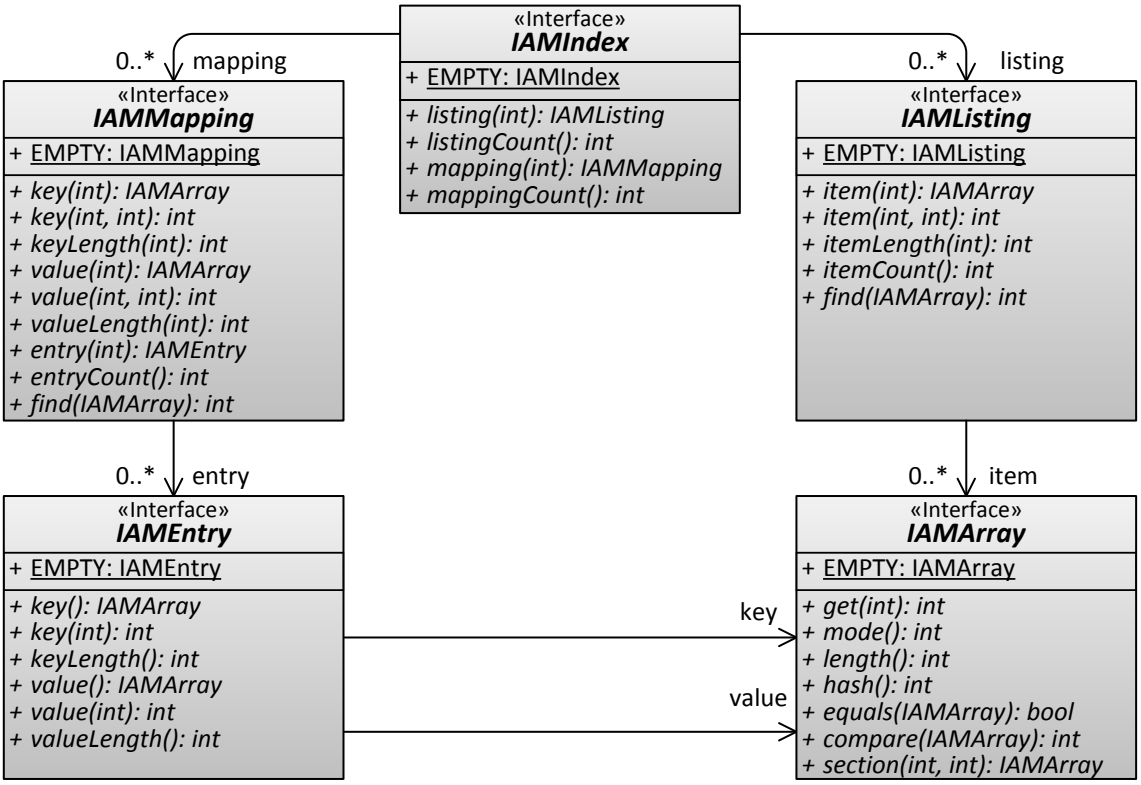
1 IAM – Integer Array Model

Das «Integer Array Model» oder kurz «IAM» ist ein abstraktes Datenmodell, welches aus konstanten Auflistungen von Zahlenfolgen sowie Abbildungen von Zahlenfolgen auf Zahlenfolgen besteht.



Ausgangspunkt des Datenmodells ist ein Inhaltsverzeichnis («`IAMIndex`»), über welches auf fast beliebig viele Abbildungen («`IAMMapping`») sowie Auflistungen («`IAMListing`») zugegriffen werden kann. Die Elemente der Auflistungen sowie die Schlüssel und Werte der Einträge («`IAMEntry`») in den Abbildungen sind Zahlenfolgen («`IAMArray`»). Da alle Datenelemente des Modells unveränderlich sind und keinen Verweis auf ihren Besitzer bereitstellen, gibt es jeweils nur eine leere Instanz.

1.1 Schnittstelle «IAMIndex»

`+ bee.creative.iam.IAMIndex`

Ein «`IAMIndex`» ist ein abstraktes Inhaltsverzeichnis zum positionsbasierten Zugriff auf die verwalteten Auflistungen («`IAMListing`») und Abbildungen («`IAMMapping`»).

`+ EMPTY: IAMIndex`

Dieses Feld speichert das leere Inhaltsverzeichnis.

`+ listing(index: int): IAMListing`

Diese Methode gibt die «index»-te Auflistung zurück.
Bei einem ungültigen «index» wird die leere Auflistung geliefert.

`+ listingCount(): int`

Diese Methode gibt die Anzahl der Auflistungen zurück («0…1073741823»).

`+ mapping(index: int): IAMMapping`

Diese Methode gibt die «index»-te Abbildung zurück.
Bei einem ungültigen «index» wird die leere Abbildung geliefert.

`+ mappingCount(): int`

Diese Methode gibt die Anzahl der Abbildungen zurück («0…1073741823»).

