfig-setAttributes
        login CDATA #IMPLIED
>
<!ELEMENT userConfig-setElements %cm.userConfig-setElements;>
<!ATTLIST userConfig-setElements
```

```
        login CDATA #IMPLIED
>
<!ELEMENT userConfig-setTexts %cm.userConfig-setTexts;>
<!ATTLIST userConfig-setTexts
        login CDATA #IMPLIED
>
<!ELEMENT userProxy-get (%cm.user-get;)*>
<!ELEMENT userProxy-where %cm.user-where;>
<!ELEMENT workflow-create %cm.workflow-create;>
<!ELEMENT workflow-delete %cm.workflow-delete;>
<!ELEMENT workflow-get (%cm.workflow-get;)*>
<!ELEMENT workflow-description (%cm.atom;)>
<!ELEMENT workflow-set %cm.workflow-set;>
<!ELEMENT workflow-where %cm.workflow-where;>
<!ATTLIST workflow-where
    maxResults CDATA #IMPLIED
>

<!ELEMENT attribute (%cm.atom; | %cm.attribute-get;)*>
<!ELEMENT attributeGroup (attribute | %cm.attributeGroup-get;)*>
<!ELEMENT content (%cm.content-get;)*>
<!ELEMENT group (%cm.atom; | %cm.group-get;)*>
<!ELEMENT job (%cm.job-get;)>
<!ELEMENT link (%cm.link-get;)>
<!ELEMENT obj (%cm.obj-get;)*>
<!ELEMENT objClass (%cm.atom; | %cm.objClass-get;)*>
<!ELEMENT updateData ANY>
<!ELEMENT user (%cm.atom; | %cm.user-get;)*>
<!ELEMENT userAttribute (%cm.userAttribute-get;)>
<!ELEMENT arguments (%cm.listitem;)*>
<!ELEMENT all (%cm.atom;)>
<!ELEMENT allowsMultipleSignatures (%cm.atom;)>
<!ELEMENT anchors (%cm.listitem;)*>
<!ELEMENT archivedContents ((content)* | (%cm.listitem;)*)>
<!ELEMENT attributeName (%cm.atom; | %cm.attribute-get;)*>
<!ELEMENT attributeNames (%cm.listitem;)*>
<!ELEMENT attributeGroups ((attributeGroup)* | (%cm.listitem;)*)>
<!ELEMENT attributes ((attribute)* | (%cm.listitem;)* | (dictitem)+)>
<!ELEMENT availableBlobEditors (%cm.listitem;)*>
<!ELEMENT blob (%cm.atom;)>
<!ATTLIST blob
    encoding (plain | base64 | stream) #IMPLIED
>
<!ELEMENT blobLength (%cm.atom;)>
<!ELEMENT body (%cm.atom;)>
<!ELEMENT bodyTemplateName (%cm.atom;)>
<!ELEMENT callback (%cm.atom;)>
<!ELEMENT canHaveAnchor (%cm.atom;)>
<!ELEMENT canHaveTarget (%cm.atom;)>
<!ELEMENT category (%cm.atom;)>
<!ELEMENT children ((obj)* | (%cm.listitem;)*)>
<!ELEMENT codeForSourceView (%cm.atom;)>
<!ATTLIST codeForSourceView
        objectPageUrl CDATA #REQUIRED
        linkPageUrl CDATA #REQUIRED
        useJavaScript CDATA #IMPLIED
>
<!ELEMENT collections (%cm.listitem;)>
<!ELEMENT command (%cm.atom;)>
<!ELEMENT comment (%cm.atom;)>
<!ELEMENT committedContentId (%cm.atom; | %cm.content-get;)*>
<!ELEMENT completionCheck (%cm.atom;)>
<!ELEMENT contentId (%cm.atom;)>
<!ELEMENT contentIds ((content)* | (%cm.listitem;)*)>
<!ELEMENT contents ((content)* | (%cm.listitem;)*)>
<!ELEMENT contentType (%cm.atom;)>
<!ELEMENT count (%cm.atom;)>
<!ELEMENT createPermission (%cm.atom;)>
<!ELEMENT customBlobEditorUrl (%cm.atom;)>
<!ELEMENT defaultAttributeGroup (%cm.atom; | attributeGroup)*>
<!ELEMENT defaultGroup (%cm.atom; | group)*>
<!ELEMENT deleteLogEntriesCount (%cm.atom;)>
```

```
<!ELEMENT description (%cm.atom;)>
<!ELEMENT destination (%cm.atom; | %cm.obj-get;)*>
<!ELEMENT destinationUrl (%cm.atom;)>
<!ELEMENT displayTitle (%cm.atom;)>
<!ATTLIST displayTitle
        type CDATA #IMPLIED
>
<!ELEMENT displayValueCallback (%cm.atom;)>
<!ELEMENT editField (%cm.atom;)>
<!ATTLIST editField
        parameter (length | maxlength | nilAllowed | objClasses | rows | startPub| type)
 #REQUIRED
>
<!ELEMENT editFieldSpec (%cm.atom; | %cm.dictitem;)*>
<!ELEMENT editGroups ((group)* | (%cm.listitem;)*)>
<!ELEMENT editedContent (%cm.atom; | %cm.content-get;)*>
<!ELEMENT editedContentId (%cm.atom; | %cm.content-get;)*>
<!ELEMENT editor (%cm.atom; | %cm.user-get;)*>
<!ELEMENT email (%cm.atom;)>
