XML für openUTM

Version 3.0A50

1 Inhaltsverzeichnis

2 Einleitung

2.1 XML in openUTM

Bei der Datenübertragung in verteilten Anwendungen stellt sich häufig das Problem, dass die Information über Struktur und Bedeutung der Daten an mehreren Stellen vorhanden und identisch sein muss: Eine Anwendung erstellt ein Datenpaket in einer bestimmten Form und sendet es an eine entfernte Anwendung, die genaue Kenntnisse über Aufbau und Inhalt haben muss, um die Daten korrekt auswerten zu können. Hier ist die Übertragung der Daten in Form von XML-Dokumenten von großem Vorteil: Informationen über ein Datenelement werden im Dokument mitgeliefert, die Elemente werden über ihre Namen und nicht über ihre genaue Platzierung im Dokument identifiziert. Änderungen in den Daten ziehen so nicht notwendig Änderungen im bearbeitenden Programm beim Empfänger der Daten nach sich. Außerdem sind XML – Dokumente plattformunabhängig, da sämtliche Inhalte in abdruckbarer Form zur Verfügung stehen.

2.2 Was ist XML

Mit XML (eXtensible Markup Language), einer Untermenge von SGML (Standard Generalized Markup Language), steht dem EDV – Anwender eine Metasprache zur Verfügung, die vielfältige Anwendungsmöglichkeiten bietet. In der [XML–Spezifikation] der W3C sind Format, Aufbau und mögliche Inhalte eines XML – Dokumentes festgelegt.
Jeder hat nun die Möglichkeit, z.B. mit Hilfe einer DTD (Document Type Definition) oder eines XML Schemas (siehe Abschnitt 3.10 XML Schema Validierung) eine eigene Sprachsyntax zu definieren, d.h. er kann den Namensraum von Elementen, Attributen und Elementinhalten einschränken und Zusammenhänge beschreiben.
Es gibt Programme (XML-Parser), die ein Dokument validieren können. D.h. ein Parser kann überprüfen, ob es gemäß der XML–Spezifikation ein "wohlgeformtes" (wellformed) XML–Dokument ist, bzw., falls eine DTD oder ein XML Schema vorliegt, ob es gemäß dieser DTD oder des Schemas ein "gültiges" (valid) Dokument ist. Mit einem Parser kann ein solches XML–Dokument in einen Objektbaum gemäß DOM konvertiert und bearbeitet werden. DOM (Document Object Model) ist ein abstraktes API zum Erzeugen, Lesen und Bearbeiten solcher Objektbäume (s. [DOM–Spezifikation]).

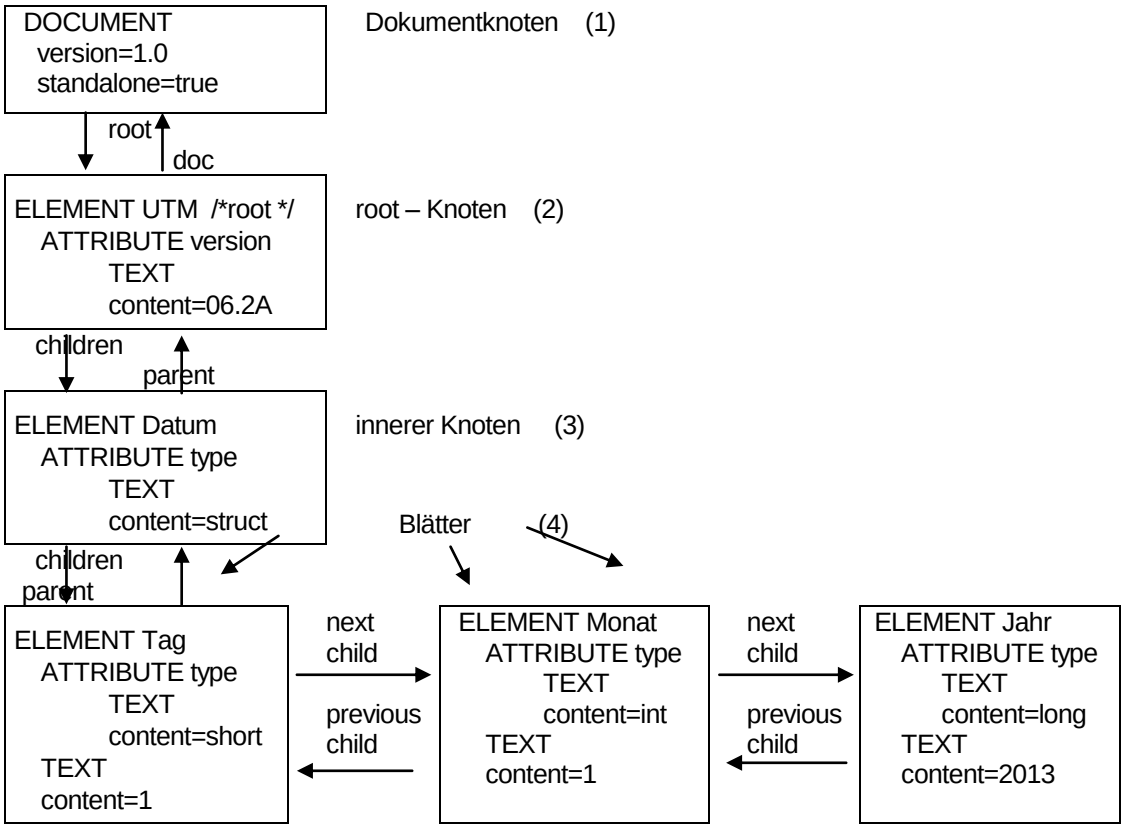
2.3 Begriffserklärungen

Im Folgenden werden einige Begriffe erklärt bzw. festgelegt.
Ein XML-Dokument ist ein Dokument, das gemäß [XML – Spezifikation] wohlgeformt ist. Weitere Begriffe wie Element, tag, Attribut siehe [XML – Spezifikation].
Ein (XML-) Objekt ist ein Objektbaum, der gemäß DOM einem XML-Dokument entspricht und auf dem mittels eines APIs operiert werden kann (s. [DOM – Spezifikation]). Ein XML-Objekt wird i.F. nur Objekt, ein XML-Dokument nur Dokument genannt.
In einem Objekt werden geschachtelte Elemente eines Dokuments in einer baumartigen Struktur hierarchisch angeordnet. Ausgehend von einem Knoten, Wurzel oder root genannt, existieren Kindknoten, children, die wiederum Kinder haben können. Kindknoten mit gleichem übergeordneten Knoten, parent genannt, sind miteinander verkettet und werden Geschwisterknoten oder siblings genannt. Knoten, die keine children haben, werden äußere Knoten oder Blätter genannt, Knoten mit children innere Knoten. Ein Knoten, i.F. auch Element genannt, besteht intern aus mehreren Knoten unterschiedlichen Typs (Elementknoten, Attributknoten, Textknoten).
Das Erzeugen eines Dokumentes aus einem Objekt und umgekehrt eines Objektes aus einem Dokument wird parsen oder konvertieren genannt.
Alle Knoten unterhalb eines inneren Knotens werden mit diesem Knoten zusammen Subobjekt oder Teilbaum genannt.

So entspricht z.B. das Dokument:

```
<?xml version="1.0"?>                          entspricht Knoten (1)
<UTM version="06.2A">                          entspricht Knoten (2)
  <Datum type="struct">                        entspricht Knoten (3)
    <Tag type="short">1</Tag>                  entspricht Knoten (4)
    <Monat type ="int">1</Monat>               entspricht Knoten (4)
    <Jahr type ="long">2013</Jahr>             entspricht Knoten (4)
  </Datum>                                     entspricht Knoten (3)
```

dem folgenden Objekt:



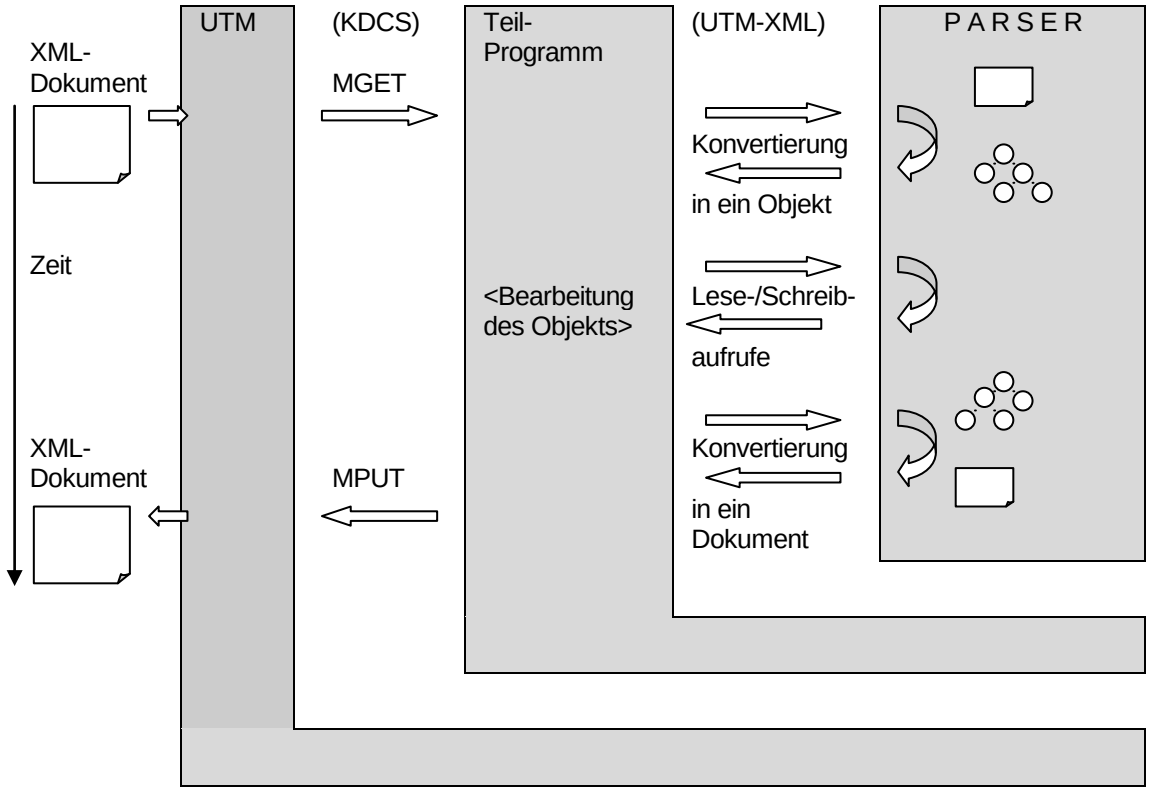
Die Knoten (2) und (3) ohne (4) bilden z.B. keinen vollständigen Teilbaum. Die Knoten (3) und (4) bilden einen Teilbaum des Objekts, der Knoten 'Tag' ist Geschwisterknoten von 'Monat' und 'Jahr'.
Vom Element <Monat type="int">1</Monat> ist

- <Monat type="int"> der Start-tag,
- Monat der tag- oder Element-Name und
- 1 der Wert des Elementes.
- type="int" ist das Attribut mit 'type' als Attributnamen und 'int' als Attributwert und
- </Monat> der Ende-tag.

2.4 Anwendungsmöglichkeiten in openUTM-Umgebung

Im Folgenden wird von Server oder Sender (als dem Erzeuger des XML-Dokuments) und vom Client oder Empfänger (als dem Leser des XML-Dokuments) gesprochen.

In verteilten Anwendungen kann z.B. zum Datenaustausch ein Server ein XML – Dokument, das einer C- oder COBOL- Datenstruktur entspricht, bereitstellen. Er erzeugt das XML-Objekt, wobei zuerst immer ein leeres XML-Objekt angelegt wird. Dann kann das XML-Objekt mittels Schreibfunktionen aufgebaut werden. Am Ende wird eine Funktion aufgerufen, die das XML-Objekt in ein XML-Dokument konvertiert. Dieses kann dann mit den senderspezifischen Kommunikationsfunktionen (KDCS, CPIC, Socket, ...) an den Empfänger gesendet werden.
Der Empfänger liest das ganze Dokument mit den empfängerspezifischen Kommunikationsfunktionen und konvertiert das XML-Dokument in ein XML-Objekt. Anschließend kann er über die tag – Namen (entsprechend den Strukturelementen) die für ihn relevanten Elemente lesen, sie ändern oder neue Elemente erzeugen, die wiederum (in ein XML – Dokument konvertiert, gesendet und wieder in ein XML – Objekt rückkonvertiert) bearbeitet werden können.
Erweiterungen der Strukturen bzw. des Dokuments ändern nicht den Zugriff auf schon bestehende Elemente. So wird z.B. zuerst der Server, der die erweiterten Strukturen bereitstellt, neu übersetzt. Dann können sukzessive die Clients folgen, die die Erweiterungen benötigen. Bei Wegnahme eines Elements würden zuerst die Clients umgestellt und zum Schluss der Server.
Am Beispiel von openUTM als Client sieht das Bearbeiten von XML – Dokumenten dann etwa folgendermaßen aus:



Bei openUTM als Server werden zum Erzeugen eines Dokuments z.B. folgende Aktionen ausgeführt:

- Lesen von relevanten Daten (MGET),
- Erzeugen eines leeren (neuen) Objekts (KXLCreateNewObj),
- Erstellen des Objekts mit Schreibaufrufen (KXLWrite),
- Konvertierung in ein Dokument (KXLConvObjToDoc) und
- Versenden des XML – Dokuments (MPUT)

Weitere Anwendungsmöglichkeiten bietet die Einbettung der Schnittstelle in einem openUTM - Client.

3 Programmschnittstelle UTM-XML

3.1 Neue Funktionen in UTM-XML V3.0

Folgende neue Funktionen werden in UTM-XML V3 angeboten:

XML Schema Unterstützung

Für die Unterstützung der XML Schema Funktionalität bietet UTM-XML folgende neue Funktionen an:

KXLConvDocToObjAndValid zur Konvertierung eines XML-Dokumentes mit Schema-Validierung

KXLParseSchema zum Parsen eines XML Schemas im Speicher

KXLParseSchemaFile zum Parsen einer XML Schema Datei

KXLValidDoc zum Validieren eines XML-Objektes gegen ein Schema

KXLValidDocBuf zum Validieren eines XML-Dokumentes im Speicher gegen ein Schema

KXLFreeSchema zum Freigeben der Schema-Speicherbereiche

KXLSchemaGetRoot zum Erhalt des Root-Knotens des Schema-Objekts (nur in Cobol)

Schnittstellen-Initialisierung:

KXLInitEnv zum Initialisieren der Schnittstellen-Umgebung

Folgende neue Funktion wird in UTM-XML V3.0A40 angeboten:

KXLFindNode zum Lesen eines Elementes unabhängig vom Namensraum

Folgende Funktionalität wird ab UTM-XML V3.0A50 angeboten:

Unterstützung NetCOBOL-Compiler

Cobol-Programme können auf Unix- und Windows-Systemen auch mit dem NetCOBOL-Compiler von Fujitsu übersetzt werden.

Unterstützung Visual COBOL Compiler

Cobol-Programme können auf Unix- und Windows-Systemen auch mit dem Compiler Visual COBOL von Micro Focus übersetzt werden.

3.2 Umfang der Schnittstelle

Das UTM-XML API ist eine Programmierschnittstelle, mit der über Zugriffsfunktionen ein Objektbaum gemäß DOM bearbeitet werden kann. Mit ihr ist es unter anderem möglich, C- und Cobol-Datenstrukturen als XML – Objekte abzulegen und zu bearbeiten. Unter bestimmten Voraussetzungen können auch eigene Elementtypen verwendet werden.

Die Zugriffsfunktionen stehen für C, C++ und Cobol zur Verfügung. Sie sind ablauffähig auf BS2000, UNIX-, Linux- und Windows-Systemen.

Der Parser, auf dem das API basiert, ist der libxml2 Parser [GNOME Parser].

Im Anhang wird die Anwendung der Funktionen anhand von Beispielen gezeigt.

Das API ist in mehrere Bereiche unterteilt:

- Konvertierung der Datentypen in die t_value-Struktur und zurück, (KXLFromXxx, KXLToXxx, wobei Xxx = Short, Int, Long, Float, Double, Char, String, Struct, und KXLFromArray)
- Aufbau eines XML-Objekts, (KXLCreateNewObj, KXLWrite)
- Navigieren im XML-Objekt, (KXLSetSubObject, KXLSetRootNode, KXLSetParentNode)
- Lesen eines XML-Objekts (KXLRead, KXLReadNode, KXLReadNextSib, KXLReadChild, KXLReadNextSingleNode, KXLReadAttr),
- Konvertieren von XML-Objekten in XML-Dokumente und umgekehrt und Freigeben von Objekten (KXLConvDocToObj, KXLConvObjToDoc, KXLFreeObj),
- Zeichensatzbehandlung(KXLGetHomeEnc, KXLGet/SetDocEnc, KXLStringFromUTF8, KXLStringToUTF8),
- Namensraum-Verwaltung (KXLWriteNS, KXLDelNS, KXLReadNSList, KXLSearchNS),
- Validierungsfunktionen (KXLConvDocToObjAndValid, KXLParseSchema, KXLParseSchemaFile, KXLValidDoc, KXLValidDocBuf, KXLFreeSchema)
- Initialisierungs- und Diagnosefunktionen (KXLInitEnv, KXLTSENV, KXLGetLastParserError) und
- weitere Funktionen (KXLGetSizeofNodelist)

3.3 Parameterübergabe

Alle Inhalte der Elemente, die mit dem UTM-XML API im XML-Objekt abgelegt sind, werden als abdruckbare Zeichenfolgen übertragen, d.h. dass z.B. int-Zahlenwerte vor dem Schreiben erst in Zeichenfolgen konvertiert werden müssen (KXLFromInt), nach dem Lesen zur Weiterverarbeitung ggf. rückkonvertiert (KXLToInt). Zum Erhalt der Typ-Information wird jeder Wert als Struktur von zwei Zeigern übergeben, die auf den Typ des Wertes und den Wert als Zeichenfolge verweisen. So wird der Inhalt eines Elements immer in folgender Form dargestellt:

```
struct t_value
        {char*   pType;
         char*   pValue;
        };
```

Dabei enthält pType die Adresse einer Zeichenfolge, in der der Typ des Elementes steht, pValue die Adresse des (abdruckbaren) Elementwerts. So wird z.B. der float-Wert 1.45 als Struktur von zwei Zeigern übergeben, die auf die Strings "float" und "1.45" zeigen.

Da der Zugriff auf Attribute über die gleichen Funktionen erfolgt wie der Zugriff auf Elemente, wird der Inhalt von Attributen über die t_value-Struktur im Feld 'pValue' erwartet bzw. geliefert. Das Feld 'pType' zeigt jeweils auf die leere Zeichenfolge.

3.4 Abbildung der (C-)Datenstrukturen auf XML-Dokumente

Bei der Abbildung einer (C-)Datenstruktur auf ein XML – Dokument bleibt die Hierarchie der Struktur erhalten. So werden übergeordnete Strukturen als umschließende Elemente des XML-Dokuments (bzw. im Objektbaum als parent) dargestellt, untergeordnete Strukturen oder elementare Daten als eingeschlossene Elemente (bzw. im Objektbaum als Kindknoten).

Beispiel:

Die C-Datenstruktur

```
typedef struct Sdatum
{       short   Tag;
        int     Monat;
        long    Jahr; };
Sdatum Datum={1,1,2013}
```


wird mit dem UTM-XML API in einem XML-Dokument wie folgt dargestellt:

```
<Datum type="struct">
      <Tag type="short">1</Tag>
      <Monat type="int">1</Monat>
      <Jahr type="long">2013</Jahr>
</Datum>
```

Der Objektbaum enthält den Knoten mit Namen 'Datum' mit den Kindknoten 'Tag', 'Monat' und 'Jahr'. Die type-Attribute können wegfallen. Dann wird für innere Knoten implizit 'type="struct"' und für äußere Knoten 'type="string"' angenommen.

3.5 Erstellen eines XML-Dokuments/XML-Objekts

Ein Objektbaum entsteht entweder durch das Parsen eines XML-Dokuments (KXLConvDocToObj), oder er wird aufgebaut, um daraus ein XML-Dokument zu generieren. Dazu muss zuerst ein neues Objekt erzeugt (KXLCreateNewObj) und mit KXLWrite die einzelnen Knoten geschrieben werden. Im so entstandenen Objekt kann gelesen, geschrieben und gelöscht werden. Mit Positionieren der Zeiger im Objektbaum (KXLSet...) kann der Zugriff auf Teilbäume optimiert werden. Zum Schluss wird das Objekt mit KXLConvObjToDoc in ein XML – Dokument konvertiert und kann in der Datenhaltung (z.B. Datenbank) abgelegt oder an andere Kommunikationspartner gesendet werden.

Alle Daten, die das XML-Objekt enthalten soll, müssen sukzessive mittels KXLWrite für jeden einzelnen Wert geschrieben werden. Dadurch wird aus dem anfangs leeren XML-Objekt eine aus Elementknoten bestehende baum-artige Struktur, der **Objektbaum**, wobei jede Struktur und jedes array durch einen „inneren“ Knoten repräsentiert wird. Das zugehörige Subobjekt oder auch Teilbaum enthält die Komponenten der Struktur bzw. des arrays. Jedes einfache Datum (vom Typ short, int, long, float, double, char) und jede Zeichenfolge (string) wird durch einen einfachen Knoten oder auch Blatt repräsentiert. Der volle Name eines jeden Elementes setzt sich aus den Namen aller Knoten zusammen, die zwischen dem root-Knoten und dem jeweiligen Knoten im Objektbaum liegen.

Beim Zugriff auf einen Knoten des XML-Objekts ist ein Zeiger auf einen Knoten anzugeben, der nicht notwendigerweise der root-Knoten des XML-Objektes sein muss. Durch Positionieren mit KXLSetSubObject oder beim Schreiben eines Knoten vom Typ `struct` oder `array` erhält man die Adresse eines Knoten innerhalb des XML-Objekts, von dem aus ein Unterbaum erzeugt oder gelesen werden kann, ohne dass bei jedem Aufruf der volle Name eines Elements angegeben und der ganze Namenspfad durchlaufen werden muss.

3.6 Zeichenvorrat

Der erlaubte Zeichenvorrat für die Element- und Attributnamen und deren Inhalte ist in der [XML-Spezifikation] beschrieben. Es folgt hier eine sehr vereinfachte Beschreibung, die keinen Anspruch auf Vollständigkeit und Richtigkeit erhebt. Im Zweifelsfall ist immer die [XML-Spezifikation] maßgeblich.

1. Attribut- und tag- Namen dürfen Buchstaben, Ziffern, ".", "-", "_" und ":" enthalten, dürfen aber nicht mit einer Ziffer, "." oder "-" beginnen.

2. Der Wert eines Attributes darf, wenn er in doppelte Hochkommata (") eingeschlossen ist, alle Zeichen außer "%", "&", """ (doppeltes Hochkomma) und keine Verweise der Art "Referenz" (s. unter 3.) enthalten, wenn er in einfache Hochkommata (') eingeschlossen ist, alle Zeichen außer "%", "&", "'" (einfaches Hochkomma) und keine Verweise der Art "Referenz" (s. unter 3.) enthalten.

3. Der Inhalt eines Elementes besteht aus einer beliebigen Mischung aus
   - Elementen
   - Referenzen wie &name; oder &#nn; oder &#xnn;, wobei 'name' wie unter 1. aufgebaut und 'nn' eine Ziffern-Buchstabenfolge ist.
   - CDSects wie <![CDATA[charData]]>
   - PI's wie <? PI-Ziel charData ?>
   - Kommentaren wie <!- - charData - - >
   - charData die alle Zeichen außer "<", "&" und der jeweiligen Abschlusszeichenfolge der Einheit, wie]]>, ?> oder --> enthalten darf

4. Für die 'verbotenen' Zeichen können vordefinierte Ersatzzeichenfolgen angegeben werden:
   - "<" statt "<"
   - ">" statt ">"
   - "&" statt "&"
   - "'" statt "'" (einfaches Hochkomma)
   - """ statt """ (doppeltes Hochkomma)

3.7 Namensräume

Um in einem XML-Dokument mehrere XML-Dokumente aus unterschiedlichen Namensräumen verwenden zu können, muss gewährleistet sein, dass ein Element- oder Attributname eindeutig einem der enthaltenen Dokumente mit seinem Namensraum zugeordnet werden kann. Damit kann ein Name eindeutig interpretiert werden, auch wenn er in mehreren Spezifikationen definiert wird. Um diese Eindeutigkeit zu erhalten, wurde der Begriff des Namensraumes eingeführt (siehe [NS – Spezifikation]). Hier werden kurz die wesentlichen Merkmale der Verwendung von Namensräumen aufgeführt:

1. Eine Namensraum-Definition besteht aus einem Präfix und einer URL, unter der die Spezifikation des Namensraum zu finden ist (wird vom UTM-XML API nicht ausgewertet).
2. Jede Definition wird einem Elementknoten des XML-Objektes zugeordnet.
3. Im XML-Dokument werden Namensräume über die Attribute xmlns="<url>" (Default-Namensraum) bzw. xmlns:<prefix>="<url>" definiert, z.B.:
   <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:xsd1="http://localhost:8080/wsdl/mynamespace.xsd"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap" >
   wobei mit "xmlns" der Default-Namensraum definiert wird.
4. Damit bestimmt sich sein Gültigkeitsbereich: Der Namensraum ist für den zugeordneten Elementknoten und alle Element- und Attributknoten des zugehörigen Teilbaumes gültig, wenn dort nicht untergeordnet ein Namensraum mit demselben Präfix definiert ist.
5. Ist einem Element- oder Attributnamen ein Präfix, getrennt mit ':' vorangestellt, wird der tag-Name dem zugehörigen Namensraum zugeordnet.
6. Eine Namensraum-Definition mit leerem Präfix nennt man einen **Default-Namensraum**. Mit einer solchen Definition wird jeder Elementname (gilt nicht für Attribute!) ohne Präfix im Gültigkeitsbereich dem Default-Namensraum zugeordnet.

Folgende Besonderheiten sind abhängig von der Arbeitsweise des verwendeten Parsers:

1. Enthält ein XML-Objekt ein Element mit einem an dieser Stelle nicht definierten Präfix, so wird vom Parser intern eine Dummy-Namensraum-Definition erzeugt.
2. Dummy-Namensraum-Definitionen werden beim Aufruf von KXLReadNSList mit erfasst, können aber nicht mit KXLSearchNS gelesen werden.
3. Attribute mit nicht definiertem Präfix werden nicht akzeptiert. Beim Konvertieren bzw. Schreiben wird ein Parser Returncode zurückgegeben.

3.8 Aufbau der Element- und Attributnamen und Nodelists

Der Name eines Elementes ist eine beliebige mit `\0` terminierte abdruckbare Zeichenfolge. Sie muss den Vorgaben der [XML-Spezifikation] entsprechen (s. auch Abschnitt Zeichenvorrat) und darf zusätzlich die Zeichen "/", "[" und "]" enthalten.
Enthält ein Name ein "/", wird es als Trennzeichen für die Strukturkomponenten interpretiert, die das Datum enthalten. So können durch Angabe von "/" Komponenten einer Struktur übergeben werden: Durch den Namen "Rechnung/Adresse/Name" wird ein Element „Name“ in der Struktur "Adresse" innerhalb der Struktur "Rechnung" angesprochen. Die in `t_value` übergebenen Typ und Wert beziehen sich auf das Element "Name". Falls beim Schreiben des Knotens noch kein Knoten "Rechnung" vom Typ `struct` existiert, wird dieser implizit angelegt (ohne Inhalt). Falls noch kein Knoten "Adresse" vom Typ `struct` unter dem Knoten "Rechnung" (als direkter Kindknoten) existiert, wird dieser ebenfalls implizit angelegt (ohne Inhalt). Mehrere Elemente in einer Struktur werden in der Reihenfolge, in der sie erzeugt werden, abgelegt (keine Sortierung).
Wird ein Knoten vom Typ `struct` oder `array` explizit erzeugt, können zusätzliche Information, etwa der Typname der Struktur, als Inhalt des Knotens abgelegt werden.

Enthält ein Name ein "[", so wird es als Indexoperator bei arrays interpretiert. Der Namensteil nach "[" sollte eine (abdruckbare) positive ganze Zahl ohne führende Nullen sein (Index des Feldelementes, beginnend mit 0, max. 98 Stellen lang) und mit "]" abgeschlossen sein. So können einzelne Feldelemente angesprochen werden. Das (i+1)-te Element eines `array` wird z.B. durch Angabe von "Artikel[i]" geschrieben. Ist noch kein Knoten mit Namen "Artikel" und Typ `array` angelegt, so wird dieser implizit erzeugt. Die Sortierung von Elementen mit nicht rein numerischen Indizes oder mit führenden Nullen wird im Anhang, Abschnitt 'Bearbeitung externer Elemente' beschrieben.
Wenn der übergebene Zeiger auf den array-Knoten zeigt, wird beim Zugriff auf ein Feldelement als Name nur der Index in der Form "[i]" angegeben, wenn das (i+1)-te Feldelement geschrieben werden soll. Wird "[ ]", also kein Index angegeben, dann wird das erste freie Feldelement geschrieben. Die Elemente in einem Feld sind dem Index nach aufsteigend angeordnet
Ist ein Feldelement selber wieder eine Struktur, werden die Namen der Strukturelemente vom Index zusätzlich durch "/" getrennt, z.B.: Rechnung/Artikel[2]/Bestell-Nr.
Es muss immer der Namensteil angegeben werden, der ab dem angegebenen Knoten relevant ist. Soll z.B. ein Element mit Namen "Rechnung/Artikel[2]/Bestell-Nr." gelesen werden, und der Zeiger auf den Knoten "Rechnung/Artikel[2]" wird beim Aufruf angegeben, so wird als Name nur "Bestell-Nr." angegeben.

Enthält ein Namensteil ein ':', wird der vorhergehende Namensteil als Namensraum-Präfix interpretiert (s. Abschnitt Namensräume). Namensteile ohne Präfix werden dem Default-Namensraum zugeordnet, wenn einer definiert ist. Namensteile mit unterschiedlichen Präfixen werden als verschieden erkannt, auch wenn die zugehörigen url's gleich sind.

Indizes von arrays können nicht mit Namensraum-Präfixen versehen werden.

Es ist möglich, mehrere Elemente mit gleichem Namen unter demselben parent-Knoten zu haben. Sie werden in diesem API als **nodelist (Knotenliste)** bezeichnet. Um auf die einzelnen Elemente einer solchen Liste zugreifen zu können, kann als **Nodelist-Position(ierung)** einer der folgenden Ausdrücke angegeben werden: [+], [-], [++], [--] .
Damit wird relativ zum angegebenen Knoten unter dessen Geschwisterknoten der nächste ([+]), der vorhergehende ([-]), der letzte ([++]) oder der erste ([--]) Knoten mit gleichem Namen gesucht.
Beginnt ein Name mit einer Nodelist-Positionierung, so wird als Name der Nodelist der des angegebenen Knotens genommen.

Ist in einem Aufruf ein komplexer Element-Name angegeben, so darf darin beliebig oft eine Nodelist-Positionierung enthalten sein, allerdings nicht mehrmals direkt hintereinander.
(Hinweis: Das gilt auch für die Kombination von Array-Elementangaben und nodelist-Positionen: mehrere eckige Klammern direkt hintereinander sind nicht erlaubt.)
Die Nodelist-Positionierung bezieht sich immer auf den direkt davor liegenden Namensteil. Als letzter Namensteil von komplexen Namen ist bei Schreibaufrufen nur [+] erlaubt,
bei Lese- und Positionierungsaufrufen alle Positionierungen außer [-], wobei [--] an dieser Stelle keine Bedeutung hat.

So wie auf Elemente, deren Namen und Inhalt zugegriffen werden kann, ist das auch für Attribute möglich. Ein bestimmtes Attribut eines Elementes wird angesprochen über den Elementnamen, dem mit '/@' der Attributname angehängt wird. So ist der volle Name des Attributes 'Zweck' des Elementes 'Adresse' innerhalb der Struktur 'Bestellung': 'Bestellung/Adresse/@Zweck' (z.B. mit Inhalt 'Lieferung' oder 'Rechnung').
Mit diesem Namensaufbau können Attribute genauso wie Elemente geschrieben und gelesen werden. Attribute sind immer 'äußere Knoten', es darf also nur der letzte Namensteil mit '@' beginnen. Positionieren auf Attribute ist zwar möglich, allerdings nur beim Lesen mit lokalen Namen (KXLFindNode) sinnvoll. Steht innerhalb eines Namensteils ein '@', wird das vom Parser erst beim Parsen (Neu Erzeugen eines Objektes aus dem Dokument) als Fehler erkannt.

3.9 Zeichensätze/Encoding

Die Bearbeitung eines XML – Objektes erfolgt intern in UTF-8 (eine Ausprägung des Unicode-Zeichensatzes) [UTF8]. D.h., dass beim Aufbau eines XML-Objektes jedes Dokument in UTF-8-Code konvertiert wird. Wird in einem Dokument ein anderer Zeichensatz verwendet, muss eine gültige Encoding-Angabe im Dokumentenkopf enthalten sein. Ohne Encoding-Angabe wird UTF-8-Code angenommen.

Beim Schreiben und Lesen von Elementen werden Namen und Inhalte, auch bei Attributen, PI's, usw. automatisch vom Zeichensatz, der auf dem lokalen Rechner verwendet wird (i.F. Home Encoding genannt), nach UTF-8 bzw. umgekehrt konvertiert, so dass an der Benutzerschnittstelle keine Codekonvertierung erfolgen muss.
Beim Erzeugen eines Dokuments aus einem XML-Objekt wird dieses aus UTF-8 in den unter encoding angegebenen Zeichensatz konvertiert. Ist keiner angegeben, geht die Schnittstelle von UTF-8 Zeichen aus und konvertiert nicht.
Beim Neuanlegen eines XML-Objekts wird als Zeichensatz der im Home Encoding angegebene eingetragen. Lesen bzw. Ändern kann man diese Angabe über KXLGetDocEnc bzw. KXLSetDocEnc.