1.2 Schnittstelle «IAMListing»

+ bee.creative.iam.IAMListing

Ein «IAMListing» ist eine abstrakte Auflistung von Elementen, welche als Zahlenfolgen realisiert sind.

+ EMPTY: IAMListing

Dieses Feld speichert die leere Auflistung.

+ item(itemIndex: int): IAMArray

Diese Methode gibt das «itemIndex»-te Element dieser Auflistung zurück.
Bei einem ungültigen «itemIndex» wird die leere Zahlenfolge geliefert.

+ item(itemIndex: int, index: int): int

Diese Methode gibt die «index»-te Zahl des «itemIndex»-ten Elements zurück.
Bei einem ungültigen «index» oder «itemIndex» wird «0» geliefert.

+ itemLength(itemIndex: int): int

Diese Methode gibt die Länge des «itemIndex»-ten Elements zurück.
Bei einem ungültigen «itemIndex» wird «0» geliefert.

+ itemCount(): int

Diese Methode gibt die Anzahl der Elemente dieser Auflistung zurück («0…1073741823»).

+ find(item: IAMArray): int

Diese Methode gibt den Index des Elements zurück, das äquivalenten zum gegebenen ist.
Die Suche erfolgt linear vom ersten zum letzten Element. Bei erfolgloser Suche wird «-1» geliefert.

1.3 Schnittstelle «IAMArray»

+ bee.creative.iam.IAMArray

Ein «IAMArray» ist eine abstrakte Zahlenfolge, die als Element in Auflistungen oder Schlüssel bzw. Wert in Abbildungen vorkommt.

+ EMPTY: IAMArray

Dieses Feld speichert die leere Zahlenfolge.

+ get(index: int): int

Diese Methode gibt die «index»-te Zahl dieser Zahlenfolge zurück.
Bei einem ungültigen «index» wird «0» geliefert.

+ mode(): int

Diese Methode gibt die Größe jeder Zahl dieser Zahlenfolge zurück.
Diese Größe ist «0» für eine unspezifische Zahlenfolgen, «1» für «INT8»- sowie «UINT8»-Zahlen, «2» für «INT16»- sowie «UINT16»-Zahlen und «4» für «INT32»-Zahlen.

+ length(): int

Diese Methode gibt die Länge dieser Zahlenfolge zurück («0…1073741823»).

+ hash(): int

Diese Methode gibt den Streuwert zurück. Der Algorithmus hierfür ist:

```
int result = 0x811C9DC5;
for (int i = 0; i < this.length(); i++)
  result = (result * 0x01000193) ^ this.get(i);
return result;
```

+ equals(that: IAMArray): bool

Diese Methode gibt nur dann «true» zurück, wenn diese Zahlenfolge gleich der gegebenen Zahlenfolge «that» ist. Der Algorithmus hierfür ist:

```
if (this.length() != that.length()) return false;
for (int i = 0; i < this.length(); i++)
  if (this.get(i) != that.get(i)) return false;
return true;
```

+ compare(that: IAMArray): int

Diese Methode gibt «-1», «0» oder «1» zurück, wenn die Ordnung dieser Zahlenfolge lexikografisch kleiner, gleich bzw. größer als die Ordnung der gegebenen Zahlenfolge «that» ist. Der Algorithmus hierfür ist:

```
for (int i = 0, result; i < this.length() && i < that.length(); i++)
  if (this.get(i) < that.get(i)) return -1; else
    if (this.get(i) > that.get(i)) return +1;
if (this.length() < that.length()) return -1; else
  if (this.length() > that.length()) return +1; else
    return 0;
```

+ section(offset: int, length: int): IAMArray

Diese Methode gibt einen Abschnitt dieser Zahlenfolge ab der gegebenen Position «offset» und mit der gegebenen Länge «length» zurück.
Wenn der Abschnitt nicht innerhalb der Zahlenfolge liegt oder die Länge kleiner als «1» ist, wird eine leere Zahlenfolge geliefert.

1.4 Schnittstelle «IAMMapping»

+ bee.creative.iam.IAMMapping

Eine «IAMMapping» ist eine abstrakte Abbildung von Schlüsseln auf Werte, welche beide als Zahlenfolgen realisiert sind.

+ EMPTY: IAMMapping

Dieses Feld speichert die leere Abbildung.

+ key(entryIndex: int): IAMArray

Diese Methode gibt den Schlüssel des «entryIndex»-ten Eintrags zurück.
Bei einem ungültigen «entryIndex» wird die leere Zahlenfolge geliefert.

+ key(entryIndex: int, index: int): int

Diese Methode gibt die «index»-te Zahl des Schlüssels des «entryIndex»-ten Eintrags zurück.
Bei einem ungültigen «index» oder «entryIndex» wird «0» geliefert.

