1 IAM – Integer Array Model

«IAM – Integer Array Model» beschreibt einerseits ein abstraktes Datenmodell aus Listen und Abbildungen sowie andererseits ein binäres und optimiertes Datenformat zur Auslagerung dieser Listen und Abbildungen in eine Datei. Ziel des Datenformats ist es, entsprechende Dateien per *file-mapping* in den Arbeitsspeicher abzubilden und darauf sehr effiziente Lese- und Suchoperationen ausführen zu können. Die Modifikation der Daten ist nicht vorgesehen.

1.1 Datenmodell

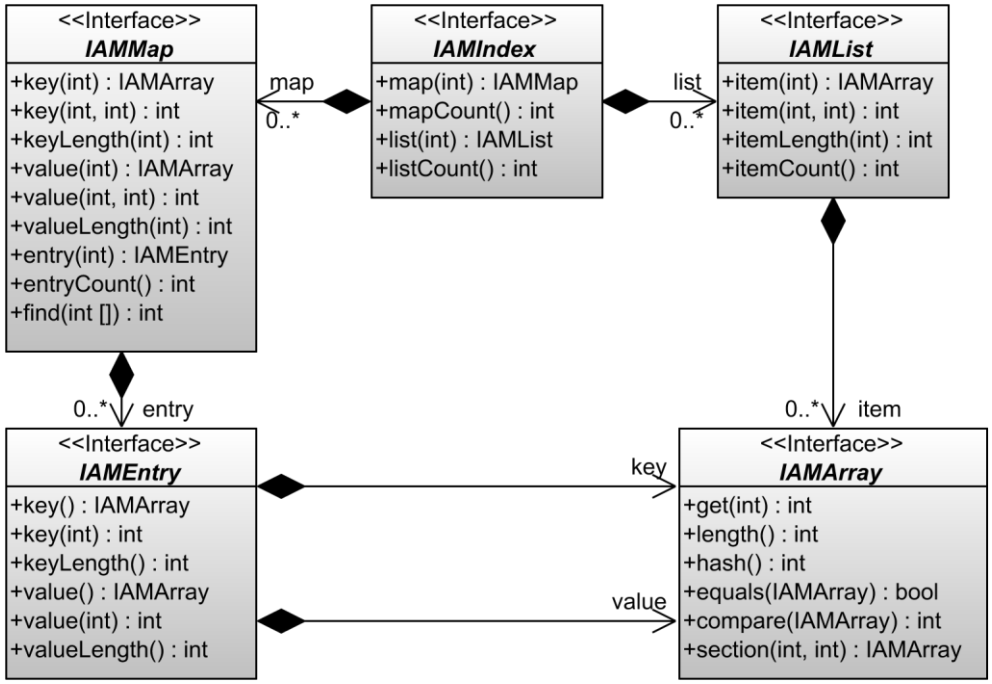

Abbildung 1 Klassendiagramm «IAM – Integer Array Model»

Die Schnittstelle «`IAMIndex`» bildet den Ausgangspunkt des Datenmodells. Über diese Schnittstelle kann auf die Listen («`IAMList`») und Abbildungen («`IAMMap`») zugegriffen werden. Die Elemente der Listen sind ebenso Zahlenfolgen («`IAMArray`»), wie die die Schlüssel und Werte der Einträge («`IAMEntry`») der Abbildungen.

IAMIndex	
Ein «`IAMIndex`» ist eine abstrakte Zusammenstellung beliebig vieler Listen («`IAMList`») und Abbildungen («`IAMMap`»).	
Methode	Beschreibung
`map(index)`	Diese Methode gibt die «index»-te Abbildung zurück. Bei einem ungültigen «index» wird eine leere Abbildung geliefert.
`mapCount()`	Diese Methode gibt die Anzahl der Abbildungen zurück («`0…1073741823`»).
`list(index)`	Diese Methode gibt die «index»-te Liste zurück. Bei einem ungültigen «index» wird eine leere Liste geliefert.
`listCount()`	Diese Methode gibt die Anzahl der Listen zurück («`0…1073741823`»).