Der auf dem lokalen Rechner verwendete Zeichensatz ist im Headerfile libxml/kxlinc.h unter _KXL_HOME_ENC_STR (abdruckbare Namen, s. Tabelle) und _KXL_HOME_UTF8_STR (Name in UTF-8) deklariert. Standardmäßig ist EBCDIC für BS2000 und UTF-8 für alle anderen Plattformen eingestellt. Diese Voreinstellung kann mit KXLGetHomeEnc gelesen werden. Beim ersten Aufruf von KXLInitEnv kann ein anwenderspezifisches Home Encoding angegeben werden.
Die wichtigsten Code-Konvertierungsroutinen sind im Parser bzw. in UTM-XML deklariert. Die folgende Liste gibt an, welche Routinen vorhanden sind und mit welchen Encoding-Angaben sie angesprochen werden können:

Encoding-Namen	Encoding
UTF8, UTF-8	UTF-8
UTF-16BE	UTF16BigEndian
UTF-16LE	UTF16LittleEndian
ASCII	ASCII
ISO-8859-1, …, ISO-8859-16	ISO-8859-1, …, ISO-8859-16
CP850	Codepage 850 (DOS-Latin-1)
EDF03DRV	BS2000 EBCDIC DF03 (deutsche Version) mit Euro-Zeichen (0x9F)
EDF03IRV	BS2000 EBCDIC DF03 (internationale Version) mit Euro-Zeichen (0x9F)
EDF04DRV	BS2000 EBCDIC DF04 (deutsche Referenz-Version)
EDF04_01	BS2000 EBCDIC DF041

Außerdem sind Aliasnamen vordefiniert. Mit ihnen können bestehende Encoding-Handler-Routinen über einen anderen Namen angesprochen werden. Ihre Zuordnung kann mit der Funktion **KXLGetEncodingAlias** gelesen und mit der Funktion **KXLSetEncodingAlias** geändert oder gelöscht werden. Folgende Aliasnamen sind vorhanden:

Alias-Name	Original Encoding-Name
EBCDIC	EDF04DRV
OSD_EBCDIC_DF04_DRV	EDF04DRV
OSD_EBCDIC_DF04_01	EDF04_01
EDF041	EDF04_01

Falls Sie mit anderen Zeichensätzen arbeiten wollen, können Sie eigene Konvertierungsroutinen mit der Parserfunktion xmlNewCharEncodingHandler deklarieren. Siehe dazu [libxml internationalization support]. Diese Aufrufe sind z.B. an der gekennzeichneten Stelle in der Funktion KXLInitEncHdlr möglich. Außerdem besteht die Möglichkeit, die Umsetztabellen der EBCDIC Encodingfunktionen (EDFnnxxx, nn=03, 04) direkt im Sourcecode zu ändern und die geänderten statt der Standard-Tabellen einzubinden. Siehe dazu Kapitel 7 "Einsatz" und im Anhang "Nutzung benutzerspezifischer Encoding-Funktionen"

Folgende Übersicht zeigt die Übergänge zwischen den verschiedenen Zeichensätzen. Dabei bedeutet:
H-Enc = Home Encoding = Zeichensatz der lokalen Plattform
D-Enc = document encoding = Zeichensatz des vorliegenden Dokuments
UTF-8 = internal encoding = Interner vom Parser verwendeter Zeichensatz

API – Input / - Output	UTM-XML-Funktion	Parser
XML-Dokument (D-Enc)	→ KXLConvDocToObj →	XML – Objekt (UTF-8)
XML-Dokument (D-Enc)	← KXLConvObjToDoc ←	XML – Objekt (UTF-8)
root-name /-content (H-Enc)	→ KXLCreateNewObj →	XML – Objekt (UTF-8)
element-name/- content (H-Enc)	→ KXLWrite →	node im XML-Objekt (UTF-8)
element-name (H-Enc)	→ KXLRead →	node im XML-Objekt (UTF-8)
element-content (H-Enc)	←(return)	←
element-name/-content (H-Enc)	← KXLReadXxx ←	node im XML-Objekt (UTF-8)
element-content (H-Enc)	←(return)	←
element-name (H-Enc)	→ KXLSetSubObject →	node im XML-Objekt (UTF-8)

In der Regel wird H-Enc = D-Enc (z.B. = EBCDIC im BS2000) sein, man kann aber auch z.B. EBCDIC-Dokumente auf Windows- oder UNIX-Systemen bearbeiten.

Dabei ist aber zu beachten, dass die üblichen Transfertools Textdokumente automatisch konvertieren. Das heißt z.B., dass ein mit FTP vom BS2000 nach Windows übertragenes Dokument zwar das Attribut "encoding='EBCDIC'" enthält, auf Windows aber z.B. in ISO-8859-1 verfügbar ist.

Es ist ratsam, XML-Dokumente, die plattformübergreifend bearbeitet werden, mit dem encoding-Attribut encoding="UTF-8" zu erstellen und ohne Codeumsetzung zu transferieren.
Eine weitere Möglichkeit ist im Anhang im Abschnitt " Bearbeitung von Dokumenten ohne encoding-Attribut" beschrieben.

Beispiel zur Deklaration von Konvertierungsroutinen zu einem privaten Zeichensatz:

```
#include <libxml/encoding.h>
/* input for parser function in UTF-8! */
const static char UTF8_EDF031string[7]=  {69,68,70,48,51,49,0};
/* ASCII for "EDF031\0" */
/* conversion routines EDF031ToUTF8 and UTF8ToEDF031 must be written by user
*/
int   EDF031ToUTF8 (unsigned char* out, int *outlen, const unsigned char*
in, int *inlen);
int   UTF8ToEDF031 (unsigned char* out, int *outlen, const unsigned char*
in, int *inlen);
xmlCharEncodingHandlerPtr EncHdlr;

EncHdlr = xmlNewCharEncodingHandler ( UTF8_EDF031string, EDF031ToUTF8,
UTF8ToEDF031);
if ( EncHdlr == NULL )
{ /* declaration failure -> error handling */ }
```

3.10 XML Schema Validierung

XML Schema ist eine Empfehlung des W3C zum Definieren von XML-Dokumentstrukturen. (siehe [XML-Schema Spezifikation]) Anders als bei den klassischen XML DTDs wird die Struktur in Form eines XML-Dokuments beschrieben. Darüber hinaus wird eine große Anzahl von Datentypen unterstützt. Das Parsen eines auf einem XML-Schema basierenden XML-Dokuments beinhaltet daher stets eine Prüfung auf strukturelle Gültigkeit und die Prüfung der korrekten Typverwendung. [XML-Schema – Definition]

Mit den Funktionen zur Schema Validierung (siehe Kapitel UTM-XML API für C/C++, Abschnitt XML Schema Validierung) unterstützt das UTM-XML API das Parsen von XML-Schemata und die Validierung von XML-Dokumenten gegen ein XML-Schema.

3.11 Dateibehandlung

Mit den Funktionen zur Schema Validierung wird nicht nur auf Daten im Speicher, sondern auch auf Dateien zugegriffen. Beim Parsen eines Schemas kann z.B. eine Datei als Parameter angegeben werden. Außerdem können Teile des Schemas in weiteren Dateien stehen, die über include-, import- oder andere Schema-Elemente referenziert werden. Die Dateien werden über ihre URL identifiziert. Um den Dateizugriff auf den unterschiedlichen Plattformen einheitlich zu gestalten bzw. überhaupt erst zu ermöglichen, werden in UTM-XML mit Hilfe von Parserfunktionen zwei Handler deklariert, die den Dateizugriff steuern. Außerdem ist es für den Anwender möglich, selbst solche Handler-Routinen zu deklarieren.
Achtung: Mit dem Einsatz von anwenderspezifischen Handler-Routinen greifen Sie in den hier beschriebenen Ablauf der UTM-XML-Funktionen ein. D.h. die Dateibehandlung erfolgt nicht mehr so wie in den Schnittstellenfunktionen beschrieben.

3.11.1 Entities Loader

Bei der Initialisierung von UTM-XML (Funktion KXLInitEnv) wird eine schnittstellenspezifische Handler-Routine KXLResolveEntity deklariert, die folgende Funktionalität bietet:

Die allgemeine Struktur einer URL vom Typ http oder ftp ist folgendermaßen:

<URL-Typ>://[<Benutzer>[:<Passwort>]@]<Server>[:<Port>]/<Pfad>?<Anfrage>#<Fragment>

wobei bei Angabe von Dateinamen der letzte Teil (?<Anfrage>#<Fragment>) wegfällt und <Pfad> weiter in <directory>/<filename> unterteilt werden kann.

In den Funktionen KXLConvDocToObjAndValid , KXLParseSchema und KXLParseSchemaFile kann mit dem Parameter pDir ein lokales Dateiverzeichnis angegeben werden, so dass Dateien, die im XML- oder Schema-Dokument referenziert sind, statt aus dem Netz aus diesem lokalen Dateiverzeichnis gelesen werden können.

Im BS2000 kann bei pDir eine Benutzerkennung (mit '$', ohne '.', z.B. "$UTMXML" angegeben werden. Sollen die Dateien aus der aktuellen Benutzerkennung referenziert werden, wird bei pDir die leere bzw. nur aus Leerzeichen bestehende Zeichenfolge angegeben.

Dabei wird eine URL mit dem UTM-XML-spezifischen Entity-Resolver folgendermaßen umgesetzt:

Ist <URL-Typ> = "http" oder "ftp" , wird der Namensteil

<URL-Typ>://[<Benutzer>[:<Passwort>]@]<Server>[:<Port>]/<directory> durch das lokale Directory pDir=<mydir> ersetzt, abgeschlossen durch das betriebssystemspezifische Datei-Trennzeichen (/ bzw. \ bzw. .) (bzw. leere Zeichenfolge, wenn <mydir> die leere Zeichenfolge ist).

Ist <URL-Typ> = "file", wird der Namensteil "file://" am Anfang gelöscht. Die Angabe des lokalen Directorys wird ignoriert. Andere Dateinamen werden nicht umgesetzt.

Beispiel:. "http://www.w3.org/2001/XMLSchema" wird umgesetzt in "<mydir>/XMLSchema" . Sollen die Dateien im lokalen Verzeichnis gelesen werden, wird <mydir> = "" angegeben, d.h. die URL wird in "XMLSchema" umgesetzt.

Es ist für den Anwender möglich, eine eigene Routine für den oben genannten Handler gemäß der Beschreibung des Parsers [IO Interfaces] einzuhängen. Hierbei ist Folgendes zu beachten:

- Mit der Parserfunktion xmlSetExternalEntityLoader kann eine eigene ResolveEntity-Handler- Routine deklariert werden, die eine gegebene URL durch einen anwenderspezifischen Dateinamen ersetzt. Die Deklaration eines anwenderspezifischen ResolveEntity-Handlers muss nach der Initialisierung der UTM-XML-Schnittstelle (KXLInitEnv) erfolgen.
- Wenn Sie beim Parser einen eigenen ResolveEntity-Handler deklarieren, ist dadurch das beschriebene Verhalten mit Lesen der Dateien aus dem beim Parameter pDir angegebenen lokalen Dateiverzeichnis ausgeschaltet.
- Sie können den ResolveEntity-Handler wieder auf den Standard von UTM-XML setzen, wenn Sie nach der Deklaration eines eigenen ResolveEntity-Handlers erneut die Funktion KXLInitEnv aufrufen.

3.11.2 IO-Handler im BS2000

Da die Parserfunktionen im BS2000 intern mit ASCII-Zeichenfolgen arbeiten, alle anderen Funktionen, wie Dateibehandlungsaufrufe (open, read), EBCDIC-Zeichenfolgen erwarten, ist es notwendig, einen UTM-XML-eigenen IO-Handler zu deklarieren. Er besteht aus vier Funktionen:

- vom Typ xmlInputMatchCallback, die prüft, ob der IO-Handler diese Datei bearbeiten soll
- vom Typ xmlInputOpenCallback, die die Datei mit dem angegebenen Dateinamen öffnet
- vom Typ xmlInputReadCallback, die die Datei einliest, und
- vom Typ xmlInputCloseCallback, die die Datei schließt.

Diese Funktionen werden vom Parser aufgerufen, wenn eine Datei eingelesen werden soll.

In der Initialisierung von UTM-XML (Funktion KXLInitEnv) wird der schnittstellenspezifische IO-Handler mit den Funktionen KXLIOHandler<func> mit <func> = Match/Open/Read/Close deklariert, der folgende Funktionalität bietet:

Beginnt der übergebene Dateiname mit "file://" oder enthält er keine URL-Typ-Angabe, gibt die Match-Routine 1 zurück (d.h. die Datei wird behandelt). Die Funktionen Open/Read/Close öffnen, lesen bzw. schließen die Datei unter Berücksichtigung der Encoding-Problematik im BS2000.

Es ist für den Anwender zusätzlich (bzw. statt dessen) möglich, auf eigenes Risiko eigene Routinen für die oben genannten Handler gemäß der Beschreibung des Parsers einzuhängen. Hierbei ist Folgendes zu beachten:

- Wenn Sie beim Parser einen eigenen IO-Handler im BS2000 deklarieren (mit xmlRegisterInputCallbacks), ist die Deklarationsreihenfolge wichtig, da bei der Bearbeitung einer URL alle IO-Handler-Funktionen vom Typ xmlInputMatchCallback aufgerufen werden, bis zum ersten positiven Return. D.h. ein anwenderspezifischer IO-Handler muss vor der Initialisierung des IO-Handlers von UTM-XML deklariert werden.
- Sie können den IO-Handler wieder auf den Standard von UTM-XML setzen, wenn Sie nach der Deklaration eines eigenen IO-Handlers erneut die Funktion KXLInitEnv aufrufen.
- Achtung: Der Parser legt eine Tabelle der deklarierten IO-Handler an, die rekursiv abgearbeitet wird. D.h. falls ein IO-Handler die Behandlung zurückweist (return 0 von IO-Handler-Match-Routine), wird der vorher deklarierte IO-Handler aufgerufen. Diese Tabelle ist begrenzt, d.h. es sind nur max. 15 Deklarationsaufrufe möglich.
- Die Funktion xmlRegisterInputCallback und die Funktionstypen xmlInput<func>Callback mit <func> = Match/Open/Read/Close sind im Parser-Headerfile xmlIO.h deklariert.

3.12 Anmerkungen

1. Alle (externen) Funktionen und Prozeduren beginnen mit dem Präfix KXL.
2. Alle Funktionen, die mit dem Präfix xml beginnen, sind Parserfunktionen. Sie stehen nur in C zur Verfügung und sind in diesem Papier nicht weiter beschrieben. Parserfunktionen erwarten Input in UTF-8-Code, auch im BS2000.
3. Im BS2000 fällt die Längenbeschränkung der externen Namen auf 7 bzw. 8 Zeichen weg. Dementsprechend müssen C-Teilprogramme, die Funktionen dieses APIs aufrufen, als LLM übersetzt und gebunden werden.

4 UTM-XML API für C / C++

Das API ist in folgende neun Gruppen unterteilt:

1. zur Konvertierung der Datentypen in die t_value-Struktur und zurück,
2. zum Aufbau eines XML-Objekts,
3. zum Navigieren im XML-Objekt,
4. zum Lesen eines XML-Objekts,
5. zum Konvertieren von XML-Objekt in XML-Dokument und umgekehrt und zum Freigeben von XML-Dokumenten,
6. zur Behandlung von Zeichensätzen,
7. zur Verwaltung von Namensräumen,
8. zum Validieren gegen XML Schema-Dateien und
9. zur Initialisierung und zur Diagnose

Zum Aufruf der Schnittstelle ist das Headerfile `libxml/kxlinc.h` im Programm anzugeben. Es enthält alle notwendigen Definitionen und Funktionsprototypen.

Der Returncode ist bei allen Funktionen, bei denen er definiert ist, optional. D.h., statt dem Zeiger auf einen Returncode kann der NULL-Zeiger übergeben werden. Dann wird kein Returncode zurückgegeben.

4.1 Initialisierung der UTM-XML-Schnittstelle

Funktionsprototyp:

```
void    KXLInitEnv(char * encFuncName,
                   xmlCharEncodingInputFunc encFuncToUTF8,
                   xmlCharEncodingOutputFunc encFuncFromUTF8,
                   short* pRetCode);
```

Parameter:

Typ	Name	Bemerkung
char *	encFuncName	In: Name des Home Encodings
xmlCharEncodingInputFunc	encFuncToUTF8	In: Encodingfunktion nach UTF-8
xmlCharEncodingOutputFunc	encFuncFromUTF8	In: Encodingfunktion von UTF-8
short*	pRetCode	Out:Returncode, wenn nicht NULL angegeben wurde

Um die Umgebung für die UTM-XML-Schnittstelle einzurichten, müssen verschiedene Funktionen des Parsers für UTM-XML und die schnittstellenspezifische Traceumgebung initialisiert werden. Mit Aufruf dieser Funktion wird die Traceumgebung initialisiert (Aufruf von KXLTSENV), und verschiedene Parser-spezifische Handler (Encoding-Handler, Entity-Resolver, IO-Handler (nur BS2000)) und ggf. ein anwenderspezifisches Home Encoding deklariert. Außerdem erfolgt ein Versions- und Konsistenz-Check.

Wird encFuncName angegeben, wird ein anwenderspezifisches Home Encoding definiert. Ist das Encoding bereits an der Schnittstelle bekannt (s. Abschnitt Zeichensätze/Encoding), kann für encFuncToUTF8 und encFuncFromUTF8 NULL angegeben werden. Dann werden die vordefinierten Encoding-Funktionen verwendet. Ist das Encoding nicht bekannt oder sollen eigene Encodingfunktionen verwendet werden, müssen hier die Adressen von zwei Funktionen angegeben werden, die folgender Deklaration entsprechen:

```
int homeEncodingToUTF8 (unsigned char* out, int *outlen,
                        const unsigned char* in, int *inlen);
int UTF8ToHomeEncoding (unsigned char* out, int *outlen,
                        const unsigned char* in, int *inlen);
```


Dabei haben die Parameter folgende Bedeutung:
out Adresse des Ausgabepuffers, in den die konvertierte Zeichenfolge geschrieben wird
outlen in: Länge des Ausgabepuffers, out: Länge der konvertierten Ausgabe-Zeichenfolge
in Adresse des Eingabepuffers, der die zu konvertierende Zeichenfolge enthält
inlen in: Länge des Eingabepuffers (Anzahl Bytes), out: Länge der verarbeiteten Zeichenfolge
Rückgabewert (int) = n, Länge der konvertierten Ausgabezeichenfolge,
-2, wenn beim Konvertieren ein Fehler erkannt wurde,
-1, bei anderen Fehlern.

Wird encFuncName nicht angegeben (NULL), wird kein anwenderspezifisches Home Encoding definiert.

Anmerkungen:

1. Diese Funktion muss nur dann explizit aufgerufen werden, wenn ggf. vor Aufruf von UTM-XML-Funktionen direkt Funktionen des Parsers aufgerufen werden, die die oben genannten Handler verwenden oder wenn ein anwenderspezifisches Home Encoding definiert werden soll. Sie muss dann vor allen anderen UTM-XML-Funktionen **und in jedem Task/Prozess** aufgerufen werden. Bei Verwendung der UTM-XML-Schnittstelle in einer UTM-Anwendung muss dieser Aufruf task- bzw. prozessspezifisch, z.B. in einem Start-Exit erfolgen.
2. Beim ersten Aufruf einer UTM-XML Funktion, die xmlNodePtr, xmlNsPtr oder xmlSchemaPtr zurückgibt, bzw. Funktionen zur Zeichenfolgenkonvertierung (d.h. bei KXLCreateNewObj, KXLConvDocToObj, KXLConvDocToObjAndValid, KXLParseSchema, KXLParseSchemaFile, KXLStringToUTF8, KXLStringFromUTF8, KXLGetEncodingAlias, KXLSetEncodingAlias, nicht bei den Konvertierungsfunktionen der t_value-Strukturen, den Konvertierungsfunktionen von Zeichenfolgen nach/von UTF-8 und bei den Aufrufen von KXLGetHomeEnc und KXLTSENV) wird implizit die Funktion KXLInitEnv aufgerufen. Dabei wird kein Encoding angegeben (KXLInitEnv (NULL, NULL, NULL, &retCode)). Bei Fehler wird ein entsprechender Returncode zurückgegeben.
3. Es ist für den Anwender zusätzlich möglich, eigene Routinen für die oben genannten Handler gemäß der Beschreibung des Parsers einzuhängen. (siehe Abschnitt 3.11 Dateibehandlung) Rufen Sie danach erneut explizit diese Funktion auf, werden der EntityResolver und IO-Handler (im BS2000) wieder auf die Default-Routinen von UTM-XML gesetzt. (siehe auch Abschnitt 3.11 Dateibehandlung)
4. Die Konsistenz-Checks und die Einstellung des Home-Encodings werden nur beim ersten Aufruf der Funktion (pro Task/Prozess) durchgeführt.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_ALLOC_ERROR (11)	S	mit malloc/realloc konnte kein zusätzlicher Speicherplatz angefordert werden
KXL_RC_VERSION_ERR (23)	A	Versionsstände der UTM-XML-Komponenten inkonsistent
KXL_RC_PARSER_VERS_NOT_ SUPP (24)	A	alte Version des XML-Parsers eingebunden (< 2.6.20) -> neuere Parser-Version einbinden
KXL_RC_ENC_NAME_ERR (27)	A	Encoding Name zu lang (> 31 Zeichen)
KXL_RC_UNKNOWN_ ENCODING (30)	A	Encoding-Name ist dem Parser nicht bekannt -> zum Encoding gehörige Konvertierungsroutinen beim Parser deklarieren
KXL_RC_ENCODING_CHANGE_ ERR (36)	A/S	für das angegebene Home Encoding konnte kein Encoding Handler eingerichtet werden
KXL_RC_IO_HNDLR_INIT_ERR (45)	A	zu viele IO-Handler definiert

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

4.2 Typ-Konvertierungsfunktionen

Um dem Anwender eine komfortable Bedienung der Schnittstelle zu ermöglichen, werden Konvertierungsfunktionen für bestimmte Datentypen bereitgestellt. Dadurch, dass die Konvertierung vor dem schreibenden bzw. nach dem lesenden Schnittstellenaufruf stattfindet, bietet sich dem Anwender die Möglichkeit, andere Konvertierungsroutinen, auch für eigene Datentypen, zu erstellen und diese Daten an der Schnittstelle zu übergeben (siehe auch Kapitel "Bearbeitung externer Dokumente").

Die C-Datentypen `short, int, long, float, double und char` werden im Folgenden als **einfache Datentypen** bezeichnet, die Datentypen `string` (Zeichenfolge) und struct (Struktur) als **komplexe Datentypen**.

4.2.1 Konvertierung in die `t_value` – Struktur

Einfache Datentypen

Funktionsprototypen:

```
t_value KXLFromShort(short s);
t_value KXLFromInt(int i);
t_value KXLFromLong(long l);
t_value KXLFromFloat(float f);
t_value KXLFromDouble(double d);
t_value KXLFromChar(char c);
```

Parameter:

Typ	Name	Bemerkung
short	s	In: short Wert, der konvertiert werden soll
int	i	In: int Wert, der konvertiert werden soll
long	l	In: long Wert, der konvertiert werden soll
float	f	In: float Wert, der konvertiert werden soll
double	d	In: double Wert, der konvertiert werden soll
char	c	In: char Wert, der konvertiert werden soll

Funktionsergebnis:

Typ	Bemerkung
t_value	Struktur mit dem umgewandelten Wert

Bei den Konvertierungsroutinen für die einfachen Datentypen `short, int, long, float, double` und `char` ist als Parameter jeweils der umzuwandelnde Wert `short, int, long, float, double` oder `char` anzugeben. Die Funktionen liefern als Returnwert eine Struktur des Typs `t_value`, die aus zwei Zeigern auf Zeichenfolgen (`char*`) besteht. Die erste Zeichenfolge enthält je nach Typ abdruckbar "short", "int", "long", "float", "double", "char", der zweite abdruckbar den Wert.

Der Speicherbereich für diese Zeichenfolgen wird von den Konvertierungsfunktionen verwaltet. Ein erneuter Aufruf einer Konvertierungsfunktion überschreibt das Ergebnis der vorhergehenden. D.h. der Aufrufer muss sich das Ergebnis ggf. in seine eigenen Speicherbereiche kopieren.

Beispiel:

```
t_value     tval;
double      Summe = 99.90;
int         i;
char        tval_Type[11];
char        tval_Value[201];
tval = KXLFromDouble (Summe);
i = MIN (strlen(tval.pType),10);
strncpy (tval_Type, tval.pType, i );
tval_Type[i] = '\0';
i = MIN (strlen(tval.pValue), 200);
strncpy (tval_Value, tval.pValue,i);
tval_Value[i] = '\0';
```

Ergebnis: tval_Type = "double"
tval_Value = "99.90"

Komplexe Datentypen

Funktionsprototypen:

```
t_value KXLFromString(char* s);
t_value KXLFromStruct(char* s);
t_value KXLFromArray(void);
```

Parameter:

Typ	Name	Bemerkung
char *	s	In: Wert, der konvertiert werden soll

Funktionsergebnis:

Typ	Bemerkung
t_value	Struktur mit dem umgewandelten Wert

Auch für die komplexen Datentypen `string` (`\0` terminierte Zeichenfolge), `struct`(Struktur) und `array` (Vektor, Feld) werden Konvertierungsfunktionen angeboten. Die Konvertierung für struct und array wird nur dann benötigt, wenn explizit ein XML-Subobjekt vom Typ `struct` oder `array` erzeugt werden soll (s. Abschnitt Programmschnittstelle..., Aufbau der Elementnamen). Als Wert beim Typ `struct` kann eine Zeichenfolge übergeben werden. Es kann so zum Beispiel der Typname der Struktur eingetragen werden, der Wert-Parameter kann aber auch mit NULL versorgt werden. Die Konvertierungsfunktion für `array` hat keinen Wert-Parameter (void).

Parameter der Funktion ist der umzuwandelnde Wert `char*`. Als Returnwert liefern die Funktionen eine Struktur des Typs `t_value`, die aus zwei Zeigern auf Zeichenfolgen (`char*`) besteht. Die erste Zeichenfolge enthält je nach Typ abdruckbar "string", "struct" oder "array", der zweite Zeiger enthält die Original-Adresse des Eingabewertes (keine Kopie).
Achtung: Die Routinen überprüfen nicht die Korrektheit des zu konvertierenden Wertes!

Beispiel:

```
t_value     tval;
char *      structtyp = "Adressen";
tval = KXLFromStruct (structtyp);
```

Ergebnis: tval.pType ->"struct"
tval.pValue ->"Adressen"

4.2.2 Konvertierung von der `t_value` - Struktur

Beim Lesen eines Objekts werden analog zum Schreiben an der Aufrufschnittstelle Werte vom Typ `t_value` zurückgegeben, die dann vom Aufrufer über Konvertierungsroutinen wieder umgewandelt werden können. Die inversen Typkonvertierungsfunktionen werden für dieselben einfachen Datentypen bereitgestellt, und zusätzlich für die Datentypen `string` und `struct`.

Einfache Datentypen

Funktionsprototypen:

```
short   KXLToShort   (t_value tval, short* pRetCode);
int     KXLToInt     (t_value tval, short* pRetCode);
long    KXLToLong    (t_value tval, short* pRetCode);
float   KXLToFloat   (t_value tval, short* pRetCode);
double  KXLToDouble  (t_value tval, short* pRetCode);
char    KXLToChar    (t_value tval, short* pRetCode);
```

Parameter der Funktion:

Typ	Name	Bemerkung
t_value	tval	In: umzuwandelnder Wert
short*	pRetCode	Out: Returncode, wenn nicht NULL angegeben wurde

Funktionsergebnis:

Typ	Bemerkung
short	konvertierter short Wert
int	konvertierter int Wert
long	konvertierter long Wert
float	konvertierter float Wert
double	konvertierter double Wert
char	konvertierter char Wert

Die Funktionen konvertieren die Werte in `t_value` in die gewünschten Typen, soweit dies möglich ist. Bei unterschiedlichem Ausgangs- und Zieltyp wird ein Returncode gesetzt, die Funktion wird jedoch ausgeführt, ggf. wird der Wert beim Konvertieren abgeschnitten (z.B. bei float -> int). Ist einer der Zeiger von t_value nicht gesetzt (NULL), wird 0 bzw. Leerzeichen (als char) zurückgegeben.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_DIFF_TYPES(2)	I	type in t_value und angeforderter type verschieden. Der Ergebniswert wird ggf. abgeschnitten.
KXL_RC_INVALID_T_VALUE (17)	A	einer der in t_value angegebenen Zeiger ist NULL ->Wert korrigieren

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

Beispiel:

```
t_value tval;
char * T_type = "float";
char * T_value = "15.90";
float f;
tval.pType = T_type;
tval.pValue = T_value;
short Retcode;
f = KXLToFloat (tval, &Retcode);
```


Komplexe Datentypen

Funktionsprototypen:

```
char*   KXLToString(t_value tval, short* pRetCode);
char*   KXLToStruct(t_value tval, short* pRetCode);
```

Parameter der Funktion:

Typ	Name	Bemerkung
t_value	tval	In: umzuwandelnder Wert
short*	pRetCode	Out: Returncode, wenn nicht NULL angegeben wurde

Funktionsergebnis:

Typ	Bemerkung
char *	konvertierter string

Die Funktionen konvertieren die Werte in `t_value` in die gewünschten Typen. Bei unterschiedlichem Ausgangs- und Zieltyp wird ein Returncode gesetzt. Das gilt auch dann, wenn bei `t_value.pType` die leere Zeichenfolge angegeben wurde. Als Returnwert wird der Zeiger `t_value.pValue` zurückgegeben.
Die Funktion KXLToStruct ist nur dann notwendig, wenn auf die in einem struct-Knoten abgelegte Information (s.o., etwa Typname der Struktur) zugegriffen werden soll.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_INVALID_T_VALUE (17)	A	einer der in t_value angegebenen Zeiger ist NULL ->Wert korrigieren

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

Beispiel:

```
t_value tval;
char * T_type = "struct";
char * T_value = "Adressen";
char * structtyp;
tval.pType = T_type;
tval.pValue = T_value;
short Retcode;
structtyp= KXLToStruct (tval, &Retcode);
```

4.3 Schreibfunktionen

4.3.1 Erzeugen eines XML-Objekts

Funktionsprototyp:

```
xmlNodePtr    KXLCreateNewObj(char* pName,t_value tval,short* pRetCode);
```

Parameter:

Typ	Name		Bemerkung
char*	pName	In:	Name des root-Elements
t_value	tval	In:	t_value des root-Elements
short*	pRetCode	Out:	Returncode, wenn nicht NULL angegeben wurde

Funktionsergebnis:

Typ	Bemerkung
xmlNodePtr	Pointer auf den root-Knoten (Wurzel) eines leeren XML-Objekts

Die Funktion erzeugt den 'Kopf' eines XML-Dokuments und legt den root-Knoten mit dem angegebenen Namen `pName` und Inhalt `tval` an. Ist `pName` der NULL-Pointer, oder zeigt er auf die leere Zeichenfolge, wird der root-Knoten mit Name UTM-XML und Versionsattribut aufgebaut. Sind die Zeiger `pType` und `pValue` NULL, bzw. zeigen auf die leere Zeichenfolge ("\0"), wird kein type-Attribut erzeugt, bzw. kein Elementinhalt abgelegt.
Der Returnwert der Funktion ist ein Zeiger auf den root-Knoten eines sonst leeren XML-Objekts. Wenn gewünscht liefert die Funktion einen Returncode zurück.

Beim Erstellen eines neuen XML-Objektes wird die unter _KXL_HOME_ENC_STR definierte Encoding-Angabe als Encoding-Wert abgelegt (s. Abschnitt Zeichensätze). Lesen und Ändern dieses Wertes ist mit **KXLGetDocEnc** bzw. **KXLSetDocEnc** möglich.

Beim ersten Aufruf wird ggf. implizit die Funktion KXLInitEnv aufgerufen (s. Abschnitt 4.1 Initialisierung der UTM-XML-Schnittstelle). Bei Fehler wird kein Objekt erzeugt und ein entsprechender Returncode zurückgegeben.
Konnte kein XML Dokument erzeugt werden, wird ein entsprechender Returncode zurückgegeben (KXL_RC_NO_ROOT_CREATED(4)). Genauere Informationen über den Fehler kann man über KXLGetLastParserError abfragen oder in der Tracedatei verfolgen.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_NO_ROOT_CREATED (4)	S	der Parser konnte kein neues Objekt erzeugen; ->siehe Meldung des Parsers. (Tracedatei oder KXLGetLastParserError)
KXL_RC_INVALID_NAME (19)	A	beim Analysieren des Elementnamens wurde ein Syntaxfehler bei der Verwendung von "[", "]" "/" und @ entdeckt -> gültigen Namen angeben
KXL_RC_VERSION_ERR (23)	A	Versionsstände der UTM-XML-Komponenten inkonsistent
KXL_RC_PARSER_VERS_NOT_ SUPP (24)	A	alte Version des XML-Parsers eingebunden (< 2.6.20) -> neuere Parser-Version einbinden
KXL_RC_UNKNOWN_ENCODING (30)	A	Encoding-Name ist dem Parser nicht bekannt -> zum Encoding gehörige Konvertierungsroutinen beim Parser deklarieren
KXL_RC_ENCODING_CHANGE_ ERR (36)	A/S	für das angegebene Home Encoding konnte kein Encoding Handler eingerichtet werden

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

Beispiel:

```
t_value    tval = {NULL, NULL};
short      Retcode;
xmlNodePtr pRoot;
pRoot = KXLCreateNewObj ("mydoc", tval, &Retcode);
```

erzeugt den root-Knoten <mydoc/>

4.3.2 Schreiben eines Elements

Funktionsprototyp:

```
xmlNodePtr   KXLWrite(xmlNodePtr pNode,char* pName,t_value tval,
   short* pRetCode)
```

Parameter:

Typ	Name	Bemerkung	
xmlNodePtr	pNode	In:	Pointer auf einen Knoten des XML-Objekts.
char*	pName	In:	Name des Elements oder des Attributes
t_value	tval	In:	t_value des Elements oder Wert des Attributes
short*	pRetCode	Out:	Returncode, wenn nicht NULL angegeben wurde

Funktionsergebnis:

Typ	Bemerkung
xmlNodePtr	Pointer auf den geschriebenen Knoten

Beim Aufruf der Funktion sind der Zeiger auf einen Knoten des XML-Objekts, der Name des Elements und die den Typ und den Wert des Elements enthaltende Struktur `t_value` anzugeben. Als Rückgabewert wird der Zeiger auf den geschriebenen Knoten (bzw: im Fehlerfall NULL oder der Zeiger auf den Knoten, bei dem der Fehler aufgetreten ist) und der Returncode an den Aufrufer übergeben.