+ keyLength(entryIndex: int): int

Diese Methode gibt die Länge des Schlüssel des «entryIndex»-ten Eintrags zurück («0…1073741823»).
Bei einem ungültigen «entryIndex» wird «0» geliefert.

+ value(entryIndex: int): IAMArray

Diese Methode gibt den Wert des «entryIndex»-ten Eintrags zurück.
Bei einem ungültigen «entryIndex» wird die leere Zahlenfolge geliefert.

+ value(entryIndex: int, index: int): int

Diese Methode gibt die «index»-te Zahl des Werts des «entryIndex»-ten Eintrags zurück.
Bei einem ungültigen «index» oder «entryIndex» wird «0» geliefert.

+ valueLength(entryIndex: int): int

Diese Methode gibt die Länge des Werts des «entryIndex»-ten Eintrags zurück («0…1073741823»).
Bei einem ungültigen «entryIndex» wird «0» geliefert.

+ entry(entryIndex: int): IAMEntry

Diese Methode gibt den «entryIndex»-ten Eintrag dieser Auflistung zurück.
Bei einem ungültigen «entryIndex» wird der leere Eintrag geliefert.

+ entryCount(): int

Diese Methode gibt die Anzahl der Einträge dieser Auflistung zurück («0…1073741823»).

+ find(key: IAMArray): int

Diese Methode gibt den Index des Eintrags zurück, dessen Schlüssel äquivalenten zum gegebenen Schlüssel ist.
Die Suche erfolgt ordnungs- oder streuwertbasiert, d.h. indiziert. Bei erfolgloser Suche wird «-1» geliefert.

1.5 Schnittstelle «IAMEntry»

+ bee.creative.iam.IAMEntry

Ein «IAMEntry» ist ein abstrakter Eintrag einer Abbildung und besteht aus einem Schlüssel und einem Wert, jeweils als Zahlenfolge.

+ EMPTY: IAMEntry

Dieses Feld speichert den leeren Eintrag.

+ key(): IAMArray

Diese Methode gibt den Schlüssel dieses Eintrags zurück.

+ key(index: int): int

Diese Methode gibt die «index»-te Zahl des Schlüssels zurück.
Bei ungültigem «index» wird «0» geliefert.

+ keyLength(): int

Diese Methode gibt die Länge des Schlüssels zurück («0…1073741823»).

+ value(): IAMArray

Diese Methode gibt den Wert als Zahlenfolge zurück.

+ value(index: int): int

Diese Methode gibt die «index»-te Zahl des Werts zurück.
Bei ungültigem «index» wird «0» geliefert.

+ valueLength(): int

Diese Methode gibt die Länge des Werts zurück («0…1073741823»).

2 IAM-Datenformat

Das binäre optimierte Datenformat des «IAM» lagert Auflistungen und Abbildungen derart in eine Datei aus, dass diese als «memory-mapped-file» in den Arbeitsspeicher abgebildet und darauf sehr effiziente Lese- und Suchoperationen ausgeführt werden können.

Die primitiven Datenformate «**INT8**», «**INT16**» und «**INT32**» stehen für eins, zwei bzw. vier Byte lange, vorzeichenbehaftete, ganze Zahlen. Die vorzeichenlosen Varianten hierzu sind «**UINT8**», «**UINT16**» bzw. «**UINT32**». Die Bytereihenfolge in den mehr als ein Byte langen Formate sollte der nativen Bytereihenfolge der Zielplattform entsprechen, um eine maximale Leistungsfähigkeit zu erreichen.

Die Speicherbereiche einiger Datenfelder der nachfolgend aufgeführten Datenstrukturen besitzen mehrere Interpretationen und sind nur in bestimmten Datenstrukturvarianten present. Solche Datenfelder bzw. ihre Datenformate sind dazu mit den tiefgestellten Varianten gekennzeichnet und können leicht an den verbundenen Tabellenzellen erkennen.

Wenn diese Zahl direkt gelesen werden kann, liegen die Daten in der nativen Bytereihenfolge der Zielplattform vor.

2.1 Datenstruktur «IAM_INDEX»

Diese binäre Datenstruktur kodiert einen «**IAMIndex**» mit den darin eingebetteten Abbildungen («**IAM_MAPPING**») und Auflistungen («**IAM_LISTING**»).

