# BEX – Binary Encoded XML

«BEX – Binary Encoded XML» beschreibt eine Vereinfachung des «Document Object Model» («DOM») sowie ein binäres Datenformat zur redundanzarmen Abbildung der Daten eines «DOM» Dokuments. Ziel dieses Formats ist es, eine leichtgewichtige, nur lesende «DOM»-Implementation darauf aufsetzen zu können, welche signifikant weniger Arbeitsspeicher verbraucht, als eine zumeist auch modifizierende Implementation einer Standard «XML» Bibliothek. Die Modifikation der Daten ist nicht vorgesehen.

## Datenmodell

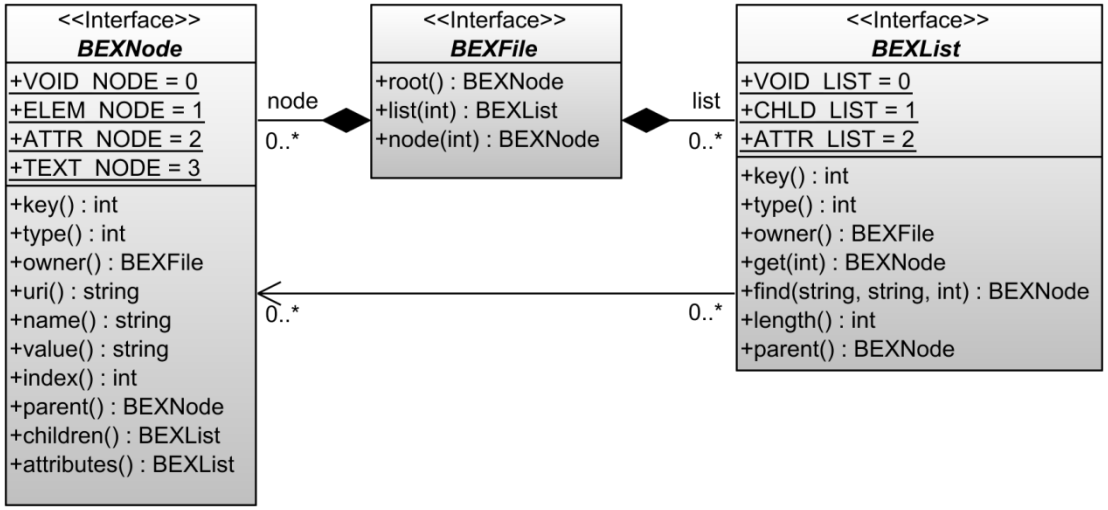


Abbildung 1 Klassendiagramm «BEX – Binary Encoded XML»

Die Schnittstelle «`BEXFile`» bildet den Ausgangspunkt des Datenmodells und steht für ein Dokument (vgl. «XML» Datei). Element-, Text- und Attributknoten werden homogen über die Schnittstelle «`BEXNode`» repräsentiert. Die Kind- und Attributknotenlisten von Elementknoten werden über die Schnittstelle «`BEXList`» vereinheitlicht abgebildet.

| **BEXFile** | |
|---|---|
| Die Schnittstelle «`BEXFile`» definiert die Verwaltung aller Element-, Text- und Attributknoten sowie aller Kind- und Attributknotenlisten, die in einem Dokument (vgl. «XML» Datei) enthalten sind. | |
| Methode | Beschreibung |
| `root()` | Diese Methode gibt das Wurzelelement des Dokuments zurück. |
| `list(key)` | Diese Methode gibt die Knotenliste mit dem gegebenen Identifikator zurück. Wenn der Identifikator unbekannt ist, wird eine undefinierte Knotenliste geliefert. Der gegebene Identifikator kann von dem der gelieferten Knotenliste abweichen. |
| `node(key)` | Diese Methode gibt den Knoten mit dem gegebenen Identifikator zurück. Wenn der Identifikator unbekannt ist, wird ein undefinierter Knoten geliefert. Der gegebene Identifikator kann von dem des gelieferten Knoten abweichen. |

Tabelle 1 Schnittstelle «BEXFile»