Der angegebene Zeiger kann ein beliebiger Knoten des XML-Objektes sein. Als Namen muss immer der Namensteil angegeben werden, der ab dem angegebenen Knoten relevant ist (s. Kapitel Programmschnittstelle..., Abschnitt Aufbau der Elementnamen). Namen müssen den Vorgaben der [XML-Spezifikation] entsprechen, zusätzlich können Namen die Strukturtrenner "/", "[" und "]" enthalten. Bei leeren Namensteilen (Index in [ ] bzw zwischen zwei /) wird ein Returncode zurückgegeben, andere syntaktische Fehler erkennt der Parser beim Aufruf KXLConvDocToObj.

Bei leerem Namen wird das Element bearbeitet, auf das der Zeiger (pNode) zeigt. Das Überschreiben von Nodelist-Elementen ist im Wesentlichen nur mit Angabe des Zeigers auf das jeweilige Element und Angabe des leeren Namens möglich.

Bei Strukturen kann man als Wert z.B. einen Strukturtypnamen übergeben, bei Typ `array` muss der pValue-Pointer den Wert NULL haben.
Zeigt pType auf die leere Zeichenfolge, wird für das gewünschte Element und alle implizit angelegten Elemente vom Typ "struct" kein type-Attribut erzeugt. Implizit wird für alle Knoten ohne type-Attribut type="struct" (innere Knoten) bzw. type="string" (äußere Knoten) angenommen.

Beim Schreiben eines Feldelementes wird nicht überprüft, ob alle Elemente des Feldes vom gleichen Typ sind.

Tritt ein Fehler auf, bleiben alle bis dahin implizit erzeugten Elemente bestehen.

Ist die Komponente mit vorgegebenem Namen und Typ noch nicht vorhanden, wird sie als neuer Knoten hinter den letzten Geschwister-Knoten angehängt. Feldelemente sind nach dem Index aufsteigend sortiert. Ist die Komponente mit vorgegebenem Namen und Typ bereits im angegebenen (Sub-)Objekt vorhanden, wird der vorhandene Inhalt durch den angegebenen Wert ersetzt. Wird als Wert der NULL-Zeiger übergeben, wird der vorhandene Inhalt gelöscht.

Ist die Komponente mit vorgegebenem Namen, aber unterschiedlichem Typ bereits vorhanden, wird das Schreiben mit Returncode KXL_RC_TYPE_MISMATCH abgewiesen. Dabei gelten auch type = 'string' oder 'struct' und das leere type-Attribut als unterschiedlich. Soll die Komponente dennoch überschrieben werden, muss sie erst mit gesetztem delete-Flag gelesen (und damit gelöscht) werden. Anschließend kann sie wie gewünscht neu geschrieben werden.

Achtung ! Das gilt auch für schon vorhandene Unterbäume: Wird der Wert eines inneren Knotens (z.B. Wert = typedef eines struct-Knotens) überschrieben, ist damit auch der ganze Unterbaum (d.h. die Struktur) überschrieben, d.h. gelöscht.

Endet der angegebene Elementname mit '[+]', wird der vorangehende Namensteil als Nodelistname interpretiert. (s. Abschnitt "Aufbau von Element- und Attributnamen und Nodelists") und als neues Element in der Nodelist erzeugt. Es wird weder sortiert noch überprüft, ob schon ein Element dieser Nodelist existiert bzw. ob die type-Angaben übereinstimmen. Endet der Name mit [++], [-] oder [--], wird der Aufruf mit Returncode abgewiesen.

Ist als Name nur "[+]" angegeben, wird ein neues Element der Nodelist mit Name des aktuell angegebenen Knotens angelegt. Enthält ein Namensteil (nicht am Anfang oder Ende) eine Nodelist-Positionierung, so wird wie bei Positionierungsaufrufen auf das zweite/nächste ([+]), letzte ([++]) oder erste ([--]) positioniert. [-] wird mit Returncode abgewiesen. Beginnt ein Name mit einer Nodelist-Positionierung, bezieht sich die Positionierung auf die Nodelist, zu der der angegebene Knoten gehört. Ist bei Angabe von [+] bei einem inneren Namensteil kein weiteres Element der angegebenen Nodelist vorhanden, wird es implizit erzeugt. Wird bei den anderen Nodelist-Positionierungen kein Element gefunden, wird ein Returncode zurückgegeben.

Enthält der angegebene Name eckige Klammern mit anderem Inhalt als oben angegeben, wird er als Index eines Arrays interpretiert.

Ist als Name ein Attributnamen angegeben, d.h. der letzte Namensteil beginnt mit '@', wird für das übergeordnete Element ein Attribut mit dem Wert, auf den t_value.pValue zeigt, erzeugt. Ist dort schon ein Attribut dieses Namens vorhanden, wird es überschrieben bzw. gelöscht, wenn der NULL-Zeiger übergeben wurde. Der Wert von t_value.pType wird ignoriert. Beginnt ein anderer Namensteil (außer dem letzten) mit '@', wird das Schreiben mit Returncode KXL_RC_INVALID_NAME abgewiesen.

Jeder einzelne Namensteil kann ein Namensraum-Präfix in der Form "prefix:name" enthalten. Ein Name ist nur dann gleich, wenn auch das Präfix gleich ist (siehe auch Abschnitt Namensräume). Über das Präfix wird ein Element intern einer Namensraum-Definition zugeordnet. Namen ohne Präfix werden dem Default-Namensraum zugeordnet. Wird für ein Element keine gültige Namensraum-Definition gefunden, wird eine Dummy-Definition erzeugt.

Beispiel: Im Namen "student/ns1:fach/ns2:punkte" gehört "student" zum Default-Namensraum, "fach" zum Namensraum ns1, "punkte" zum Namensraum ns2.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_NO_NODE_FOUND (1)	I	Knoten mit dem angegebenen Namen nicht gefunden
KXL_RC_ALLOC_ERROR (11)	S	mit malloc/realloc konnte kein zusätzlicher Speicherplatz angefordert werden
KXL_RC_TYPE_MISMATCH(12)	A	beim Überschreiben eines Knotens stimmt der vorhandene Typ nicht mit dem angegebenen Typ überein ->wenn Überschreiben gewünscht, Knoten erst löschen, dann neu schreiben
KXL_RC_NO_CHILD_CREATED (13)	S/A	Fehler beim Erzeugen eines neuen Knotens ->Meldung des Parsers auswerten
KXL_RC_NO_TYPE_ATTR_FOUND(14)	S/A	beim Überschreiben eines Knotens wurde kein Typ – Attribut gefunden, kein Überschreiben möglich.
KXL_RC_NO_XML_NODE (15)	A	der Zeiger auf den aktuellen Knoten ist NULL -> gültigen Zeiger angeben
KXL_RC_EMPTY_NAME(18)	A	beim Analysieren des Elementnamens wurde ein leerer Namensteil gefunden, Schreiben abgebrochen, bzw. Name leer -> gültigen Namen angeben.
KXL_RC_INVALID_NAME (19)	A	beim Analysieren des Elementnamens wurde ein Syntaxfehler bei der Verwendung von "[", "]" "/" und @ entdeckt -> gültigen Namen angeben
KXL_RC_NAMESPACE_WRITE_ERROR (33)	S/A	Der Parser konnte die Namensraum-Definition nicht ordnungsgemäß schreiben, siehe Parser-Fehlermeldungen

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

Beispiel:
float Preis=1.75;
KXLWrite(pNode, "Preis", KXLFromfloat(Preis), NULL);

4.4 Konvertierungsfunktionen

4.4.1 Konvertierung eines XML-Objekts in ein XML-Dokument

Funktionsprototyp:

```
char *  KXLConvObjToDoc    (xmlNodePtr pNode, char* pStylesheet,
                            int* pBufLen, short* pRetCode);
```

Parameter:

Typ	Name	Bemerkung	
xmlNodePtr	pNode	In:	Pointer auf den root-Knoten des XML-Objekts
char*	pStylesheet	In:	Inhalt der Stylesheet PI
int*	pBufLen	Out:	Länge des Puffers, wenn nicht NULL angegeben wurde
short*	pRetCode	Out:	Returncode, wenn nicht NULL angegeben wurde

Funktionsergebnis:

Typ	Bemerkung
char*	Puffer mit dem XML-Dokument

Um aus einem XML-Objekt ein XML-Dokument zu erzeugen, muss die Funktion KXLConvObjToDoc aufgerufen werden, die als Eingabeparameter den Zeiger auf den root-Knoten des XML-Objekts erwartet. Aus dem XML-Objekt in UTF-8 wird das Dokument in den unter encoding angegebenen Zeichensatz konvertiert. Ist keiner angegeben, bleibt das Dokument im UTF-8 Encoding. Die Encoding-Angabe kann mit KXLSetDocEnc erzeugt bzw. geändert werden.
Ist für den angegebenen Zeichensatz keine Konvertierungsroutine deklariert (s. Abschnitt Zeichensätze), wird kein Dokument erzeugt und ein entsprechender Returncode zurückgegeben.

Die Funktion konvertiert das XML-Objekt in ein XML-Dokument und gibt die Adresse des Puffers zurück, der das XML-Dokument enthält und \0 terminiert ist (im Fehlerfall der NULL-Pointer). Wenn gewünscht (pBufLen nicht NULL), wird auch die Länge des Puffers zurückgegeben. Hinter dem 'Kopf' des Dokuments (<?xml...?>) wird ein stylesheet PI mit angegebenem Inhalt (in der Form <?xml-stylesheet ... ?>) eingefügt, wenn pStylesheet nicht NULL ist bzw. auf die leere Zeichenfolge zeigt. Ist schon ein stylesheet PI vorhanden, wird dieses mit dem neuen PI überschrieben. Der Speicherbereich des Puffers kann mit xmlFree() wieder freigegeben werden.

Anmerkungen:

- Eine Ausgabeformatierung mit Zeilenvorschüben erfolgt nur bei Elementen, die ausschließlich weitere Elemente als Kindknoten enthalten. Bei Elementen, die Text oder Entities als Kindknoten enthalten, werden beim Aufbau des Dokuments keine Zeilenvorschübe eingefügt, da sonst der Inhalt des Dokumentes verfälscht würde.
- Es ist auch möglich, ein Dokument ohne encoding-Attribut in einem anderen Encoding als UTF-8 zu erzeugen. Dazu siehe Anhang, Abschnitt "Bearbeitung von Dokumenten ohne encoding-Attribut".
- Konnte das XML Dokument nicht erzeugt werden, wird ein entsprechender Returncode zurückgegeben (KXL_RC_NO_CONVERSION_TO_DOC(7)). Genauere Informationen über den Fehler kann man über KXLGetLastParserError abfragen oder in der Tracedatei verfolgen.
- Wird als Root-Knoten der eines Schema-Objektes angegeben, kann natürlich auch ein Schema-Dokument erzeugt werden. Den Root Knoten eines Schemas erhält man in C über den Pointer auf die Schema-Struktur folgendermaßen:

  ```
  xmlSchemaPtr->doc->children
  ```

  Für die Cobol-Schnittstelle steht dazu die Funktion KXLSchemaGetRoot zur Verfügung.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_NO_CONVERSION_ TO_DOC (7)	S	Fehler beim Erzeugen des Dokuments ->siehe Meldung des Parsers. (Tracedatei oder KXLGetLastParserError)
KXL_RC_INVALID_DOC_PTR (10)	S/A	der angegebene Objektknoten ist keinem Objekt zugeordnet; mögl. Grund: Objekt zerstört oder fehlerhaft erzeugt ->Fehlerunterlagen erzeugen
KXL_RC_NO_XML_NODE (15)	A	der Zeiger auf den aktuellen Knoten ist NULL -> gültigen Zeiger angeben
KXL_RC_CONVERSION_ ERROR (22)	S/A	Fehler bei der Code-Konvertierung

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

4.4.2 Konvertierung eines XML-Dokuments in ein XML-Objekt

Funktionsprototyp:

```
xmlNodePtr    KXLConvDocToObj(char* pBuffer, short* pRetCode);
```

Parameter:

Typ	Name	Bemerkung
char*	pBuffer	In: Puffer mit dem XML-Dokument
short*	pRetCode	Out: Returncode, wenn nicht NULL angegeben wurde

Funktionsergebnis:

Typ	Bemerkung
xmlNodePtr	Pointer auf den root-Knoten des XML-Objekts

Zur Bearbeitung muss ein XML-Dokument in ein XML-Objekt konvertiert werden. Dazu wird die Funktion KXLConvDocToObj aufgerufen, die als Eingabeparameter den Puffer mit dem kompletten (\0 terminierten!) XML-Dokument hat. Das Ergebnis der Funktion ist der Zeiger auf den root-Knoten des XML-Objekts. Im Fehlerfall, z.B. wenn ein nicht gültiges Dokument (not valid) vorliegt, wird NULL zurückgegeben.

Enthält ein zu konvertierendes Dokument eine Encoding-Angabe, wird dies im Objektbaum hinterlegt, alle Namen und Inhalte werden nach UTF-8 konvertiert und im Objektbaum abgelegt.
Ist keine Encoding-Angabe vorhanden, wird im Normalfall UTF-8 angenommen. Es ist auch möglich, ein Dokument ohne encoding-Attribut in einem anderen Encoding als UTF-8 zu verarbeiten. Dazu siehe Anhang, Abschnitt "Bearbeitung von Dokumenten ohne encoding-Attribut".
Für die Encoding-Angabe muss eine Codekonvertierungsroutine (von und nach UTF-8) deklariert sein, andernfalls wird ein Fehler zurückgegeben und kein Objekt erzeugt (siehe dazu Abschnitt Zeichensätze).

Beim ersten Aufruf wird ggf. implizit die Funktion KXLInitEnv aufgerufen (s. Abschnitt 4.1 Initialisierung der UTM-XML-Schnittstelle). Bei Fehler wird kein Objekt erzeugt und ein entsprechender Returncode zurückgegeben.
Ist das XML Dokument nicht wohlgeformt, wird ein entsprechender Returncode zurückgegeben (KXL_RC_PARSER_ERROR(20)). Genauere Informationen über den Fehler kann man über KXLGetLastParserError abfragen oder in der Tracedatei verfolgen.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_NO_MALLOC_FOR_ PARSER_1(5)	S	nicht genug Speicherplatz für das Objekt vorhanden -> Programmspeicher vergrößern
KXL_RC_NO_MALLOC_FOR_ PARSER_2(6)	S	nicht genug Speicherplatz für das Objekt vorhanden -> Programmspeicher vergrößern
KXL_RC_NO_CODE_ CONVERSION (8)	A	Beginn des Dokuments entspricht nicht dem XML Start tag '<?xml' oder '<?XML' in EBCDIC oder ASCII ->Dokument korrigieren bzw. Codekonvertierung vornehmen
KXL_RC_EMPTY_DOC (9)	A	konvertiertes Objekt enthält keine Knoten
KXL_RC_ALLOC_ERROR (11)	S	mit malloc/realloc konnte kein zusätzlicher Speicherplatz angefordert werden
KXL_RC_PARSER_ERROR (20)	S/A	Fehler im Parser entdeckt ->siehe Meldung des Parsers. (Tracedatei oder KXLGetLastParserError)
KXL_RC_NO_CONTEXT_FOR_ PARSER (21)	S	Fehler beim Anlegen parserinterner Verwaltungsbereiche -> evtl. Speicherengpass beseitigen
KXL_RC_CONVERSION_ERRO R (22)	S/A	Fehler bei der Code-Konvertierung
KXL_RC_VERSION_ERR (23)	A	Versionsstände der UTM-XML-Komponenten inkonsistent
KXL_RC_PARSER_VERS_NOT_ SUPP (24)	A	alte Version des XML-Parsers eingebunden (< 2.6.20) -> neuere Parser-Version einbinden
KXL_RC_UNKNOWN_ENCODIN G (30)	A	Encoding-Name ist dem Parser nicht bekannt -> zum Encoding gehörige Konvertierungsroutinen beim Parser deklarieren
KXL_RC_ENCODING_CHANGE_ ERR (36)	A/S	für das angegebene Home Encoding konnte kein Encoding Handler eingerichtet werden

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

4.4.3 Freigeben des Speichers eines XML-Objekts

Funktionsprototyp:

```
void    KXLFreeObj(xmlNodePtr pNode, short* pRetCode);
```

Parameter:

Typ	Name	Bemerkung
xmlNodePtr	pNode	In: Pointer auf den root-Knoten des XML-Objekts
short*	pRetCode	Out: Returncode, wenn nicht NULL angegeben wurde

Mit diesem Aufruf wird der für das XML-Objekt benötigte Speicher freigegeben. Er sollte immer dann erfolgen, wenn nicht mehr mit dem betreffenden XML-Objekt gearbeitet wird.
Anschließend darf auf Adressen des freigegebenen Objekts nicht mehr zugegriffen werden.

Achtung: Das XML-Objekt eines Schemas sollte nicht mit KXLFreeObj, sondern zusammen mit der gesamten XML-Schema-Struktur durch den Aufruf von KXLFreeSchema freigegeben werden.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_INVALID_DOC_PTR (10)	S/A	der angegebene Objektknoten ist keinem Objekt zugeordnet; möglicher Grund: Objekt zerstört oder fehlerhaft erzeugt ->Fehlerunterlagen erzeugen
KXL_RC_NO_XML_NODE (15)	A	der Zeiger auf den aktuellen Knoten ist NULL -> gültigen Zeiger angeben

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

4.5 Navigieren im XML-Objekt

4.5.1 Positionieren auf ein XML-Subobjekt

Funktionsprototyp:

```
xmlNodePtr    KXLSetSubObject(xmlNodePtr pNode, char* pName,
short* pRetCode);
```

Parameter:

Typ	Name	Bemerkung
xmlNodePtr	pNode	In:Pointer auf den Ausgangs-Knoten des XML-Objekts
char*	Name	In:Name des Elements, auf das positioniert werden soll
short*	pRetCode	Out:Returncode, wenn nicht NULL angegeben wurde

Funktionsergebnis:

Typ	Bemerkung
xmlNodePtr	Pointer auf den gesuchten Knoten des XML-Objekts

Sowohl beim Schreiben als auch beim Lesen kann es sinnvoll sein, den Zugriff auf einen Unterbaum zu beschränken, so dass nicht immer der vollqualifizierte Name angegeben werden muss. Das erhöht die Performance, da beim Zugriff auf einen Knoten nicht über alle Stufen des Objektbaumes gesucht werden muss. Daher gibt es die Möglichkeit, auf einen Teilbaum des XML-Objekts zu positionieren. Die Funktion liefert den Zeiger auf einen Knoten, der die Wurzel eines Teilbaums im XML-Objekt, entsprechend der angegebenen Struktur bzw. array bildet. Über diesen Zeiger kann mit teilqualifizierten Namen auf die Komponenten der Struktur bzw. des arrays mit den Zugriffsfunktionen (z.B. KXLWrite/KXLRead) zugegriffen werden.

Bei Angabe von leeren Namensteilen (Index in [ ] bzw zwischen zwei /) wird ein Returncode zurückgegeben.

Im Fehlerfall wird der NULL – Pointer zurückgegeben.

Enthält der Name eine der Nodelist-Positionierungen [--], [+] oder [++], kann auf das erste, 2. oder das letzte Nodelist-Element positioniert werden. [-] kann nur zu Beginn des Namens angegeben werden. Dann wird auf das dem angegebenen Knoten vorhergehende Nodelist-Element positioniert. Steht [-] an anderer Stelle, wird der Aufruf mit Returncode abgewiesen. Mit [+] zu Beginn eines Namens wird auf das dem angegebenen Knoten folgende Nodelist-Element positioniert.

Mit dem zurückgegebenen Zeiger kann man mit Angabe weiterer Nodelist-Positionierungen auf weitere Elemente der nodelist positionieren bzw. sie lesen.

Enthalten die eckigen Klammern andere Zeichen, werden sie als Index eines Arrays interpretiert.

Wird mit Namensräumen gearbeitet, müssen die Namensteile mit den entsprechenden Präfixen angegeben werden.

Anmerkung:
Das Positionieren auf Attribute ist möglich, allerdings nicht sinnvoll. Zur Namensangabe siehe Abschnitt Aufbau von Element- und Attributnamen und Nodelists

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_NO_NODE_FOUND (1)	I	Knoten mit dem angegebenen Namen nicht gefunden
KXL_RC_ALLOC_ERROR (11)	S	mit malloc/realloc konnte kein zusätzlicher Speicherplatz angefordert werden
KXL_RC_NO_XML_NODE (15)	A	der Zeiger auf den aktuellen Knoten ist NULL -> gültigen Zeiger angeben
KXL_RC_INVALID_NAME (19)	A	beim Analysieren des Elementnamens wurde ein Syntaxfehler bei der Verwendung von "[", "]" "/" und @ entdeckt -> gültigen Namen angeben
KXL_RC_CONVERSION_ERROR (22)	S/A	Fehler bei der Code-Konvertierung
KXL_RC_NODE_TYPE_NOT_ ALLOWED (28)	A	bei diesem Aufruf ist nur die Angabe eines Element- bzw. Attributknotens erlaubt

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

4.5.2 Positionieren auf den root-Knoten des XML-Objekts

Funktionsprototyp:

```
xmlNodePtr    KXLSetRootNode(xmlNodePtr pNode, short* pRetCode);
```

Parameter:

```
Typ       Name       Bemerkung
xmlNodePtr pNode     In:   Pointer auf einen Knoten des XML-Objekts.
short*    pRetCode   Out:  Returncode, wenn nicht NULL angegeben wurde
```

Funktionsergebnis:

```
Typ           Bemerkung
xmlNodePtr    Pointer auf den root-Knoten des XML-Objekts
```

In manchen Fällen ist es notwendig, vom Arbeiten in einem Teilbaum wieder auf das gesamte XML-Objekt überzugehen, sei es, dass ein Zugriff an einer anderen Stelle notwendig ist, sei es, dass die Arbeit an dem XML-Objekt abgeschlossen ist und es konvertiert und verschickt werden soll.
Als Parameter wird ein beliebiger Knoten des XML-Objekts übergeben. Zurückgegeben wird der root-Knoten des XML-Objekts oder im Fehlerfall der NULL – Pointer.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_INVALID_DOC_PTR (10)	S/A	der angegebene Objektknoten ist keinem Objekt zugeordnet; möglicher Grund: Objekt zerstört oder fehlerhaft erzeugt ->Fehlerunterlagen erzeugen
KXL_RC_NO_XML_NODE (15)	A	der Zeiger auf den aktuellen Knoten ist NULL -> gültigen Zeiger angeben

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

Beispiel zum Erkennen des Rootknotens:

```
if (KXLSetRootNode(pNode, NULL) == pNode)
{ /* root erreicht, stop working */ }
```

4.5.3 Positionieren auf das nächst-höhere XML-Subobjekt

Funktionsprototyp:

```
xmlNodePtr    KXLSetParentNode(xmlNodePtr pNode, short* pRetCode);
```

Parameter:

Typ	Name	Bemerkung	
xmlNodePtr	pNode	In:	Pointer auf einen Knoten des XML-Objekts.
short*	pRetCode	Out:	Returncode, wenn nicht NULL angegeben wurde

Funktionsergebnis:

Typ	Bemerkung
xmlNodePtr	Pointer auf den nächsthöheren Knoten des XML-Objekts

In manchen Fällen ist es sinnvoll, von einem Teilbaum wieder auf den nächst-höheren Teilbaum zu positionieren, um z.B. auf die der aktuellen Struktur folgende Struktur zuzugreifen.
Als Parameter wird ein beliebiger Knoten des XML-Objekts übergeben. Zurückgegeben wird der nächsthöhere Knoten innerhalb des XML-Objekts.
Bei Angabe des Dokumentknotens wird NULL zurückgegeben.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_NO_XML_NODE (15)	A	der Zeiger auf den aktuellen Knoten ist NULL -> gültigen Zeiger angeben

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

4.6 Lesefunktionen

In einem bestehenden XML-Objekt (neu aufgebaut oder aus einem XML-Dokument konvertiert) können die einzelnen Elemente des Objekts mit Hilfe von Lesefunktionen gelesen werden.

4.6.1 Lesen mit Namen

Funktionsprototyp:

```
t_value       KXLRead(xmlNodePtr pNode, char* pName, int delete,
                      short* pRetCode)
```

Parameter:

Typ	Name	Bemerkung
xmlNodePtr	pNode	In: Pointer auf den Knoten des XML-Objekts.
char*	pName	In: Name der Elements
int	delete	In: KXL_TRUE / KXL_FALSE
short*	pRetCode	Out: Returncode, wenn nicht NULL angegeben wurde

Funktionsergebnis:

Typ	Bemerkung
t_value	t_value des Elements

Mit der Funktion KXLRead kann man ein Element über seinen Namen ansprechen. KXLRead liefert in der Struktur vom Typ `t_value` den Typ und den Wert des Datums zurück, die mit den im Abschnitt Typ-Konvertierungsfunktionen beschriebenen Funktionen umgewandelt werden können. Ist kein type-Attribut vorhanden, wird als Typ die leere Zeichenfolge zurückgegeben. Für innere Knoten entspricht das type="struct", für äußere type="string".

Wie beim Schreiben kann das Lesen auf zwei Arten erfolgen: Bei Angabe des root-Knotens kann ein Datum über den vollqualifizierten Namen angesprochen werden, das Lesen kann aber auch mehrstufig erfolgen: Mit KXLSetSubObject kann im XML-Objekt positioniert werden. Mit dem so erhaltenen Zeiger auf ein SubObjekt kann mit KXLRead und dem Komponentennamen bzw. Feldindex (derForm [i]) auf die einzelnen Daten zugegriffen werden.

Bei Angabe eines leeren Namens (Leerzeichen) wird Typ und Wert des angegebenen Knotens (pNode) zurückgegeben. Dies ist allerdings nur für Element- und nicht für Attributknoten möglich. Diese können direkt nur über den Aufruf KXLReadNode gelesen werden.

Das erste Element einer nodelist wird wie einzelne Elemente über einen Knoten und die Angabe des dazu relativen Namens gelesen. Bei Angabe einer Nodelist-Positionierung ([+],[-],[++] oder [--]) wird ausgehend von dem aktuellen Knoten das nächste Element der nodelist mit angegebenen Namen mit [+], das vorherige mit [-], das letzte mit [++] bzw. das erste mit [--] gelesen. Es dürfen nicht mehrere Klammern aufeinanderfolgen, z.B. name[++][-] wird mit Returncode KXL_RC_INVALID_NAME abgewiesen.

Enthält der angegebene Name eckige Klammern mit anderem Inhalt als '+','-', '++', '--' wird er als Index eines Arrays interpretiert.

Wird mit Namensräumen gearbeitet, müssen die Namensteile mit den entsprechenden Präfixen angegeben werden.

Ist das zu lesende Element ein Attributknoten, wird der Inhalt des Attributes über t_value.pValue zurückgegeben. t_value.pType zeigt auf die leere Zeichenfolge.

Falls gewünscht, wird der Knoten nach dem Lesen gelöscht. Lesen mit Löschen kann man nur auf einfachen Daten-Elementen der untersten Ebene, d.h. Blättern des XML-Objekts und Attributen, ausführen. Soll ein ganzer Teilbaum gelöscht werden, gibt es zwei Möglichkeiten: Entweder müssen alle Knoten einzeln gelöscht werden, oder der root – Knoten des Teilbaums wird mit einem beliebigen Wert, aber der gleichen Typ – Angabe überschrieben (damit ist der Teilbaum implizit gelöscht) und anschließend mit Delete = KXL_TRUE gelesen und gelöscht.

Die Puffer, auf die die Zeiger in der Struktur t_value zeigen, werden beim nächsten KXLRead-Aufruf überschrieben. Ggf. müssen die Werte in eigenen Speicherbereichen gesichert werden.

Im Fehlerfall enthält die zurückgegebene t_value Variable zwei NULL – Pointer.

Anmerkung:
Attribute, die Namensräume definieren, können nicht gelesen werden. Siehe dazu KXLSearchNS

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_NO_NODE_FOUND (1)	I	Knoten mit dem angegebenen Namen nicht gefunden bzw. es existiert kein weiteres Element der angegebenen nodelist
KXL_RC_ALLOC_ERROR (11)	S	mit malloc/realloc konnte kein zusätzlicher Speicherplatz angefordert werden, kein Lesen möglich
KXL_RC_NO_XML_NODE (15)	A	der Zeiger auf den aktuellen Knoten ist NULL -> gültigen Zeiger angeben
KXL_RC_NO_SUBTREE_DELETION (16)	A	beim Lesen eines Knotens mit Löschen wurde der Knoten nicht gelöscht, da es kein äußerer Knoten ist. ->soll der ganze Teilbaum gelöscht werden, muss jeder Knoten einzeln gelöscht werden.
KXL_RC_INVALID_NAME (19)	A	beim Analysieren des Elementnamens wurde ein Syntaxfehler bei der Verwendung von "[", "]" ,"/" und "@" entdeckt -> gültigen Namen angeben
KXL_RC_CONVERSION_ERROR (22)	S/A	Fehler bei der Code-Konvertierung
KXL_RC_ATTR_NAME_ERROR (25)	A	Der Aufbau des Attributnamens ist nicht korrekt; nach '@' keine '/', '[' oder ']' erlaubt
KXL_RC_ATTR_READ_WRITE_ ERR (26)	A	Fehler beim Lesen oder Schreiben eines Attributes
KXL_RC_NODE_TYPE_NOT_ ALLOWED (28)	A	bei diesem Aufruf ist nur die Angabe eines Element- bzw. Attributknotens erlaubt

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

Beispiel:

```
float Preis;
Preis = KXLToFloat(KXLRead(pNode, "preis", KXL_FALSE, NULL), &RetCode);
```

Ist das Element 'preis' nicht vorhanden, hat RetCode den Wert KXL_RC_NO_NODE_FOUND.

4.6.2 Direktes Lesen

Funktionsprototyp:

```
t_value  KXLReadNode(xmlNodePtr pNode ,int FullName, char** ppName,
                     short* pRetCode);
```

Parameter:

Typ	Name	Bemerkung
xmlNodePtr	pNode	In: Pointer auf einen Knoten des XML-Objekts.
int	FullName	In: KXL_TRUE / KXL_FALSE
char**	ppName	Out: Pointer auf den Namen des Knotens.
short*	pRetCode	Out: Returncode, wenn nicht NULL angegeben wurde

Funktionsergebnis:

Typ	Bemerkung
t_value	t_value des Elements

Mit dieser Funktion kann ein Element des XML-Objekts direkt gelesen werden, ohne dass der Aufrufer den Namen kennt. Vom aktuellen Knoten des XML-Objekts wird der Name des Elements und in der Struktur t_value der Typ und der Wert des Elements zurückgegeben. Ist kein type-Attribut vorhanden, wird als Typ die leere Zeichenfolge zurückgegeben. Für innere Knoten entspricht das type="struct", für äußere type="string".
Bei FullName=KXL_TRUE wird der vollständige Pfadname des Elements (ausgehend vom root-Knoten, aber ohne diesen) ausgegeben, bei FullName=KXL_FALSE der Namensteil, der dem gelesenen Knoten entspricht. Der Name des root-Knotens kann also nur mit FullName=KXL_FALSE gelesen werden. Die Namensteile enthalten das Namensraum-Präfix, wenn vorhanden.
Im Fehlerfall werden NULL – Pointer zurückgegeben.
Die Puffer, in denen die Daten zurückgegeben werden (t_value, ppName), werden beim nächsten Leseaufruf überschrieben.

Anmerkungen:

1. Es wird nicht erkannt, ob der gelesene Knoten ein Element einer nodelist repräsentiert und wenn ja, welches.
2. Wird ein Attribut gelesen, wird dem zurückgegebenen Attributnamen ein '@' vorangestellt.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_INVALID_DOC_PTR (10)	S/A	der angegebene Objektknoten ist keinem Objekt zugeordnet; möglicher Grund: Objekt zerstört oder fehlerhaft erzeugt ->Fehlerunterlagen erzeugen
KXL_RC_ALLOC_ERROR (11)	S	mit malloc/realloc konnte kein zusätzlicher Speicherplatz angefordert werden, kein Lesen möglich
KXL_RC_NO_XML_NODE (15)	A	der Zeiger auf den aktuellen Knoten ist NULL -> gültigen Zeiger angeben
KXL_RC_CONVERSION_ERROR (22)	S/A	Fehler bei der Code-Konvertierung
KXL_RC_NODE_TYPE_NOT_ALLOWED (28)	A	bei diesem Aufruf ist nur die Angabe eines Element- bzw. Attributknotens erlaubt

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

Beispiel:

```
xmlNodePtr  pRoot;
t_value     tval;
char  Name[64];
char *pName = &Name[0];
tval = KXLReadNode (pRoot, KXL_FALSE, &pName, &Retcode);
/* output of type, name and value of root node: */
sprintf("root node %s %s = %s \n", tval.pType, pName, tval.pValue);
```

4.6.3 Sequentielles Lesen

Funktionsprototyp:

```
t_value KXLReadNextSib(xmlNodePtr pNode ,char** ppName,
        xmlNodePtr* ppNode, short* pRetCode);
t_value  KXLReadChild(xmlNodePtr pNode,char** ppName,
        xmlNodePtr* ppNode, short* pRetCode);
t_value  KXLReadNextSingleNode(xmlNodePtr pNode ,char** ppName,
        xmlNodePtr* ppNode, short* pRetCode);
t_value  KXLReadAttr(xmlNodePtr pNode ,char** ppName,
        xmlNodePtr* ppNode, short* pRetCode)
```

Parameter:

Typ	Name	Bemerkung
xmlNodePtr	pNode	In: Pointer auf einen Knoten des XML-Objekts
char**	ppName	Out:Pointer auf den Namen des Knotens
xmlNodePtr*	ppNode	Out:Pointer auf den neu gefundenen Knoten des XML-Objekts
short*	pRetCode	Out:Returncode, wenn nicht NULL angegeben wurde

Funktionsergebnis:

Typ	Bemerkung
t_value	t_value des Elements

Mit diesen Funktionen können alle Elemente des XML-Objekts sequentiell gelesen werden, ohne dass der Aufrufer den Namen kennt. Ausgehend vom aktuellen Knoten wird vom nächsten Knoten im XML-Objekt der Name des Elements, in der Struktur t_value der Typ und der Wert des Elements und der Zeiger auf den gelesenen Knoten zurückgegeben. Ist kein type-Attribut vorhanden, oder wird ein Attribut gelesen, wird als Typ die leere Zeichenfolge zurückgegeben. Für innere Knoten entspricht das type="struct", für äußere type="string".
Im Fehlerfall werden NULL – Pointer zurückgegeben. Die Knoten werden in derselben Reihenfolge gelesen wie sie geschrieben worden sind. Ausnahme bilden die Elemente eines arrays, die beim Schreiben aufsteigend geordnet werden.
Abhängig davon, ob nur die einfachen Daten-Elemente (Blätter) oder Attribute gelesen werden sollen oder alle Knoten (auch Struktur- und array-Knoten), sind unterschiedliche Funktionen notwendig.