<!ELEMENT emptyAttributeGroups ((%cm.listitem;)* | (attributeGroup)*)>
<!ELEMENT encryptedPassword (%cm.atom;)>
<!ELEMENT execLogin (%cm.atom;)>
<!ELEMENT execPerm (%cm.atom;)>
<!ELEMENT expectedPath (%cm.atom;)>
<!ELEMENT exportBlob (%cm.atom;)>
<!ATTLIST exportBlob
    encoding (plain | base64 | stream) #IMPLIED
>
<!ELEMENT exportFiles (%cm.listitem;)*>
<!ELEMENT exportMimeType (%cm.atom;)>
<!ELEMENT externalAttrNames ((attribute)* | (%cm.listitem;)*)>
<!ELEMENT externalAttributes (%cm.dictitem;)*>
<!ELEMENT externalEditExtension (%cm.atom;)>
<!ELEMENT externalEditMimeType (%cm.atom;)>
<!ELEMENT externalUserAttrNames ((userAttribute)* | (%cm.listitem;)*)>
<!ELEMENT extractToDirectory (%cm.atom;)>
<!ELEMENT filter (%cm.atom;)>
<!ELEMENT file (%cm.atom;)>
<!ELEMENT filePrefix (%cm.atom;)>
<!ELEMENT frameNames (%cm.listitem;)*>
<!ELEMENT freeLinks ((link)* | (%cm.listitem;)*)>
<!ELEMENT fromDate (%cm.atom;)>
<!ELEMENT getKeys (%cm.listitem;)*>
<!ELEMENT globalPermissions (%cm.listitem;)*>
<!ELEMENT goodAttributeGroupAttributes ((%cm.listitem;)* | (attribute)*)>
<!ELEMENT goodAttributes ((%cm.listitem;)* | (attribute)*)>
<!ELEMENT goodMandatoryAttributes ((attribute)* | (%cm.listitem;)*)>
<!ELEMENT goodPresetAttributes ((attribute)* | (%cm.listitem;)*)>
<!ELEMENT goodPresetFromParentAttributes ((attribute)* | (%cm.listitem;)*)>
<!ELEMENT groupName (%cm.atom; | %cm.group-get;)*>
<!ELEMENT groupNames (%cm.listitem;)*>
<!ELEMENT groupText (%cm.atom;)>
<!ELEMENT groups ((group)* | (%cm.listitem;)*)>
<!ELEMENT hasChildren (%cm.atom;)>
<!ELEMENT hasGlobalPermission (%cm.atom;)>
<!ATTLIST hasGlobalPermission
        permission CDATA #REQUIRED
>
<!ELEMENT hasPassword (%cm.atom;)>
<!ATTLIST hasPassword
        password CDATA #REQUIRED
>
<!ELEMENT hasSuperLinks (%cm.atom;)>
<!ELEMENT helpText (%cm.atom;)>
<!ATTLIST helpText
        lang (en | de | it | fr | es) #IMPLIED
>
<!ELEMENT hierarchy (%cm.listitem;)*>
<!ATTLIST hierarchy
        maxDepth CDATA #IMPLIED
        maxLines CDATA #IMPLIED
        document (0 | 1) #IMPLIED
```

```
        generic (0 | 1) #IMPLIED
        image (0 | 1) #IMPLIED
        publication (0 | 1) #IMPLIED
        template (0 | 1) #IMPLIED
>
<!ELEMENT condition (%cm.atom; | %cm.listitem;)*>
<!ATTLIST condition
    subject CDATA #REQUIRED
    verb CDATA #REQUIRED
>
<!ELEMENT id (%cm.atom;)>
<!ELEMENT ids (%cm.atom; | %cm.listitem;)*>
<!ELEMENT index (%cm.atom;)>
<!ELEMENT inputType (%cm.atom;)>
<!ELEMENT isActive (%cm.atom;)>
<!ELEMENT isCommitted (%cm.atom;)>
<!ELEMENT isComplete (%cm.atom;)>
<!ELEMENT isDefaultGroup (%cm.atom;)>
<!ELEMENT isEdited (%cm.atom;)>
<!ELEMENT isEmpty (%cm.atom;)>
<!ELEMENT isEnabled (%cm.atom;)>
<!ELEMENT isExternalLink (%cm.atom;)>
<!ELEMENT isFreeLink (%cm.atom;)>
<!ELEMENT isGoodDestination (%cm.atom;)>
<!ATTLIST isGoodDestination
        linkId CDATA #REQUIRED
>
<!ELEMENT isGoodParent (%cm.atom;)>
<!ATTLIST isGoodParent
        objectId CDATA #REQUIRED
>
<!ELEMENT isIncludeLink (%cm.atom;)>
<!ELEMENT isInlineReferenceLink (%cm.atom;)>
<!ELEMENT isLinkFromCommittedContent (%cm.atom;)>
<!ELEMENT isLinkFromEditedContent (%cm.atom;)>
<!ELEMENT isLinkFromReleasedContent (%cm.atom;)>
<!ELEMENT isoDateTime (%cm.atom;)>
<!ELEMENT isOwnerOf (%cm.atom;)>
<!ATTLIST isOwnerOf
        login CDATA #REQUIRED
>
<!ELEMENT isQueued (%cm.atom;)>
<!ELEMENT isRelatedLink (%cm.atom;)>
<!ELEMENT isReleased (%cm.atom;)>
<!ELEMENT isRoot (%cm.atom;)>
<!ELEMENT isSearchableInCM (%cm.atom;)>
<!ELEMENT isSearchableInTE (%cm.atom;)>
<!ELEMENT isSuperUser (%cm.atom;)>
<!ELEMENT isWritable (%cm.atom;)>