Datenfeld	Format	Anzahl	Beschreibung
HEADER	UINT32	1	Dieses Feld speichert den Wert «0xF00DBA5E» 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.
mappingCount	UINT32	1	Dieses Feld speichert die Anzahl der Abbildungen («0…1073741823»).
listingCount	UINT32	1	Dieses Feld speichert die Anzahl der Listen («0…1073741823»).
mappingOffset	UINT32	mapCount+1	Dieses Feld speichert die Startpositionen der Abbildungen im Speicherbereich «mappingData». Die «i»-te Abbildung beginnt bzw. endet an den Positionen «mappingOffset[i]» bzw. «mappingOffset[i+1]» und hat eine Länge von «mappingOffset[i+1]-mappingOffset[i]». Die Startposition «mappingOffset[0]» ist «0». Die minimale Länge ist «4».
listingOffset	UINT32	listCount+1	Dieses Feld speichert die Startpositionen der Auflistungen im Speicherbereich «listingData». Die «i»-te Auflistung beginnt bzw. endet an den Positionen «listingOffset[i]» bzw. «listingOffset[i+1]» und hat eine Länge von «listingOffset[i+1]-listingOffset[i]». Die Startposition «listingOffset[0]» ist «0». Die minimale ist «3».
mappingData	UINT32	mapOffset [mapCount]	Dieses Feld speichert die kodierten Abbildungen («IAM_MAPPING»).
	IAM_MAPPING	mapCount	
listingData	UINT32	listOffset [listCount]	Dieses Feld speichert die kodierten Auflistungen («IAM_LISTING»).
	IAM_LISTING	listCount	

2.2 Datenstruktur «IAM_LISTING»

Diese binäre Datenstruktur kodiert ein «**IAMListing**» und respektiert hierbei folgende Aspekte:

- Die Zahlen in den Elementen können als «**INT8**», «**INT16**» oder «**INT32**» kodiert sein («ID=1…3»).
- Die Elemente können eine homogene Länge oder heterogene Längen haben («IL=0»). Die heterogenen Längen können als «**UINT8**», «**UINT16**» oder «**UINT32**» kodiert sein («IL=1…3»).

Datenfeld	Format	Anzahl	Beschreibung
HEADER	UINT32	1	Dieses Feld speichert den Wert «0xF00D200(ID:IL)» und kennzeichnet damit die Datenstruktur. Die Bitpaare «ID» und «IL» bestimmen den Elementdatentyp bzw. den Elementlängentyp.
itemCount	UINT32	1	Dieses Feld speichert die Anzahl der Elemente («0…1073741823»).
$itemLength_{IL=0}$	UINT32	1	Dieses Feld speichert die Länge eines Elements («0…1073741823»).
$itemOffset_{IL=1…3}$	$UINT8_{IL=1}$ $UINT16_{IL=2}$ $UINT32_{IL=3}$	itemCount+1	Dieses Feld speichert die Startpositionen der Elemente im Speicherbereich «itemData». Das «i»-te Element beginnt bzw. endet an den Positionen «itemOffset[i]» bzw. «itemOffset[i+1]» und hat eine Läge von «itemOffset[i+1]-itemOffset[i]». Die Startposition «itemOffset[0]» ist «0».
$-_{IL=1…2}$	UINT8	0…3	Dieses Feld ergänzt den Speicherbereich «itemOffset» auf eine restlos mit «UINT32» füllbare Größe.
$itemData_{IL=0}$	$INT8_{ID=1}$ $INT16_{ID=2}$ $INT32_{ID=3}$	itemSize* itemCount	Dieses Feld speichert die Zahlen der Elemente.
$itemData_{IL=1…3}$	$INT8_{ID=1}$ $INT16_{ID=2}$ $INT32_{ID=3}$	itemOffset [itemCount]	
$-_{IL=1…3}$	UINT8	0…3	Dieses Feld ergänzt den Speicherbereich «itemData» auf eine restlos mit «INT32» füllbare Größe.

2.3 Datenstruktur «IAM_MAPPING»

Diese binäre Datenstruktur kodiert ein «**IAMMapping**» und respektiert hierbei folgende Aspekte:

- Die Zahlen in den Schlüsseln bzw. Werten können als «**INT8**», «**INT16**» oder «**INT32**» kodiert sein («KD=1…3», «VD=1…3»).
- Die Schlüssel bzw. Werte können eine homogene Länge oder heterogene Längen haben («KL=0», «VL=0»). Die heterogenen Längen können als «**UINT8**», «**UINT16**» oder «**UINT32**» kodiert sein («KL=1…3», «VL=1…3»).
- Die Suche eines Schlüssel kann statt streuwertbasiert auch ordnungsbasiert erfolgen(«RL=0»). Die Indizes in der Streuwerttabelle können als «**UINT8**», «**UINT16**» oder «**UINT32**» kodiert sein («RL=1…3»).