Tabelle 1-1 Schnittstelle «IAMIndex»

IAMList	
Eine «`IAMList`» ist eine abstrakte, geordnete Liste von Elementen, welche selbst Zahlenfolgen («`IAMArray`») sind.	
Methode	Beschreibung
`item(itemIndex)`	Diese Methode gibt das «itemIndex»-te Element als Zahlenfolge zurück. Bei einem ungültigen «itemIndex» wird eine leere Zahlenfolge geliefert.
`item(itemIndex, index)`	Diese Methode gibt die «index»-te Zahl des «itemIndex»-ten Elements zurück. Bei einem ungültigen «index» oder «itemIndex» wird «`0`» geliefert.
`itemLength(itemIndex)`	Diese Methode gibt die Länge der Zahlenfolge des «itemIndex»-ten Elements zurück. Bei einem ungültigen «itemIndex» wird «0» geliefert.
`itemCount()`	Diese Methode gibt die Anzahl der Elemente zurück («`0…1073741823`»).

Tabelle 1-2 Schnittstelle «IAMList»

IAMArray	
Ein «`IAMArray`» ist eine abstrakte Zahlenfolge, welche in einer Liste («`IAMList`») für die Elemente sowie einer Abbildung («`IAMMap`») für die Schlüssel und Werte der Einträge («`IAMEntry`») verwendet wird.	
Methode	Beschreibung
`get(index)`	Diese Methode gibt die «index»-te Zahl zurück. Bei einem ungültigen «index» wird «`0`» geliefert.
`length()`	Diese Methode gibt die Länge der Zahlenfolge zurück («`0…1073741823`»).

`hash()`	Diese Methode gibt den Streuwert zurück. Der Algorithmus hierfür ist:
	`int result = 0x811C9DC5;`
	`for (int i = 0; i < length(); i++)`
	`  result = (result * 0x01000193) ^ get(i);`
	`return result;`

`equals(value)`	Diese Methode gibt nur dann «true» zurück, wenn diese Zahlenfolge gleich der gegebenen Zahlenfolge ist. Der Algorithmus hierfür ist:
	`if (length() != value.length()) return false;`
	`for (int i = 0; i < length(); i++)`
	`  if (get(i) != value.get(i)) return false;`
	`return true;`

`compare(value)`	Diese Methode gibt eine Zahl kleiner, gleich oder größer als «`0`» zurück, wenn die Ordnung dieser Zahlenfolge lexikografisch kleiner, gleich bzw. größer als die der gegebenen Zahlenfolge ist. Der Algorithmus hierfür ist:
	`for (int i = 0, result; i < min(length(), value.length()); i++)`
	`  if (get(i) < value.get(i)) return -1; else`
	`  if (get(i) > value.get(i)) != 0) return +1;`
	`return length() - value.length();`

`section(offset, length)`	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.

Tabelle 1-3 Schnittstelle «IAMArray»

IAMMap	
Eine «`IAMMap`» ist eine abstrakte Abbildung von Schlüsseln auf Werte, welche beide selbst Zahlenfolgen («`IAMArray`») sind.	
Methode	Beschreibung
`key(entryIndex)`	Diese Methode gibt den Schlüssel des «entryIndex»-ten Eintrags als Zahlenfolge zurück. Bei einem ungültigen «entryIndex» wird eine leere Zahlenfolge geliefert.
`key(entryIndex, index)`	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)`	Diese Methode gibt die Länge der Zahlenfolge des Schlüssels des «entryIndex»-ten Eintrags zurück («`0…1073741823`»). Bei einem ungültigen «entryIndex» wird «0» geliefert.
`value(entryIndex)`	Diese Methode gibt den Wert des «entryIndex»-ten Eintrags als Zahlenfolge zurück. Bei einem ungültigen «entryIndex» wird eine leere Zahlenfolge geliefert.
`value(entryIndex, index)`	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)`	Diese Methode gibt die Länge der Zahlenfolge des Werts des «entryIndex»-ten Eintrags zurück («`0…1073741823`»). Bei einem ungültigen «entryIndex» wird «`0`» geliefert.
`entry(entryIndex)`	Diese Methode gibt den «entryIndex»-ten Eintrag zurück. Bei einem ungültigen «entryIndex» wird ein leerer Eintrag geliefert.
`entryCount()`	Diese Methode gibt die Anzahl der Einträge zurück («`0…1073741823`»).
`find(key)`	Diese Methode gibt den Index des Eintrags zurück, dessen Schlüssel äquivalenten zum gegebenen Schlüssel ist. Bei erfolgloser Suche wird «`-1`» geliefert.