<!ELEMENT lastChanged %cm.date;>
<!ATTLIST lastChanged
        type CDATA #IMPLIED
>
<!ELEMENT lastExecEnd %cm.date;>
<!ATTLIST lastExecEnd
        type CDATA #IMPLIED
>
<!ELEMENT lastExecResult (%cm.atom;)>
<!ELEMENT lastExecStart %cm.date;>
<!ATTLIST lastExecStart
        type CDATA #IMPLIED
>
<!ELEMENT lastLogEntry %cm.logEntry;>
<!ELEMENT lastOutput (%cm.atom;)>
<!ELEMENT linkListAttributes ((attribute)* | (%cm.listitem;)*)>
<!ELEMENT localizedTitle (%cm.atom;)>
<!ELEMENT localizedHelpText (%cm.atom;)>
<!ELEMENT log (%cm.listitem;)*>
<!ELEMENT logEntries (%cm.listitem;)*>
<!ELEMENT logEntryId (%cm.atom;)>
<!ELEMENT execResult (%cm.atom;)>
<!ELEMENT execStart %cm.date;>
```

```
<!ATTLIST execStart
    type CDATA #IMPLIED
>
<!ELEMENT execEnd %cm.date;>
<!ATTLIST execEnd
    type CDATA #IMPLIED
>
<!ELEMENT logText (%cm.atom;)>
<!ELEMENT logTime %cm.date;>
<!ELEMENT logType (%cm.atom;)>
<!ELEMENT logTypes (%cm.listitem;)*>
<!ELEMENT login (%cm.atom;)>
<!ELEMENT maxDocs (%cm.atom;)>
<!ELEMENT maxSize (%cm.atom;)*>
<!ELEMENT mandatoryAttributes ((attribute)* | (%cm.listitem;)*)>
<!ELEMENT method (%cm.atom;)>
<!ELEMENT mimeType (%cm.atom;)>
<!ELEMENT minRelevance (%cm.atom;)>
<!ELEMENT minSize (%cm.atom;)*>
<!ELEMENT mode (%cm.atom;)>
<!ELEMENT name (%cm.atom;)>
<!ATTLIST name
        type CDATA #IMPLIED
>
<!ELEMENT nameLike (%cm.atom;)>
<!ELEMENT namevalue (name, value)>
<!ELEMENT next (%cm.atom; | %cm.obj-get;)*>
<!ELEMENT nextEditGroup (%cm.atom; | %cm.group-get;)*>
<!ELEMENT nextExecStart %cm.date;>
<!ATTLIST nextExecStart
        type CDATA #IMPLIED
>
<!ELEMENT nextSignGroup (%cm.atom; | %cm.group-get;)*>
<!ELEMENT now %cm.date;>
<!ATTLIST now
        type CDATA #IMPLIED
>
<!ELEMENT today %cm.date;>
<!ELEMENT objClasses ((%cm.listitem;)* | (objClass)*)>
<!ELEMENT objType (%cm.atom;)>
<!ELEMENT objectId (%cm.atom; | %cm.obj-get;)*>
<!ELEMENT objectsToRoot ((obj)* | (%cm.listitem;)*)>
<!ELEMENT offsetStart (%cm.atom;)>
<!ELEMENT offsetLength (%cm.atom;)>
<!ELEMENT outputType (%cm.atom;)>
<!ELEMENT owner (%cm.atom; | %cm.user-get;)*>
<!ELEMENT parent (%cm.atom; | %cm.obj-get;)*>
<!ELEMENT parser (%cm.atom;)>
<!ELEMENT password (%cm.atom;)>
<!ELEMENT path (%cm.atom;)>
<!ELEMENT permission ((group)* | (%cm.listitem;)*)>
<!ATTLIST permission
        permission (permissionCreateChildren | permissionWrite | permissionRead |
 permissionRoot) #REQUIRED
>
<!ELEMENT permissionGrantedTo (%cm.atom;)>
<!ATTLIST permissionGrantedTo
        permission (permissionCreateChildren | permissionWrite | permissionRead |
 permissionRoot) #REQUIRED
        user CDATA #IMPLIED
        group CDATA #IMPLIED
>
<!ELEMENT prefixPath (%cm.atom;)>
<!ELEMENT presetAttributes (dictitem)*>
<!ELEMENT presetFromParentAttributes ((attribute)* | (%cm.listitem;)*)>
<!ELEMENT previous (%cm.atom; | %cm.obj-get;)*>
<!ELEMENT queuePos (%cm.atom;)>
<!ELEMENT query (%cm.atom;)>
<!ELEMENT realName (%cm.atom;)>
<!ELEMENT reasonsForIncompleteState (%cm.listitem;)*>
<!ELEMENT receiver (%cm.atom;)>
<!ELEMENT recordSetCallback (%cm.atom;)>
```

```
<!ELEMENT recursive (%cm.atom;)>
<!ELEMENT relatedLinks ((link)* | (%cm.listitem;)*)>
<!ELEMENT releasedContent (%cm.atom; | %cm.content-get;)*>
<!ELEMENT releasedContentId (%cm.atom; | %cm.content-get;)*>
<!ELEMENT releasedVersions ((content)* | (%cm.listitem;)*)>
<!ELEMENT rootPermissionFor (%cm.atom;)>
<!ATTLIST rootPermissionFor
        user CDATA #IMPLIED
        group CDATA #IMPLIED
>
<!ELEMENT schedule (%cm.listitem;)*>
<!ELEMENT script (%cm.atom;)>
<!ELEMENT setKeys (%cm.listitem;)*>
<!ELEMENT signatureAttrNames ((attribute)* | (%cm.listitem;)*)>