| **BEXNode** | |
|---|---|
| Die Schnittstelle «`BEXNode`» definiert die homogene Sicht auf Element-, Text- und Attributknoten. In besonderen Fällen wird sie auch zur Abbildung undefinierter Knoten verwendet. Die aufsteigende Navigation von einem Kind- bzw. Attributknoten zu dessen Elternknoten ist optional. | |
| Methode | Beschreibung |
| `key()` | Diese Methode gibt den Identifikator dieses Knoten zurück. |
| `type()` | Diese Methode gibt die Typkennung dieses Knoten zurück. Die Typkennung ist bei einem Attributknoten «`1`», bei einem Elementknoten «`2`», bei einem Textknoten «`3`» und bei einem undefinierten Knoten «`0`». |
| `owner()` | Diese Methode gibt das diesen Knoten verwaltende Objekt zurück. |
| `uri()` | Diese Methode gibt den URI des Namensraums dieses Knoten als Zeichenkette zurück. Der URI eines Textknoten, eines Element- bzw. Attributknoten ohne Namensraum sowie eines undefinierten Knoten ist leer. |
| `name()` | Diese Methode gibt den Namen dieses Knoten als Zeichenkette zurück. Der Name eines Textknoten sowie eines undefinierten Knoten ist leer. |
| `value()` | Diese Methode gibt den Wert dieses Knoten als Zeichenkette zurück. Der Wert eines Elementknoten ohne Kindknoten sowie eines undefinierten Knoten ist leer. Der Wert eines Elementknoten mit Kindknoten entspricht dem Wert seines ersten Kindknoten. |
| `index()` | Diese Methode gibt die Position dieses Knoten in der Kind- bzw. Attributknotenliste des Elternknotens zurück (optional). Die Position eines undefinierten Knoten ist «`-1`». Wenn die Navigation zum Elternknoten deaktiviert ist, ist die Position jedes Knoten «`-1`». |
| `parent()` | Diese Methode gibt den Elternknoten dieses Knoten zurück (optional). Der Elternknoten des Wurzelelementknoten sowie eines undefinierten Knoten ist ein undefinierter Knoten. Wenn die Navigation zum Elternknoten deaktiviert ist, ist der Elternknoten jedes Knoten ein undefinierter Knoten. |
| `children()` | Diese Methode gibt die Kindknotenliste dieses Knoten zurück. Die Kindknotenliste eines Text- bzw. Attributknoten sowie eines undefinierten Knoten ist eine undefinierte Knotenliste. |
| `attributes()` | Diese Methode gibt die Attributknotenliste dieses Knoten zurück. Die Kindknotenliste eines Text- bzw. Attributknoten sowie eines undefinierten Knoten ist eine undefinierte Knotenliste. |

Tabelle 2 Schnittstelle «BEXNode»

| **BEXList** | |
|---|---|
| Die Schnittstelle «`BEXList`» definiert die homogene Sicht auf Kind- und Attributknotenlisten. Die aufsteigende Navigation von einer Knotenliste zu deren Elternknoten ist optional. | |
| Methode | Beschreibung |
| `key()` | Diese Methode gibt den Identifikator dieser Knotenliste zurück. |
| `type()` | Diese Methode gibt die Typkennung dieser Knotenliste zurück. Die Typkennung ist bei einer Attributknotenliste «`1`», bei einer allgemeinen Kindknotenliste «`2`» und bei einer undefinierten Knotenliste «`0`». |
| `owner()` | Diese Methode gibt das diese Knotenliste verwaltende Objekt zurück. |
| `get(index)` | Diese Methode gibt den «index»-ten Knoten dieser Knotenliste zurück. Bei einem ungültigen «index» wird ein undefinierter Knoten geliefert. |
| `find(uri, name, start)` | Diese Methode sucht linear ab der gegebenen «start»-Position den ersten Element- bzw. Attributknoten mit der gegebenen «uri» sowie dem gegebenen «name» und gibt dessen Position zurück. Bei einer erfolglosen Suche wird «`-1`» geliefert. Ein leerer «uri» bzw. «name» wird bei der Suche ignoriert, d.h. der gesuchte Knoten hat einen beliebigen URI bzw. Namen. Bei einer negativen «start»-Position wird immer «`-1`» geliefert. |
| `length()` | Diese Methode gibt die Länge dieser Knotenliste zurück. Die Länge ist bei einer undefinierten Knotenliste «0». |
| `parent()` | Diese Methode gibt den Elternknoten dieser Knotenliste zurück (optional). Der Elternknoten ist bei einer undefinierten Knotenliste ein undefinierter Knoten. Wenn die Navigation zum Elternknoten deaktiviert ist, ist der Elternknoten jeder Knotenliste ein undefinierter Knoten. |

Tabelle 3 Schnittstelle «BEXList»

## Datenformat

Das Datenformat baut auf dem «IAM – Integer Array Model» auf und nutzt das dort definierte, primitiven Datenformat «UINT32» sowie das strukturierte Datenformate «IAM_INDEX».

**BEX_FILE**

Die Datenstruktur «BEX_FILE» kodiert die Daten eines «BEXFile».

Alle Text- und Elementknoten sind in den Zeilen der Kindknotentabelle so abgelegt, dass jede Kindknotenliste einen Abschnitt aufeinanderfolgender Zeilen bildet und sich die Zeilenbereiche keiner zwei Kindknotenlisten überlagern. Analog gilt dies für die Attributknoten und Attributknotenlisten in der Attributknotentabelle. Beide Tabellen werden spaltenweise gespeichert.