Datenfeld	Format	Anzahl	Beschreibung
HEADER	UINT32	1	Dieses Feld speichert den Wert «0xF00D1(00:KD:KL:RL:VD:VL)» und kennzeichnet damit die Datenstruktur. Die Bitpaare «KD», «KL», «RL», «VD» und «VL» bestimmen den Schlüsseldatentyp, Schlüssellängentyp, Streuwertdatentyp, Wertdatentyp bzw. den Wertlängentyp.
entryCount	UINT32	1	Dieses Feld speichert die Anzahl der Einträge («0…1073741823»).

rangeMask$_{RL=1…3}$	UINT32	1	Dieses Feld speichert die Bitmaske zur Umrechnung des Streuwerts eines Schlüssels («IAMArray.hash()») in den Index des einzigen Schlüsselbereichs, in dem ein gesuchter Schlüssel enthalten sein kann («1…536870911»).
			Die Bitmaske muss eine um «1» verringerte Potenz von «2» sein. Ein Algorithmus zur Ermittlung der Bitmaske ist:
			`int result = 2;`
			`while (result < entryCount)`
			`  result = result << 1;`
			`return (result - 1) & 536870911;`

rangeData$_{RL=1…3}$	UINT8$_{RL=1}$ UINT16$_{RL=2}$ UINT32$_{RL=3}$	rangeMask+2	Dieses Feld speichert die Startindizes der Schlüsselbereiche. Ein Schlüsselbereich enthält die Indizes aller der Einträge, die einen Schlüssel mit einem unter Beachtung der Bitmaske äquivalenten Streuwert besitzen. Der «i»-te Schlüsselbereich beginnt bzw. endet mit den Indizes «rangeData[i]» bzw. «rangeData[i+1]» und enthält «rangeData[i+1]-rangeData[i]» Indizes. Der Startindex «rangeData[0]» ist «0».
-$_{RL=1…2}$	UINT8	0…3	Dieses Feld ergänzt den Speicherbereich «rangeData» auf eine restlos mit «UINT32» füllbare Größe.
keyLength$_{KL=0}$	UINT32	1	Dieses Feld speichert die Länge eines Schlüssels («0…1073741823»).
keyOffset$_{KL=1…3}$	UINT8$_{KL=1}$ UINT16$_{KL=2}$ UINT32$_{KL=3}$	entryCount+1	Dieses Feld speichert die Startpositionen der Schlüssel im Speicherbereich «keyData». Der «i»-te Schlüssel beginnt bzw. endet an den Positionen «keyOffset[i]» bzw. «keyOffset[i+1]» und hat eine Läge von «keyOffset[i+1]-keyOffset[i]». Die Startposition «keyOffset[0]» ist «0».
-$_{KL=1…2}$	UINT8	0…3	Dieses Feld ergänzt den Speicherbereich «keyOffset» auf eine restlos mit «UINT32» füllbare Größe.
keyData$_{KD=0}$	INT8$_{KD=1}$ INT16$_{KD=2}$ INT32$_{KD=3}$	keySize* entryCount	Dieses Feld speichert die Zahlen der Schlüssel.
keyData$_{KD=1…3}$	INT8$_{KD=1}$ INT16$_{KD=2}$ INT32$_{KD=3}$	keyOffset [entryCount]	
-$_{KD1…2}$	UINT8	0…3	Dieses Feld ergänzt den Speicherbereich «keyData» auf eine restlos mit «UINT32» füllbare Größe.
valueLength$_{VL=0}$	UINT32	1	Dieses Feld speichert die Länge eines Werts («0…1073741823»).
valueOffset$_{VL=1…3}$	UINT8$_{VL=1}$ UINT16$_{VL=2}$ UINT32$_{VL=3}$	entryCount+1	Dieses Feld speichert die Startpositionen der Werte im Speicherbereich «valueData». Der «i»-te Wert beginnt bzw. endet an den Positionen «valueOffset[i]» bzw. «valueOffset[i+1]» und hat eine Läge von «valueOffset[i+1]-valueOffset[i]». Die Startposition «valueOffset[0]» ist «0».
-$_{VL=1…2}$	UINT8	0…3	Dieses Feld ergänzt den Speicherbereich «valueOffset» auf eine restlos mit «UINT32» füllbare Größe.
valueData$_{VD=0}$	INT8$_{VD=1}$ INT16$_{VD=2}$ INT32$_{VD=3}$	valueSize* entryCount	Dieses Feld speichert die Zahlen der Werte.
valueData$_{VD=1…3}$	INT8$_{VD=1}$ INT16$_{VD=2}$ INT32$_{VD=3}$	valueOffset [entryCount]	
-$_{VD1…2}$	UINT8	0…3	Dieses Feld ergänzt den Speicherbereich «valueData» auf eine restlos mit «UINT32» füllbare Größe.

3 IAM-Austauschformate

Die Datenaustauschformate für «IAM» sind menschenlesbare Textformate, welche durch manuelle Bearbeitung angepasst und mit Hilfe eines Transformationswerkzeugs in das binäre Datenformat überführt werden können.

3.1 Textdatenformat «STRING»

Dieser Datentyp steht für eine Zeichenkette (vgl. «XML-Schema-Data-Type: string»).

3.2 Textdatenformat «INTEGER»

Dieser Datentyp steht für eine vorzeichenlose Dezimalzahl (vgl. «XML-Schema-Data-Type: int»).

3.3 Textdatenformat «FINDMODE»

Dieser textbasierte Aufzählungstyp definiert die Schlüsselsuchmodus, für welche ein «**IAM_MAPPING**» kodiert ist oder werden soll.