<!ELEMENT signatureAttributes (namevalue)*>
<!ELEMENT signatureDefs (%cm.dictitem;)*>
<!ELEMENT sortKey1 (%cm.atom;)>
<!ELEMENT sortKey2 (%cm.atom;)>
<!ELEMENT sortKey3 (%cm.atom;)>
<!ELEMENT sortKeyLength1 (%cm.atom;)>
<!ELEMENT sortKeyLength2 (%cm.atom;)>
<!ELEMENT sortKeyLength3 (%cm.atom;)>
<!ELEMENT sortOrder (%cm.atom;)>
<!ELEMENT sortType1 (%cm.atom;)>
<!ELEMENT sortType2 (%cm.atom;)>
<!ELEMENT sortType3 (%cm.atom;)>
<!ELEMENT sortValue (%cm.atom;)>
<!ELEMENT source (%cm.atom; | %cm.obj-get;)*>
<!ELEMENT sourceContent (%cm.atom; | %cm.content-get;)*>
<!ELEMENT sourceTagAttribute (%cm.atom;)>
<!ELEMENT sourceTagName (%cm.atom;)>
<!ELEMENT state (%cm.atom;)>
<!ELEMENT subLinks ((link)* | (%cm.listitem;)*)>
<!ELEMENT superLinks ((link)* | (%cm.listitem;))>
<!ELEMENT superObjects ((obj)* | (%cm.listitem;))>
<!ELEMENT suppressExport (%cm.atom;)>
<!ELEMENT systemConfigFormattedTime (%cm.atom;)>
<!ELEMENT target (%cm.atom;)>
<!ELEMENT taskText (%cm.atom;)>
<!ELEMENT taskType (%cm.atom;)>
<!ELEMENT textLinks ((link)* | (%cm.listitem;)*)>
<!ELEMENT templateName (%cm.atom;)>
<!ELEMENT thumbnail (%cm.atom;)>
<!ELEMENT timeStamp (%cm.atom;)>
<!ELEMENT timeZone (%cm.atom;)>
<!ELEMENT title (%cm.atom;)>
<!ATTLIST title
        lang (en | de | it | fr | es) #IMPLIED
        type CDATA #IMPLIED
>
<!ELEMENT toclist ((obj)* | (%cm.listitem;)*)>
<!ELEMENT type (%cm.atom;)>
<!ELEMENT untilDate %cm.date;>
<!ELEMENT updateRecords (%cm.listitem;)*>
<!ELEMENT updateRecordCount (%cm.atom;)>
<!ELEMENT updateRecordId (%cm.atom;)>
<!ELEMENT userConfigFormattedTime (%cm.atom;)>
<!ELEMENT userLocked (%cm.atom;)>
<!ELEMENT userLogin (%cm.atom; | %cm.user-get;)*>
<!ELEMENT userText (%cm.atom;)>
<!ELEMENT users ((user)* | (%cm.listitem;)*)>
<!ELEMENT userProxy ANY>
<!ELEMENT validContentTypes (%cm.listitem;)*>
<!ELEMENT validControlActionKeys (%cm.listitem;)*>
<!ELEMENT validCreateObjClasses ((objClass)* | (%cm.listitem;)*)>
<!ELEMENT validEditFieldKeys (%cm.listitem;)*>
<!ELEMENT validEditFieldTypes (%cm.listitem;)*>
<!ELEMENT validFrom %cm.date;>
<!ATTLIST validFrom
        type CDATA #IMPLIED
>
<!ELEMENT validObjClasses ((objClass)* | (%cm.listitem;)*)>
```

```
<!ELEMENT validPermissions (%cm.listitem;)*>
<!ELEMENT validSortKeys (%cm.listitem;)*>
<!ELEMENT validSortOrders (%cm.listitem;)*>
<!ELEMENT validSortTypes (%cm.listitem;)*>
<!ELEMENT validSubObjClassCheck (%cm.atom;)>
<!ELEMENT validSubObjClasses ((objClass)* | (%cm.listitem;)*)>
<!ELEMENT validUntil %cm.date;>
<!ATTLIST validUntil
        type CDATA #IMPLIED
>
<!ELEMENT values (%cm.listitem;)*>
<!ELEMENT version (%cm.atom;)>
<!ELEMENT visibleExportTemplates ((obj)* | (%cm.listitem;)*)>
<!ELEMENT visibleName (%cm.atom;)>
<!ELEMENT visiblePath (%cm.atom;)>
<!ELEMENT wantedTags (%cm.listitem;)*>
<!ELEMENT workflowComment (%cm.atom;)>
<!ELEMENT workflowModification (%cm.atom;)>
<!ELEMENT workflowName (%cm.atom; | %cm.workflow-get;)*>
<!ELEMENT xmlBlob (%cm.atom;)>
<!ELEMENT xmldtd (%cm.atom;)>

<!ELEMENT appName (%cm.atom;)>
<!ELEMENT baseDir (%cm.atom;)>
<!ELEMENT binDir (%cm.atom;)>
<!ELEMENT configDir (%cm.atom;)>
<!ELEMENT dataDir (%cm.atom;)>
<!ELEMENT debugChannels (%cm.atom;)>
<!ELEMENT debugLevel (%cm.atom;)>
<!ELEMENT instanceDir (%cm.atom;)>
<!ELEMENT libDir (%cm.atom;)>
<!ELEMENT logDir (%cm.atom;)>
<!ELEMENT rootConfigPath (%cm.atom;)>
<!ELEMENT scriptDir (%cm.atom;)>
<!ELEMENT shareDir (%cm.atom;)>
<!ELEMENT tmpDir (%cm.atom;)>
```

28

28 Fehlermeldungen

10005: Die Anzahl der Response-Elemente entspricht nicht der Anzahl der Request-Elemente in der Anfrage.