Die Spalten der Kindknotentabelle sind «chldUriRef», «chldNameRef», «chldContentRef», «chldAttributesRef» und «chldParentRef», die der Attributknotentabelle sind «attrUriRef», «attrNameRef», «attrValueRef» und «attrParentRef». Sie alle enthalten Referenzen auf separat abgelegte Zeichenketten bzw. Tabellenabschnitte.

Die Zeichenketten werden in den Tabellen «attrUriText», «attrNameText», «attrValueText», «chldUriText», «chldNameText» und «chldValueText» abgelegt. Die Tabellenabschnitte sind dagegen in den Tabellen «chldListRange» und «attrListRange» kodiert.

In der Kindknotentabelle referenzieren die Spalten «chldUriRef» und «chldNameRef» je eine Zeichenkette in «chldUriText» bzw. «chldNameText», die Spalte «chldContentRef» entweder eine Zeichenkette in «chldValueText» oder eine Kindknotenliste in «chldListRange» und die Spalte «chldAttributesRef» eine Attributknotenliste in «attrListRange». In der Attributknotentabelle referenzieren die Spalten «attrUriRef», «attrNameRef» und «attrValueRef» je eine Zeichenkette in «attrUriText», «attrNameText» bzw. «attrValueText».

Wenn die Navigation von Kind- bzw. Attributknoten zu deren Elternknoten deaktiviert ist, sind die Spalten «chldParentRef» und «attrParentRef» leer. Andernfalls enthalten diese Spalten die Zeilennummern des Elternknoten in der Kindknotentabelle. Das Wurzelelement verweist hierbei auf sich selbst.

Bei einem Kindknoten verweist die Spalte «chldContentRef» mit einem positiven Wert auf eine Zeichenkette. Diese ist bei einem Textknoten dessen Wert und bei einem Elementknoten der Wert seines einzigen Kindknoten, welcher ein Textknoten ist. Ein leerer Elementknoten verweist damit immer auf die leere Zeichenkette. Ein negativer Wert in der Spalte «chldContentRef» verweist indirekt auf die Kindknotenliste eines Elementknoten. Zur Auflösung des Verweiseses muss dieser negiert werden. Bei einem Textknoten enthalten die Spalte «chldUriRef», «chldNameRef» und «chldAttributesRef» immer «0». Bei einem Elementknoten enthält die Spalte «chldNameRef» niemals den Wert «0».

Wenn die Spalte «chldUriRef» bzw. «attrUriRef» ausschließlich der Wert «0» enthielte, wird diese Spalte als leer gespeichert.

| Feld | Format | Anzahl | Beschreibung |
|---|---|---|---|
| HEADER | UINT32 | 1 | Dieses Feld speichert den Wert «0xBE10BA5E» und kennzeichnet damit die Datenstruktur und die Bytereihenfolge. Wenn diese Zahl direkt gelesen werden kann, liegen die Daten in der nativen Bytereihenfolge der Zielplattform vor. |
| rootRef | UINT32 | 1 | Dieses Feld speichert die Zeilennummer des Wurzelelementknoten in der Kindknotentabelle. |
| fileData | IAM_INDEX | 1 | Dieses Feld speichert einen «IAMIndex» mit «17» Listen («IAMList») und ohne Abbildungen («IAMMap»).<br>Die ersten «6» Listen (Tabellen «attrUriText», «attrNameText», «attrValueText», «chldUriText», «chldNameText» und «chldValueText») kodieren jeweils duplikatfreie Auflistung von Zeichenketten. Eine Zeichenkette liegt hierbei in nullterminierter «UTF8»-Kodierung als Byte- und damit Zahlenfolge («IAMArray») vor. Das erste Element einer solchen Auflistung ist immer die Zahlenfolge der leeren Zeichenkette. Die Ordnung der restlichen Elemente sollte deren Nutzungshäufigkeit entsprechen.<br>Die nächsten «9» Listen (Tabellen ) enthalten jeweils genau ein Element (Zahlenfolge), welches einer Spalte der Kind- bzw. Attributknotentabelle entspricht (Spalten «attrUriRef», «attrNameRef», «attrValueRef», «attrParentRef», «chldUriRef», «chldNameRef», «chldContentRef», «chldAttributesRef» und «chldParentRef»).<br>Die letzten «2» Listen (Tabellen «chldListRange» und «attrListRange») enthalten jeweils genau ein Element (Zahlenfolge), welches die Nummern der Zeilen in den Kind- bzw. Attributknotentabellen enthalten, ab denen die Kind- bzw. Attributknotenlisten beginnen, wobei das Ende einer Knotenliste zugleich der Beginn der nächsten Knotenliste ist. Die ersten beiden Elemente einer solchen Liste beschreiben immer eine leere Knotenliste. Die Ordnung der restlichen Knotenlisten sollte deren sollte deren Navigationspfad/Nutzungshäufigkeit entsprechen. |

Tabelle 4 Datenstruktur «BEX_FILE»