Tabelle 1-4 Schnittstelle «IAMMap»

IAMEntry	
Ein «`IAMEntry`» ist ein abstrakter Eintrag einer Abbildung («`IAMMap`») und besteht aus einem Schlüssel und einem Wert, welche selbst Zahlenfolgen («`IAMArray`») sind.	
Methode	Beschreibung
`key()`	Diese Methode gibt den Schlüssel als Zahlenfolge zurück.
`key(index)`	Diese Methode gibt die «index»-te Zahl des Schlüssels zurück. Bei einem ungültigen «index» wird «`0`» geliefert.
`keyLength()`	Diese Methode gibt die Länge der Zahlenfolge des Schlüssels zurück («`0…1073741823`»).
`value()`	Diese Methode gibt den Wert als Zahlenfolge zurück.
`value(index)`	Diese Methode gibt die «index»-te Zahl des Werts zurück. Bei einem ungültigen «index» wird «`0`» geliefert.
`valueLength()`	Diese Methode gibt die Länge der Zahlenfolge des Werts zurück («`0…1073741823`»).

Tabelle 1-5 Schnittstelle «IAMEntry»

1.2 Datenformat

Die primitiven Datenformate «INT8», «INT16» und «INT32» stehen für eins, zwei bzw. vier Byte große, vorzeichenbehaftete, ganze Zahlen. Die vorzeichenlosen Varianten hiervon sind «UINT8», «UINT16» und «UINT32». Die Bytereihenfolge in den zwei und vier Byte großen Formate entspricht der nativen Bytereihenfolge der Zielplattform, in welcher die Daten per file-mapping angebunden werden.

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 Datenstrukturvarianten in spitzen Klammern gekennzeichnet.

IAM_INDEX

Die Datenstruktur «IAM_INDEX» kodiert einen «IAMIndex». Auf die kodierten Daten der Abbildungen («IAM_MAP») und Listen («IAM_LIST») kann wahlfrei zugegriffen werden.

Feld	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.
mapCount	UINT32	1	Dieses Feld speichert die Anzahl der Abbildungen («0…1073741823»).
listCount	UINT32	1	Dieses Feld speichert die Anzahl der Listen («0…1073741823»).
mapOffset	UINT32	mapCount+1	Dieses Feld speichert die Startpositionen der Abbildungen im Speicherbereich «mapData». Die «i»-te Abbildung beginnt und endet dort an den Positionen «mapOffset[i]» bzw. «mapOffset[i+1]» und hat damit eine Länge von «mapOffset[i+1]−mapOffset[i]». Die Startposition «mapOffset[0]» ist «0». Die minimale Länge einer Abbildung ist «4».
listOffset	UINT32	listCount+1	Dieses Feld speichert die Startpositionen der Listen im Speicherbereich «listData». Die «i»-te Liste beginnt und endet dort an den Positionen «listOffset[i]» bzw. «listOffset[i+1]» und hat damit eine Länge von «listOffset[i+1]−listOffset[i]». Die Startposition «listOffset[0]» ist «0». Die minimale Länge einer Liste ist «3».
mapData	UINT32	mapOffset [mapCount]	Dieses Feld speichert die kodierten Abbildungen («IAM_MAP»).
listData	UINT32	listOffset [listCount]	Dieses Feld speichert die kodierten Listen («IAM_LIST»).