20005: Um den Verwalter eines Benutzers oder einer Benutzergruppe zu setzen, benötigt der angemeldete Benutzer das globale Recht `permissionGlobalUserEdit`.

20006: Der angemeldete Benutzer kann andere Benutzer nicht in Gruppen aufnehmen oder aus Gruppen entfernen, da er das globale Recht `permissionGlobalUserEdit` nicht hat.

20007: Der angemeldete Benutzer kann keine globalen oder dateispezifischen Rechte erteilen, da er die hierfür erforderlichen Rechte (`permissionGlobalRoot` bzw. `permissionRoot`) nicht hat.

20008: Der angemeldete Benutzer kann keine globalen oder dateispezifischen Rechte entziehen, da er die hierfür erforderlichen Rechte (`permissionGlobalRoot` bzw. `permissionRoot`) nicht hat.

20009: Der angemeldete Benutzer hat nicht das erforderliche Recht (`permissionGlobalUserEdit`), um einen anderen Benutzer zu sperren oder zu entsperren.

20012: Der Benutzer darf die Transaktion nicht ausführen.

20013: Der angemeldete Benutzer benötigt das globale Recht `permissionGlobalExport`, um die Versionsfunktion `debugExport` verwenden oder auf eines der Versionsfelder `encodedExportBlob`, `exportBlob` und `exportFiles` zugreifen zu können.

20014: Der angemeldete Benutzer darf die Felder der Version nicht auslesen, da er das Leserecht (`permissonRead`) für die Datei, zu der die Version gehört, nicht hat.

20015: Der angemeldete Benutzer darf die Parameter der Datei nicht auslesen, da er das Leserecht (`permissonRead`) für die Datei nicht hat.

20016: Der Benutzer hat nicht die Rechte, die erforderlich sind, um eine Datei mit der angegebenen Vorlage in diesem Ordner anzulegen. Diese Rechte sind das globale Anlegerecht der Vorlage und das Dateianlegerecht in dem betreffenden Ordner.

20020: Der angemeldete Benutzer hat versucht, die Arbeitsversion einer Datei freizugeben, ohne dazu berechtigt zu sein.

20021: Der eingeloggte Benutzer hat versucht, den Workflow einer Datei abzubrechen und einen neuen zu beginnen, ohne dazu berechtigt zu sein.

33008: Der Wert des angegebenen Bezeichners ist keine URL und kann auch nicht in eine URL umgewandelt werden.

36001: Beim Aufruf eines zusätzlichen (kundenspezifischen) Befehls wurde die konfigurierte, damit assoziierte Tcl-Prozedur nicht gefunden.

36024: Diese Meldung betrifft den Portal Manager. Bei einem Kommando, das genau einen der genannten Werte erwartet, wurde keiner dieser Werte angegeben.

36025: Diese Meldung betrifft den Portal Manager. Bei einem Kommando, das genau einen der genannten Werte erwartet, wurde keiner dieser Werte angegeben.

40005: Der Benutzer ist nicht der Administrator der Datei und muss deshalb der Bearbeiter sein, um die Workflowaktion ausführen zu können.

40009: Der Benutzer hat die Datei bereits unterschrieben. Da er nicht der Administrator der Datei ist und der Workflow Mehrfachunterzeichnungen verbietet, darf er kein zweites Mal unterschreiben.

40010: Die Datei kann nicht eingereicht oder abgezeichnet, sondern nur freigegeben werden.

40011: Die `commit`-Operation kann nicht ausgeführt werden, da die Datei noch nicht von allen Bearbeitergruppen im Workflow bearbeitet wurde. Nur der Dateiadministrator und ein Superuser hat das Recht, die Datei trotzdem einzureichen oder freizugeben.

40012: Die Datei kann nicht eingereicht oder freigegeben werden, da der Arbeitscontent unvollständig ist. Dies bedeutet, dass die Arbeitsversion defekte Links oder Felder mit nicht zulässigen Inhalten enthält.

40013: Der Benutzer ist weder Administrator der Datei noch Superuser und kann die Datei nicht vorzeitig freigeben.

