BEX – Binary Encoded XML

BEX – Binary Encoded XML beschreibt eine nur lesbare 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.

Datenmodell

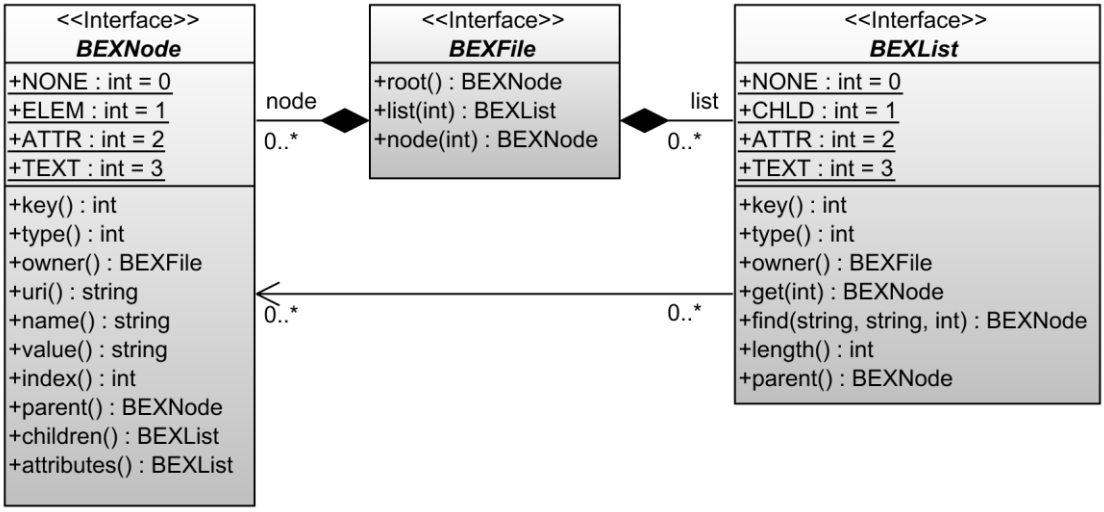

Abbildung 1 BEX – Binary Encoded XML

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.
`node(key)`	Diese Methode gibt den Knoten mit dem gegebenen Identifikator zurück. Wenn der Identifikator unbekannt ist, wird ein undefinierter Knoten geliefert.

Tabelle 1 BEXFile

BEXNode	
Die Schnittstelle `BEXNode` definiert die homogene Schnittstelle eines Element-, Text- bzw. 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 Elementknoten `1`, bei einem Attributknoten `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 BEXNode

BEXList	
Die Schnittstelle `BEXList` definiert die homogene Schnittstelle die Kind- bzw. Attributknotenlisten.	
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 allgemeinen Kindknotenliste `1`, bei einer Attributknotenliste `2` und bei einer undefinierten Knotenliste `0`. Bei einer Kindknotenliste, die nur aus einem Textknoten besteht, ist die Typkennung `3`.
`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 BEXList

Datenformat

Das Datenformat ist von dem des `IAM – Integer Array Model` abhängig und verwendet 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 als ein Auszug aufeinanderfolgender Zeilen beschrieben werden kann 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 *uri* (`chldUriRef`), *name* (`chldNameRef`), *content* (`chldContentRef`), *attributes* (`chldAttributesRef`) und *parent* (`chldParentRef`), die der Attributknotentabelle sind *uri* (`attrUriRef`), *name* (`attrNameRef`), *value* (`attrValueRef`) und *parent* (`attrParentRef`). Sie alle enthalten Referenzen auf separat abgelegte Nutzdaten.

In der Kindknotentabelle referenzieren die Spalten *uri* und *name* je eine Zeichenkette in `chldUriText` bzw. `chldNameText`, die Spalte *content* entweder eine Zeichenkette in `chldValueText` oder eine Kindknotenliste in `chldListRange` und die Spalte *attributes* eine Attributknotenliste in `attrListRange`. In der Attributknotentabelle referenzieren die Spalten *uri*, *name* und *value* je eine Zeichenkette in `attrUriText`, `attrNameText` bzw. `attrValueText`.

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

Bei einem Textknoten referenziert die *content* Spalte eine Zeichenkette und enthalten die *uri* Spalte, die *name* Spalte sowie die *attributes* Spalte immer `0`. Bei einem Elementknoten enthält die *name* Spalte niemals die `0`. Die *content* Spalte referenziert mit einem positiven Wert eine Kindknotenliste. Mit einem negativen Wert verweist sie dagegen auf die Zeichenkette des Textknoten, der der einzige Kindknoten des Elementknoten ist. Dieser Verweis ist zur Auflösung im Vorzeichen umzukehren. Demzufolge referenziert ein leerer Elementknoten immer auf die leere Kinknotenliste.

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 Referenz auf den Wurzelelementknoten des Dokuments.
`nodeData`	`IAM_INDEX`	`1`	Dieses Feld speichert einen `IAMIndex` mit `17` Listen (`IAMList`) und ohne Abbildungen. Die ersten `6` Listen (`attrUriText`, `attrNameText`, `attrValueText`, `chldUriText`, `chldNameText` und `chldValueText`) kodieren jeweils duplikatfreie, sortierte Auflistung von Zeichenketten. Eine Zeichenkette liegt hierbei in nullterminierter UTF8-Kodierung als Byte- und damit Zahlenfolge (`IAMArray`) vor. Die Sortierung erfolgt lexikografisch auf den Zahlenfolgen. Das erste Element einer solchen Auflistung ist immer die Zahlenfolge der leeren Zeichenkette. Die nächsten `9` Listen (`attrUriRef`, `attrNameRef`, `attrValueRef`, `attrParentRef`, `chldUriRef`, `chldNameRef`, `chldContentRef`, `chldAttributesRef` und `chldParentRef`) enthalten jeweils genau ein Element (Zahlenfolge), welches einer Spalte der Kind- bzw. Attributknotentabelle entspricht. Die letzten `2` Listen (`chldListRange` und `attrListRange`) enthalten jeweils genau ein Element (Zahlenfolge), welches die Nummern der Zeilen in den Kind- bzw. Attributknotentabellen, ab denen die Kind- bzw. Attributknotenlisten beginnen, wobei das Ende einer Liste gleich dem Beginn der nächsten Liste ist. Die Listen sind gemäß ihrer Länge sortiert, sodass die leeren Knotenlisten immer den Index `0` haben.

Tabelle 4 BEX_FILE