KXLReadNextSib	liest den Inhalt des nächsten ‚Geschwister'-Knotens (gleiche Ebene). Ist der aktuelle Knoten ein Attribut, werden hier sequentiell weitere Attribute gelesen.
KXLReadChild	liest den Inhalt des (ersten) Kindknotens (eine Ebene tiefer, keine Attribute),
KXLReadNextSingleNode	liest den Inhalt des nächsten Blattes (unterste Ebene, keine Attribute),
KXLReadAttr	liest den Inhalt des (ersten) Attributes des vorgegebenen Knotens

Als Name wird bei KXLReadNextSingleNode der vollständige Pfadname des Elements (ausgehend vom root-Knoten) zurückgegeben, sonst nur der Namensteil, der dem gelesenen Knoten entspricht.

Die Namensteile enthalten das Namensraum-Präfix, wenn vorhanden.
Die Puffer, in denen die Daten zurückgegeben werden (t_value, ppName), werden beim nächsten Leseaufruf überschrieben.

Anmerkungen:

1. Es wird nicht erkannt, ob der gelesene Knoten ein Element einer nodelist repräsentiert und wenn ja, welches.
2. Wird ein Attribut gelesen, wird dem zurückgegebenen Attributnamen ein '@' vorangestellt.
3. Attribute, die Namensräume definieren, können mit den diesen Aufrufen nicht gelesen werden. Siehe dazu KXLSearchNS.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_NO_NODE_FOUND (1)	I	kein weiterer Knoten mehr gefunden, d.h.: - bei KXLReadNext Sib: kein Geschwisterknoten im aktuellen Teilbaum bzw. kein weiteres Attribut vorhanden - bei KXLReadChild: kein Kindknoten - bei KXLReadNextSingleNode: kein Blatt (im ganzen Objekt) - bei KXLReadAttr: kein Attribut vorhanden
KXL_RC_INVALID_DOC_PTR (10)	S/A	der angegebene Objektknoten ist keinem Objekt zugeordnet mögl. Grund: Objekt zerstört oder fehlerhaft erzeugt ->Fehlerunterlagen erzeugen
KXL_RC_ALLOC_ERROR (11)	S	mit malloc/realloc konnte kein zusätzlicher Speicherplatz angefordert werden, kein Lesen möglich
KXL_RC_NO_XML_NODE (15)	A	der Zeiger auf den aktuellen Knoten ist NULL -> gültigen Zeiger angeben
KXL_RC_CONVERSION_ERROR (22)	S/A	Fehler bei der Code-Konvertierung
KXL_RC_NODE_TYPE_NOT_ALLOWED (28)	A	bei diesen Aufruf ist nur die Angabe eines Elementknotens erlaubt

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

Beispiele:

1. Ausgabe aller einfachen Elemente (Blätter) eines Objekts:

```
xmlNodePtr  pRoot, pAct, pNew;
t_value     tval;
char  Name[64];
char *pName = &Name[0];
pAct = pRoot;
while (1)
{ tval = KXLReadNextSingleNode (pAct, &pName, &pNew, &Retcode);
   if (Retcode == KXL_RC_NO_NODE_FOUND)
      break;
   /* output of Type, Name and Value of node: */
   sprintf("%s %s = %s \n", tval.pType, pName, tval.pValue);
   pAct = pNew;
}
```

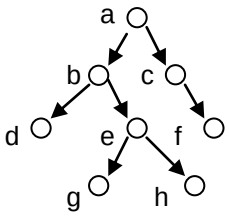

2. Ausgabe aller Knoten eines Objekts:

```
void ReadSubtree(xmlNodePtr pNode)
{   t_value  tval;
    char *   pName;
    xmlNodePtr pAct;
    xmlNodePtr pNew;
    tval = KXLReadChild(pNode, &pName, &pNew, NULL); /* read child node */
    while (pNew != NULL)                    /* no more children in subtree */
    {   /* output of Type, Name and Value of node */
        sprintf("%s %s = %s \n", tval.pType, pName, tval.pValue);
        ReadSubtree (pNew);    /* recursive call: read subtree of child*/
        pAct = pNew;
        /* read next sibling */
        tval = KXLReadNextSib(pAct, &pName, &pNew, NULL);
    }
    return;
} /* end ReadSubtree */
xmlNodePtr  pRoot; /* root – Pointer of Object */
ReadSubtree(pRoot);
```

3. Ausgabe aller Attribute eines Elements:

```
xmlNodePtr  pAct, pNew;
t_value     tval;
char  Name[64];
char *pName = &Name[0];
short Retcode = KXL_RC_OK;
/* read first attribute */
tval = KXLReadAttr (pAct, &pName, &pNew, &Retcode);
while (Retcode == KXL_RC_OK)
{  /* output of name and value of node: */
   sprintf("%s = %s \n", pName, tval.pValue);
   pAct = pNew;
   /* read next attribute */
   tval = KXLReadNextSib (pAct, &pName, &pNew, &Retcode);
}
```

In folgendem Objektbaum:



werden mit Beispiel 1 die Knoten d, g, h, f gelesen,
mit Beispiel 2: a, b, d, e, g, h, c, f

4.6.4 Lesen mit lokalem Namen

Funktionsprototyp:

```
t_value       KXLFindNode(xmlNodePtr pNode, char* pName, char** ppUrl,
              char** ppPrefix, xmlNodePtr* ppNode, short* pRetCode)
```

Parameter:

Typ	Name	Bemerkung
xmlNodePtr	pNode	In: Pointer auf den Knoten des XML-Objekts.
char*	pName	In: Name der Elements
char**	ppUrl	Out: Pointer auf die Url des neu gefundenen Knoten des XML-Objekts
char**	ppPrefix	Out: Pointer auf den Präfix des neu gefundenen Knoten des XML-Objekts
xmlNodePtr*	ppNode	Out: Pointer auf den neu gefundenen Knoten des XML-Objekts
short*	pRetCode	Out: Returncode, wenn nicht NULL angegeben wurde

Funktionsergebnis:

Typ	Bemerkung
t_value	t_value des Elements

Mit der Funktion KXLFindNode kann man ein Element über seinen lokalen Namen, d.h. ohne Berücksichtigung der Namensraumzugehörigkeit, ansprechen.

Der mögliche Aufbau eines Element-Namens ist im Abschnitt "Aufbau der Element- und Attributnamen und Nodelists" beschrieben. Das erste Element einer nodelist wird wie einzelne Elemente über einen Knoten und die Angabe des dazu relativen Namens gelesen. Bei Angabe einer Nodelist-Positionierung ([+],[-],[++] oder [--]) wird ausgehend von dem aktuellen Knoten das nächste Element der nodelist mit angegebenen Namen mit [+], das vorherige mit [-], das letzte mit [++] bzw. das erste mit [--] gelesen. Es dürfen nicht mehrere Klammern aufeinanderfolgen, z.B. name[++][-] wird mit Returncode KXL_RC_INVALID_NAME abgewiesen. Enthält der angegebene Name eckige Klammern mit anderem Inhalt als '+','-', '++', '--' wird er als Index eines Arrays interpretiert.

Wird mit Namensräumen gearbeitet, müssen alle Namensteile ausser dem letzten mit den entsprechenden Präfixen angegeben werden. Bei KXLFindNode kann im Gegensatz zu KXLRead im letzten Namensteil kein Namensraum-Präfix angegeben werden.

KXLFindNode liefert in der Struktur vom Typ `t_value` den Typ und den Wert des Datums zurück, die mit den im Abschnitt Typ-Konvertierungsfunktionen beschriebenen Funktionen umgewandelt werden können. Ist kein type-Attribut vorhanden, wird als Typ die leere Zeichenfolge zurückgegeben. Für innere Knoten entspricht das type="struct", für äußere type="string". Ist das zu lesende Element ein Attributknoten, wird der Inhalt des Attributes über t_value.pValue zurückgegeben. t_value.pType zeigt auf die leere Zeichenfolge. Im Fehlerfall enthält die zurückgegebene t_value Variable zwei NULL – Pointer.

Zusätzlich werden mit ppUrl und ppPrefix die Adressen von Zeichenfolgen zurückgegeben, die die Url und den aktuellen Prefix des Namensraumes enthalten, zu dem das gelesene Element gehört. Für den Default-Namensraum wird als Prefix die leere Zeichenfolge zurückgegeben. Im Fehlerfall und wenn das Element zu keinem Namensraum gehört, wird für Prefix und Url die leere Zeichenfolge zurückgegeben.

In ppNode wird der Pointer auf den gefundenen Knoten zurückgegeben. Im Fehlerfall wird der NULL – Pointer zurückgegeben.

Wie beim Schreiben kann das Lesen auf zwei Arten erfolgen: Bei Angabe des root-Knotens kann ein Datum über den vollqualifizierten Namen angesprochen werden, das Lesen kann aber auch mehrstufig erfolgen: Mit KXLSetSubObject kann im XML-Objekt positioniert werden. Mit dem so erhaltenen Zeiger auf ein SubObjekt kann mit KXLFindNode und dem Komponentennamen bzw. Feldindex (derForm [i]) auf die einzelnen Daten zugegriffen werden.

Bei Angabe eines leeren Namens (Leerzeichen) wird Typ und Wert des angegebenen Knotens (pNode) zurückgegeben.

Die Puffer, auf die die Zeiger in t_value, ppUrl und ppPrefix zeigen, werden beim nächsten KXLFindNode-Aufruf überschrieben. Ggf. müssen die Werte in eigenen Speicherbereichen gesichert werden.

Anmerkung:
Attribute, die Namensräume definieren, können mit KXLFindNode nicht gelesen werden. Siehe dazu KXLSearchNS.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_NO_NODE_FOUND (1)	I	Knoten mit dem angegebenen Namen nicht gefunden bzw. es existiert kein weiteres Element der angegebenen nodelist
KXL_RC_ALLOC_ERROR (11)	S	mit malloc/realloc konnte kein zusätzlicher Speicherplatz angefordert werden, kein Lesen möglich
KXL_RC_NO_XML_NODE (15)	A	der Zeiger auf den aktuellen Knoten ist NULL -> gültigen Zeiger angeben
KXL_RC_INVALID_NAME (19)	A	beim Analysieren des Elementnamens wurde ein Syntaxfehler bei der Verwendung von "[", "]" ,"/" und "@" entdeckt -> gültigen Namen angeben
KXL_RC_CONVERSION_ERROR (22)	S/A	Fehler bei der Code-Konvertierung
KXL_RC_ATTR_NAME_ERROR (25)	A	Der Aufbau des Attributnamens ist nicht korrekt; nach '@' keine '/', '[' oder ']' erlaubt
KXL_RC_ATTR_READ_WRITE_ ERR (26)	A	Fehler beim Lesen oder Schreiben eines Attributes
KXL_RC_NODE_TYPE_NOT_ ALLOWED (28)	A	bei diesem Aufruf ist nur die Angabe eines Element- bzw. Attributknotens erlaubt

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

Beispiel:

```
{   t_value tval;
    char ** ppUrl;
    char ** ppPrefix;
    xmlNodePtr pAct = pRoot;  // initiated with root node of document
    xmlNodePtr pNew;
    /* find all param child node */
    tval = KXLFindNode(pAct, "param", ppUrl, ppPrefix, &pNew, NULL);
    while (pNew != NULL)                    /* more children in subtree */
    {   /* output of name, value and namespace info of node */
        sprintf("value of {%s:%s}param = %s \n",
                ppPrefix, ppUrl, tval.pValue);
        /* read next param child node */
        tval = KXLFindNode(pNew, "[+]", ppUrl, ppPrefix, &pNew, NULL);
    }
    /* output of name, value and namespace info of node */
    sprintf("value of {%s:%s}param = %s \n", ppPrefix, ppUrl, tval.pValue);
```

Für das Dokument

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<application xmlns:utm="http://www.xml.test/test/utm"
         xmlns:fhs="http://www.xml.test/test/fhs"
         xmlns:uds="http://www.xml.test/test/uds"
         xmlns="http://www.xml.test/test/utm">
  <comment>openUTM application start parameter</comment>
  <param>START STARTNAME=TEST</param>
  <param>START FILEBASE=TEST.UTM</param>
  <utm:comment>format and database parameter</utm:comment>
  <fhs:param>MAPLIB=MAPLIB.FHS.TEST.UTM</fhs:param>
  <uds:param>DATABASE=UDSCONF</uds:param>
  <utm:comment>continue utm parameter</utm:comment>
  <utm:param>START ASYNTASKS=4</utm:param>
  <utm:param>START TASKS=5</utm:param>
</application>
```

werden folgende Meldungen ausgegeben:

```
value of {:http://www.xml.test/test/utm}param = START STARTNAME=TEST
value of {:http://www.xml.test/test/utm}param = START FILEBASE=TEST.UTM
value of {fhs:http://www.xml.test/test/fhs}param = MAPLIB=MAPLIB.FHS.TEST.UTM
value of {uds:http://www.xml.test/test/uds}param = DATABASE=UDSCONF
value of {utm:http://www.xml.test/test/utm}param = START ASYNTASKS=4
value of {utm:http://www.xml.test/test/utm}param = START TASKS=5
```


4.6.5 Abfragen der Größe einer Nodelist

Funktionsprototyp:

```
int     KXLGetSizeofNodelist(xmlNodePtr pNode, char *pName, short* pRetCode);
```

Parameter:

Typ	Name	Bemerkung
xmlNodePtr	pNode	In: Pointer auf einen Knoten des XML-Objekts.
char *	pName	In: Name der Nodelist
short*	pRetCode	Out: Returncode, wenn nicht NULL angegeben wurde

Funktionsergebnis:

Typ	Bemerkung
int	Anzahl der Elemente der nodelist

Diese Funktion liefert die Anzahl der Elemente der nodelist mit angegebenem Namen, die als Geschwisterknoten zu dem angegebenen Knoten existieren, zurück. Der Name darf keine Strukturelemente, wie /, [,], @ enthalten. Das Namensraum-Präfix muss mitangegeben werden, wenn vorhanden (siehe Abschnitt Namensräume).
Wird als Name der NULL-Pointer oder die leere Zeichenfolge angegeben, wird die Größe der nodelist mit Namen des aktuellen Knotens ermittelt.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_NO_NODE_FOUND (1)	I	kein Element der angegebenen nodelist gefunden
KXL_RC_NO_XML_NODE (15)	A	der Zeiger auf den aktuellen Knoten ist NULL -> gültigen Zeiger angeben
KXL_RC_INVALID_NAME (19)	A	beim Analysieren des Elementnamens wurde ein Syntaxfehler bei der Verwendung von "[", "]" ,"/" und "@" entdeckt -> gültigen Namen angeben

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

4.7 Zeichensatzbehandlung

4.7.1 Lesen des HomeZeichensatzes

Funktionsprototyp:
```
char*   KXLGetHomeEnc(short* pRetCode);
```

Parameter:
```
Typ     Name         Bemerkung
short* pRetCode     Out:  Returncode, wenn nicht NULL angegeben wurde
```

Funktionsergebnis:
```
Typ     Bemerkung
char*   (abdruckbar) Home Encoding
```

Diese Funktion liefert den Namen des aktuellen Home Encodings zurück.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

4.7.2 Lesen des Dokument-Zeichensatzes

Funktionsprototyp:
```
char*   KXLGetDocEnc(xmlNodePtr pNode, short* pRetCode);
```

Parameter:
```
Typ       Name       Bemerkung
xmlNodePtr pNode    In:   Pointer auf einen Knoten des XML-Objekts.
short*    pRetCode  Out:  Returncode, wenn nicht NULL angegeben wurde
```

Funktionsergebnis:
```
Typ     Bemerkung
char*   Name des Dokument-Encodings
```

Diese Funktion liefert den Encoding Namen des Dokumentes, zu dem der angegebene Knoten gehört, zurück. Beim nächsten Aufruf wird der Rückgabewert überschrieben.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_INVALID_DOC_PTR (10)	S/A	der angegebene Objektknoten ist keinem Objekt zugeordnet; möglicher Grund: Objekt zerstört oder fehlerhaft erzeugt ->Fehlerunterlagen erzeugen
KXL_RC_ALLOC_ERROR (11)	S	mit malloc/realloc konnte kein zusätzlicher Speicherplatz angefordert werden
KXL_RC_NO_XML_NODE (15)	A	der Zeiger auf den aktuellen Knoten ist NULL -> gültigen Zeiger angeben

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

4.7.3 Ändern des Dokument-Zeichensatzes

Funktionsprototyp:

```
void    KXLSetDocEnc(xmlNodePtr pNode, char* pEnc, short* pRetCode);
```

Parameter:

Typ	Name	Bemerkung	
xmlNodePtr	pNode	In:	Pointer auf einen Knoten des XML-Objekts.
char*	pEnc	In:	Name des Dokument Encodings
short*	pRetCode	Out:	Returncode, wenn nicht NULL angegeben wurde

Die Funktion ändert die Encoding-Angabe in dem Objekt, zu dem der angegebene Knoten gehört. Es erfolgt zusätzlich eine Prüfung, ob dem Parser diese Encoding-Angabe bekannt ist. Die Encoding-Angabe des Dokuments wird gelöscht, wenn `pEnc` NULL ist oder auf die leere Zeichenfolge zeigt.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_INVALID_DOC_PTR (10)	S/A	der angegebene Objektknoten ist keinem Objekt zugeordnet; möglicher Grund: Objekt zerstört oder fehlerhaft erzeugt ->Fehlerunterlagen erzeugen
KXL_RC_ALLOC_ERROR (11)	S	mit malloc/realloc konnte kein zusätzlicher Speicherplatz angefordert werden
KXL_RC_NO_XML_NODE (15)	A	der Zeiger auf den aktuellen Knoten ist NULL -> gültigen Zeiger angeben
KXL_RC_UNKNOWN_ENCODING (30)	A	angegebenes Encoding ist dem Parser nicht bekannt -> zum Encoding gehörige Konvertierungsroutinen beim Parser deklarieren

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

4.7.4 Konvertieren einer Zeichenfolge nach UTF-8

Funktionsprototyp:

```
char*   KXLStringToUTF8(char* pString, short* pRetCode);
```

Parameter:

Typ	Name	Bemerkung	
char*	pString	In:	Zeichenfolge im Home Encoding
short*	pRetCode	Out:	Returncode, wenn nicht NULL angegeben wurde

Funktionsergebnis:

Typ	Bemerkung
char*	Zeichenfolge in UTF-8

Die Funktion konvertiert die angegebene Zeichenfolge vom Home Encoding nach UTF-8. Es erfolgt zusätzlich eine Prüfung, ob dem Parser dieses Encoding bekannt ist.

Beim ersten Aufruf wird ggf. implizit die Funktion KXLInitEnv aufgerufen (s. Abschnitt 4.1 Initialisierung der UTM-XML-Schnittstelle).

Hinweis: Der Puffer, in dem die konvertierte Zeichenfolge zurückgegeben wird, wird beim nächsten Aufruf von KXLStringToUTF8 überschrieben.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_ALLOC_ERROR (11)	S	mit malloc/realloc konnte kein zusätzlicher Speicherplatz angefordert werden
KXL_RC_VERSION_ERR (23)	A	Versionsstände der UTM-XML-Komponenten inkonsistent
KXL_RC_PARSER_VERS_NOT_ SUPP (24)	A	alte Version des XML-Parsers eingebunden (< 2.6.20) -> neuere Parser-Version einbinden
KXL_RC_UNKNOWN_ENCODING (30)	A	Home Encoding ist dem Parser nicht bekannt -> zum Encoding gehörige Konvertierungs-routinen beim Parser deklarieren
KXL_RC_INVALID_PARAMETER (32)	A	Parameterangabe nicht korrekt , z.B. NULL-Zeiger angegeben, obwohl nicht erlaubt
KXL_RC_ENCODING_CHANGE_ ERR (36)	A/S	für das Home Encoding konnte kein Encoding Handler eingerichtet werden

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

4.7.5 Konvertieren einer Zeichenfolge von UTF-8 ins Home Encoding

Funktionsprototyp:

```
char*  KXLStringFromUTF8(char* pString, short* pRetCode);
```

Parameter:

```
Typ       NameBemerkung
char*     pString    In:   Zeichenfolge in UTF-8
short*    pRetCode   Out:  Returncode, wenn nicht NULL angegeben wurde
```

Funktionsergebnis:

```
Typ     Bemerkung
char*   Zeichenfolge im Home Encoding
```

Die Funktion konvertiert die angegebene Zeichenfolge von UTF-8 ins Home Encoding. Es erfolgt zusätzlich eine Prüfung, ob dem Parser dieses Encoding bekannt ist.
Beim ersten Aufruf wird ggf. implizit die Funktion KXLInitEnv aufgerufen (s. Abschnitt 4.1 Initialisierung der UTM-XML-Schnittstelle).

Hinweis: Der Puffer, in dem die konvertierte Zeichenfolge zurückgegeben wird, wird beim nächsten Aufruf von KXLStringFromUTF8 überschrieben.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_ALLOC_ERROR (11)	S	mit malloc/realloc konnte kein zusätzlicher Speicherplatz angefordert werden
KXL_RC_VERSION_ERR (23)	A	Versionsstände der UTM-XML-Komponenten inkonsistent
KXL_RC_PARSER_VERS_NOT_ SUPP (24)	A	alte Version des XML-Parsers eingebunden (< 2.6.20) -> neuere Parser-Version einbinden
KXL_RC_UNKNOWN_ENCODING (30)	A	Home Encoding ist dem Parser nicht bekannt -> zum Encoding gehörige Konvertierungs-routinen beim Parser deklarieren
KXL_RC_INVALID_PARAMETER (32)	A	Parameterangabe nicht korrekt , z.B. NULL-Zeiger angegeben, obwohl nicht erlaubt

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

4.7.6 Lesen des Original Encodings zum angegebenen Alias-Namen

Funktionsprototyp:

```
char*  KXLGetEncodingAlias(char* alias, short* pRetCode);
```

Parameter:

Typ	Name	Bemerkung
char*	alias	In: Alias-Name eines Encodings
short*	pRetCode	Out: Returncode, wenn nicht NULL angegeben wurde

Funktionsergebnis:

Typ	Bemerkung
char*	Original Encoding Name

Die Funktion sucht das Encoding, dem der angegebene Alias-Name zugeordnet ist. Zurückgegeben wird der Name des Original-Encodings oder NULL, wenn der Alias-Name nicht bekannt ist.

Beim ersten Aufruf wird ggf. implizit die Funktion KXLInitEnv aufgerufen (s. Abschnitt 4.1 Initialisierung der UTM-XML-Schnittstelle).

Hinweise:

1. Der Puffer, in dem die konvertierte Zeichenfolge zurückgegeben wird, wird beim nächsten Aufruf von KXLGetEncodingAlias überschrieben.
2. Zur Behandlung verschiedener Zeichensätze und deren Namen siehe auch Abschnitt 3.9 Zeichensätze/Encoding.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_ALLOC_ERROR (11)	S	mit malloc/realloc konnte kein zusätzlicher Speicherplatz angefordert werden
KXL_RC_CONVERSION_ERROR (22)	S/A	Fehler bei der Code-Konvertierung
KXL_RC_VERSION_ERR (23)	A	Versionsstände der UTM-XML-Komponenten inkonsistent
KXL_RC_PARSER_VERS_NOT_ SUPP (24)	A	alte Version des XML-Parsers eingebunden (< 2.6.20) -> neuere Parser-Version einbinden
KXL_RC_UNKNOWN_ENCODING (30)	A	Home Encoding ist dem Parser nicht bekannt -> zum Encoding gehörige Konvertierungs-routinen beim Parser deklarieren
KXL_RC_INVALID_PARAMETER (32)	A	Parameterangabe nicht korrekt , z.B. NULL-Zeiger angegeben, obwohl nicht erlaubt

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

4.7.7 Zuordnen eines Alias-Namens zu einem Original Encoding

Funktionsprototyp:

```
void    KXLSetEncodingAlias(char* alias, char* encName, short* pRetCode);
```

Parameter:

Typ	Name	Bemerkung
char*	alias	In: Alias-Name eines Encodings
char*	encName	In: Original-Name eines Encodings
short*	pRetCode	Out: Returncode, wenn nicht NULL angegeben wurde

Die Funktion ordnet den angegebenen Alias-Namen dem gewünschten Encoding zu. Ist für `encName` NULL angegeben, wird die aktuelle Zuordnung des Alias-Namens gelöscht.
Beim ersten Aufruf wird ggf. implizit die Funktion KXLInitEnv aufgerufen (s. Abschnitt 4.1 Initialisierung der UTM-XML-Schnittstelle).

Hinweise:

1. Bei `encName` darf nur ein definierter Original-Encoding Name angegeben werden. Alias-Namen werden abgelehnt.
2. Wenn bei `alias` der Name eines definierten Original encodings angegeben wird, wird bei Dokumenten mit diesem Encoding der Encoding Handler des Encodings verwendet, dem `alias` zugeordnet wurde.
3. Die Deklaration von Alias-Namen erfolgt task- bzw. prozessspezifisch. D.h. bei Verwendung der UTM-XML-Schnittstelle in einer UTM-Anwendung muss dieser Aufruf in jedem Task bzw. Prozess, z.B. in einem Start-Exit, erfolgen.
4. Zur Behandlung verschiedener Zeichensätze und deren Namen siehe auch Abschnitt 3.9 Zeichensätze/Encoding.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_ALLOC_ERROR (11)	S	mit malloc/realloc konnte kein zusätzlicher Speicherplatz angefordert werden
KXL_RC_CONVERSION_ERROR (22)	S/A	Fehler bei der Code-Konvertierung
KXL_RC_VERSION_ERR (23)	A	Versionsstände der UTM-XML-Komponenten inkonsistent
KXL_RC_PARSER_VERS_NOT_ SUPP (24)	A	alte Version des XML-Parsers eingebunden (< 2.6.20) -> neuere Parser-Version einbinden
KXL_RC_UNKNOWN_ENCODING (30)	A	(Home) Encoding ist dem Parser nicht bekannt -> vorher zum Encoding gehörige Konvertierungs-routinen beim Parser deklarieren
KXL_RC_INVALID_PARAMETER (32)	A	Parameterangabe nicht korrekt , z.B. NULL-Zeiger angegeben, obwohl nicht erlaubt
KXL_RC_PARSER_ENCODING_E RR (46)	S/A	Parser konnte Funktion nicht ausführen -> überprüfen, ob beim Setzen der Encoding-Name bzw. beim Löschen der Alias-Name definiert ist.
KXL_RC_IS_ALIAS_ENCODING (47)	A	angegebene Encoding-Name ist Alias-Name -> Original Encoding Name angeben

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

4.8 Namensraum-Verwaltung

4.8.1 Schreiben einer Namensraum-Definition

Funktionsprototyp:

```
xmlNsPtr     KXLWriteNS (xmlNodePtr pNode, char* prefix, char* url,
                         short* pRetCode )
```

Parameter:

Typ	Name	Bemerkung
xmlNodePtr	pNode	In: Pointer auf einen Knoten des XML-Objekts
char*	prefix	In: Präfix der Namensraum-Definition
char*	url	In: Url der Namensraum-Definition
short*	pRetCode	Out: Returncode, wenn nicht NULL angegeben wurde

Funktionsergebnis:

Typ	Bemerkung
xmlNsPtr	Pointer auf die geschriebene Namensraum-Definition

Mit der Funktion `KXLWriteNS` kann ein Namensraum definiert bzw. umdefiniert werden. D.h. einem Namensraum-Präfix prefix wird eine bestimmte Url url zugeordnet. Diese Namensraum-Definition wird einem bestimmten Elementknoten pNode des XML-Objekts zugeordnet.

Prefix und url dürfen nicht NULL-Pointer sein. Für die Definition eines Default-Namensraumes ist für prefix die leere Zeichenfolge anzugeben.

Ist diesem Knoten schon eine Namensraum-Definition mit dem angegebenen Präfix zugeordnet, wird diese überschrieben. Ist diesem Knoten noch keine Namensraum-Definition mit dem angegebenen Präfix zugeordnet, wird ein neuer Namensraum-Eintrag angelegt, und der gesamte Unterbaum nach Elementen und Attributen mit diesem Präfix durchsucht. Sind sie einer Dummy-Definition zugeordnet, werden diese durch Verweise auf die neu erstellte Namensraum-Definition ersetzt.

Der Eintrag, auf den der zurückgegebene Zeiger zeigt, enthält Verweise auf UTF-8-Zeichenfolgen. Zum Konvertieren ins Home Encoding steht die Funktion `KXLStringFromUTF8` zur Verfügung.

Zur Verwendung von Namensräumen und deren Gültigkeitsbereich siehe Abschnitt Namensräume.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_ALLOC_ERROR (11)	S	mit malloc/realloc konnte kein zusätzlicher Speicherplatz angefordert werden, kein Lesen möglich
KXL_RC_NO_XML_NODE (15)	A	der Zeiger auf den aktuellen Knoten ist NULL -> gültigen Zeiger angeben
KXL_RC_CONVERSION_ERRO R (22)	S/A	Fehler bei der Code-Konvertierung
KXL_RC_NODE_TYPE_NOT_AL LOWED (28)	A	bei diesem Aufruf ist nur die Angabe eines Elementknotens erlaubt
KXL_RC_INVALID_PARAMETER (32)	A	Parameterangabe nicht korrekt , z.B. NULL-Zeiger angegeben, obwohl nicht erlaubt
KXL_RC_NAMESPACE_WRITE_ ERROR (33)	S/A	Der Parser konnte die Namensraum-Definition nicht ordnungsgemäß schreiben, siehe Parser-Fehlermeldungen

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

4.8.2 Löschen einer Namensraum-Definition

Funktionsprototyp:

```
xmlNodePtr KXLDelNS (xmlNodePtr pNode, char* prefix, short* pRetCode )
```

Parameter:

Typ	Name	Bemerkung	
xmlNodePtr	pNode	In:	Pointer auf einen Knoten des XML-Objekts.
char*	prefix	In:	Präfix der Namensraum-Definition
short*	pRetCode	Out:	Returncode, wenn nicht NULL angegeben wurde

Funktionsergebnis:

Typ	Bemerkung
xmlNodePtr	Pointer auf einen Knoten des XML-Objekts oder NULL

Mit der Funktion KXLDelNS kann ein Namensraum mit dem Präfix `prefix`, der einem bestimmten Elementknoten `pNode` zugeordnet ist, gelöscht werden.

`prefix` darf nicht der NULL-Pointer sein. Für das Löschen eines Default-Namensraumes ist für `prefix` die leere Zeichenfolge anzugeben.

Ist diesem Knoten keine Namensraum-Definition mit dem angegebenen Präfix zugeordnet, wird ein entsprechender Returncode zurückgegeben.

Gibt es in dem Unterbaum des angegebenen Knotens einen Element- oder Attributknoten, der auf den zu löschenden Namensraum verweist, wird das Löschen nicht durchgeführt und der Pointer des gefundenen Knotens zurückgegeben. Andernfalls wird der NULL-Pointer zurückgegeben.

Zur Verwendung von Namensräumen siehe Abschnitt Namensräume.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_ALLOC_ERROR (11)	S	mit malloc/realloc konnte kein zusätzlicher Speicherplatz angefordert werden, kein Lesen möglich
KXL_RC_NO_XML_NODE (15)	A	der Zeiger auf den aktuellen Knoten ist NULL -> gültigen Zeiger angeben
KXL_RC_CONVERSION_ERROR (22)	S/A	Fehler bei der Code-Konvertierung
KXL_RC_NODE_TYPE_NOT_ALLOWED (28)	A	bei diesem Aufruf ist nur die Angabe eines Elementknotens erlaubt
KXL_RC_NAMESPACE_NOT_FOUND (31)	I	Gesuchten Namensraum nicht gefunden
KXL_RC_INVALID_PARAMETER (32)	A	Parameterangabe nicht korrekt , z.B. NULL-Zeiger angegeben, obwohl nicht erlaubt
KXL_RC_NAMESPACE_DEL_ERROR (34)	A	Die Definition kann nicht gelöscht werden, da sie noch von mindestens einem Element- oder Attributknoten referenziert wird.

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

4.8.3 Lesen einer Liste von Namensraum-Definitionen

Funktionsprototyp:

```
xmlNsPtr*     KXLReadNSList (xmlNodePtr pNode, short* pRetCode )
```

Parameter:

Typ	Name	Bemerkung	
xmlNodePtr	pNode	In:	Pointer auf einen Knoten des XML-Objekts.
short*	pRetCode	Out:	Returncode, wenn nicht NULL angegeben wurde

Funktionsergebnis:

Typ	Bemerkung
XmlNsPtr*	Pointer auf ein array von Namensraum-Definitionen

Mit der Funktion KXLReadNSList kann für einen bestimmten Elementknoten pNode die Liste aller für ihn relevanten Namensraum-Definitionen ausgegeben werden. D.h. es wird ein array von Pointern zurückgegeben, die auf die Namensraum-Definitionen zeigen, die für den angegebenen Knoten gültig sind, also nicht nur eigene, sondern auch bei Elternknoten definierte Namensräume. Es werden auch dummy-Einträge (mit leerer url/href) zurückgegeben. Der Aufbau eines solchen Namensraum-Eintrags ist im Parser-Headerfile libxml/tree.h als Struktur xmlNs definiert.

Die Einträge, auf die der zurückgegebene Zeiger zeigt, enthält Verweise auf UTF-8-Zeichenfolgen. Zum Konvertieren ins Home Encoding steht die Funktion KXLStringFromUTF8 zur Verfügung.

Zur Verwendung von Namensräumen siehe Abschnitt Namensräume.

Achtung: Das array von Pointern auf die Namensraum-Definitionen, das von dem Aufruf zurückgegeben wird, muss vom Anwender mit xmlFree (Parserfunktion) explizit freigegeben werden.

Returncodes:

Name (Wert)	Fehler -typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_INVALID_DOC_PTR (10)	S/A	der angegebene Objektknoten ist keinem Objekt zugeordnet; möglicher Grund: Objekt zerstört oder fehlerhaft erzeugt ->Fehlerunterlagen erzeugen
KXL_RC_NO_XML_NODE (15)	A	der Zeiger auf den aktuellen Knoten ist NULL -> gültigen Zeiger angeben
KXL_RC_NODE_TYPE_NOT_ ALLOWED (28)	A	bei diesem Aufruf ist nur die Angabe eines Elementknotens erlaubt
KXL_RC_NAMESPACE_NOT_ FOUND (31)	I	Gesuchten Namensraum nicht gefunden

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

Beispiel:

```
xmlNodePtr pNode;
xmlNsPtr * pNScur;
int i;
short rcode;
pnode = …;
/* read all namespaces valid for pNode */
pNScur = KXLReadNSList (pNode, &rcode);
if (rcode == KXL_RC_OK)
{ /* build output */
  printf ("namespaces found:\n");
  for (i=0;pNScur[i] != NULL; i++)
  { /* loop over all namespaces found */
    if (pNScur[i]->prefix == NULL)
      printf ("<default> : ");
    else
      printf ("%s  : ", pNScur[i]->prefix);
    printf ("%s \n", pNScur[i]->href);
  }
```


4.8.4 Suchen einer Namensraum-Definition

Funktionsprototyp:

```
xmlNsPtr      KXLSearchNS (xmlNodePtr pNode, char* prefix, char* url,
              short* pRetCode )
```

Parameter:

Typ	Name	Bemerkung
xmlNodePtr	pNode	In: Pointer auf einen Knoten des XML-Objekts.
char*	prefix	In: Präfix der gesuchten Namensraum-Definition
char*	url	In: Url der gesuchten Namensraum-Definition
short*	pRetCode	Out: Returncode, wenn nicht NULL angegeben wurde

Funktionsergebnis:

Typ	Bemerkung
xmlNsPtr	Pointer auf die gefundene Namensraum-Definition

Mit der Funktion KXLSearchNS wird für einen bestimmten Elementknoten pNode eine für ihn relevante Namensraum-Definition gesucht.

Ist die Url url angegeben, wird nach der Namensraum-Definition mit der angegebenen Url gesucht. Ist als url der NULL-Pointer angegeben, wird nach der Namensraum-Definition mit angegebenem Präfix prefix gesucht. Ist diese auch leer oder der NULL-Pointer, wird nach dem für den Elementknoten gültigen Default-Namensraum gesucht. Zurückgegeben wird der Pointer auf den gefundenen Namensraum-Eintrag oder NULL, wenn kein passender gefunden wurde. Namensraum-Einträge mit leerer Url werden nicht zurückgegeben.

Der Aufbau eines solchen Namensraum-Eintrags ist im Parser-Headerfile libxml/tree.h als Struktur xmlNs definiert.

Der Eintrag, auf den der zurückgegebene Zeiger zeigt, enthält Verweise auf UTF-8-Zeichenfolgen. Zum Konvertieren ins Home Encoding steht die Funktion KXLStringFromUTF8 zur Verfügung.

Zur Verwendung von Namensräumen siehe Abschnitt Namensräume.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_INVALID_DOC_PTR (10)	S/A	der angegebene Objektknoten ist keinem Objekt zugeordnet mögl. Grund: Objekt zerstört oder fehlerhaft erzeugt ->Fehlerunterlagen erzeugen
KXL_RC_ALLOC_ERROR (11)	S	mit malloc/realloc konnte kein zusätzlicher Speicherplatz angefordert werden, kein Lesen möglich
KXL_RC_NO_XML_NODE (15)	A	der Zeiger auf den aktuellen Knoten ist NULL -> gültigen Zeiger angeben
KXL_RC_CONVERSION_ERRO R (22)	S/A	Fehler bei der Code-Konvertierung
KXL_RC_NODE_TYPE_NOT_ ALLOWED (28)	A	bei diesem Aufruf ist nur die Angabe eines Elementknotens erlaubt
KXL_RC_NAMESPACE_NOT_ FOUND (31)	I	Gesuchten Namensraum nicht gefunden
KXL_RC_INVALID_PARAMETER (32)	A	Parameterangabe nicht korrekt , z.B. NULL-Zeiger angegeben, obwohl nicht erlaubt

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

4.9 XML Schema Validierung

4.9.1 Konvertierung eines XML-Dokuments in ein XML-Objekt mit Schemavalidierung

Funktionsprototyp:

```
xmlNodePtr    KXLConvDocToObjAndValid (char* pBuffer, int validate,
                                       char* pDir, short* pRetCode);
```

Parameter:

Typ	Name	Bemerkung
char*	pBuffer	In: Puffer mit dem XML-Dokument
int	validFlag	In: gibt an, ob validiert werden soll oder nicht
char*	pDir	In: Name des Directories für Dateireferenzierungen
short*	pRetCode	Out: Returncode, wenn nicht NULL angegeben wurde

Funktionsergebnis:

Typ	Bemerkung
xmlNodePtr	Pointer auf den root-Knoten des XML-Objekts

Zur Bearbeitung muss ein XML-Dokument in ein XML-Objekt konvertiert werden. Dazu kann die Funktion KXLConvDocToObjAndValid aufgerufen werden, die als Eingabeparameter den Puffer mit dem kompletten (\0 terminierten!) XML-Dokument hat. Wenn gewünscht, wird das XML-Dokument gegen das erste Schema, das im Dokument bei schemaLocation angegeben ist, validiert. Das Ergebnis der Funktion ist der Zeiger auf den root-Knoten des XML-Objekts. Im Fehlerfall, z.B. wenn ein nicht gültiges Dokument (not valid) vorliegt, wird NULL zurückgegeben.

Enthält ein zu konvertierendes Dokument eine Encoding-Angabe, wird dies im Objektbaum hinterlegt, alle Namen und Inhalte werden nach UTF-8 konvertiert und im Objektbaum abgelegt.
Ist keine Encoding-Angabe vorhanden, wird UTF-8 angenommen. Für die vorliegende Encoding-Angabe muss eine Codekonvertierungsroutine (von und nach UTF-8) deklariert sein, andernfalls wird ein Fehler zurückgegeben und kein Objekt erzeugt (siehe dazu Abschnitt Zeichensätze).

Enthält das Dokument ein schemaLocation Element und ist der Parameter validate nicht 0, wird das XML-Dokument gegen das erste in schemaLocation angegebene XML-Schema validiert. Dabei werden alle angegebenen URLs (d.h. die bei schemaLocation und dort bei include, import, redefine… angegebenen) in lokale Dateinamen umgewandelt, wenn der Parameter pDir nicht NULL ist (siehe 3.11.1 Entities Loader). Das Schema wird intern geparst, der verwendete Speicher jedoch bei Beendigung der Funktion wieder freigegeben. D.h. das Schema kann nicht weiter bearbeitet werden.
Ist das XML-Dokument nicht wohlgeformt oder keine gültige Instanz des Schemas, wird ein entsprechender Returncode zurückgegeben (Returncodes 20, 38, 39, 44). Genauere Informationen über den Fehler kann man über KXLGetLastParserError abfragen oder in der Tracedatei verfolgen.

Ist der Parameter validate = 0, entspricht die Funktionalität der Funktion KXLConvDocToObj.

Beim ersten Aufruf wird ggf. implizit die Funktion KXLInitEnv aufgerufen (s. Abschnitt 4.1 Initialisierung der UTM-XML-Schnittstelle). Bei Fehler wird kein Objekt erzeugt und ein entsprechender Returncode zurückgegeben.

Returncodes:

Name (Wert)	Fehler -typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt; Dokument ist Instanz des angegebenen Schemas
KXL_RC_NO_NODE_FOUND (1)	I	Angabe der SchemaLocation nicht gefunden
KXL_RC_NO_MALLOC_FOR_ PARSER_1(5)	S	nicht genug Speicherplatz für das Objekt vorhanden -> Programmspeicher vergrößern
KXL_RC_NO_MALLOC_FOR_ PARSER_2(6)	S	nicht genug Speicherplatz für das Objekt vorhanden -> Programmspeicher vergrößern

Name (Wert)	Fehler -typ	Bemerkung -> Maßnahme
KXL_RC_NO_CODE_ CONVERSION (8)	A	Beginn des Dokuments entspricht nicht dem XML Start tag '<?xml' oder '<?XML' in EBCDIC oder ASCII ->Dokument korrigieren bzw. Codekonvertierung vornehmen
KXL_RC_EMPTY_DOC (9)	A	konvertiertes Objekt enthält keine Knoten
KXL_RC_ALLOC_ERROR (11)	S	mit malloc/realloc konnte kein zusätzlicher Speicherplatz angefordert werden
KXL_RC_NO_XML_NODE (15)	S	interner Fehler beim Lesen der SchemaLocation
KXL_RC_INVALID_NAME (19)	S	interner Fehler beim Lesen der SchemaLocation
KXL_RC_PARSER_ERROR (20)	S/A	Parser hat einen Fehler entdeckt ->siehe Meldung des Parsers. (Tracedatei oder KXLGetLastParserError)
KXL_RC_NO_CONTEXT_FOR_ PARSER (21)	S	Fehler beim Anlegen parserinterner Verwaltungsbereiche -> evtl. Speicherengpass beseitigen
KXL_RC_CONVERSION_ERROR (22)	S/A	Fehler bei der Code-Konvertierung
KXL_RC_VERSION_ERR (23)	A	Versionsstände der UTM-XML-Komponenten inkonsistent
KXL_RC_PARSER_VERS_NOT_ SUPP (24)	A	alte Version des XML-Parsers eingebunden (< 2.6.20) -> neuere Parser-Version einbinden
KXL_RC_NODE_TYPE_NOT_ ALLOWED (28)	S	interner Fehler beim Lesen der SchemaLocation
KXL_RC_UNKNOWN_ ENCODING (30)	A	Encoding-Name ist dem Parser nicht bekannt -> zum Encoding gehörige Konvertierungsroutinen beim Parser deklarieren
KXL_RC_NAMESPACE_NOT_ FOUND (31)	I	Gesuchten Namensraum nicht gefunden
KXL_RC_INVALID_PARAMETER (32)	A	Parameterangabe nicht korrekt , z.B. NULL-Zeiger angegeben, obwohl nicht erlaubt
KXL_RC_ENCODING_CHANGE_ ERR (36)	A/S	für das angegebene Home Encoding konnte kein Encoding Handler eingerichtet werden
KXL_RC_PARSER_CONTEXT_ ERR (37)	S	Parser-interner Fehler beim Erzeugen von Parser-internen Verwaltungsbereichen -> evtl. Speicherengpass beseitigen
KXL_RC_SCHEMA_PARSE_INT_ ERR (38)	A	Fehler beim Parsen des Schemas ->siehe Meldung bzw. Returncode des Parsers. (Tracedatei oder KXLGetLastParserError)
KXL_RC_SCHEMA_VALID_INT_ ERR (39)	A	Fehler beim Validieren des Schemas ->siehe Meldung bzw. Returncode des Parsers. (Tracedatei oder KXLGetLastParserError)
KXL_RC_NO_SCHEMA_GIVEN (40)	A	die Schema-Eingabe-Adresse ist NULL
KXL_RC_DOC_NOT_VALID(44)	A	das Dokument ist keine gültige Instanz des angegebenen Schemas ->siehe Meldung des Parsers. (Tracedatei oder KXLGetLastParserError)

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

4.9.2 Parsen eines XML Schemas aus dem Speicher

Funktionsprototyp:

```
xmlSchemaPtr  KXLParseSchema(char* pBuffer, char* pDir, short* pRetCode);
```

Parameter:

Typ	Name	Bemerkung
char*	pBuffer	In: Puffer mit dem XML-Schema-Dokument
char*	pDir	In: Dateiverzeichnis für referenzierte Dateien
short*	pRetCode	Out: Returncode, wenn nicht NULL angegeben wurde

Funktionsergebnis:

Typ	Bemerkung
xmlSchemaPtr	Pointer auf die XML-Schema-Struktur

Diese Funktion prüft, ob der Eingabepuffer ein gültiges XML Schema enthält. Dabei wird intern ein Dokumentenbaum für dieses Schema aufgebaut und weitere Verwaltungsdaten in der internen XML-Schema-Struktur abgelegt. Wenn der Parameter pDir nicht NULL ist (siehe 3.11.1 Entities Loader), werden alle angegebenen URLs, z.B. in einer include- oder import-Anweisung, in lokale Dateinamen umgewandelt. Mit der zurückgegebenen Adresse xmlSchemaPtr, die auf diesen Bereich verweist, kann die Validierung eines XML-Dokumentes gegen dieses Schema mit KXLValidDoc bzw. KXLValidDocBuf angestoßen werden.

Ist das XML Schema nicht wohlgeformt, wird ein entsprechender Returncode zurückgegeben (KXL_RC_SCHEMA_PARSE_INT_ERR (38)). Genauere Informationen über den Fehler kann man über KXLGetLastParserError abfragen oder in der Tracedatei verfolgen.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_EMPTY_DOC(9)	A	leerer Eingabepuffer
KXL_RC_VERSION_ERR (23)	A	Versionsstände der UTM-XML-Komponenten inkonsistent
KXL_RC_PARSER_VERS_NOT_ SUPP (24)	A	alte Version des XML-Parsers eingebunden (< 2.6.20) -> neuere Parser-Version einbinden
KXL_RC_UNKNOWN_ ENCODING (30)	A	Encoding-Name ist dem Parser nicht bekannt -> zum Encoding gehörige Konvertierungsroutinen beim Parser deklarieren
KXL_RC_INVALID_PARAMETER (32)	A	Adresse des Eingabepuffers ist NULL
KXL_RC_ENCODING_CHANGE _ERR (36)	A/S	für das angegebene Home Encoding konnte kein Encoding Handler eingerichtet werden
KXL_RC_PARSER_CONTEXT_ ERR (37)	S	Parser-interner Fehler beim Erzeugen von Parser-internen Verwaltungsbereichen -> evtl. Speicherengpass beseitigen
KXL_RC_SCHEMA_PARSE_INT _ERR (38)	A	Fehler beim Parsen des Schemas ->siehe Meldung bzw. Returncode des Parsers. (Tracedatei oder KXLGetLastParserError)

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

4.9.3 Parsen eines XML Schemas aus einer Datei

Funktionsprototyp:

```
xmlSchemaPtr KXLParseSchemaFile(char* pFilename, char* pDir, short*
pRetCode);
```

Parameter:

Typ	Name	Bemerkung
char*	pFilename	In: Datei mit dem XML-Schema-Dokument
char*	pDir	In: Dateiverzeichnis für referenzierte Dateien
short*	pRetCode	Out: Returncode, wenn nicht NULL angegeben wurde

Funktionsergebnis:

Typ	Bemerkung
xmlSchemaPtr	Pointer auf die XML-Schema-Struktur

Diese Funktion prüft, ob die angegebene Datei pFilename ein gültiges XML Schema enthält. Dabei wird intern ein Dokumentenbaum für dieses Schema aufgebaut und weitere Verwaltungsdaten in der internen XML-Schema-Struktur abgelegt. Wenn der Parameter pDir nicht NULL ist (siehe 3.11.1 Entities Loader), werden alle angegebenen URLs, z.B. in einer include- oder import-Anweisung, in lokale Dateinamen umgewandelt. Der Dateiname pFilename wird grundsätzlich nicht auf pDir umgelenkt. Mit der zurückgegebenen Adresse xmlSchemaPtr, die auf diesen Bereich verweist, kann die Validierung eines XML-Dokumentes gegen dieses Schema mit KXLValidDoc bzw. KXLValidDocBuf angestoßen werden.

Ist das XML Schema nicht wohlgeformt, wird ein entsprechender Returncode zurückgegeben (KXL_RC_SCHEMA_PARSE_INT_ERR (38)). Genauere Informationen über den Fehler kann man über KXLGetLastParserError abfragen oder in der Tracedatei verfolgen.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_VERSION_ERR (23)	A	Versionsstände der UTM-XML-Komponenten inkonsistent
KXL_RC_PARSER_VERS_NOT_SUPP (24)	A	alte Version des XML-Parsers eingebunden (< 2.6.20) -> neuere Parser-Version einbinden
KXL_RC_INVALID_PARAMETER (32)	A	Adresse des Dateinamens ist NULL
KXL_RC_PARSER_CONTEXT_ERR (37)	S	Parser-interner Fehler beim Erzeugen von Parser-internen Verwaltungsbereichen -> evtl. Speicherengpass beseitigen
KXL_RC_SCHEMA_PARSE_INT_ERR (38)	A	Fehler beim Parsen des Schemas ->siehe Meldung bzw. Returncode des Parsers. (Tracedatei oder KXLGetLastParserError)

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

4.9.4 Validieren eines XML-Objekts gegen ein XML Schema

Funktionsprototyp:

```
void   KXLValidDoc(xmlNodePtr pNode, xmlSchemaPtr pSchema, short* pRetCode);
```

Parameter:

Typ	Name	Bemerkung
xmlNodePtr	pNode	In: Pointer auf den Knoten des XML-Objekts
xmlSchemaPtr	pSchema	In: Pointer auf die XML-Schema-Struktur
short*	pRetCode	Out: Returncode (Pflichtparameter!)

Funktionsergebnis:

keines

Diese Funktion prüft, ob das XML-Dokument, das von dem unter xmlNodePtr gegebenen XML-Objekt repräsentiert wird, eine Instanz des unter xmlSchemaPtr vorgegebenen XML Schema ist. Für diese Funktion muss ein Returncodefeld angegeben werden.

Tritt beim Validieren ein Fehler auf, wird ein entsprechender Returncode zurückgegeben (Returncodes 39, 44). Genauere Informationen über den Fehler kann man über KXLGetLastParserError abfragen oder in der Tracedatei verfolgen.

Wird der Pflichtparameter pRetcode nicht angegeben (=NULL), wird eine Fehlermeldung in die Tracedatei ausgegeben.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_INVALID_DOC_PTR (10)	S/A	der angegebene Objektknoten ist keinem Objekt zugeordnet; mögl. Grund: Objekt zerstört oder fehlerhaft erzeugt ->Fehlerunterlagen erzeugen
KXL_RC_NO_XML_NODE(15)	A	der Zeiger auf den aktuellen Knoten ist NULL -> gültigen Zeiger angeben
KXL_RC_PARSER_CONTEXT_ ERR (37)	S	Parser-interner Fehler beim Erzeugen von Parser-internen Verwaltungsbereichen -> evtl. Speicherengpass beseitigen
KXL_RC_SCHEMA_VALID_INT_ ERR (39)	A	Fehler beim Validieren ->siehe Meldung bzw. Returncode des Parsers. (Tracedatei oder KXLGetLastParserError)
KXL_RC_NO_SCHEMA_GIVEN (40)	A	die Schema-Eingabe-Adresse ist NULL
KXL_RC_DOC_NOT_VALID (44)	A	das Dokument ist keine gültige Instanz des angegebenen Schemas ->siehe Meldung des Parsers. (Tracedatei oder KXLGetLastParserError)

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

4.9.5 Validieren eines XML-Dokuments im Speicher gegen ein XML Schema

Funktionsprototyp:

```
void   KXLValidDocBuf(char* pDocBuf, xmlSchemaPtr pSchema, short* pRetCode);
```

Parameter:

Typ	Name	Bemerkung
char*	pDocBuf	In: Puffer, der das XML-Dokument enthält
xmlSchemaPtr	pSchema	In: Pointer auf die XML-Schema-Struktur
short*	pRetCode	Out: Returncode (Pflichtparameter!)

Funktionsergebnis:

keines

Diese Funktion prüft, ob das XML-Dokument, das im Puffer pDocBuf steht, eine Instanz des unter xmlSchemaPtr vorgegebenen XML Schema ist. Für diese Funktion muss ein Returncodefeld angegeben werden.

Tritt beim Validieren ein Fehler auf, wird ein entsprechender Returncode zurückgegeben (Returncodes 39, 44). Genauere Informationen über den Fehler kann man über KXLGetLastParserError abfragen oder in der Tracedatei verfolgen.

Wird der Pflichtparameter pRetcode nicht angegeben (=NULL), wird eine Fehlermeldung in die Tracedatei ausgegeben.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_EMPTY_DOC(9)	A	leerer Eingabepuffer
KXL_RC_INVALID_PARAMETER (32)	A	die Adresse des Eingabepuffers ist NULL
KXL_RC_PARSER_CONTEXT_ERR (37)	S	Parser-interner Fehler beim Erzeugen von Parser-internen Verwaltungsbereichen -> evtl. Speicherengpass beseitigen
KXL_RC_SCHEMA_VALID_INT_ERR (39)	A	Fehler beim Validieren ->siehe Meldung bzw. Returncode des Parsers. (Tracedatei oder KXLGetLastParserError)
KXL_RC_NO_SCHEMA_GIVEN(40)	A	die Schema-Eingabe-Adresse ist NULL
KXL_RC_DOC_NOT_VALID(44)	A	das Dokument ist keine gültige Instanz des angegebenen Schemas ->siehe Meldung des Parsers. (Tracedatei oder KXLGetLastParserError)

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

4.9.6 Freigeben eines Schema-Baumes im Speicher

Funktionsprototyp:

```
void    KXLFreeSchema(xmlSchemaPtr pSchema, short* pRetCode);
```

Parameter:

Typ	Name	Bemerkung
xmlSchemaPtr	pSchema	In: Pointer auf die XML-Schema-Struktur
short*	pRetCode	Out: Returncode, wenn nicht NULL angegeben wurde

Funktionsergebnis:

keines

Mit diesem Aufruf wird der für das XML-Schema benötigte Speicher freigegeben. Er sollte immer dann erfolgen, wenn nicht mehr mit dem betreffenden XML-Schema gearbeitet wird.
Anschließend darf auf Adressen der freigegebenen Schema-Struktur einschließlich des XML-Objektes des Schemas nicht mehr zugegriffen werden.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_NO_SCHEMA_GIVEN (40)	A	die Schema-Eingabe-Adresse ist NULL

4.10 Diagnosefunktionen

4.10.1 Trace-Initialisierung

Funktionsprototyp:
```
KXLTSENV();
```

Parameter:
```
 keine
```

Mit der Funktion KXLTSENV wird die Tracefunktion der UTM-XML-Schnittstelle initialisiert. Abhängig von der Einstellung des Tracemodus (in der Jobvariable bzw. Umgebungsvariable KXLTRAC) werden Tracesätze in Dateien geschrieben, die im Fehlerfall ausgewertet werden können.

Implizit wird der Aufruf bei jedem KXLCreateNewObj, KXLConvDocToObj und KXLParseSchema abgesetzt, also bei jedem Neuaufbau eines Objektes. Er kann jedoch auch explizit abgesetzt werden. Dann muss der headerfile `libxml/kxltrace.h` angegeben werden.

Die Initialisierung des Traces erfolgt prozess- bzw. taskspezifisch. D.h. jeder Prozess/Task muss einen solchen Aufruf absetzen, bevor dieser XML – Tracesätze schreibt.

Die Einstellung des Tracemodus, der Aufbau der Dateinamen und Tracesätze, ist im Kapitel "Diagnose", Abschnitt "Trace" beschrieben.

Hinweis für BS2000:
Im BS2000 wird bei jedem KXLTSENV – Aufruf die Jobvariable mit Linknamen KXLTRAC neu gelesen. Ist in der Zwischenzeit der Tracemodus geändert worden, wird er ins Programm übernommen.

4.10.2 Information über den letzten Parser-Fehler

Funktionsprototyp:
```
int     KXLGetLastParserError(char** ppErrmsg, short* pRetCode);
```

Parameter:

Typ	Name	Bemerkung
char **	ppErrmsg	Out: Pointer auf die letzte Parser-Fehlermeldung
short*	pRetCode	Out: Returncode, wenn nicht NULL angegeben wurde

Funktionsergebnis:

Typ	Bemerkung
int	Parser Returncode

Wird im Parser (beim Parsen, Validieren oder auch bei anderen Aufrufen) ein Fehler erkannt, legt der Parser intern einen Fehlercode ab, und/oder gibt eine Meldung (in die Tracedatei) aus, je nachdem, aus welchem Kontext der Fehler erkannt wurde. Dem Aufrufer wird in diesem Fall einer der Returncodes zurückgegeben, die auf einen Parserfehler schließen lassen (s.u.). Über den Aufruf KXLGetLastParserError erhält der Aufrufer den letzten Parser-Returncode bzw. die letzte Parser-Meldung. Das jeweils nicht versorgte Feld wird mit 0 (Returncode) bzw. "\0' (Meldung) überschrieben. Die Parser Returncodes sind im Headerfile libxml/xmlerror.h unter xmlParserErrors definiert.

Liste der wegen Parserfehler zurückgegebenen Returncodes:

- KXL_RC_NO_ROOT_CREATED (4)
- KXL_RC_NO_CONVERSION_TO_DOC (7)
- KXL_RC_NO_CHILD_CREATED (13)
- KXL_RC_NO_TYPE_ATTR_FOUND(14)
- KXL_RC_NO_XML_NODE (15)
- KXL_RC_PARSER_ERROR (20)
- KXL_RC_NO_CONTEXT_FOR_PARSER (21)
- KXL_RC_CONVERSION_ERROR (22)
- KXL_RC_ATTR_READ_WRITE_ERR (26)
- KXL_RC_PARSER_CONTEXT_ERR (37)
- KXL_RC_SCHEMA_PARSE_INT_ERR (38)
- KXL_RC_SCHEMA_VALID_INT_ERR (39)
- KXL_RC_DOC_NOT_VALID (44)

Anmerkung:
Ist der UTM-XML-Trace initialisiert, werden Meldungen auch in die Tracedatei ausgegeben. Da auf einen Fehler hin ggf. mehrere Parsermeldungen ausgegeben werden, ist es sinnvoll, diese in der Tracedatei zu verfolgen.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt

wobei
I= Information, A=Anwenderfehler, S=Systemfehler

5 UTM-XML API für COBOL

5.1 Allgemeines

Da die C-Schnittstelle nicht unverändert auf Cobol abgebildet werden kann, gibt es einen Adaptermodul KXLCOB.c mit eigener Schnittstelle KXLFUNC(im Folgenden beschrieben), die von Cobol – Teilprogrammen aufgerufen wird.

Zum Aufruf der Schnittstelle ist das Copy-Element KXLCOBOL im Programm anzugeben. Es enthält die Definition des Parameterblocks und der Bedingungsnamen.
Abgesehen von folgenden Besonderheiten der Cobol-Schnittstelle entspricht die Bedeutung der Aufrufe, ihrer Parameter und Returnwerte der des C - APIs und ist dort ausführlich beschrieben.

5.1.1 Parameter

Als erster Parameterbereich wird eine Struktur gemäß Definition von KC-XML-PARAMETER übergeben. Sie steht im Copy – Element KXLCOBOL zur Verfügung und muss bei allen Aufrufen der Schnittstelle angegeben werden. Weitere Parameter (Zeichenfolgen) sind abhängig vom OPCODE anzugeben.
Für alle Aufrufe gilt:

- Als Eingabe-Parameter muss KC-XML-PARAMETER mit VERSION=1 versorgt werden.
- Zurückgegeben wird immer RTCODE in KC-XML-PARAMETER

D.h. im Cobol-API kann man, anders als beim C-API, nicht auf die Rückgabe des Returncodes verzichten.

Alle (je nach Funktionsaufruf benötigten) Längenfelder im Parameterbereich sind beim Aufruf der Cobol-Schnittstelle zu versorgen. Eine Ausnahme bildet das Feld BUFFER-RETURN-LTH, in dem KXLConvObjToDoc die reale Länge des konvertierten Dokuments an das Cobol-Programm zurückgibt und das Feld TYPE-RETURN-LTH bei den Leseaufrufen KXLREADXxx, in dem die Länge des zurückgegebenen Typ-strings, zurückgegeben wird.

Bei Angaben von Längen < 0 wird ein entsprechender Returncode zurückgegeben.

5.1.2 Übergabe von Zeichenfolgen

Da Cobol und C unterschiedliche Definitionen von Zeichenfolgen haben (Cobol - Zeichenfolgen mit festen Längen, evtl. rechts mit Leerzeichen aufgefüllt, C - Zeichenfolgen mit abschließendem '/0' (ein Byte mit Wert X'00')), verhält sich der Adaptermodul wie folgt:

- Von Cobol erhaltene Zeichenfolgen werden, abzüglich der "aufgefüllten" Leerzeichen in einen neuen Puffer kopiert und mit '/0' abgeschlossen. Die so erhaltenen C-Zeichenfolgen werden an die C-Schnittstelle weitergegeben.
- Die von der C-Schnittstelle zurückgegebenen Zeichenfolgen werden (ohne das abschließende Nullbyte) in die von Cobol bereitgestellte Zeichenfolgevariable kopiert. Je nach Länge der C-Zeichenfolge und der im Parameterbereich für diese Variable angegebenen Länge wird mit Leerzeichen aufgefüllt oder abgeschnitten.
- Wird an der C-Schnittstelle ein NULL-Zeiger bzw. die leere Zeichenfolge erwartet, muss vom Cobol-Programm das Längenfeld mit 0 vorbelegt werden. Wird von der C-Schnittstelle im Fehlerfall der NULL-Zeiger zurückgegeben, wird dem Cobol-Programm das Rückgabefeld nicht verändert, der Returncode muss ausgewertet werden. Bei Rückgabe von leeren Werten ("emptystring", wenn etwa beim Lesen kein type-Attribut oder kein Inhalt vorhanden ist), wird das entsprechende Rückgabefeld mit Leerzeichen aufgefüllt und ggf. das Feld TYPE-RETURN-LTH mit 0 versorgt.

5.1.3 Bedingungsvariablen

Einige Variablen im Parameterbereich sind als Bedingungsvariablen definiert. Mit den zugehörigen Bedingungsnamen (88er Datenelementen) werden sie wie folgt angesprochen:

- z.B. OPCODE setzen für KXLCreateNewObj: SET KC-XML-CREATE-OBJECT TO TRUE.
- Abfragen: IF KC-XML-CREATE-OBJECT THEN ...

5.2 Returnwerte

Vom Cobol-API werden zusätzlich folgende Returnwerte zurückgegeben:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_COBOL_PARAM_ ERROR(50)	A	Parameterfehler vom Cobol Programmaufruf -> gültigen OPCODE bzw. ELEMENT-TYPE angeben
KXL_RC_BUFFER_TOO_SMALL (51)	A	Puffer des Cobolprogramms für einen Returnwert zu klein -> Puffer vergrößern
KXL_RC_INVALID_LENGTH (52)	A	Der angegebene Längenwert ist ungültig (negativ) -> Angabe korrigieren
KXL_RC_ADD_PARAM_ EXPECTED(53)	A	ein weiterer Parameter wurde erwartet

5.3 Funktionen

5.3.1 Typkonvertierung in die TVALUE - Struktur

Abbildung der Funktionen
- KXLFromShort
- KXLFromInt
- KXLFromLong
- KXLFromFloat
- KXLFromDouble
- KXLFromChar
- KXLFromString
- KXLFromStruct
- KXLFromArray

Eingabeparameter:

KC-XML-PARAMETER mit:	OPCODE	= KC-XML-CONVERT-FROM
	ELEMENT-TYPE	= KC-XML-SHORT / KC-XML-INT / KC-XML-LONG / KC-XML-FLOAT / KC-XML-DOUBLE / KC-XML-CHAR / KC-XML-STRING / KC-XML-STRUCT / KC-XML-ARRAY
	ELEM-VALUE-LTH	= Länge des Feldes ELEMENT-VALUE, bei KXLFromString oder KXLFromStruct oder = 0 bei KXLFromArray
	TYPE-LTH	= Länge des Feldes TVALUE-TYPE
	VALUE-LTH	= Länge des Feldes TVALUE-VALUE
	VAL-xxx	= Inhalt des zu konvertierenden Elements des Typs xxx bei xxx= SHORT / INT / LONG / FLOAT / DOUBLE

ELEMENT-VALUE mit dem Inhalt des Elements (Zeichenfolge) bei ELEMENT-TYPE = KC-XML-STRING / KC-XML-STRUCT / KC-XML-ARRAY

Ausgabeparameter:
TVALUE-TYPE mit dem Typ (abdruckbar) des Elements
TVALUE-VALUE mit dem abdruckbaren Wert des Elements

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
TVALUE-TYPE,
TVALUE-VALUE
(,ELEMENT-VALUE).

Anmerkungen :
- ein short Wert entspricht PIC 9(4) COMP
- ein int/long Wert entspricht PIC 9(8) COMP
- ein float Wert entspricht COMP-1 im BS2000
- ein double Wert entspricht COMP-2 im BS2000

5.3.2 Typkonvertierung aus der TVALUE - Struktur

Abbildung der Funktionen

- KXLToShort
- KXLToInt
- KXLToLong
- KXLToFloat
- KXLToDouble
- KXLToChar
- KXLToString
- KXLToStruct

Eingabeparameter:

KC-XML-PARAMETER mit: OPCODE = KC-XML-CONVERT-TO
ELEMENT-TYPE = KC-XML-SHORT / KC-XML-INT / KC-XML-LONG / KC-XML-FLOAT / KC-XML-DOUBLE / KC-XML-CHAR / KC-XML-STRING / KC-XML-STRUCT
ELEM-VALUE-LTH = Länge des Feldes ELEMENT-VALUE
TYPE-LTH = Länge des Feldes TVALUE-TYPE
VALUE-LTH = Länge des Feldes TVALUE-VALUE

TVALUE-TYPE mit dem Typ (abdruckbar) des Elements
TVALUE-VALUE mit dem abdruckbaren Wert des Elements

Ausgabeparameter:

KC-XML-PARAMETER mit VAL-xxx (bei KC-XML-xxx mit xxx = SHORT / INT / LONG / FLOAT / DOUBLE)
oder
ELEMENT-VALUE mit dem Inhalt des Elements (bei KC-XML-STRING / KC-XML-STRUCT)

Aufruf:
```
CALL "KXLFUNC" USING  KC-XML-PARAMETER,
                      TVALUE-TYPE,
                      TVALUE-VALUE
                      (,ELEMENT-VALUE).
```

5.3.3 KXLCreateNewObj

Eingabeparameter:

KC-XML-PARAMETER mit: OPCODE = KC-XML-CREATE-OBJECT
ELEM-NAME-LTH = Länge des Feldes ELEMENT-NAME
TYPE-LTH = Länge des Feldes TVALUE-TYPE
VALUE-LTH = Länge des Feldes TVALUE-VALUE

ELEMENT-NAME Name des Root-Elements
TVALUE-TYPE Typ (abdruckbar) des Root-Elements
TVALUE-VALUE Wert (abdruckbar) des Root-Elements

Ausgabeparameter:

KC-XML-PARAMETER mit NODE-REFERENCE

Aufruf:
```
CALL "KXLFUNC" USING  KC-XML-PARAMETER,
                      ELEMENT-NAME,
                      TVALUE-TYPE,
                      TVALUE-VALUE.
```


5.3.4 KXLWrite

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-WRITE
NODE-REFERENCE = Referenz des akt. Knotens
ELEM-NAME-LTH = Länge des Feldes ELEMENT-NAME
TYPE-LTH = Länge des Feldes TVALUE-TYPE
VALUE-LTH = Länge des Feldes TVALUE-VALUE

ELEMENT-NAME Name des Root-Elements
TVALUE-TYPE Typ (abdruckbar) des Root-Elements
TVALUE-VALUE Wert (abdruckbar) des Root-Elements

Ausgabeparameter:
KC-XML-PARAMETER mit NEW-NODE-REFERENCE

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
ELEMENT-NAME,
TVALUE-TYPE,
TVALUE-VALUE.

Anmerkung: Ist TYPE-LTH=0, wird der Parameter TVALUE-TYPE nicht ausgewertet und kein type-Attribut erzeugt.

5.3.5 KXLConvObjToDoc

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-CONVERT-OBJ-TO-DOC
BUFFER-LTH = maximale Länge des Ausgabepuffers
NODE-REFERENCE = Referenz des root-Knotens des XML-Objekts
VALUE-LTH = Länge des Feldes STYLESHEET-VALUE

STYLESHEET-VALUE Wert der Stylesheet PI

Ausgabeparameter:
KC-XML-PARAMETER mit BUFFER-RETURN-LTH
DOCUMENT-BUFFER

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
STYLESHEET-VALUE,
DOCUMENT-BUFFER.

5.3.6 KXLConvDocToObj

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-CONVERT-DOC-TO-OBJ
BUFFER-LTH = Länge des Dokuments

DOCUMENT-BUFFER XML-Dokument

Ausgabeparameter:
KC-XML-PARAMETER mit NEW-NODE-REFERENCE

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
DOCUMENT-BUFFER.

5.3.7 KXLFreeObj

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-FREE-OBJECT
NODE-REFERENCE = Referenz des aktuellen root-Knotens

Ausgabeparameter: keine zusätzlichen

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER.

Anmerkung: Mit der Freigabe eines Objektes sind alle Referenzen (NODE-REFERENCE, NEW-NODE-REFERENCE) der zugehörigen Elemente ungültig und dürfen nicht mehr verwendet werden.

5.3.8 KXLSetSubObject

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-SET-SUBOBJECT
NODE-REFERENCE = Referenz des aktuellen Knotens
ELEM-NAME-LTH = Länge des Feldes ELEMENT-NAME

ELEMENT-NAME Name des zu suchenden Knotens

Ausgabeparameter:
KC-XML-PARAMETER mit NEW-NODE-REFERENCE

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
ELEMENT-NAME.

5.3.9 KXLSetRootNode

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-SET-ROOT-NODE
NODE-REFERENCE = Referenz des aktuellen Knotens

Ausgabeparameter:
KC-XML-PARAMETER mit NEW-NODE-REFERENCE

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER.

5.3.10 KXLSetParentNode

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-SET-PARENT-NODE
NODE-REFERENCE = Referenz des aktuellen Knotens

Ausgabeparameter:
KC-XML-PARAMETER mit NEW-NODE-REFERENCE

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER.

5.3.11 KXLRead

Eingabeparameter:

KC-XML-PARAMETER mit: OPCODE = KC-XML-READ
NODE-REFERENCE = Referenz des aktuellen Knotens
DELETION-FLAG = KC-XML-FALSE / -TRUE
ELEM-NAME-LTH = Länge des Feldes ELEMENT-NAME
TYPE-LTH = Länge des Feldes TVALUE-TYPE
VALUE-LTH = Länge des Feldes TVALUE-VALUE

ELEMENT-NAME Name des zu lesenden Knotens

Ausgabeparameter:

KC-XML-PARAMETER mit: TYPE-RETURN-LTH = Rückgabelänge des Feldes TVALUE-TYPE

TVALUE-TYPE Typ des Elements
TVALUE-VALUE Wert des Elements

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
ELEMENT-NAME,
TVALUE-TYPE,
TVALUE-VALUE.

Anmerkung: Enthält das gelesene Element kein type-Attribut, wird TYPE-RETURN-LTH=0 zurückgegeben. TVALUE-TYPE wird dann nicht verändert.

5.3.12 KXLReadNode

Eingabeparameter:

KC-XML-PARAMETER mit: OPCODE = KC-XML-READ-NODE
NODE-REFERENCE = Referenz des aktuellen Knotens
ELEM-NAME-LTH = Länge des Feldes ELEMENT-NAME
TYPE-LTH = Länge des Feldes TVALUE-TYPE
VALUE-LTH = Länge des Feldes TVALUE-VALUE
FULL-NAME = KC-XML-FULL-NAME/ KC-XML-PARTIAL-NAME

Ausgabeparameter:

KC-XML-PARAMETER mit: TYPE-RETURN-LTH = Rückgabelänge des Feldes TVALUE-TYPE

ELEMENT-NAME Name des gelesenen Elements
TVALUE-TYPE Typ (abdruckbar) des Elements
TVALUE-VALUE Wert (abdruckbar) des Elements

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
ELEMENT-NAME,
TVALUE-TYPE,
TVALUE-VALUE.

Anmerkung: Enthält das gelesene Element kein type-Attribut, wird TYPE-RETURN-LTH=0 zurückgegeben. TVALUE-TYPE wird dann nicht verändert.

5.3.13 KXLReadNextSib

Eingabeparameter:

KC-XML-PARAMETER mit: OPCODE = KC-XML-READ-NEXT-SIB
NODE-REFERENCE = Referenz des aktuellen Knotens
ELEM-NAME-LTH = Länge des Feldes ELEMENT-NAME
TYPE-LTH = Länge des Feldes TVALUE-TYPE
VALUE-LTH = Länge des Feldes TVALUE-VALUE

Ausgabeparameter:

KC-XML-PARAMETER mit NEW-NODE-REFERENCE
TYPE-RETURN-LTH

ELEMENT-NAME Name des gelesenen Elements
TVALUE-TYPE Typ (abdruckbar) des Elements
TVALUE-VALUE Wert (abdruckbar) des Elements

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
ELEMENT-NAME,
TVALUE-TYPE,
TVALUE-VALUE.

Anmerkung: Enthält das gelesene Element kein type-Attribut, wird TYPE-RETURN-LTH=0 zurückgegeben. TVALUE-TYPE wird dann nicht verändert.

5.3.14 KXLReadChild

Analog zu KXLReadNextSib mit
OPCODE = KC-XML-READ-CHILD als Eingabeparameter.

5.3.15 KXLReadNextSingleNode

Analog zu KXLReadNextSib mit
OPCODE = KC-XML-READ-NEXT-SINGLE-NODE als Eingabeparameter.

5.3.16 KXLReadAttr

Eingabeparameter:

KC-XML-PARAMETER mit: OPCODE = KC-XML-READ-ATTR
NODE-REFERENCE = Referenz des aktuellen Knotens
ELEM-NAME-LTH = Länge des Feldes ELEMENT-NAME
VALUE-LTH = Länge des Feldes TVALUE-VALUE

Ausgabeparameter:

KC-XML-PARAMETER mit NEW-NODE-REFERENCE

ELEMENT-NAME Name des gelesenen Attributs
TVALUE-VALUE Wert (abdruckbar) des Attributs

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
ELEMENT-NAME,
TVALUE-VALUE.

5.3.17 KXLFindNode

Eingabeparameter:

KC-XML-PARAMETER mit: OPCODE = KC-XML-FIND-NODE
NODE-REFERENCE = Referenz des aktuellen Knotens
ELEM-NAME-LTH = Länge des Feldes ELEMENT-NAME
TYPE-LTH = Länge des Feldes TVALUE-TYPE
VALUE-LTH = Länge des Feldes TVALUE-VALUE
PREFIX2-LTH = Länge des Feldes PREFIX
URL2-LTH = Länge des Feldes URL

ELEMENT-NAME Name des zu lesenden Knotens

Ausgabeparameter:

KC-XML-PARAMETER mit NEW-NODE-REFERENCE
TYPE-RETURN-LTH = Rückgabelänge des Feldes TVALUE-TYPE

TVALUE-TYPE Typ des Elements
TVALUE-VALUE Wert des Elements
PREFIX Präfix des Namensraumes oder SPACES, wenn das gefundene Element keinem oder dem Default-Namensraum zugeordnet ist.
URL Url des Namensraumes oder SPACES, wenn das gefundene Element keinem Namensraum zugeordnet ist

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
ELEMENT-NAME,
TVALUE-TYPE,
TVALUE-VALUE,
PREFIX,
URL.

Anmerkung: Enthält das gelesene Element kein type-Attribut, wird TYPE-RETURN-LTH=0 zurückgegeben. TVALUE-TYPE wird dann nicht verändert.

5.3.18 KXLGetSizeofNodelist

Eingabeparameter:

KC-XML-PARAMETER mit: OPCODE = KC-XML-GET-SIZEOF-NODELIST
ELEM-NAME-LTH = Länge des Feldes ELEMENT-NAME
NODE-REFERENCE = Zeiger auf das aktuelle Element

ELEMENT-NAME Name des Elements

Ausgabeparameter:

KC-XML-PARAMETER mit NODELIST-LENGTH

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
ELEMENT-NAME.

5.3.19 KXLGetHomeEnc

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-GET-HOME-ENC
ELEM-NAME-LTH = Länge des Feldes ENCODING-NAME

Ausgabeparameter:
ENCODING-NAME Name des Home Encodings

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
ENCODING-NAME.

5.3.20 KXLGetDocEnc

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-GET-DOC-ENC
NODE-REFERENCE = Referenz des akt. Knotens
ELEM-NAME-LTH = Länge des Feldes ENCODING-NAME

Ausgabeparameter:
ENCODING-NAME Name des Dokument-Encodings

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
ENCODING-NAME.

5.3.21 KXLSetDocEnc

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-SET-DOC-ENC
NODE-REFERENCE = Referenz des akt. Knotens
ELEM-NAME-LTH = Länge des Feldes ENCODING-NAME

ENCODING-NAME Name des Dokument-Encodings

Ausgabeparameter: keine zusätzliche

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
ENCODING-NAME.

5.3.22 KXLStringToUTF8

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-STRING-TO-UTF8
BUFFER-LTH = Länge des Feldes INPUT-STRING
BUFFER-RETURN-LTH = Länge des Feldes OUTPUT-STRING

INPUT-STRING

Ausgabeparameter: OUTPUT-STRING

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
INPUT-STRING,
OUTPUT-STRING.

5.3.23 KXLStringFromUTF8

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-STRING-FROM-UTF8
BUFFER-LTH = Länge des Feldes INPUT-STRING
BUFFER-RETURN-LTH = Länge des Feldes OUTPUT-STRING

INPUT-STRING

Ausgabeparameter: OUTPUT-STRING

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
INPUT-STRING,
OUTPUT-STRING.

5.3.24 KXLGetEncodingAlias

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-GET-ENCODING-ALIAS
BUF1-LTH = Länge des Feldes ALIAS-NAME
BUF2-LTH = Länge des Feldes ENCODING-NAME

ALIAS-NAME Alias-Name

Ausgabeparameter:
ENCODING-NAME Encoding-Name

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
ALIAS-NAME,
ENCODING-NAME.

5.3.25 KXLSetEncodingAlias

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-SET-ENCODING-ALIAS
BUF1-LTH = Länge des Feldes ALIAS-NAME
BUF2-LTH = Länge des Feldes ENCODING-NAME

ALIAS-NAME Alias-Name
ENCODING-NAME Encoding-Name

Ausgabeparameter: keine zusätzlichen

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
ALIAS-NAME,
ENCODING-NAME.

Hinweis: Soll die aktuelle Zuordnung des Aliasnamens gelöscht werden, muss BUF2-LTH auf 0 gesetzt werden.

5.3.26 KXLWriteNS

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-WRITE-NAMESPACE
NODE-REFERENCE = Referenz des aktuellen Knotens
PREFIX-LTH = Länge des Feldes PREFIX
URL-LTH = Länge des Feldes URL

PREFIX Präfix des Namensraumes
URL Url des Namensraumes

Ausgabeparameter: keine zusätzlichen

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
PREFIX,
URL.

Hinweis: PREFIX-LTH ist auf max. 16 und URL-LTH auf max. 128 beschränkt, da sonst der Eintrag über den Aufruf KXLReadNSList nicht gelesen werden kann.

5.3.27 KXLDelNS

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-DELETE-NAMESPACE
NODE-REFERENCE = Referenz des aktuellen Knotens
PREFIX-LTH = Länge des Feldes PREFIX

PREFIX Präfix des Namensraumes

Ausgabeparameter: keine zusätzlichen

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
PREFIX.

5.3.28 KXLReadNSList

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-READ-NAMESPACE-LIST
NODE-REFERENCE = Referenz des aktuellen Knotens
BUFFER-LTH = Länge des Feldes NS-TBL-BUFFER

Ausgabeparameter:
KC-XML-PARAMETER mit BUFFER-RETURN-LTH
NS-TBL-BUFFER Tabelle mit den Namensräumen

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
NS-TBL-BUFFER.

Struktur der Tabelle, die in NS-TBL-BUFFER zurückgegeben wird:

```
41 NS-TBL-ENTRIES.
  43 NS-TBL-LTH     PIC 9(8) COMP-5.
  43 NS-ENTRY       OCCURS 0 TO 31 DEPENDING ON NS-TBL-LTH.
    45 NS-PREFIX    PIC X(16).
    45 NS-URL       PIC X(128).
```

Hinweis: Anders als in der C-Schnittstelle werden die Zeichenfolgen im Home Encoding übergeben.

5.3.29 KXLSearchNS

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-SEARCH-NAMESPACE
NODE-REFERENCE = Referenz des aktuellen Knotens
PREFIX-LTH = Länge des Feldes PREFIX
URL-LTH = Länge des Feldes URL

PREFIX Präfix des Namensraumes oder
URL Url des Namensraumes

Ausgabeparameter: PREFIX, URL

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
PREFIX,
URL.

Hinweis: Anders als in der C-Schnittstelle werden die Zeichenfolgen im Home Encoding übergeben.

5.3.30 KXLConvDocToObjAndValid

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-CONVERT-DOC-AND-VALID
BUFFER-LTH = Länge des Feldes DOC-BUFFER
BUF2-LTH = Länge des Feldes LOCAL-DIR
VALIDATION-FLAG = KC-XML-FALSE / -TRUE

DOC-BUFFER XML-Dokument
LOCAL-DIR lokales Dateiverzeichnis

Ausgabeparameter:
KC-XML-PARAMETER mit NEW-NODE-REFERENCE

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
DOC-BUFFER,
LOCAL-DIR.

Hinweis:
- Der Parameter LOCAL-DIR entspricht dem Parameter pDir des C-APIs.
- Wenn nicht mit lokalen Dateien gearbeitet werden soll, muss der Parameter LOCAL-DIR LOW-VALUE enthalten.

5.3.31 KXLParseSchema

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-SCHEMA-PARSE-BUF
BUFFER-LTH = Länge des Feldes SCHEMA-BUFFER
BUF2-LTH = Länge des Feldes LOCAL-DIR

SCHEMA-BUFFER XML-Schema
LOCAL-DIR lokales Dateiverzeichnis

Ausgabeparameter: SCHEMA-REFERENCE in KC-XML-PARAMETER

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
SCHEMA-BUFFER,
LOCAL-DIR.

Hinweis:
- Der Parameter LOCAL-DIR entspricht dem Parameter pDir des C-APIs.
- Wenn nicht mit lokalen Dateien gearbeitet werden soll, muss der Parameter LOCAL-DIR LOW-VALUE enthalten.

5.3.32 KXLParseSchemaFile

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-SCHEMA-PARSE-FILE
URL-LTH = Länge des Dateinamens FILENAME
BUF2-LTH = Länge des Feldes LOCAL-DIR

FILENAME Name der Datei, die das XML-Schema enthält
LOCAL-DIR lokales Dateiverzeichnis

Ausgabeparameter: SCHEMA-REFERENCE in KC-XML-PARAMETER

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
FILENAME,
LOCAL-DIR.

Hinweis:
- Der Parameter LOCAL-DIR entspricht dem Parameter pDir des C-APIs.
- Wenn nicht mit lokalen Dateien gearbeitet werden soll, muss der Parameter LOCAL-DIR LOW-VALUE enthalten.

5.3.33 KXLValidDocBuf

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-SCHEMA-VALIDATE-DOC
BUFFER-LTH = Länge des Feldes DOC-BUFFER
SCHEMA-REFERENCE = Pointer auf die XML-Schema-Struktur

DOC-BUFFER XML-Dokument

Ausgabeparameter: keine

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
DOC-BUFFER.

5.3.34 KXLValidDoc

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-SCHEMA-VALIDATE-OBJ
NODE-REFERENCE = Referenz des Root-Knotens des Dokuments
SCHEMA-REFERENCE = Pointer auf die XML-Schema-Struktur

Ausgabeparameter: keine

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER.

5.3.35 KXLFreeSchema

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-SCHEMA-FREE
SCHEMA-REFERENCE = Pointer auf die XML-Schema- Struktur

Ausgabeparameter: keine

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER.

5.3.36 KXLTSENV

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-INIT-TRACE

Ausgabeparameter: keine

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER.

5.3.37 KXLInitEnv

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-INIT-ENV
ENC-FUNC-TO-UTF8 = Adresse der Code-Umsetzfunktion vom Home Encoding nach UTF-8
ENC-FUNC-FROM-UTF8 = Adresse der Code-Umsetzfunktion von UTF-8 ins Home Encoding
BUFFER-LTH = Länge des Feldes ENCODING-NAME

ENCODING-NAME Name des gewünschten Home Encodings

Ausgabeparameter: keine

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
ENCODING-NAME.

Achtung:
1. Die Angabe des home Encoding Namens muss in UTF-8 erfolgen.
2. Es ist zu beachten, dass die angegebenen Umsetzfunktionen C - Zeichenfolgen (mit abschließendem '/0') verarbeiten und erzeugen, da diese Funktionen in UTM-XML von C-Modulen aufgerufen werden.

5.3.38 KXLSchemaGetRoot (Root Knoten eines Schema-Objektes anfordern)

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-SCHEMA-GET-ROOT
SCHEMA-REFERENCE = Adresse der XML-Schema-Struktur

Ausgabeparameter:
KC-XML-PARAMETER mit NEW-NODE-REFERENCE

Returncode: Falls SCHEMA-REFERENCE NULL-Pointer, wird KC-XML-RC-INVALID-PARAMETER zurückgegeben.

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER.

Funktion:
Diese Funktion erwartet einen Pointer auf eine Schema-Struktur und gibt den Root-Pointer des dazugehörigen Schema-Objekts zurück. Mit diesem kann man über KXLConvObjToDoc das Schema-Dokument erzeugen.
Diese Funktion ist nur an der COBOL-Schnittstelle verfügbar.

Achtung: Das Schema-Objekt darf nicht mit KXLFreeObj freigegeben werden. Wenn die Schema-Struktur nicht mehr benötigt wird, sollte diese mit KXLFreeSchema freigegeben werden.

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KC-XML-RC-OK (0)	I	Funktion ordnungsgemäß ausgeführt
KC-XML-RC-INVALID-PARAMETER (32)	A	Adresse der XML-Schema-Struktur ist NULL

5.3.39 KXLGetLastParserError

Eingabeparameter:
KC-XML-PARAMETER mit: OPCODE = KC-XML-GET-PARSER-ERRMSG
BUFFER-LTH = Länge des Feldes ERR-MESSAGE

Ausgabeparameter:
KC-XML-PARAMETER mit PARSER-RTCODE
ERR-MESSAGE

Aufruf: CALL "KXLFUNC" USING KC-XML-PARAMETER,
ERR-MESSAGE.

6 Installation

UTM-XML wird nur per download im Internet zur Verfügung gestellt. Sie können sich die plattformspezifischen Anwendungs- ('rt') und Source-Pakete ('dev') von UTM-XML über die folgende Webseite herunterladen:

http://de.ts.fujitsu.com/products/software/openseas/openutm.html

Nach dem Herunterladen müssen Sie die Pakete ggf. auf die Zielplattformen transferieren. Die Pakete sind komprimiert und müssen mit den entsprechenden Tools (tar, winzip) dekomprimiert werden.

Das Anwendungs-Paket enthält alle Bestandteile, die Sie für die Erstellung der UTM-XML Teilprogramme zusätzlich zu UTM benötigen:

- Dokumentation: Lizenz, readme-Dateien, Funktionsbeschreibung (dieses Dokument)
- COBOL - Copy-Element
- C - Headerfiles
- Sourcen der UTM - Beispielprogramme in Cobol und C
- Bibliotheken bzw. BS2000-Lademodule

Das Source-Paket enthält die UTM-XML Sourcen, die Sourcen vom GNOME Parser libxml2, die Lizenz und die Beschreibungen.

Für den Einsatz von UTM-XML reicht es aus, die Bestandteile des Anwendungspakets zu installieren.

6.1.1 Unix-Plattformen

Für die unterschiedlichen Plattformen und Ausprägungen (32- / 64-Bit) stehen verschiedene Pakete auf der oben genannten Webseite zur Verfügung. Aus dem Paketnamen ist Zielplattform und Ausprägung ersichtlich. Dabei bezeichnet "rt" das Anwendungspaket UTM-XML inkl. der Sourcen der Beispielprogramme in COBOL und C. Die mit "dev" bezeichneten Pakete enthalten alle UTM-XML-Sourcen und Parser-Sourcen mit Headerfiles und Beispielen.

Die genaue Aufstellung der verfügbaren Pakete ist in der Freigabemitteilung (readme) enthalten.

Für UNIX- und Linux-Plattformen müssen Sie die Pakete nach dem Herunterladen ggfs. auf die Zielmaschine transferieren.

Danach müssen Sie Pakete mit Suffix .Z mit uncompress und Pakete mit Suffix .gz mit gunzip dekomprimieren. Die (daraus entstandenen) Pakete mit dem Suffix .tar werden mit tar -xvf entpackt.

6.1.2 BS2000

Für BS2000 stehen die Pakete in einem zip-Archiv. Nach dem Download entpacken Sie die zip-Datei, die folgende Unterverzeichnisse enthält:

- ftp mit der LMS-Bibliothek SYSLIB.UTM-XML.<version>.RT bzw. -.DEV
- openft mit der LMS-Bibliothek SYSLIB.UTM-XML. <version>.RT bzw. -.DEV
- doc mit den Dokumenten (liesmich, API-Beschreibung, Lizenz)

Übertragen Sie die LMS-Bibliothek aus dem Unterverzeichnis ftp auf Ihren BS2000 Rechner, wenn Sie die Übertragung mit ftp durchführen (im Binärmodus). Oder übertragen Sie die LMS-Bibliothek aus dem Unterverzeichnis openft auf Ihren BS2000 Rechner, wenn Sie die Übertragung mit openFT durchführen (Dateityp binär, Übertragungsmodus transparent).

Danach stehen sie als LMS-Bibliotheken zum Einsatz zur Verfügung.

Die folgenden Listen geben einen Überblick über die Inhalte der LMS-Bibliotheken:

SYSLIB.UTM-XML. <version>.RT (Anwendungsbibliothek UTM-XML) enthält

- die XML Objekte für BS2000/OSD
- das COBOL - Copy-Element
- die C - Headerfiles
- die UTM - Beispielprogramme in Cobol und C (Sourcen und Module)
- die Dokumentation

SYSLIB.UTM-XML. <version>.DEV (Sourcebibliothek UTM-XML) enthält

- alle verwendeten libxml2 Parser-Sourcen mit C-Headerfiles
- alle UTM-XML-Sourcemodule mit C-Headerfiles

6.1.3 Windows

Die folgenden Pakete stehen für Windows auf der oben genannten Webseite zur Verfügung:

- utmxmlrt_<version>_win_32.zip: Anwendungspaket UTM-XML inkl. der Beispielprogramme in COBOL und C
- utmxmldev_<version>_win_32.zip: Sourcepaket UTM-XML

Nach dem Herunterladen müssen Sie die Pakete ggf. auf die Zielmaschine transferieren. Danach sind sie mit winzip zu entpacken. Mit dem Skript install.cmd werden die Komponenten in die entsprechenden Verzeichnisse kopiert. Dazu muss für das Anwendungs-Paket die Umgebungsvariable $UTMPATH auf das aktuelle UTM-Directory gesetzt werden. Wird die zip-Datei entpackt und das zugehörige skript install.cmd gestartet, wird im UTM-Directory ein Dateibaum mit Namen xml und den entsprechenden Unterordnern erzeugt, in die die einzelnen Komponenten installiert werden. Für das Source-Paket werden mit dem Aufruf von install.cmd die Komponenten in einen Dateibaum mit Namen xml/opensource im aktuellen Verzeichnis kopiert.

7 Einsatz

7.1.1 Unix-Plattformen

Beim Übersetzen von Komponenten, die die XML-Schnittstelle aufrufen, sind die Verzeichnisse <utmxml_home>/include und bei Cobol <utmxml_home>/copy-cobol85 bzw. <utmxml_home>/netcobol zu berücksichtigen. Dabei ist <utmxml_home> das Verzeichnis, in dem das Anwendungspaket von UTM-XML installiert wurde.

COBOL-Teilprogramme, die das COPY-Element KXLCOBOL verwenden, dürfen bei Verwendung des MicroFocus Compiler nicht mit der Option NOMF übersetzt werden, da sonst Übersetzungsfehler auftreten.

Bei Verwendung anderer Compiler sollten die Felder VAL-FLOAT und VAL-DOUBLE entsprechend angepasst werden. Falls es keine Entsprechung für Float- und Double-Typen gibt, sollten die Felder wie folgt gesetzt werden:

```
43 VAL-FLOAT        PIC X(4).
43 VAL-DOUBLE       PIC X(8).
```

Die entsprechenden Funktionen KXLFromFloat, KXLFromDouble, KXLToFloat,

KXLToDouble können dann von COBOL aus nicht genutzt werden.

Beim Binden einer UTM-Anwendung (Client/Server) mit Nutzung der UTM-XML-Schnittstelle müssen folgende Bibliotheken (aus <utmxml_home>\lib) berücksichtigt werden:

- libutmxml.a (statisch) bzw.
- libutmxml.so/sl (dynamisch)

Außerdem muss mit Angabe von -lm die Bibliothek der mathematischen Funktionen eingebunden werden.

Die genaue Realisierung kann der Beispielanwendung (Bestandteil von openUTM) entnommen werden.

Zum Einsatz benutzerspezifischer bzw. zur Anpassung bestehender Encoding-Funktionen siehe im Anhang den Abschnitt "Nutzung benutzerspezifischer Encoding-Funktionen".

7.1.2 BS2000

Übersetzen von C-Programmen:

Zum Aufruf der Schnittstelle geben Sie das Headerfile `libxml/kxlinc.h` im Programm an. Die Namen der Headerfiles libxml/<includename> entsprechen im BS2000 den Include-Dateien LIBXML.<includename>. Der BS2000-C-Compiler setzt die Namen entsprechend um.

Beim Übersetzen der Programme, die die Schnittstellenaufrufe enthalten, weisen Sie die Bibliothek SYSLIB.UTM-XML. <version>.RT mit der USER-INCLUDE-LIBRARY-Option zu. C-Teilprogramme, die Aufrufe des UTM-XML API enthalten, müssen als LLM übersetzt und gebunden werden.

Im Modul KXLCVLT.C sind die Umsetztabellen von UTF-8 nach EBCDIC und umgekehrt enthalten. Wenn Sie diese Tabellen nur geringfügig ändern wollen, können Sie die Tabellen im Source (in der SYSLIB.UTM-XML. <version>.DEV enthalten) entsprechend ändern und mit dem C-Compiler übersetzen. Folgende Compiler-Option muss angegeben werden:

```
MODIFY-MODULE-PROPERTIES LOWER-CASE-NAMES=*YES
```


Übersetzen von Cobol-Programmen:

COBOL Teilprogramme, die die UTM-XML-Schnittstellenaufrufe enthalten, müssen das Copy-Element KXLCOBOL enthalten (COPY-Anweisung). Beim Übersetzen der Programme verknüpfen Sie die Bibliothek SYSLIB.UTM-XML. <version>.RT mit einem der Linknamen COBLIB, COBLIBn (n=1,…,9). Beim Übersetzen muss folgende Compileroption angegeben werden:

P[ERMIT]-S[TANDARD]-D[EVIATION]=YES

Binden:

Wenn Sie ein Teilprogramm, das die Aufrufe der XML-Schnittstelle enthält, in Ihre UTM-Anwendung einbinden wollen, fügen Sie eine Resolve-Anweisung auf die SYSLIB.UTM-XML. <version>.RT ein.

Wird die UTM-Anwendung basierend auf der BLS-Schnittstelle generiert, so kann statt dessen die SYSLIB.UTM-XML. <version>.RT in der Startprozedur mit einem Linknamen BLSLIBnn zugewiesen werden.

Haben Sie das Modul KXLCVLT.C geändert, müssen Sie den übersetzten Modul ggf. explizit einbinden. Zum Einsatz benutzerspezifischer bzw. zur Anpassung bestehender Encoding-Funktionen siehe im Anhang den Abschnitt "Nutzung benutzerspezifischer Encoding-Funktionen".

7.1.3 Windows

Beim Übersetzen von Komponenten, die die UTM-XML-Schnittstelle aufrufen, sind die Verzeichnisse <utmxml_home>\include und bei Cobol <utmxml_home>\copy-cobol85 bzw. <utmxml_home>\netcobol zu berücksichtigen. Dabei ist <utmxml_home> das Verzeichnis, in dem das Anwendungspaket von UTM-XML installiert wurde.

Beim Binden einer UTM-Anwendung (Client/Server) mit Nutzung der UTM-XML-Schnittstelle müssen folgende Bibliotheken (aus <utmxml_home>\dll) berücksichtigt werden:

- libutmxml.lib und
- libutmxml.dll

Außerdem muss bei COBOL-Nutzung folgendes Objekt (aus <utmxml_home>\obj) eingebunden werden:

- kxlcob2c.obj

Die genaue Realisierung kann dem Quickstart Kit (Bestandteil von openUTM) entnommen werden.
Zum Einsatz benutzerspezifischer bzw. zur Anpassung bestehender Encoding-Funktionen siehe im Anhang den Abschnitt "Nutzung benutzerspezifischer Encoding-Funktionen".

8 Diagnose

8.1 Trace

Zur Diagnose bei laufender Anwendung können Traces für die UTM-XML-Schnittstelle eingeschaltet werden. Sie werden über Umgebungs- bzw. Jobvariablen gesteuert, die vor dem Start der Anwendung gesetzt werden müssen.

Prozess (auf Unix- und Windows-Systemen) bzw. **Task** (in BS2000) bezeichnet i.F. einen UTM-, UPIC- oder anderen UTM-Client-Prozess.
Jeder Prozess bzw. Task schreibt den Trace in eine eigene Datei, die in zwei Generationen (alt und neu) existieren kann.
Die maximale Größe einer Tracedatei beträgt ca. 2000 KB. Sobald diese Größe erreicht wird, wird auf eine zweite Datei umgeschaltet. Hat auch diese das Limit erreicht, wird wieder in die erste Datei geschrieben.

Eine Tracedatei besitzt folgenden Namen:
KXL*pid*.*n* (UNIX- und Windows-Systemen) bzw.
KXL*tsn*.*n* (BS2000/OSD), wobei
KXL einen XML-Trace kennzeichnet,
pid die Prozess-ID des Prozesses, 5-stellig,
tsn die ID des Tasks, 4-stellig, und
n die Nummer der Generation: 1 oder 2 ist.
Den jüngeren Trace erkennen Sie anhand der Zeitstempel in den Tracedaten.

Beispiel: KXL00341.1: XML-Tracedatei Nummer 1 für den Prozess 00341
KXL00341.2: XML-Tracedatei Nummer 2 für den Prozess 00341

Zur Initialisierung der Tracefunktionen der UTM-XML-Schnittstelle wird die Funktion KXLTSENV() aufgerufen. Implizit wird der Aufruf bei KXLInitEnv und bei jedem KXLCreateNewObj und KXLConvDocToObj abgesetzt, also bei jedem Neuaufbau eines Objektes. Er kann jedoch auch explizit abgesetzt werden. Dann muss der headerfile `libxml/kxltrace.h` angegeben werden:
`#include <libxml/kxltrace.h>`
Die Initialisierung erfolgt prozess- bzw. taskspezifisch. D.h. jeder Prozess/Task muss einen solchen Aufruf absetzen, bevor dieser UTM-XML-Tracesätze schreibt.
Bei der Initialisierung wird über den Makro getenv (bzw. GETJV im BS2000) aus der Umgebungsvariablen bzw. Jobvariablen ((Link)Name KXLTRAC) der gewünschte Tracemodus abgefragt und gesetzt, der Name der Tracedatei aufgebaut und diese geöffnet.

Hinweis für BS2000:
Im BS2000 wird bei jedem KXLTSENV- Aufruf die Jobvariable mit Linknamen KXLTRAC neu gelesen. Ist in der Zwischenzeit der Tracemodus geändert worden, wird er ins Programm übernommen.

Es ist möglich, folgende drei Tracemodi anzugeben:

E (Error): Aktiviert den Fehlertrace. Nach jedem Returncode ungleich KXL_RC_OK wird eine Fehlermeldung in die Trace-Datei geschrieben:
<Zeitstempel> <funktion> error: <Fehlertext>

I (Interface): Aktiviert den Schnittstellentrace für die UTM-XML-Aufrufe. Bei jedem Aufruf einer UTM-XML-Funktion wird folgender Satz in die Trace-Datei geschrieben:
<Zeitstempel> <funktion> init <eingabe-parameter>
Bei jedem Rücksprung aus einer UTM-XML-Funktion wird folgender Satz in die Trace-Datei geschrieben:
<Zeitstempel> <funktion> exit <rückgabe-parameter>
Da die Tracemodi hierarchisch aufgebaut sind, enthält der Tracemode I auch den Tracemode E.

F (Full): Aktiviert den vollen UTM-XML-Trace. Zusätzlich zum Umfang des Tracemodus I wird bei jeder Konvertierung das XML-Dokument in die Tracedatei ausgegeben. Ist das Dokument größer als 4096 Zeichen, werden nur 2048 Byte vom Anfang und 2048 Byte vom Ende des Dokuments ausgegeben.

Unter <eingabe-parameter> und <rückgabe-parameter> werden die Eingabe- und Rückgabe-Parameter der jeweiligen Funktion in folgender Form protokolliert und mit Kommata voneinander getrennt.:

xmlNodePtr	Adresse in der Form	XML-A=nnnnnnnn
char * (Doku.)	Adresse in der Form	DOC-A=nnnnnnnn
xmlSchemaPtr	Adresse in der Form	SCA-A=nnnnnnnn
char * (stylesheet)	Zeichenfolge in der Form	S=<string>
char * (Name)	Zeichenfolge in der Form	N=<string>
t_value	2 Zeichenfolgen in der Form	T=<string>, V=<string>
short (RetCode)	Nummernwert des Returncodes	RC=nn
int (Länge)	ganzzahliger Längenwert	LTH=nn
int (NameType)	Nummernwert des Elementtyps	ET=nn
int (FullName)	Bool-Wert f. Namensaufbereitung	FULL=0/1
int (Delete)	Bool-Wert f. Löschen	DEL=0/1
short/int/long	Ein-/Rückgabe der Typ-Konvertierungsfunktionen	SH/IN/LG=<wert>
float/double/char *	Ein-/Rückgabe der Typ-Konvertierungsfunktionen	FL/DO/ST=<wert>
char *(Name)	Encoding-Name	ENC=<string>
Ptr	Encoding-Adresse	ENC-A= nnnnnnnn
int	Parser Returncode	P-RC=nn
char *(message)	Parser Error Message	MSG=<string>
int	Sizeof Nodelist	SIZ=nn
int	Version	V=nn
int (Präfix-Länge)	Präfix Längenwert	P_LTH=nn
char *(Präfix)	Namespace Präfix	NS-P=<string>
char *(Url)	Namespace URL	NS-U=<string>

Bei FL/DO = <wert> wird der hexadezimale Wert der Variablen protokolliert. Die nachfolgende Zeile (abdruckbare Wiedergabe) ist nicht von Bedeutung.

Wenn, z.B. im Fehlerfall, Ausgabeparameter nicht protokolliert werden, sind die Werte
- bei Adressen NULL,
- bei Namen leere Zeichenfolge '\0',
- bei t_value {NULL, NULL}.

Die Bedeutung der Werte von ET sind folgende:

0	NoType
1	IsSingleType
2	IsStruct
3	IsArray
4	IsElemOfArray

Die Bedeutung der Werte von P-RC sind in der Parser-Headerdatei xmlerror.h in xmlParserErrors beschrieben. Ist P-RC = -1, liegt ein Parser-interner oder Parameterfehler vor.

Eine Beispiel einer Tracedatei ist beim Beispiel im Anhang abgebildet.

Anmerkung:
Wird im Parser ein Fehler entdeckt, wird ggf. eine Meldung ausgegeben. Diese Ausgabe erfolgt in die aktuelle Tracedatei, wenn der Trace initialisiert ist, sonst auf SYSOUT bzw. stderr.

8.1.1 Tracemodus ablegen auf UNIX- und Windows-Systemen

Für das UTM-XML API wird der Tracemodus in der Umgebungsvariablen KXLTRAC abgelegt. Mögliche Angaben (E/I/F) siehe oben. Jeder Prozess liest beim ersten Aufruf von KXLTSENV diese Umgebungsvariable und initialisiert den Trace mit dem gewünschten Tracemodus.
Umgebungsvariablen werden auf UNIX-Systemen mit folgendem Kommando gesetzt:

```
KXLTRAC = wert
export KXLTRAC
```

Auf UNIX-Systemen gelten die Umgebungsvariablen jeweils für eine Shell; für eine Anwendung in einer anderen Shell können andere Werte gelten.
Auf Windows kann die Umgebungsvariable vor dem Start des Programms mit dem Kommando SET gesetzt werden. Es wird empfohlen, die Umgebungsvariablen über Arbeitsplatz – Eigenschaften - Umgebung zu definieren.

8.1.2 Tracemodus ablegen unter BS2000/OSD

Für das UTM-XML API wird der gewünschte Tracemodus in einer Jobvariable mit Linknamen (Kettungsnamen) KXLTRAC abgelegt. Jeder Task liest beim Aufruf von KXLTSENV diese Jobvariable und initialisiert den Trace mit dem gewünschten Tracemodus.

Wenn das Software-Produkt JV als Subsystem geladen ist, können die Jobvariablen z.B. unter BS2000/OSD wie folgt gesetzt werden:

1. Jobvariable erzeugen:
   CREATE-JV JV-NAME=FULLTR
2. Wert an die Jobvariable übergeben:
   MODIFY-JV JV[-CONTENTS]=FULLTR, SET-VALUE=‘F‘
3. Taskspezifischen Jobvariablen-Link setzen:
   SET-JV-LINK LINK-NAME=KXLTRAC, JV-NAME=FULLTR
4. Taskspezifischen Jobvariablen-Link anzeigen:
   SHOW-JV-LINK JV[-NAME]=FULLTR
5. Taskspezifischen Jobvariablen-Link löschen:
   REMOVE-JV-LINK LINK-NAME=KXLTRAC

Unter BS2000/OSD sind die Jobvariablen-Links taskspezifisch. Einer zweiten Anwendung unter der gleichen Kennung können andere Jobvariablen zugewiesen werden.

9 Besonderheiten bei Nutzung des UTM-XML API unter openUTM

Das grundlegende Prinzip der DOM – Schnittstelle ist folgendes:
Im Programmspeicher wird ein Objekt in Baumstruktur aufgebaut, das interaktiv bearbeitet werden kann. Innerhalb des Objektes kann positioniert werden. D.h. ausgehend von einem inneren Knoten kann auf einem Teilbaum gearbeitet werden, ohne dass bei jedem Zugriff auf einen Knoten der vollständige Pfad vom root zum Knoten durchlaufen werden muss (Performance!).

Zum Versenden an Kommunikationspartner oder zum Archivieren wird das Objekt in ein Dokument konvertiert.

Bei der Verwendung des UTM-XML API in openUTM, d.h. in einem UTM – Teilprogramm, kann nach Teilprogrammende wegen möglichem Prozesswechsel nicht mehr auf das Objekt und die evtl. vorhandenen Positionierungen zugegriffen werden. Daher gibt es beim interaktiven Bearbeiten eines Dokuments nur zwei Möglichkeiten:

1. Das XML – Dokument wird in UTM-Speicherbereichen, z.B. im KB oder GSSB, abgelegt. Bei jedem Teilprogramm-Start (=Dialogschritt-Beginn) wird das Dokument gelesen und in ein Objekt konvertiert, nach der Bearbeitung vor Teilprogramm-Ende (=Dialogschritt-Ende) wird das Objekt wieder zurückkonvertiert und als Dokument abgelegt/gesendet. Diese Möglichkeit ist insbesondere bei großen Dokumenten inperformant.
2. Es wird ein Teilprogramm mit PGWT verwendet. Auch bei Dialogschritt-Ende (MPUT, PGWT, MGET) bleibt das Objekt im Speicher erhalten. Es kann also nach einer Ausgabe am und Eingabe vom Bildschirm auf dem vorhandenen Objekt weitergearbeitet werden.

   Nachteile:
   - Es erfolgt keine Sicherung, d.h. UTM – seitige Transaktions-Sicherung (z.B. vom Dokument in KB oder GSSB) erfolgt frühestens beim Teilprogramm-Ende. Dann ist aber auch das Objekt im Speicher nicht mehr verfügbar (sollte also vor Teilprogramm-Ende freigegeben werden).
   - je ein UTM – Prozess ist von jedem PGWT – Teilprogramm 'blockiert', d.h. auch nach Dialogschritt-Ende wird der Prozess nicht für andere Benutzer freigegeben. Es müssen also im ungünstigsten Fall soviel Prozesse wie Benutzer vorhanden sein.

Bei Verwendung des UTM-XML API in openUTM – Clients, wie z.B. UPIC, stellt sich das Problem nicht, da der Kontext über Dialogschrittgrenzen erhalten bleibt.

10 Anhang

10.1 Returncodes

Die Werte des Returncodes, der von den Funktionen dieser Schnittstelle zurückgegeben wird, sind im Headerfile `libxml/kxlinc.h` mit #define definiert, also standardmäßig int - Werte. Um bei Fehlern die Diagnose zu erleichtern, kann man sich bei C/C++ - Programmen den Returncode als Zeichenfolge ausgeben lassen. Mit der Definition des symbolischen Namens KXL_GEN_STRING vor Inkludieren des `libxml/kxlinc.h`:

```
#define KXL_GEN_STRING
#include <libxml/kxlinc.h>
```

kann man mit KXL_RCODE_STRINGS[returncode] auf die gewünschte Zeichenfolge (max. 32 Zeichen) zugreifen.

Die Werte KC-XML-RC-xxx von RTCODE in COBOL entsprechen den Werten KXL_RC_xxx in `libxml/kxlinc.h` in C.
Namen, Wert und Bedeutung der Returncodes sind wie folgt. Dabei kann in der Spalte Fehlertyp: A (Anwenderfehler), S (Systemfehler) oder I(nformation) stehen

Returncodes:

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_OK (0)	I	Funktion ordnungsgemäß ausgeführt
KXL_RC_NO_NODE_FOUND (1)	I	Knoten mit dem angegebenen Namen nicht gefunden
KXL_RC_DIFF_TYPES(2)	I	type in t_value und angeforderter type verschieden
KXL_RC_NO_ROOT_CREATED (4)	S	der Parser konnte kein neues Objekt erzeugen; ->siehe Meldung des Parsers. (Tracedatei oder KXLGetLastParserError)
KXL_RC_NO_CONVERSION_TO_ DOC (7)	S	Fehler beim Erzeugen des Dokuments -> Meldung des Parsers auswerten (Tracedatei oder KXLGetLastParserError)
KXL_RC_EMPTY_DOC (9)	A	konvertiertes Objekt enthält keine Knoten
KXL_RC_INVALID_DOC_PTR (10)	S/A	der angegebene Objektknoten ist keinem Objekt zugeordnet; möglicher Grund: Objekt zerstört oder fehlerhaft erzeugt ->Fehlerunterlagen erzeugen
KXL_RC_ALLOC_ERROR (11)	S	mit malloc/realloc konnte kein zusätzlicher Speicherplatz angefordert werden
KXL_RC_TYPE_MISMATCH(12)	A	beim Überschreiben eines Knotens stimmt der vorhandene Typ nicht mit dem angegebenen Typ überein ->wenn Überschreiben gewünscht, Knoten erst löschen, dann neu schreiben
KXL_RC_NO_CHILD_CREATED (13)	S/A	Fehler beim Erzeugen eines neuen Knotens ->Meldung des Parsers auswerten
KXL_RC_NO_TYPE_ATTR_ FOUND(14)	S/A	beim Lesen/Überschreiben eines Knotens wurde kein Typ – Attribut gefunden, kein Positionieren/Überschreiben möglich.
KXL_RC_NO_XML_NODE (15)	A	der Zeiger auf den aktuellen Knoten ist NULL -> gültigen Zeiger angeben
KXL_RC_NO_SUBTREE_	A	beim Lesen eines Knoten mit Löschen wurde der

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
DELETION (16)		Knoten nicht gelöscht, da es kein äußerer Knoten ist. ->soll der ganze Teilbaum gelöscht werden, muss jeder Knoten einzeln gelöscht werden.
KXL_RC_INVALID_T_VALUE (17)	A	einer der in t_value angegebenen Zeiger ist NULL ->Wert korrigieren
KXL_RC_EMPTY_NAME(18)	A	beim Analysieren des Elementnamens wurde ein leerer Namensteil gefunden, Schreiben abgebrochen, bzw. Name leer -> gültigen Namen angeben.
KXL_RC_INVALID_NAME (19)	A	beim Analysieren des Elementnamens wurde ein Syntaxfehler bei der Verwendung von "[", "]" ,"/" und "@" entdeckt -> gültigen Namen angeben
KXL_RC_PARSER_ERROR (20)	S/A	Fehler im Parser entdeckt ->siehe Meldung des Parsers. (Tracedatei oder KXLGetLastParserError)
KXL_RC_NO_CONTEXT_FOR_ PARSER (21)	S	Fehler beim Anlegen parserinterner Verwaltungsbereiche -> evtl. Speicherengpass beseitigen
KXL_RC_CONVERSION_ERROR (22)	S/A	Fehler bei der Code-Konvertierung
KXL_RC_VERSION_ERR (23)	A	Versionsstände der UTM-XML-Komponenten inkonsistent
KXL_RC_PARSER_VERS_NOT_ SUPP (24)	A	vorhandene Parserversion wird nicht unterstützt
KXL_RC_ATTR_NAME_ERROR (25)	A	Der Aufbau des Attributnamens ist nicht korrekt; nach '@' keine '/', '[' oder ']' erlaubt
KXL_RC_ATTR_READ_WRITE_ ERR (26)	A	Fehler beim Lesen oder Schreiben eines Attributes
KXL_RC_ENC_NAME_ERR (27)	A	Encoding Name zu lang (> 31 Zeichen)
KXL_RC_NODE_TYPE_NOT_ ALLOWED (28)	A	bei diesem Aufruf ist nur die Angabe eines Element- bzw. Attributknotens erlaubt
KXL_RC_NO_PARENT_NODE (29)	A	angegebener Knoten besitzt keinen Verweis auf einen Vaterknoten
KXL_RC_UNKNOWN_ENCODING (30)	A	Encoding-Name ist dem Parser nicht bekannt -> zum Encoding gehörige Konvertierungs-routinen beim Parser deklarieren
KXL_RC_NAMESPACE_NOT_ FOUND (31)	I	Gesuchten Namensraum nicht gefunden
KXL_RC_INVALID_PARAMETER (32)	A	Parameterangabe nicht korrekt , z.B. NULL-Zeiger angegeben, obwohl nicht erlaubt
KXL_RC_NAMESPACE_WRITE_ ERROR (33)	S/A	Der Parser konnte die Namensraum-Definition nicht ordnungsgemäß schreiben, siehe Parser-Fehlermeldungen
KXL_RC_NAMESPACE_DEL_ERR (34)	A	Die Namensraum-Definition konnte nicht gelöscht werden. Es sind noch Elemente zugeordnet.
KXL_RC_ELEM_BEL_TO_ NAMSP (35)	A	Dem aktuellen Namensraum ist mindestens ein Element zugeordnet.
KXL_RC_ENCODING_CHANGE_ ERR (36)	A/S	für das angegebene Home Encoding konnte kein Encoding Handler eingerichtet werden
KXL_RC_PARSER_CONTEXT_ ERR (37)	S	Parser-interner Fehler beim Erzeugen von Parser-internen Verwaltungsbereichen -> evtl. Speicherengpass beseitigen
KXL_RC_SCHEMA_PARSE_INT_ ERR (38)	A	Fehler beim Parsen des Schemas ->siehe Meldung des Parsers. (Tracedatei oder KXLGetLastParserError)

Name (Wert)	Fehler-typ	Bemerkung -> Maßnahme
KXL_RC_SCHEMA_VALID_INT_ ERR (39)	A	Fehler beim Validieren ->siehe Meldung des Parsers. (Tracedatei oder KXLGetLastParserError)
KXL_RC_NO_SCHEMA_GIVEN (40)	A	die Schema-Eingabe-Adresse ist NULL
KXL_RC_FILE_OPEN_ERR (41)	S	Fehler beim Öffnen einer Datei
KXL_RC_FILE_READ_ERR (42)	S	Fehler beim Lesen einer Datei
KXL_RC_FILE_CLOSE_ERR (43)	S	Fehler beim Schließen einer Datei
KXL_RC_DOC_NOT_VALID (44)	A	das Dokument ist keine gültige Instanz des angegebenen Schemas ->siehe Meldung des Parsers. (Tracedatei oder KXLGetLastParserError)
KXL_RC_IO_HNDLR_INIT_ERR (45)	A	zu viele IO-Handler definiert
KXL_RC_PARSER_ENCODING_ ERR (46)	S/A	Parser konnte Funktion nicht ausführen -> überprüfen, ob beim Setzen der Encoding-Name bzw. beim Löschen der Alias-Name definiert ist.
KXL_RC_IS_ALIAS_ENCODING (47)	A	angegebene Encoding-Name ist Alias-Name -> Original Encoding Name angeben
KXL_RC_COBOL_PARAM_ ERROR(50)	A	Parameterfehler vom Cobol Programmaufruf -> gültigen OPCODE bzw. ELEMENT-TYPE angeben
KXL_RC_BUFFER_TOO_SMALL (51)	A	Puffer des Cobolprogramms für einen Returnwert zu klein -> Puffer vergrößern
KXL_RC_INVALID_LENGTH (52)	A	Der angegebene Längenwert ist ungültig (negativ oder zu groß) -> Angabe korrigieren
KXL_RC_ADD_PARAM_ EXPECTED (53)	A	Vom Cobol-Programm wurden nicht genug Parameter übergeben -> Programmierung überprüfen
KXL_RC_NOT_SUPPORTED (99)	A	Funktion (noch) nicht unterstützt

Folgende Returncodewerte werden nicht zurückgegeben, sind jedoch (als Ober- und Untergrenze) definiert:

KXL_RC_NIL (-1)	nicht gültig
KXL_MAX_RC(99)	höchster Returncode

Folgende Returncodewerte werden nicht zurückgegeben, werden jedoch von den Dateibehandlungsfunktionen im Fehlerfall in die Tracedatei geschrieben :

KXL_RC_FILE_OPEN_ERR (41)	Fehler beim Öffnen einer Datei
KXL_RC_FILE_READ_ERR (42)	Fehler beim Lesen einer Datei
KXL_RC_FILE_CLOSE_ERR (43)	Fehler beim Schließen einer Datei

Bei Fehlern, die der Parser feststellt (z.B. XML – Syntaxfehler im Dokument), wird eine Meldung in die Tracedatei ausgegeben.

10.2 Nutzung benutzerspezifischer Encoding-Funktionen

Zur Nutzung benutzerspezifischer Encoding-Funktionen gibt es mehrere Möglichkeiten:

1. Nach den Prototypen (siehe Abschnitt "Initialisierung der UTM-XML-Schnittstelle") werden eigene Funktionen entwickelt, die die Code-Umsetzung vornehmen. Sie müssen entweder über den ersten Aufruf von KXLInitEnv() oder mit einem Aufruf von xmlNewEncodingHandler() direkt dem Parser bekannt gemacht werden.
2. Eine bestehende Codetabelle (im Modul KXLCVLT.C) wird angepasst, das Modul neu übersetzt und eingebunden. Die Änderungen werden im entsprechenden Encoding wirksam (auf Windows nicht möglich!).
3. Eine neue Codetabelle wird nach der Struktur bestehender Tabellen aufgebaut. Die nach den Prototypen aufgebaute Funktion ruft jeweils nur eine von UTM-XML zur Verfügung gestellte Funktion mit der jeweiligen Codetabelle auf. Diese Funktionen müssen wie bei Möglichkeit 1 dem Parser bekannt gemacht werden. Diese Funktionen können auch in Cobol ohne großen Aufwand erstellt werden (siehe Beispielprogramm KXLCOBST.CBL).

Zur Realisierung von 2. oder 3. ist im folgenden der Aufbau der Tabellen erklärt.

10.2.1 Codetabelle UserEncoding -> UTF-8

Die folgende Tabelle zeigt den Aufbau der Codetabelle vom User Encoding nach UTF-8 am Beispiel der Tabelle für EDF03DRV. Für jedes Zeichen in EDF03DRV mit Binär-Wert i steht an der Position i (0 .. 255) in der Tabelle der entsprechende hexadezimale UTF-8-Code. Nicht definierte Zeichen werden in 0x1a umgesetzt.

```
unsigned char xmlunicodetable_EDF03DRV[256] = {
         0x00, 0x01, 0x02, 0x03, 0x1a, 0x09, 0x1a, 0x7f, /* X'00' - X'07' */
         0x1a, 0x1a, 0x1a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f, /* X'08' - X'0F' */
         0x10, 0x11, 0x12, 0x13, 0x1a, 0x1a, 0x08, 0x1a, /* X'10' - X'17' */
         0x18, 0x19, 0x1a, 0x1a, 0x1c, 0x1d, 0x1e, 0x1f, /* X'18' - X'1F' */
         0x1a, 0x1a, 0x1a, 0x1a, 0x1a, 0x0a, 0x17, 0x1b, /* X'20' - X'27' */
         0x1a, 0x1a, 0x1a, 0x1a, 0x1a, 0x05, 0x06, 0x07, /* X'28' - X'2F' */
         0x1a, 0x1a, 0x16, 0x1a, 0x1a, 0x1a, 0x1a, 0x04, /* X'30' - X'37' */
         0x1a, 0x1a, 0x1a, 0x1a, 0x14, 0x15, 0x1a, 0x1a, /* X'38' - X'3F' */
         0x20, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, /* X'40' - X'47' */
         0x1a, 0x1a, 0x60, 0x2e, 0x3c, 0x28, 0x2b, 0xe4, /* X'48' - X'4F' */
         0x26, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, /* X'50' - X'57' */
         0x1a, 0x1a, 0x21, 0x24, 0x2a, 0x29, 0x3b, 0x1a, /* X'58' - X'5F' */
         0x2d, 0x2f, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, /* X'60' - X'67' */
         0x1a, 0x1a, 0x5e, 0x2c, 0x25, 0x5f, 0x3e, 0x3f, /* X'68' - X'6F' */
         0x1a, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, /* X'70' - X'77' */
         0x1a, 0x1a, 0x3a, 0x23, 0x40, 0x27, 0x3d, 0x22, /* X'78' - X'7F' */
         0x1a, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, /* X'80' - X'87' */
         0x68, 0x69, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, /* X'88' - X'8F' */
         0x1a, 0x6a, 0x6b, 0x6c, 0x6d, 0x6e, 0x6f, 0x70, /* X'90' - X'97' */
         0x71, 0x72, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, 0x20ac,/* X'98' - X'9F'*/
         0x1a, 0x1a, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78, /* X'A0' - X'A7' */
         0x79, 0x7a, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, /* X'A8' - X'AF' */
         0x1a, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, /* X'B0' - X'B7' */
         0x1a, 0x1a, 0x1a, 0xd6, 0xc4, 0xdc, 0x1a, 0x1a, /* X'B8' - X'BF' */
         0x1a, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, /* X'C0' - X'C7' */
         0x48, 0x49, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, /* X'C8' - X'CF' */
         0x1a, 0x4a, 0x4b, 0x4c, 0x4d, 0x4e, 0x4f, 0x50, /* X'D0' - X'D7' */
         0x51, 0x52, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, /* X'D8' - X'DF' */
         0x1a, 0x1a, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58, /* X'E0' - X'E7' */
         0x59, 0x5a, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, 0x1a, /* X'E8' - X'EF' */
         0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, /* X'F0' - X'F7' */
         0x38, 0x39, 0x1a, 0xf6, 0x1a, 0xfc, 0x1a, 0xdf  /* X'F8' - X'FF' */
         };
```

Im Cobol sieht die entsprechende Tabelle wie folgt aus:

```
01 EBCDIC-TO-UTF8-TAB.
  03 TAB-00-07 PIC N(8)  VALUE NX"00000001000200030001A0009001A007F".
  03 TAB-08-0F PIC N(8)  VALUE NX"001A001A001A000B000C000D000E000F".
…
  03 TAB-F0-F7 PIC N(8)  VALUE NX"00300031003200330034003500360037".
  03 TAB-F8-FF PIC N(8)  VALUE NX"00380039001A00F6001A00FC001A00DF".
```

10.2.2 Codetabelle UTF-8 -> UserEncoding

Die folgende Tabelle zeigt den Aufbau der Codetabelle von UTF-8 ins User Encoding am Beispiel der Tabelle für EDF03DRV. Da UTF-8-Zeichen ein bis drei Byte umfassen können, ist die Tabelle mehrstufig aufgebaut und berücksichtigt die Standard-Belegung der führenden Bits:

0xxx xxxx	1-Byte Zeichen
110x xxxx	1. Byte eines 2-Byte-Zeichens
1110 xxxx	1. Byte eines 3-Byte Zeichens
10xx xxxx	Folgebyte eines 2- oder 3-Byte Zeichen

Für jedes 1-Byte-Zeichen in UTF-8 mit Binär-Wert i steht an der Position i (0 .. 127) in der Tabelle (table part 1) der entsprechende hexadezimale EDF03DRV-Code.

Für die 2-Byte-Zeichen erfolgt die Umsetzung zweistufig:

Im ersten Schritt wird abhängig vom 1. Byte, das standardmäßig zwischen C0 und DF liegt, in der Tabelle (table part 2) der Index einer Tabelle in table part 4 gelesen. In dieser 64 Byte langen Tabelle steht abhängig vom 2. Byte (ohne Berücksichtigung der ersten 2 Bits) der entsprechende Hexadezimalwert des Zeichens in EDF03DRV.

Für die 3-Byte-Zeichen erfolgt die Umsetzung dreistufig:

Im ersten Schritt wird abhängig vom 1. Byte, das standardmäßig zwischen E0 und EF liegt, in der Tabelle (table part 3) der Index einer Tabelle in table part 4 gelesen. In dieser 64 Byte langen Tabelle steht abhängig vom 2. Byte (ohne Berücksichtigung der ersten 2 Bits) der Index der Tabelle in table part 4,in der abhängig vom 3. Byte (ohne Berücksichtigung der ersten 2 Bits) der entsprechende Hexadezimalwert des Zeichens in EDF03DRV.

Nicht definierte Zeichen werden in \x00 umgesetzt., ebenso die nicht definierten Indizes in table part 2 und part 3 und in den Tabellen der 2. Stufe bei Umsetzung von 3-Byte-Zeichen. Die erste 64-Byte-Tabelle in table part 4 besteht nur aus \x00 – Werten, d.h. alle nicht definierten Werte werden auf \x00 gesetzt.

Alle weiteren Tabellen in table part 4 können anwenderspezifisch aufgebaut werden und müssen über die entsprechenden Stellen in table part 2 und part 3 referenziert werden.

```
unsigned char const xmltranscodetable_EDF03DRV[256 + 48 + 4 * 64] = {

/* table part 1: 1 Byte code */
        "\x00\x01\x02\x03\x37\x2d\x2e\x2f\x16\x05\x15\x0b\x0c\x0d\x0e\x0f" /* X'00' - X'0F' */
        "\x10\x11\x12\x13\x3c\x3d\x32\x26\x18\x19\x3f\x27\x1c\x1d\x1e\x1f" /* X'10' - X'1F' */
        "\x40\x5a\x7f\x7b\x5b\x6c\x50\x7d\x4d\x5d\x5c\x4e\x6b\x60\x4b\x61" /* X'20' - X'2F' */
        "\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\x7a\x5e\x4c\x7e\x6e\x6f" /* X'30' - X'3F' */
        "\x7c\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xd1\xd2\xd3\xd4\xd5\xd6" /* X'40' - X'4F' */
        "\xd7\xd8\xd9\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xbb\xbc\xbd\x6a\x6d" /* X'50' - X'5F' */
        "\x4a\x81\x82\x83\x84\x85\x86\x87\x88\x89\x91\x92\x93\x94\x95\x96" /* X'60' - X'6F' */
        "\x97\x98\x99\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xfb\x4f\xfd\xff\x07" /* X'70' - X'7F' */
/* table part2: 2 Byte code 1st step */
    "\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00" /* 2 Byte code 1st step */
    "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
/* table part3: 3 Byte code 1st step */
    "\x00\x00\x02\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00" /* 3 Byte code 1st step */
/* table part4: 2/3 Byte code 2nd step: */
    "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00" /* first Byte not valid */
    "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
    "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
    "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
```

```
    "\x00\x00\x00\x00\xbc\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00" /* first Byte  = C3 */
                                                                   /* X'c380' - X'c38f' */
    "\x00\x00\x00\x00\x00\x00\xbb\x00\x00\x00\x00\x00\xbd\x00\x00\xff" /* X'c390' - X'c39f' */
    "\x00\x00\x00\x00\x4f\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00" /* X'c3a0' - X'c3af' */
    "\x00\x00\x00\x00\x00\x00\xfb\x00\x00\x00\x00\x00\xfd\x00\x00\x00" /* X'c3b0' - X'c3bf' */

    "\x00\x00\x03\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00" /* 3 Byte code 2nd step */
    "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
    "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
    "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"

    "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00" /* X'E28280' - */
    "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
    "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x9f\x00\x00"
    "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00" /* - X'E282bf' */
};
```

Im Cobol sieht die entsprechende Tabelle wie folgt aus:

```
 01 UTF8-TO-EBCDIC-TAB.
   03 TAB-00-0F PIC N(8) VALUE NX"00010203372D2E2F1605150B0C0D0E0F".
   03 TAB-10-1F PIC N(8) VALUE NX"101112133C3D322618193F271C1D1E1F".