40014: Der Benutzer ist weder Administrator der Datei noch Superuser und kann sie nicht vorzeitig freigeben.

40015: Der Benutzer ist weder Administrator der Datei noch Superuser, sodass er Mitglied in der letzten Prüfergruppe sein muss, um die Datei freigeben zu können.

40016: Es gibt eine eingereichte Version, sodass nur Mitglieder der verbleibenden unterschriftsberechtigten Gruppen die Datei ablehnen dürfen.

40017: Es wurde versucht, ein Unterschriftenfeld mehrfach zu verwenden. Dies ist nicht zulässig.

50009: Der angegebene Name enthält unzulässige Zeichen. Folgende Zeichen können in Namen verwendet werden: A-Z, a-z, 0-9 und _.

50014: Bei einem `sortKey`-Feld wurde ein Feld angegeben, das nicht existiert.

50015: Für das `sortOrder`-Feld wurde ein unzulässiger Wert angegeben.

50024: Die angegebene Spezifikation der `presetAttributes` (vorbelegte Felder) ist nicht korrekt. Es konnte daraus keine gültige Liste gebildet werden.

50025: Der angegebene Name enthält unzulässige Zeichen. Folgende Zeichen können in Namen verwendet werden: A-Z, a-z, 0-9 und _.

50034: Der angegebene Name enthält unzulässige Zeichen. Folgende Zeichen können in Namen verwendet werden: A-Z, a-z, 0-9 und _.

50037: Der angegebene Name enthält unzulässige Zeichen. Folgende Zeichen können in Namen verwendet werden: A-Z, a-z, 0-9 und _.

50038: Beim angegebenen Feld wurde in der Liste der `presetAttributes` der genannte Wert angegeben, der jedoch für dieses Feld unzulässig ist.

50040: Das angegebene Tag enthält unzulässige Zeichen.

50042: Die erlaubten Dateinamenserweiterungen können im Systemkonfigurationseintrag `mimeTypes` konfiguriert werden.

50043: Die angegebene Definition enthält ein nicht existierendes Feld oder eine nicht existierende Benutzergruppe.

50044: Der angegebene Name enthält unzulässige Zeichen. Folgende Zeichen können in Namen verwendet werden: A-Z, a-z, 0-9 und _.

50045: In einem Update-Record wurde ein unbekannter Update-Typ angegeben.

50063: Job-Namen dürfen nicht mit `system` und nicht mit dem Unterstrich beginnen. Erlaubte Zeichen in Job-Namen sind a - z, A - Z, 0 - 9 und der Unterstrich.

55000: Die Zeichenfolge besteht aus zwei Zeichen, die sich jeweils im Bereich von 0 bis 9, a bis z oder A bis Z befinden. Die Zeichen werden zur Verschlüsselung verwendet.

55001: Mit diesem Fehler werden Payloads beantwortet, die in einer Version des XML-Schnittstellenprotokolls formuliert sind, die der Server nicht unterstützt. Zum Verhalten des Servers in diesen Fällen siehe das Handbuch *Die XML-Schnittstelle*.

55003: Der Link gehört zu einem Tag, in dem es nicht erlaubt ist, einen Anker anzugeben.

55004: Der Link gehört zu einem Tag, das Frame-Targets nicht unterstützt.

55005: Die beim Kopieren oder Verschieben angegebene Datei kann nicht als Ziel akzeptiert werden. Ursache kann sein: eine Datei mit dem Namen der betreffenden Datei existiert bereits in dem Ordner; die Datei ist kein Ordner, liegt innerhalb des zu verschiebenden Hierarchiezweiges oder darf keine Unterordner mit der Vorlage der Quelldatei haben. Möglicherweise haben Sie auch nicht die erforderlichen Rechte.

55011: Der angegebene Name enthält unzulässige Zeichen. Folgende Zeichen können in Namen verwendet werden: A-Z, a-z, 0-9 und _.

55015: Die angegebene oder bei der Operation verwendete Namenserweiterung ist nicht in den `validContentTypes` der Vorlage enthalten, das der Datei zugeordnet ist.

55023: Die angegebene Vorlage kann bei dieser Datei nicht eingetragen werden. Es existiert nicht oder ist nicht in den `validSubObjClasses` der Vorlage des darüber liegenden Ordners enthalten.

55036: Einem Link konnte der angegebene Feldname nicht zugewiesen werden, weil es das Feld in der Version, zu der der Link gehört, nicht gibt.

55046: Dieser Fehler tritt auf, wenn man versucht, ein Miniaturbild (Thumbnail) für eine Datei zu erzeugen, die nicht vom Typ *Bild* oder *Ressource* ist.

55047: Der angegebene Zeichensatz existiert nicht oder wurde nicht definiert.

55054: Bei dem Dateibefehl `createAndLoad` wurde als `file` ein Pfad mit mehr als zwei Komponenten angegeben, oder der Pfad ist absolut oder verweist auf das übergeordnete Verzeichnis.

55055: Bei dem Versuch, auf ein temporäres Verzeichnis zuzugreifen, nachdem der Upload Wizard aufgerufen wurde, wurde ein Pfad angegeben, der aus mehreren Teilen besteht. Dies ist nicht zulässig.

60014: Dies ist ein interner Fehler. Bitte wenden Sie sich an den Support (support@infopark.de).