Tabelle 1-6 Datenstruktur «IAM_INDEX»

IAM_LIST

Diese Datenstruktur kodiert eine «IAMListe». Hierbei werden die folgenden, technischen Aspekte unterscheiden: Die Zahlen in den Elementen können als «INT8» («ID=1»), «INT16» («ID=2») oder «INT32» («ID=3») kodiert werden. Die Elemente können eine homogene Länge («IL=0») oder heterogene Längen besitzen. Die heterogenen Längen können als «UINT8» («IL=1»), «UINT16» («IL=2») oder «UINT32» («IL=3») kodiert werden.

Feld	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 sowie die 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 und endet dort an den Positionen «itemOffset[i]» bzw. «itemOffset[i+1]» und hat damit 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.

Tabelle 1-7 Datenstruktur «IAM_LIST»

IAM_MAP

Diese Datenstruktur kodiert eine «IAMMap». Hierbei werden die folgenden, technischen Aspekte unterscheiden: Die Zahlen in den Schlüsseln bzw. Werten können als «INT8» («KD=1», «VD=1»), «INT16» («KD=2», «VD=2») oder «INT32» («KD=3», «VD=3») kodiert werden. Die Schlüssel bzw. Werte können eine homogene Länge («KL=0», «VL=0») oder heterogene Längen besitzen. Die heterogenen Längen können als «UINT8» («KL=1», «VL=1»), «UINT16» («KL=2», «VL=2») oder «UINT32» («KL=3», «VL=3») kodiert werden. Die Ermittlung eines Schlüssel kann über eine binäre Suche («RL=0») oder eine Streuwerttabelle erfolgen. Die Indizes in der Streuwerttabelle können als «UINT8» («RL=1»), «UINT16» («RL=2») oder «UINT32» («RL=3») kodiert werden.

Feld	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 sowie 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 (vgl. «IAMArray.hash()») in den Index des einzigen Schlüsselbereichs, in dem der gesuchte 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 und endet mit den Indizes «rangeData[i]» bzw. «rangeData[i+1]» und enthält damit «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 und endet dort an den Positionen «keyOffset[i]» bzw. «keyOffset[i+1]» und hat damit 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>	keySize* entryCount	Dieses Feld speichert die Zahlen der Schlüssel.
	INT16<KD=2>		
	INT32<KD=3>		
keyData<KD=1…3>	INT8<KD=1>	keyOffset [entryCount]	
	INT16<KD=2>		
	INT32<KD=3>		
–<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>	entryCount+1	Dieses Feld speichert die Startpositionen der Werte im Speicherbereich «valueData». Der «i»-te Wert beginnt und endet dort an den Positionen «valueOffset[i]» bzw. «valueOffset[i+1]» und hat damit eine Läge von «valueOffset[i+1]–valueOffset[i]». Die Startposition «valueOffset[0]» ist «0».
	UINT16<VL=2>		
	UINT32<VL=3>		
–<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>	valueSize* entryCount	Dieses Feld speichert die Zahlen der Werte.
	INT16<VD=2>		
	INT32<VD=3>		
valueData<VD=1…3>	INT8<VD=1>	valueOffset [entryCount]	
	INT16<VD=2>		
	INT32<VD=3>		
–<VD1…2>	UINT8	0…3	Dieses Feld ergänzt den Speicherbereich «valueData» auf eine restlos mit «UINT32» füllbare Größe.

Tabelle 1-8 Datenstruktur «IAM_MAP»

2 Datenaustauschformate

Datenaustauschformate für «IAM» sind menschenlesbaren Textdateien, welche durch manuelle Bearbeitung angepasst und mit Hilfe eines Transformationswerkzeugs in das optimierte, binäre Datenformat «IAM_INDEX» überführt werden können.