Die Ausprägungen «""», «"A"» und «"AUTO"» geben an, dass der Schlüsselsuchmodus bei der Kodierung vom genutzten Werkzeug gewählt wird. Im Gegensatz dazu geben die Ausprägungen «"S"» und «"SORTED"» sowie «"H"» und «"HASHED"» an, dass das «**IAM_MAPPING**» für eine ordnungsbasierte bzw. streuwertbasierte Suche kodiert ist oder werden soll.

3.4 Textdatenformat «BYTEORDER»

Dieser textbasierte Aufzählungstyp definiert die Bytereihenfolgen, in denen ein «**IAM_INDEX**» kodiert ist oder werden soll.

Die Ausprägungen «""», «"A"» und «"AUTO"» geben an, dass die Bytereihenfolge bei der Kodierung vom genutzten Werkzeug gewählt wird. Im Gegensatz dazu geben die Ausprägungen «"B"» und «"BIGENDIAN"» sowie «"L"» und «"LITTLEENDIAN"» an, dass der «**IAM_INDEX**» in der Bytereihenfolge «big-endian» bzw. «little-endian» kodiert ist oder werden soll.

3.5 Textdatenformat «ARRAYFORMAT»

Dieser textbasierte Aufzählungstyp definiert das Zahlenfolgenformat, in welchem eine Zahlenfolge zur Kodierung angegeben werden kann.

Das Zahlenfolgenformat kann i.d.R. nur bei der Kodierung berücksichtigt werden, da beim Dekodieren nicht zweifelsfrei erkannt werden kann, ob eine Zahlenfolge eine Zeichenkette darstellt und in welchem Format diese kodiert ist.

Die Ausprägungen «""», «"A"» und «"ARRAY"» geben an, dass die Zahlen der Folge als Dezimalzahlen mit optionalem Vorzeichen leerzeichenseparierт angegeben werden (z.B. «""», «"12"», «"12 -34 5 -6"»). Im Gegensatz dazu geben die Ausprägungen «"B"» und «"BINARY"» an, dass die Zahlen der Folge als zweistellige hexadezimale Zahlen in großen Buchstaben und ohne Trennzeichen angegeben werden (z.B. «""», «"12"», «"12ABF0"»). Bei den Ausprägungen «"UTF-8"», «"UTF-16"» und «"UTF-32"» steht jeder Token des «8/16/32-Bit UCS Transformation Format» für eine Zahl der Folge, wobei ein Zeichen mit bis zu vier Token kodiert sein kann und bei den Ausprägungen «"CP-1252"», «"ISO-8859-1"» und «"ISO-8859-15"» steht jedes Zeichen des entsprechenden 8-Bit-Formats für eine Zahl der Folge.

3.6 INI-Datenformat

Die INI-Datei beginnt mit dem Inhaltsverzeichnis «**[IAM_INDEX]**», welches von bis zu «**1073741823**» Abschnitten für Abbildungen «**[IAM_MAPPING]**» bzw. Auflistungen «**[IAM_LISTING]**» gefolgt wird.

3.6.1 INI-Abschnitt «[IAM_INDEX]»

Dieser INI-Abschnitt kodiert einen «**IAMIndex**» mit einer bestimmten Anzahl von Abbildungen und Auflistungen sowie der Bytereihenfolge zur Kodierung des Binärformats.

Datenfeld	Format	Anzahl	Beschreibung
byteOrder	BYTEORDER	1	Diese Eigenschaft gibt die Bytereihenfolge an.
mappingCount	INTEGER	1	Diese Eigenschaft gibt die Anzahl der Abbildungen an.
listingCount	INTEGER	1	Diese Eigenschaft gibt die Anzahl der Auflistungen an.

3.6.2 INI-Abschnitt «[IAM_MAPPING]»

Dieser INI-Abschnitt kodiert Einträge eines «**IAMMapping**» und kann für eine Abbildung sogar mehrfach vorkommen.

Datenfeld	Format	Anzahl	Beschreibung
index	INTEGER	1	Diese Eigenschaft gibt den Index der Abbildung an.
findMode	FINDMODE	1	Diese Eigenschaft gibt den Schlüsselsuchmodus an.
keyFormat	ARRAYFORMAT	1	Diese Eigenschaft gibt das Zahlenfolgenformat der Schlüssel an.
valueFormat	ARRAYFORMAT	1	Diese Eigenschaft gibt das Zahlenfolgenformat der Werte an.
…	STRING	1…1073741823	Alle weiteren Eigenschaften geben jeweils einen Eintrag der Abbildung an. Name und Wert der Eigenschaften geben dabei Schlüssel und Wert eines Eintrags an.

3.6.3 INI-Abschnitt «[IAM_LISTING]»

Dieser INI-Abschnitt beschreibt Elemente eines «**IAMListing**» und kann für eine Auflistung sogar mehrfach vorkommen.