60015: Dies ist ein interner Fehler. Bitte wenden Sie sich an den Support (support@infopark.de).

60016: Diese Meldung betrifft den Portal Manager. Der angegebene Pfad existiert nicht.

60033: Das angegebene Layout wird beim Export benötigt, ist jedoch nicht vorhanden oder nicht freigegeben.

80001: Bevor ein Ordner gelöscht werden kann, müssen die darin enthaltenen Dateien gelöscht werden.

80009: Das Feld kann nicht gelöscht werden, da es noch in den angegebenen Vorlagen verwendet wird.

80010: Das angegebene Feld kann nicht gelöscht werden, da es noch in den angegebenen Workflows für Unterschriften verwendet wird.

80011: Der Workflow kann nicht gelöscht werden, da er in den angegebenen Vorlagen im vorbelegten `workflowName`-Feld referenziert wird.

80014: Bei den genannten Benutzern muss der Wert des Feldes geändert werden, um den Aufzählungswert löschen zu können.

85013: Requests in der Request-Warteschlange des Search Engine Servers werden temporär nicht abgearbeitet, während die exportierten Dokumente live geschaltet werden.

85015: Der Request an die XML-Schnittstelle wurde nicht bearbeitet, weil ein vorhergehender Request, der als `preclusive` markiert war, fehlgeschlagen ist.

85020: Es wurde versucht, den Hauptinhalt einer Version zu bearbeiten. Dies ist jedoch nicht möglich, wenn die Datei, zu der die Version gehört, auf einer Vorlage basiert, bei der ein Hauptlayout angegeben wurde.

85028: Felder werden automatisch zur Basisgruppe hinzugefügt, wenn sie aus anderen Feldergruppen gelöscht oder in die Vorlage aufgenommen werden.

85031: Der Content ist ein eingereichter oder freigegebener Content und damit nicht änderbar.

85033: Die aufgeführten Felder können nicht in die Liste `mandatoryAttributes` aufgenommen werden, weil sie nicht in der Felderliste der Vorlage enthalten sind.

85034: Die aufgeführten Felder können zur Liste `presetAttributes` nicht hinzugefügt werden, weil sie nicht in der Felderliste der Vorlage enthalten sind.

85035: Die aufgeführten Felder können zur Liste `presetFromParentAttributes` nicht hinzugefügt werden, weil sie nicht in der Felderliste der Vorlage enthalten sind.

85036: Der in der Vorlage angegebene Dateityp lässt keine Unterdateien zu. Der Fehler tritt auf, wenn man einer Vorlage, mit der kein Ordner (sondern ein anderer Dateityp) definiert wird, eine erlaubte Vorlage für darin enthaltene Dateien zuzuweisen versucht.

85037: Es wurde versucht, bei einer Vorlage, die Layouts, Bilder oder Ressourcen definiert, ein Hauptinhaltslayout anzugeben. Bei solchen Vorlagen kann kein Hauptinhaltslayout angegeben werden.

85038: In der Definition einer Vorlage wurde versucht, den Hauptinhalt vorzubelegen. Dies ist jedoch nicht möglich, wenn ein Hauptinhaltslayout angegeben ist.

85039: In der Definition einer Vorlage wurde versucht, den Hauptinhalt mit dem Wert des Hauptinhalts des darüber liegenden Ordners vorzubelegen. Dies ist jedoch nicht möglich, wenn ein Hauptinhaltslayout angegeben ist.

85041: Kontextlinks müssen auf CMS-Dateien verweisen.

90007: Das angegebene Feld kommt im XML-Dokument vor, in der zum Dokument gehörenden Vorlage ist es jedoch nicht enthalten.

100000: Der angegebene kundenspezifische Befehl (*engl.* "custom command") konnte nicht ausgeführt werden.

100001: Bei der Ausführung eines Custom Commands konnte nicht in eine Binärdatei geschrieben werden.

100016: Die angegebenen Werte können nicht verwendet werden. Der Wert für den Konfigurationseintrag `master.maxSlaves` muss immer höher sein als der für `master.minIdleSlaves`.

100029: Dies ist ein Datenbankfehler.

100030: Dies ist ein Datenbankfehler.

100031: Dies ist ein Datenbankfehler.

100032: Dies ist ein Datenbankfehler.

100037: Dies ist ein Datenbankfehler.

100038: Dies ist ein Datenbankfehler.

100039: Eine CMS-Applikation hat versucht, auf eine nicht vorhandene Datei lesend zuzugreifen. Der Fehler kann beispielsweise dann auftreten, wenn das Streaming-Interface eine Datei zu einem Ticket ausliefern möchte, diese Datei jedoch gelöscht wurde.

100040: Dies ist ein Datenbankfehler.

100043: Dies ist ein Datenbankfehler.

100044: Dies ist ein Datenbank- oder ein interner Systemfehler. Bitte wenden Sie sich an den Support (support@infopark.de).

100052: Die Datei, in der die Logins und Passwörter aller Benutzer gespeichert werden, kann nicht erzeugt werden.

100054: Dies ist ein Datenbankfehler.

100055: Dies ist ein Datenbankfehler.

100056: Dies ist ein Datenbankfehler.

100110: Dies ist ein Datenbankfehler.