 …
   03 TAB-E2828X PIC N(8) VALUE NX"00000000000000000000000000000000".
   03 TAB-E2829X PIC N(8) VALUE NX"00000000000000000000000000000000".
   03 TAB-E282AX PIC N(8) VALUE NX"0000000000000000000000009F000000".
   03 TAB-E282BX PIC N(8) VALUE NX"00000000000000000000000000000000".
```

10.2.3 Deklaration benutzerspezifischer Encoding-Funktionen

Wird eine Code-Tabelle wie oben beschrieben aufgebaut, können die Encoding-Funktionen wie folgt mit UTM-XML-Mitteln implementiert werden.

In C sehen dann die Funktionen wie folgt aus:

```
int EDF03DRVToUTF8 (unsigned char* out, int *outlen,
                    const unsigned char* in, int *inlen)
{
    int result;
    result = EBCDxToUTF8 (out, outlen, in, inlen,
                          xmlunicodetable_EDF03DRV);
    return result;
}

int UTF8ToEDF03DRV (unsigned char* out, int *outlen,
                    const unsigned char* in, int *inlen)
{
    int result;
    result = UTF8ToEBCDx (out, outlen, in, inlen,
                          xmltranscodetable_EDF03DRV);
    return result;
}
```

In Cobol sehen die Funktionen wie folgt aus:

```
      *  encoding functions UTF-8 -> EBCDIC, EBCDIC -> UTF-8
      *
       ENTRY "CVUTOE"   USING OUTBUF OUTLEN INBUF INLEN.
      *
       CALL "UTF8ToEBCDx"
            USING OUTBUF, OUTLEN, INBUF, INLEN, UTF8-TO-EBCDIC-TAB.
       EXIT PROGRAM.
      *
       ENTRY "CVETOU"   USING OUTBUF OUTLEN INBUF INLEN.
      *
       CALL "EBCDxToUTF8"
            USING OUTBUF, OUTLEN, INBUF, INLEN, EBCDIC-TO-UTF8-TAB.
       EXIT PROGRAM.
```

10.3 Bearbeitung von Dokumenten ohne encoding-Attribut

Bei plattformübergreifender Verarbeitung von Dokumenten ist es in manchen Fällen problematisch, wenn ein Dokument ein encoding-Attribut enthält. Wenn bei Übertragung des Dokumentes (z.B. mit ftp) eine Code-Umsetzung erfolgt, entspricht das aktuelle Encoding nicht mehr dem angegebenen encoding-Attribut. Das führt bei den verarbeitenden Programmen zu Problemen. Umgekehrt sollen Dokumente ohne encoding-Attribut an der UTM-XML-Schnittstelle verarbeitet werden können, die in einem dem Anwender bekannten Encoding (ungleich UTF-8) zur Verfügung stehen.

Beispiel:

Ein Dokument wird im BS2000 erzeugt, z.B. mit encoding="EBCDIC", und wird mit ftp auf Windows übertragen. Textdateien werden i.d.R. automatisch von EBCDIC in ein ASCII-Encoding umgewandelt. Tools wie z.B. xml-Editoren können die Datei nicht verarbeiten, wenn die Datei das Attribut encoding="EBCDIC" enthält.

Zur Lösung des Problems soll ein mit der UTM-XML-Schnittstelle erzeugtes XML-Dokument zwar in einem bestimmten Encoding erstellt werden (insbes. im BS2000 in einer EBCDIC-Variante), aber ohne encoding-Attribut.

Das ist ein Verhalten, das im Gegensatz zum Vorgehen des libxml2 Parsers steht (siehe http://xmlsoft.org/encoding.html), der jedes Dokument ohne encoding-Attribut als UTF-8-encodiertes Dokument interpretiert.

Um das gewünschte Verhalten trotzdem zu erhalten, wird mit dem Encoding "NONE" eine Sonderbehandlung durchgeführt.

Mit dem Alias-Namen "NONE" ist folgendes Verhalten verbunden:

- Vor Verarbeitung eines XML-Dokumentes ohne encoding-Attribut müssen Sie encoding="NONE" mit KXLSetEncodingAlias als Aliasnamen für das gewünschte Encoding setzen (z.B. für das Home-Encoding EDF04DRV).
- Soll ein Dokument in dem gewünschten Encoding (z.B. EDF04DRV) ohne encoding Attribut erzeugt werden, setzen Sie das Dokumenten-Encoding auf NONE (KXLSetDocEncoding). Beim Erzeugen des Dokuments (KXLConvObjToDoc) wird das Dokument ohne encoding-Attribut im gewünschten Encoding erstellt.
- Soll ein Dokument ohne encoding-Attribut eingelesen werden (Voraussetzung: Encoding entspricht dem mit NONE definierten), wird in der UTM-XML-Schnittstelle das Encoding NONE zur Codeumsetzung nach UTF-8 verwendet und intern als Encoding im Objektbaum vermerkt. Ist das Encoding NONE nicht definiert, wird defaultmäßig von Encoding =UTF-8 ausgegangen. Es erfolgt keine Codeumsetzung.

Beispiel:

```
xmlNodePtr  pXMLroot;
char *     pBuffer;
short      rc =0;

/* Deklarieren des Alias-Namens NONE */
KXLSetEncodingAlias ( "NONE", "EDF03DRV", &rc);

/* Erzeugen eines Dokumentes */
pXMLroot        = KXLCreateNewObj("mydoc", {"\0", "\0"}, &rc);
/* Bearbeitung des Dokuments */
…
/* Setzen des Dokumenten-Encodings NONE */
KXLSetDocEnc ( pXMLroot, "NONE", &rc);

/* Erzeugen des Dokumentes ohne encoding Attribut */
pBuffer = KXLConvObjToDoc(pXMLroot , NULL, NULL, &rc);
```

Ausgegebenes Dokument (im Encoding EDF03DRV):

```
<?xml version="1.0"?> <mydoc>…</mydoc>
```


10.4 Bearbeitung externer Dokumente

Mit gewissen Einschränkungen ist es möglich, Dokumente, die nicht mit dem UTM-XML API erzeugt worden sind, (im Folgenden 'extern' genannt) mit den hier beschriebenen Funktionsaufrufen zu bearbeiten. Das Verhalten wird im Folgenden kurz beschrieben. Die Beschreibung erhebt aber keinen Anspruch auf Vollständigkeit.

10.4.1 Root – Knoten

Dokumente, die mit KXLCreateNewObj ohne Angabe von Name und Inhalt erzeugt wurden, haben einen API-spezifischen root – Knoten mit tag – Namen UTM-XML (<UTM-XML version="03.0"></UTM-XML>). Den root-Knoten selber kann man mit KXLReadNode lesen. Darüber hinaus gelten die gleichen Einschränkungen wie beim Zugriff auf andere Knoten.

10.4.2 Aufbau der Knoten

Die mit dem UTM-XML API erstellten Objektknoten werden folgendermaßen aufgebaut:

- Der angegebene Name des Elements wird der tag – Name
- Der Elementtyp (string, auf den t_value.pType zeigt) wird in einem Attribut mit Namen 'type' abgelegt. Ist das die leere Zeichenfolge, wird kein type-Attribut erzeugt. Intern werden solche Elemente wie type=struct (innere Knoten) bzw. type=string (äußere Knoten) behandelt.
- Der Wert des Elements (string, auf den t_value.pValue zeigt) wird als Wert des Knotens abgelegt
- Elemente, die Feldelemente darstellen (Kindknoten von Feldern mit type = array), werden mit zusätzlichem Attribut 'arrayelem = "y" abgelegt.

Beispiel:

```
t_value = {"char","y"};
KXLWrite (pNode, "bezahlt", tval, NULL);
```

erzeugt den Knoten:

```
ELEMENT bezahlt
    ATTRIBUTE type
        TEXT
            content = char
    TEXT
        content = y
```

und entspricht dem Element:

```
<bezahlt type="char"> y </bezahlt>
```

10.4.3 Schreiben von Knoten

1. Die im API beschriebenen Typ-Konvertierungsfunktionen können beliebig durch eigene Konvertierungsfunktionen ersetzt werden, wenn etwa höhere Genauigkeit bei float gewünscht ist.

2. Wenn ohne die Typ-Konvertierungsfunktionen des UTM-XML API gearbeitet wird, ist es auch möglich, Knoten beliebigen Typs zu schreiben. Der Anwender muss dann dem Schreibaufruf selbst die gewünschten Daten in einer t_value – Struktur bereitstellen. Es ist darauf zu achten, dass mehrstufige Namen (die '/' oder '[..]' enthalten) keine Namensteile, außer dem letzten, mit externen Typ-Bezeichnungen enthalten.

   Beispiel mit '/':
   Zum Schreiben eines Knotens 'eins/zwei/drei' mit Wert '123' und 'eins' mit Typ 'myType1', 'zwei' mit Typ 'struct', 'drei' mit Typ 'myType3' muss der Knoten mit eigenem Typ explizit geschrieben werden:

```
KXLWrite    pname->'eins',          tval.pType->'myType1',   tval.pValue->' '
KXLWrite    pname->'zwei/drei',     tval.pType->'myType3',   tval-pValue->'123'
```


Beispiel mit '[ ]':
Schreiben eines Knotens 'Adr[Meier]' mit 'Adr' vom Typ 'Adresse' und 'Meier' vom Typ 'Person' mit Wert 'Finkenweg 6':

```
KXLWrite    pname->'Adr',          tval.pType->'Adresse',    tval.pValue->' '
KXLWrite    pname->'[Meier]'       tval.pType->'Person',     tval.pValue->'Finkenweg 6'
```

Achtung: Beim Schreiben von Feldelementen (wie 'Meier' im Beispiel oben) ist zu beachten, dass dem Elementnamen intern ein '_' vorangestellt wird (dadurch sind numerische Angaben möglich).

3. Feldelemente werden dem (numerischen) Index nach aufsteigend geordnet. Der Index eines Feldelementes kann aber auch nicht-numerisch sein. Dann werden die Elemente nach vorangestellten Ziffern numerisch geordnet, bei gleichem "Nummernpräfix" werden die Indizes lexikalisch geordnet. So werden z.B. die Elemente [0],[1],[2],[12],[a],[0a],[01a],[1a],[1b],[1a2],[12a] in folgender Reihenfolge abgelegt:
   [a],[0],[0a],[1],[01a],[1a],[1a2],[1b],[2],[12],[12a]

10.4.4 Lesen von Knoten

Da bei Feldelementen intern dem Feldnamen ein Byte vorangestellt wird, wird beim Lesen von Namen, die in eckigen Klammern stehen, intern nach Knotennamen ohne das 1. Byte gesucht
Beispiel:
Ein Knoten mit internem Name 'pVALUE' kann mit KXLRead pname->'[VALUE]' gelesen werden.

10.5 Beispiel 1

In diesem Abschnitt wird ein komplettes Beispiel mit Aufbau der Objektstruktur gezeigt.

10.5.1 Programmcode:

Folgende C – Datenstruktur soll in einem XML – Dokument abgelegt werden:

```
typedef struct {      Sdate         date;
                      Saddress      address;
                      Sarticle      article[3];
                      double        total;
                      char          paid;
               }Saccount;
```

wobei

```
typedef struct {      short  day;
                      int    month;
                      long   year;
               }Sdate;

typedef struct {      char   name[20];
                      char   street[20];
                      int    postalCode;
                      char   town[20];
                      char   phone[20];
               }Saddress;
typedef struct {      char orderNumber[10];
                      char name[20];
                      float price;
                      int quantity;
               }Sarticle;
```


Der Programmcode sieht wie folgt aus:

```
xmlNodePtr pXMLroot, pXMLact, pXMLstruct, pXMLarr;
char *  pBuffer;
short   rc    =0;
short   day   =1;
int     month =1;
long    year  =2013;
char    name[]     = "Max Meier";
char    street[]   = "Mondstr. 55";
int     postalCode = 80808;
char    town []    = "Muenchen";
char    phone[]    = "089/12345678";
double  total = 0;
t_value tval1 = {NULL, NULL};
t_value tval2 = {"\0", "\0"};
int     i;
char    Stylesheet[ ] = "href=\"mystyle.ccs\" title=\"Compact\"\ type=\"text/ccs\"";
char *  pNewBuffer;

Sarticle art[] = {
   {"A12345", "eraser", 1.50, 1},
   {"B23456", "pencil", 1.00, 4},
   {"C34567", "CD", 4.50, 1},
    };
char *field_nr[3] = {"[0]", "[1]", "[2]" };
char *pName;

pXMLroot      = KXLCreateNewObj("mydoc", tval2, &rc);                          (1)
/* result pXMLroot is pointer to "mydoc" node */

/* write node account implicitly, node date explicitly */
pXMLstruct = KXLWrite(pXMLroot, "account/date", KXLFromStruct(NULL), &rc);     (2)
/* result pXMLstruct is pointer to struct date node */

/* write structure Sdate */
pXMLact = KXLWrite(pXMLroot,"account/date/day",KXLFromShort(day),&rc);         (3)
pXMLact = KXLWrite(pXMLstruct, "month", KXLFromInt(month),&rc);                (4)
pXMLact = KXLWrite(pXMLstruct, "year", KXLFromLong(year),&rc);                 (5)
/* node account exists; create new node address */
pXMLstruct = KXLWrite(pXMLroot, "account/address", KXLFromStruct(NULL), &rc);  (6)
/* pXMLStruct is pointer struct address node */

/* write structure address */
pXMLact = KXLWrite(pXMLstruct, "name", KXLFromString(name),&rc);                (7)
pXMLact = KXLWrite(pXMLstruct, "street", KXLFromString(street),&rc);            (8)
pXMLact = KXLWrite(pXMLstruct, "postalCode", KXLFromInt(postalCode),&rc);       (9)
pXMLact = KXLWrite(pXMLstruct, "town", KXLFromString(town),&rc);                (10)
pXMLact = KXLWrite(pXMLstruct, "phone", KXLFromString(phone),&rc);              (11)

pXMLstruct = KXLSetParentNode(pXMLstruct, &rc);
/* pXMLstruct now pointer to node account */
pXMLarr    =KXLWrite(pXMLstruct, "article", KXLFromArray( ),&rc);               (12)
/* pXMLarr pointer to array article node */

for (i = 0; i<=2; i++) /* write field elements = structures of type Sarticle */
{
  pXMLstruct=KXLWrite(pXMLarr, field_nr[i], KXLFromStruct("Sarticle"),&rc);     (13)
  /* pXMLstruct pointer to struct [0], [1] or [2]*/
  /* write structure node with value Sarticle */
  pXMLact =KXLWrite(pXMLstruct, "orderNumber",
                    KXLFromString(art[i].orderNumber),&rc);                     (14)
  pXMLact =KXLWrite(pXMLstruct, "name", KXLFromString(art[i].name),&rc);        (15)
  pXMLact =KXLWrite(pXMLstruct, "price", KXLFromFloat(art[i].price),&rc);       (16)
  pXMLact =KXLWrite(pXMLstruct,"quantity",KXLFromInt(art[i].quantity),&rc);     (17)
  /* end of write structure elements */
  /* calculate accumulated total */
  total = total + art[i].price * art[i ].quantity;
}
/* write total and paid nodes */
pXMLact =KXLWrite( pXMLroot, "account/total", KXLFromDouble(total),&rc);        (18)
pXMLact =KXLWrite( pXMLroot, "account/paid", KXLFromChar('n'),&rc);             (19)
```



```
pBuffer = KXLConvObjToDoc(pXMLroot ,(char*)&Stylesheet,NULL, &rc);              (20)
/* pBuffer  is pointer to the buffer containing the converted object */
/* now you may send pBuffer with XML-document e.g. to a communication partner */

/* examples for read calls: */
/* read the root node */
tval1 = KXLReadNode(pXMLroot, KXL_FALSE, &pName, &rc);                          (21)
/* 2 possible read calls, e.g. the name of the first article: */
/* read directly */
tval1 = KXLRead (pXMLroot, "account/article[0]/name",KXL_FALSE, &rc);           (22)
/* or position and read */
pXMLstruct = KXLSetSubObject (pXMLroot, "account/article[0]", &rc);             (23)
tval2 = KXLRead (pXMLstruct, "name", KXL_FALSE, &rc);                           (24)

/* end handling: */
pXMLact = pXMLstruct = pXMLarr = NULL;  /* clear pointer */
KXLFreeObj(pXMLroot, &rc);              /* free XML-objects */                  (25)
pXMLroot = NULL;
```

10.5.2 Programmtrace

Eine mit Tracemode=F erzeugte Tracedatei vom obigen Beispiel sieht in Auszügen wie folgt aus.
Dabei verweisen die Zahlen in Klammern am rechten Rand auf die Aufrufe im Programm, die Buchstaben in Klammern auf Anmerkungen im Anschluss.

```
/XML for UTM V03.0A0  -- TRACE 1 -- PID:02356 -- Wed Nov 23 13:23:23 2005 (a)
WIN32 osversion=2600 winversion=5.1 cpumode=protected mode
Tracemode: Full, Maximum Size: 16384 KByte

13:23:23 KXLTSENV
         Set Trace environment:
13:23:23 KXLTSENV: Trace initiated
13:23:23 KXLCreateNewObj init: N=mydoc, T=, V=                                (1)
13:23:23 KXLInitEnv init
13:23:23 KXLCheckInitVersion init, Version 30
13:23:23 KXLCheckInitVersion exit,Parser V=20620, RC=0
13:23:23 KXLCheckCreaVersion init, Version 30
13:23:23 KXLCheckCreaVersion exit,Parser V=20620, RC=0
13:23:23 KXLCheckConvVersion init, Version 30
13:23:23 KXLCheckConvVersion exit, RC=0
13:23:23 KXLCheckReadVersion init, Version 30
13:23:23 KXLCheckReadVersion exit, RC=0
13:23:23 KXLCheckTracVersion init, Version 30
13:23:23 KXLCheckTracVersion exit, RC=0
13:23:23 KXLCheckScmaVersion init, Version 30
13:23:23 KXLCheckScmaVersion exit, RC=0
13:23:23 KXLInitEncHdlr init
13:23:23 KXLInitEncHdlr exit, RC=0
13:23:23 KXLInitEnv exit,
13:23:23 KXLAnalyseName init, N=mydoc
13:23:23 KXLAnalyseName exit, N=, LTH=5, ET=1, P_LTH=-1
13:23:23 KXLCreateNewObj exit,XML-A=00325F28, RC=0
13:23:23 KXLFromStruct init, ST=(null)
13:23:23 KXLFromStruct exit, T=struct, V=(null)
13:23:23 KXLWrite init, XML-A=00325F28, N=account/date, T=struct, V=        (2)
...
13:23:23 KXLWrite exit,XML-A=003263E0,RC=0
13:23:23 KXLFromShort init, SH=   1
13:23:23 KXLFromShort exit, T=short, V=1
13:23:23 KXLWrite init, XML-A=00325F28, N=account/date/day, T=short, V=1  (3)
...
13:23:23 KXLWrite exit,XML-A=003268F8,RC=0
13:23:23 KXLFromInt init, IN=       1
13:23:23 KXLFromInt exit, T=int, V=1
13:23:23 KXLWrite init, XML-A=003263E0, N=month, T=int, V=1                 (4)
...
13:23:23 KXLWrite exit,XML-A=00326B80,RC=0
13:23:23 KXLFromLong init, LG=     7D6
13:23:23 KXLFromLong exit, T=long, V=2006
13:23:23 KXLWrite init, XML-A=003263E0, N=year, T=long, V=2006              (5)
...
```



```
13:23:23 KXLWrite exit,XML-A=00326EE8,RC=0
13:23:23 KXLFromStruct init, ST=(null)
13:23:23 KXLFromStruct exit, T=struct, V=(null)
13:23:23 KXLWrite init, XML-A=00325F28, N=account/address, T=struct, V=   (6)
...
13:23:23 KXLWrite exit,XML-A=00327308,RC=0
13:23:23 KXLFromString init, ST=Max Meier
13:23:23 KXLFromString exit, T=string, V=Max Meier
13:23:23 KXLWrite init, XML-A=00327308, N=name, T=string, V=Max Meier     (7)
...
13:23:23 KXLWrite exit,XML-A=003277C8,RC=0
13:23:23 KXLFromString init, ST=Mondstr. 55
13:23:23 KXLFromString exit, T=string, V=Mondstr. 55
13:23:23 KXLWrite init, XML-A=00327308, N=street, T=string, V=Mondstr. 55 (8)
...
13:23:23 KXLWrite exit,XML-A=00327F20,RC=0
13:23:23 KXLFromInt init, IN=   13BA8
13:23:23 KXLFromInt exit, T=int, V=80808
13:23:23 KXLWrite init, XML-A=00327308, N=postalCode, T=int, V=80808      (9)
...
13:23:23 KXLWrite exit,XML-A=00327CA0,RC=0
13:23:23 KXLFromString init, ST=Muenchen
13:23:23 KXLFromString exit, T=string, V=Muenchen
13:23:23 KXLWrite init, XML-A=00327308, N=town, T=string, V=Muenchen      (10)
...
13:23:23 KXLWrite exit,XML-A=00328130,RC=0
13:23:23 KXLFromString init, ST=089/12345678
13:23:23 KXLFromString exit, T=string, V=089/12345678
13:23:23 KXLWrite init, XML-A=00327308, N=phone, T=string, V=089/12345678 (11)
...
13:23:23 KXLWrite exit,XML-A=003284A8,RC=0
13:23:23 KXLSetParentNode init, XML-A=00327308
13:23:23 KXLSetParentNode exit, XML-A=00326228, RC=0
13:23:23 KXLFromArray init
13:23:23 KXLFromArray exit, T=array
13:23:23 KXLWrite init, XML-A=00326228, N=article, T=array, V=            (12)
...
13:23:23 KXLWrite exit,XML-A=00328D70,RC=0
13:23:23 KXLFromStruct init, ST=Sarticle
13:23:23 KXLFromStruct exit, T=struct, V=Sarticle
13:23:23 KXLWrite init, XML-A=00328D70, N=[0], T=struct, V=Sarticle       (13)
..
13:23:23 KXLWrite exit,XML-A=00328890,RC=0
13:23:23 KXLFromString init, ST=A12345
13:23:23 KXLFromString exit, T=string, V=A12345
13:23:23 KXLWrite init, XML-A=00328890, N=orderNumber, T=string, V=A12345 (14)
...
13:23:23 KXLWrite exit,XML-A=003291C0,RC=0
13:23:23 KXLFromString init, ST=eraser
13:23:23 KXLFromString exit, T=string, V=eraser
13:23:23 KXLWrite init, XML-A=00328890, N=name, T=string, V=eraser        (15)
...
13:23:23 KXLWrite exit,XML-A=00329568,RC=0
13:23:23 KXLFromFloat init, FL=         0000c03f
|...?           |
13:23:23 KXLFromFloat exit, T=float, V=1.5
13:23:23 KXLWrite init, XML-A=00328890, N=price, T=float, V=1.5           (16)
...
13:23:23 KXLWrite exit,XML-A=003298E0,RC=0
13:23:23 KXLFromInt init, IN=       1
13:23:23 KXLFromInt exit, T=int, V=1
13:23:23 KXLWrite init, XML-A=00328890, N=quantity, T=int, V=1            (17)
...
13:23:23 KXLWrite exit,XML-A=00329C00,RC=0
...
13:23:23 KXLFromDouble init, DO=         00000000 00002440
|......$@        |
13:23:23 KXLFromDouble exit, T=double, V=10.
13:23:23 KXLWrite init, XML-A=00325F28, N=account/total, T=double, V=10.  (18)
...
13:23:23 KXLWrite exit,XML-A=0032C830,RC=0
13:23:23 KXLFromChar init, CH=n
13:23:23 KXLFromChar exit, T=char, V=n
13:23:23 KXLWrite init, XML-A=00325F28, N=account/paid, T=char, V=n       (19)
```

```
...
13:23:23 KXLWrite exit,XML-A=0032CA68,RC=0
13:23:23 KXLConvObjToDoc init, XML-A=00325F28, S=href="mystyle.ccs" title="Compact"
type="text/ccs"  (20)
...
13:23:23 Output of the converted XML document:                              (b)
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="mystyle.ccs" title="Compact" type="text/ccs"?>
<mydoc>
  <account type="struct">
    <date type="struct">
      <day type="short">1</day>
      <month type="int">1</month>
      <year type="long">2006</year>
    </date>
    <address type="struct">
      <name type="string">Max Meier</name>
      <street type="string">Mondstr. 55</street>
      <postalCode type="int">80808</postalCode>
      <town type="string">Muenchen</town>
      <phone type="string">089/12345678</phone>
    </address>
    <article type="array">
      <_0 type="struct" ArrayElem="y">Sarticle
        <orderNumber type="string">A12345</orderNumber>
        <name type="string">eraser</name>
        <price type="float">1.5</price>
        <quantity type="int">1</quantity>
      </_0>
      <_1 type="struct" ArrayElem="y">Sarticle
        <orderNumber type="string">B23456</orderNumber>
        <name type="string">pencil</name>
        <price type="float">1.</price>
        <quantity type="int">4</quantity>
      </_1>
      <_2 type="struct" ArrayElem="y">Sarticle
        <orderNumber type="string">C34567</orderNumber>
        <name type="string">CD</name>
        <price type="float">4.5</price>
        <quantity type="int">1</quantity>
      </_2>
    </article>
    <total type="double">10.</total>
    <paid type="char">n</paid>
  </account>
</mydoc>

13:23:23 KXLConvObjToDoc exit, DOC-A=0032F280, LTH=1180, RC=0      ende   (20)
13:23:23 KXLReadNode init, XML-A=00325F28, FULL=0                          (21)
...
13:23:23 KXLReadNode exit, N=mydoc, T=, V=, RC=0
13:23:23 KXLRead init, XML-A=00325F28, N=account/article[0]/name, DEL=0    (22)
...
13:23:23 KXLRead exit, T=string, V=eraser, RC=0
13:23:23 KXLSetSubObject init, XML-A=00325F28, N=account/article[0]        (23)
...
13:23:23 KXLSetSubObject exit, XML-A=00328890, RC=0
13:23:23 KXLRead init, XML-A=00328890, N=name, DEL=0                       (24)
...
13:23:23 KXLRead exit, T=string, V=eraser, RC=0
13:23:23 KXLFreeObj init, XML-A=00325F28                                   (25)
13:23:23 KXLFreeObj exit, RC=0
```

Anmerkungen:

(a) Kopf der Tracedatei

(b) bei Tracemodus=F wird das konvertierte XML-Dokument mitprotokolliert; zur besseren Lesbarkeit wurde das Dokument hier formatiert

10.6 Beispiel 2

In diesem Abschnitt wird ein Beispiel mit einigen neuen Funktionen der V2.0 (Namensräume, Attribute) gezeigt.

Dabei steht folgendes Dokument in pBuffer und wird bearbeitet:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<account xmlns="http://www.mydoc.de"\
          xmlns:adr="http://www.myaddr.de"
          customerNumber="12345678" status="paid">
  <date>20/03/2013</date>
  <adr:address type="struct">
    <adr:name >Max Meier</adr:name>
    <adr:street >Mondstr. 55</adr:street>
    <adr:postalCode >80808</adr:postalCode>
    <adr:town >Muenchen</adr:town>
    <adr:phone>089/12345678</adr:phone>
    </adr:address>
  <article section="office">
    <orderNumber>A12345</orderNumber>
    <name>eraser</name>
    <price currency="Euro">1.5</price>
    <quantity>1</quantity>
  </article>
  <article  section="office">
    <orderNumber>B23456</orderNumber>
    <name>pencil</name>
    <price currency="Euro">1.</price>
    <quantity>4</quantity>
  </article>
  <article  section="music">
    <orderNumber>C34567</orderNumber>
    <name>CD</name>
    <price currency="Euro">4.5</price>
    <quantity>1</quantity>
  </article>
  <deliveryCharges currency="Euro">2.</deliveryCharges>
  <total currency="Euro">12.</total>
</account>
```

10.6.1 Programmcode

Der Programmcode sieht wie folgt aus, wobei folgende Unterfunktionen verwendet werden:

```
void stock_office(char * orderNumber, int quantity)
void stock_music(char * orderNumber, int quantity)
void stock_house(char * orderNumber, int quantity)
void finances(char * customer_no, float deliveryCharge, double total)
float conv_to_euro(char * val, float f)
static void  print_RC_and_msg(char * apiCall, short rc);

   xmlNodePtr    pXMLroot, pXMLact, pXMLattr;
   short rc=0;
   char *pName;
   char customerNumber[10]="\0";
   char orderNumber[10]="\0";
   int status = 0;
   int i;
   float f, deliveryCharges = 0;
   double total=0;
   t_value tval;
   char *pPrintBuffer;

pXMLroot    =KXLConvDocToObj(pBuffer, &rc);                          (a)
/* check returncode */
if (rc != KXL_RC_OK)
{ print_RC_and_msg ("KXLConvDocToObj", rc);
  return;
```

```
}
/* read first attribute of the root node */
tval = KXLReadAttr(pXMLroot, &pName, &pXMLact, &rc);                    (b)
while (rc == KXL_RC_OK)                                                 (b)
{ /* read custumer number and status */
    if (strcmp (pName, "@customerNumber")== 0)
         strcpy (customerNumber, tval.pValue);
    if (strcmp (pName, "@status")== 0)
         if (strcmp (tval.pValue,"paid")==0)
             status = 1;
    /* read next attribute */
    tval = KXLReadNextSib(pXMLact, &pName, &pXMLact, &rc);              (b)
}

if ( status)
/* process all articles in loop, if account is paid */
{ /* set to first article */
    pXMLact = KXLSetSubObject (pXMLroot, "article", &rc);               (c)
    while (rc == KXL_RC_OK)
        { /* read parameter and pass them on to processing */
        tval = KXLRead (pXMLact, "orderNumber", KXL_FALSE, &rc);
        strcpy(orderNumber, tval.pValue);
        tval = KXLRead (pXMLact, "quantity", KXL_FALSE, &rc);
        i = KXLToInt(tval, NULL);
        tval = KXLRead (pXMLact, "@section", KXL_FALSE, &rc);
        if (strcmp(tval.pValue, "office")==0)
            stock_office (orderNumber, i);

        if (strcmp(tval.pValue, "music")==0)
            stock_music (orderNumber, i);

        if (strcmp(tval.pValue, "house")==0)
            stock_house (orderNumber, i);

        /* set to next article ( = read all elements of the nodelist )*/
        pXMLact = KXLSetSubObject (pXMLact, "[+]", &rc);                (c)
     }

    /* read deliveryCharges and total */
    tval = KXLRead (pXMLroot, "deliveryCharges", KXL_FALSE, &rc);
    f = KXLToFloat (tval, NULL);
    tval = KXLRead (pXMLroot, "deliveryCharges/@currency",
                    KXL_FALSE, &rc);                                    (d)
    if (strcmp (tval.pValue, "Euro")==0)
        deliveryCharges = f;
    else
        deliveryCharges = conv_to_euro (tval.pValue, f);

    tval = KXLRead (pXMLroot, "total", KXL_FALSE, &rc);
    f = KXLToFloat (tval, NULL);
    tval = KXLRead (pXMLroot, "total/@currency", KXL_FALSE, &rc);       (d)
    if (strcmp (tval.pValue, "Euro")==0)
        total = f;
    else
        total = conv_to_euro (tval.pValue, f);

    /* passing on to financial section */
    finances (customerNumber, deliveryCharges, total);
}
/* clear pointer */
pXMLact =  pXMLattr = NULL;
/* free XML object */
KXLFreeObj(pXMLroot, &rc);
pXMLroot = NULL;
```

Bemerkungen zum Programmcode:

(a) Dokument wird konvertiert

(b) Lesen aller Attribute des Knotens: Das erste wird mit KXLReadAttr gelesen, alle weiteren mit KXLReadNextSib, wobei die zurückgegebene Elementadresse jeweils für den folgenden Aufruf verwendet wird. Die Namensraum-Definitionen werden hier nicht gelesen.

(c) Alle Elemente einer nodelist bearbeiten: auf das erste Element wird mit Angabe des Namens positioniert; bei den weiteren Positionieraufrufen wird als Name nur '[+]' angegeben.

(d) direktes Lesen eines Attributes

10.7 Beispiel 3

In diesem Abschnitt wird ein Beispiel mit den Schema Funktionen gezeigt.

Dabei steht folgendes Dokument in pXMLDocBuffer:

```
<?xml version="1.0" encoding="UTF-8"?>
<adr:addressbook xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xmlns:adr = "http://www.xml.test/test/ADR"
                 xsi:schemaLocation="http://www.xml.test/test/ADR
                                     http://www.xml.test/test/adr.xsd">
  <adr:address>
    <adr:name>Anke Alpha</adr:name>
    <adr:street>Astreet</adr:street>
    <adr:postalCode>A1234</adr:postalCode>
    <adr:town>Atown</adr:town>
    <adr:email>Alpha@web.de</adr:email>
  </adr:address>
  <adr:address>
    <adr:name>Bodo Beta</adr:name>
    <adr:street>Bstreet</adr:street>
    <adr:postalCode>B5678</adr:postalCode>
    <adr:town>Btown</adr:town>
  </adr:address>
</adr:addressbook>
```

und folgendes Schema in pSchemaBuffer:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.xml.test/test/ADR"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:adr="http://www.xml.test/test/ADR"
        xmlns="http://www.w3.org/2001/XMLSchema"
         version = "1.0.0" elementFormDefault = "qualified">
<xsd:element name="person" type="adr:addressTypeDe"/>
<xsd:element name="addressbook">
 <xsd:complexType>
  <xsd:sequence minOccurs="0" maxOccurs="unbounded">
   <xsd:element name="address" type="adr:addressTypeDe"
                minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
<xsd:complexType name="addressTypeDe">
 <xsd:sequence minOccurs="1" maxOccurs="1">
  <xsd:element name="name" type="xsd:string" minOccurs="1" maxOccurs="1"/>
  <xsd:element name="street" type="xsd:string" minOccurs="1" maxOccurs="1"/>
  <xsd:element name="postalCode" type="xsd:string"
               minOccurs="1" maxOccurs="1"/>
  <xsd:element name="town" type="xsd:string" minOccurs="1" maxOccurs="1"/>
  <xsd:element name="phone" type="xsd:string"
               minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element name="email" type="xsd:string"
               minOccurs="0" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>
</schema>";
```


10.7.1 Programmcode

Der Programmcode sieht wie folgt aus, wobei

- create_printable_doc das Dokument druckaufbereitet
  char * create_printable_doc (char * pBuffer);
  und
- print_RC_and_msg den Returncode und ggf. Parsermeldungen ausgibt.
  void print_RC_and_msg (char * apiCall, short rc);

```
xmlNodePtr    pXMLDoc;
xmlSchemaPtr  pXMLSchema;
char*         pXmlDocBuffer;
char*         pSchemaBuffer;
short rc=0;
char *pLocalDir = "";
char *pPrintBuffer;

 pPrintBuffer = create_printable_doc(pXMLDocBuffer);
 printf ("xml document is:\n %s\n", pPrintBuffer);
 pPrintBuffer = create_printable_doc(pSchemaBuffer);
 printf ("schema document is:\n %s\n", pPrintBuffer);

 /* convert document to DOM object */
 pXMLDoc = KXLConvDocToObj(pXMLDocBuffer, &rc);
 print_RC_and_msg ("KXLConvDocToObj", rc);
 /* check returncode */
 if (rc != KXL_RC_OK)
   return;

 /* parse schema from buffer */
 pXMLSchema = KXLParseSchema (pSchemaBuffer, pLocalDir, &rc);
 print_RC_and_msg ("KXLParseSchema", rc);
 /* same as following call with file adr.xsd containing the schema
    pXMLSchema = KXLParseSchemaFile ("adr.xsd", pLocalDir, &rc);
    print_RC_and_msg ("KXLParseSchemaFile", rc);
 /* */
 /* check returncode */
 if (rc != KXL_RC_OK)
   return;

 /* validate xml object vs. schema */
 KXLValidDoc (pXMLDoc, pXMLSchema, &rc);
 print_RC_and_msg ("KXLValidDoc", rc);
 /* instead of KXLConvDocToObj and KXLValidDoc also do the following call
    if you don't need pXMLDoc:
    KXLValidDocBuf (pXMLDocBuffer, pXMLSchema, &rc);
    print_RC_and_msg ("KXLValidDocBuf", rc);
 /* */

/* validation of XML document with Schema file adr.xsd in local directory */
 pXMLDoc =KXLConvDocToObjAndValid(pXMLDocBuffer,KXL_TRUE, pLocalDir, &rc);
 print_RC_and_msg ("KXLConvDocToObjAndValid", rc);

 /* free pointer to doc and schema object */
 KXLFreeSchema(pXMLSchema, NULL);
 KXLFreeObj(pXMLDoc, NULL);
```

10.8 Literaturverweise

[DOM – Spezifikation] DOM – Spezifikation des W3C (www – Konsortium)
web – page: http://www.w3.org/DOM

[NS – Spezifikation] Namespace – Recommendation des W3C (www – Konsortium)
web – page: http://www.w3.org/TR/REC-xml-names/

[XML – Spezifikation] XML – Spezifikation des W3C (www – Konsortium)
web – page: http://www.w3.org/XML

[XML-Schema – Definition] http://de.wikipedia.org/wiki/XML_Schema
http://www.gi-ev.de/service/informatiklexikon/informatiklexikon-detailansicht/meldung/99/

[XML-Schema – Spezifikation] web – pages: http://www.w3.org/TR/xmlschema-0/
http://www.w3.org/TR/xmlschema-1/
http://www.w3.org/TR/xmlschema-2/

[GNOME - Parser] Parser GNOME XML –library V1.0,
freeware from Daniel Veillard
web – page: http://xmlsoft.org

[UTF8] http://de.wikipedia.org/wiki/UTF-8

[libxml internalization support] http://xmlsoft.org/encoding.html

[IO-Interfaces] http://xmlsoft.org/xmlio.html