Datenfeld	Format	Anzahl	Beschreibung
index	INTEGER	1	Diese Eigenschaft gibt den Index der Auflistung an.
itemFormat	ARRAYFORMAT	1	Diese Eigenschaft gibt das Zahlenfolgenformat der Elemente an.
…	STRING	1…1073741823	Alle weiteren Eigenschaften geben jeweils ein Element der Auflistung an. Name und Wert der Eigenschaften geben dabei Index und Zahlenfolge eines Elements an. Die Namen sind lückenlos aufsteigend durchnummeriert und der erste in einem «[IAM_LISTING]» genannte Elementindex ist gleich der bisher angegebenen Elementanzahl der entsprechenden Auflistung.

3.7 XML-Datenformat

Die XML-Datei beginnt mit dem Wurzelelement «**index**» mit dem Datentyp «**IAM_INDEX**», welches das Inhaltsverzeichnis sowie die Auflistungen und Abbildungen enthält.

3.7.1 XML-Datentyp «IAM_INDEX»

Dieser XML-Datentyp kodiert einen «**IAMIndex**» mit einer bestimmten Anzahl von Abbildungen und Auflistungen sowie der Bytereihenfolge zur Kodierung des Binärformats.

Attribut	Format	Anzahl	Beschreibung
byteOrder	BYTEORDER	0…1	Dieses Attribut gibt die Bytereihenfolge an.
mappingCount	INTEGER	1	Dieses Attribut gibt die Anzahl der Abbildungen an.
listingCount	INTEGER	1	Dieses Attribut gibt die Anzahl der Auflistungen an.

Element	Format	Anzahl	Beschreibung
mapping	IAM_MAPPING	0…1073741823	Diese Elemente geben in beliebiger Reihenfolge die Einträge und Elemente von Abbildungen bzw. Auflistungen an.
listing	IAM_LISTING	0…1073741823	

3.7.2 XML-Datentyp «IAM_MAPPING»

Dieser XML-Datentyp kodiert Einträge eines «**IAMMapping**» und kann für eine Abbildung sogar mehrfach vorkommen.

Attribut	Format	Anzahl	Beschreibung
index	INTEGER	1	Diese Eigenschaft gibt den Index der Abbildung an.
findMode	FINDMODE	0…1	Dieses Attribut gibt den Schlüsselsuchmodus an.
keyFormat	ARRAYFORMAT	0…1	Dieses Attribut gibt das Zahlenfolgenformat der Schlüssel an.
valueFormat	ARRAYFORMAT	0…1	Dieses Attribut gibt das Zahlenfolgenformat der Werte an.

Element	Format	Anzahl	Beschreibung
entry	IAM_ENTRY	1…1073741823	Dieses Element gibt einen Eintrag der Abbildung an.

3.7.3 XML-Datentyp «IAM_ENTRY»

Dieser XML-Datentyp kodiert die Daten eines «**IAMEntry**» einer Abbildung.

Attribut	Format	Anzahl	Beschreibung
key	STRING	1	Dieses Attribut gibt die Schlüsselzahlenfolge des Eintrags an. Das Format zur Angabe dieser Zahlenfolge ist im Attribut «keyFormat» des übergeordneten «mapping»-Elements angegeben.
value	STRING	1	Dieses Attribut gibt die Wertzahlenfolge des Eintrags an. Das Format zur Angabe dieser Zahlenfolge ist im Attribut «valueFormat» des übergeordneten «mapping»-Elements angegeben.

3.7.4 XML-Datentyp «IAM_LISTING»

Dieser XML-Datentyp beschreibt Elemente eines «**IAMListing**» und kann für eine Auflistung sogar mehrfach vorkommen.

Attribut	Format	Anzahl	Beschreibung
index	INTEGER	1	Diese Eigenschaft gibt den Index der Auflistung an.
itemFormat	ARRAYFORMAT	0…1	Diese Eigenschaft gibt das Zahlenfolgenformat der Elemente an.

Element	Format	Anzahl	Beschreibung
item	IAM_ITEM	1…1073741823	Dieses Element gibt ein Element der Auflistung an.

3.7.5 XML-Datentyp «IAM_ITEM»

Dieser XML-Datentyp kodiert die Daten eines «**IAMArray**» einer Auflistung.

Attribut	Format	Anzahl	Beschreibung
data	STRING	1	Dieses Attribut gibt die Zahlenfolge des Elements an. Das Format zur Angabe dieser Zahlenfolge ist im Attribut «itemFormat» des übergeordneten «listing»-Elements angegeben.