100111: Die Arbeitsversion konnte nicht angelegt werden, weil es keine voreingestellte Dateinamenserweiterung gibt.

100112: Dies ist ein Datenbankfehler.

100113: Dies ist ein Datenbankfehler.

100114: Dies ist ein Datenbankfehler.

100115: Dies ist ein Datenbankfehler.

100116: Dies ist ein Datenbankfehler.

100119: Dies ist ein interner Systemfehler.

100121: Dies ist ein Datenbankfehler.

100123: Der angegebene Typ entspricht keiner internen Repräsentation eines Tasktyps.

100148: Dies ist ein Datenbankfehler.

100149: Dies ist ein Datenbankfehler.

100150: Dies ist ein Datenbankfehler.

100151: Dies ist ein Datenbankfehler.

100152: Die Applikation konnte auf die angegebene Steuerdatei nicht zugreifen. Entweder existiert die Datei nicht oder der Applikation fehlt das Leserecht.

100153: Dies ist ein Datenbankfehler.

100154: Dies ist ein Datenbankfehler.

100164: Bei der Indizierung durch den Search Engine Server wurde der externe Prozessor nicht gefunden oder die Programmdatei war nicht ausführbar.

100165: Bei der Indizierung durch den Search Engine Server sollten dem externen Prozessor Daten über eine *Pipe* übergeben werden. Dieser Vorgang ist fehlgeschlagen.

100166: Bei der Indizierung durch den Search Engine Server sollte die Ausgabe des externen Prozessors über eine *Pipe* gelesen werden. Dieser Vorgang ist fehlgeschlagen.

100167: Dies ist ein Datenbankfehler.

100168: Dieser Fehler kann unterschiedliche Ursachen haben, unter anderem, dass der Pfad nicht existierende Verzeichnisse enthält oder die Zugriffsrechte nicht ausreichen.

100172: Bei der Initialisierung der Verity-Suchmaschine wurden keine Collections gefunden. Diese werden im instanzenspezifischen Verzeichnis `data/collections` gesucht.

100184: Beim SES werden auszuführende Aktionen in die Namen von Requestdateien kodiert. Es gibt nur die beiden Aktionen *Indizieren* und *Löschen*. In Bezug auf eine solche Datei ist es zu einer Inkonsistenz gekommen. Der Fehler wird ins System-Protokoll geschrieben, und die betreffende Datei wird ignoriert.

100185: Dies ist ein Datenbankfehler.

100188: Bei dem Versuch, ein Streaming-Ticket anzulegen, ist ein Fehler aufgetreten. Möglicherweise existiert das Tickets-Verzeichnis nicht oder die Applikation darf nicht darauf zugreifen.

100197: Die Längenbeschränkung ist auf das Dateisystem oder das Betriebssystem zurückzuführen.

100209: Dieser Fehler betrifft die Template Engine. Beim Export sind mehr Fehler aufgetreten, als der Wert des Systemkonfigurationseintrags `acceptableExportFailures` zulässt.

100210: Der Portal Manager oder die Applikation, die diese Meldung ausgegeben hat, ist falsch konfiguriert. Möglicherweise wurde der Portal Manager auch nicht gestartet.

100219: Dies ist ein Datenbankfehler.

109010: Dies ist ein Channel-Check-Callback-Fehler.

109012: Dies ist ein Contentzuweisungscallback-Fehler.

109016: Während `sudo` ausgeführt wurde, traten die aufgeführten Fehler auf.

109017: Während die `generateThumbnail`-Funktion ausgeführt wurde, traten die aufgeführten Fehler auf.

109018: Die Prozedur `usersWhere` in der Datei `usermanAPI.tcl` hat den angegebenen Fehler geliefert.

109021: Die Prozedur `groupsWhere` in der Datei `usermanAPI.tcl` hat den angegebenen Fehler geliefert.

109500: Der Link-Callback kann nicht ausgeführt werden, da die Liste der Felder die falsche Anzahl Elemente enthält.

109501: Der Link-Callback konnte nicht korrekt ausgeführt werden.

109503: Thumbnails müssen im JPEG-Format gespeichert werden.

109511: Dieser Fehler wird ausgegeben und die Fehlerursache wird näher spezifiziert, wenn der Tcl-Code in einer der folgenden Funktionen einen Fehler produziert: Wertzuweisungsfunktion, Wertanzeigefunktion, Vollständigkeitscheck, Versionszuweisungsfunktion, Anlegecheck für Dateien in einem Ordner, Workflowzuweisungsfunktion.

120012: Dieser Fehler kann auftreten, wenn gesicherte Daten gelöscht wurden oder Benutzer während des Dump-Vorgangs auf dem Server gearbeitet haben.

130007: Dies ist ein fataler Datenbankfehler, der dazu führt, dass der Prozess der betreffenden CMS-Applikation terminiert wird. Die betreffende Datenbank-Aktion wurde nicht ausgeführt.

140021: In NPSOBJ-Anweisungen können nur die Alias-Namen von Tcl-Befehlen verwendet werden. Den Namen müssen im entsprechenden Systemkonfigurationseintrag (beispielsweise `tclSystemExecuteCommands` oder `tclFormatterCommands`) die tatsächlichen Prozeduraufrufe zugewiesen werden.