3.8 JSON-Datenformat

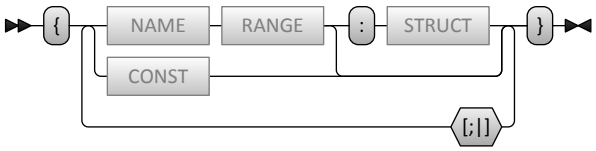
TODO

4 IAM-Notation

Die «IAM»-Notation erlaubt die Definition von Datenstrukturen zur Interpretation bzw. Beschreibung des Aufbaus von Zahlenfolgen, Auflistungen und Abbildungen eines «**IAMIndex**». In den Definitionen werden mehrere alternative Varianten «|»-separiert und mehrere serielle bzw. gemeinsame Komponenten «;»-separiert angegeben.

ARRAY ::= '{' (CONST|NAME RANGE (':' STRUCT)?) ([;|] (CONST|NAME RANGE (':' STRUCT)?))* '}'

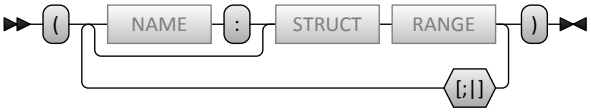
Diese EBNF-Regel definiert, wie die Struktur einer Zahlenfolge angegeben wird. Ein Datenfeld («NAME») ohne explizite Typisierung («STRUCT») steht implizit für einen Zahlenwert.



Beispiel «{4; foo[2]; bar: @TypeA; opt[0..1]: {x; y}}».

LISTING ::= '(' (NAME ':')? STRUCT RANGE ([;|] (NAME ':')? STRUCT RANGE)* ')'

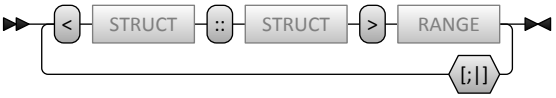
Diese EBNF-Regel definiert, wie die Struktur einer Auflistung angegeben wird. Ein Abschnitt der Auflistung wird immer Typisiert («STRUCT RANGE») und kann benannt («NAME») werden.



Beispiele: «(TypeA; TypeB[0..1]; refs: @TypeA[n])», «(TypeA[n] | TypeB[n])».

MAPPING ::= '<' STRUCT '::' STRUCT '>' RANGE ([;|] '<' STRUCT '--' STRUCT '>' RANGE)*

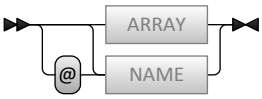
Diese EBNF-Regel definiert, wie die Struktur einer Abbildung angegeben wird.



Beispiel: «<{1}::@TypeA>; <{2}::@TypeB>», «<TypeA::@TypeB>[0..A]».

STRUCT ::= ARRAY|'@'? NAME

Diese EBNF-Regel definiert, wie die typisierende Datenstruktur einer Zahlenfolge angegeben wird. Wenn ein Datenstrukturnamen mit «@» beginnt, beschreibt dies eine Zahlenfolge mit einem Element, welches auf die Position einer Zahlenfolge, Auflistung oder Abbildung mit der jeweiligen Datenstruktur verweist.



Beispiele: «{4711}», «TypeB», «@TypeA».

NAME ::= [http://www.w3.org/TR/xml-names/#NT-NCName]

Diese EBNF-Regel definiert, wie ein Namen zum Benennen oder Referenzieren von Datenstrukturen oder Datenfeldern angegeben wird.

Beispiele: «ItemCount», «source-ref», «MAX».

COUNT ::= [0-9]+

Diese EBNF-Regel definiert, wie eine natürliche Zahl angegeben wird, z.B. als Anzahl.

Beispiele: «0», «4711».

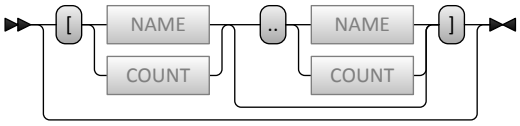
CONST ::= '-'? COUNT

Diese EBNF-Regel definiert, wie eine ganze Zahl angegeben wird, z.B. als Konstante.

Beispiele: «1», «-42».

`RANGE ::= ('[' (NAME|COUNT) ('..' (NAME|COUNT))? ']')?`

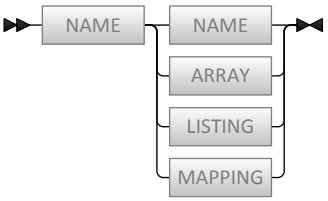
Diese EBNF-Regel definiert, wie eine Multiplizität als Bereich mit minimaler und optionaler maximaler Anzahl angegeben wird. Wenn keine Multiplizität angegeben ist, sind minimale und maximale Anzahl implizit «1». Wenn nur die minimale Anzahl angegeben ist, ist die maximale gleich der minimalen. Eine Anzahl kann als Zahlenkonstante oder Variablenname angegeben werden.



Beispiele: «», «[42]», «[COUNT]», «[min..max]».

`NAMING ::= NAME (NAME|ARRAY|LISTING|MAPPING)`

Diese EBNF-Regel definiert, wie der Name zu einer konkreten bzw. referenzierten Datenstruktur angegeben werden kann.



Beispiel: «TypeB TypeA».