2.1 INI-Datenformat

Das *ini-file* beginnt mit dem Abschnitt für das Inhaltsverzeichnis («[IAM_INDEX]»), welcher von den Abschnitten für die Abbildungen («[IAM_MAP/#]») und Listen («[IAM_LIST/#]») gefolgt wird.

[IAM_INDEX]	
Dieser Abschnitt beschreibt eine Zusammenstellung («IAMIndex») von Abbildungen und Listen mit einer bestimmten Abbildungsanzahl («mapCount»), Listenanzahl («listCount») und Bytereihenfolge («byteOrder»). Zulässige Bytereihenfolgen sind «N» für *native-order*, «B» für *big-endian* und «L» für *little-endian*. Als Rückfallwerte gelten «mapCount=0», «listCount=0» und «byteOrder=N».	
Beispiel	Beschreibung
[IAM_INDEX] mapCount=3 listCount=2 byteOrder=N	Die Zusammenstellung besteht aus drei Abbildungen («IAMMap») und zwei Listen («IAMList») in nativer Bytereihenfolge.
[IAM_INDEX] listCount=10 byteOrder=B	Die Zusammenstellung besteht aus zehn Listen («IAMList») in *big-endian* Bytereihenfolge.

Tabelle 2-1 Abschnitt «[IAM_INDEX]»

[IAM_LIST/#]	
Dieser Abschnitt beschreibt eine Liste («IAMList») mit einem bestimmten Index «#». Der Index ist dazu am Ende des Abschnittsnamens nach dem Schrägstrich «/» angegeben. Jedes Element («IAMArray») der Liste wird als Eigenschaft angegeben, deren Name dem Index des Elements entspricht. Die Elemente sind damit bei «0» beginnend lückenlos aufsteigend durchnummeriert. Die Zahlen in den Zahlenfolgen der Elemente werden mit Schrägstrich «/» separiert.	
Beispiel	Beschreibung
[IAM_LIST/0] 0=1/2/3 1=4/5/6 2=0/0/0	Die erste Liste der Zusammenstellung enthält drei homogene Elemente. Das erste Element ist eine Zahlenfolge aus «1», «2» und «3», das zweite ist eine aus «4», «5» und «6», usw.
[IAM_LIST/1] 0= 1=1/2 2=3	Die zweite Liste der Zusammenstellung enthält drei heterogene Elemente. Das erste Element ist eine leere Zahlenfolge, das zweite ist eine aus «1» und «2», usw.

Tabelle 2-2 Abschnitt «[IAM_LIST/#]»

[IAM_MAP/#]	
Dieser Abschnitt beschreibt eine Abbildung («`IAMMap`») mit einem bestimmten Index «`#`». Der Index ist dazu am Ende des Abschnittsnamens nach dem Schrägstrich «`/`» angegeben. Die Suche von Einträgen («`IAMEntry`») über einen gegebenen Schlüssel erfolge gemäß der Eigenschaft «`findMode`» über eine binäre Suche («`S`») oder eine streuwertbasierte Suche («`H`»). Jeder Eintrag der Abbildung wird als Eigenschaft angegeben, deren Name dem Schlüssel des Eintrags entspricht. Die Zahlen in den Zahlenfolgen («`IAMArray`») der Schlüssel und Werte werden mit Schrägstrich «`/`» separiert. Als Rückfallwert gilt «`findMode=H`».	
Beispiel	Beschreibung

`[IAM_MAP/0]`
`findMode=S`
`0/1=1`
`0/2=2`
`1/2=3`
	Die erste Abbildung der Zusammenstellung enthält drei homogene Einträge, auf denen binär gesucht wird. Der erste Eintrag bildet von einer Zahlenfolge aus «`0`» und «`1`» auf eine aus «`3`» ab, usw.

`[IAM_LIST/1]`
`0/1=`
`=1/2`
`0/1/2=3/4`
	Die zweite Abbildung der Zusammenstellung enthält drei heterogene Einträge, auf denen streuwertbasierte gesucht wird. Der erste Eintrag bildet von einer Zahlenfolge aus «`0`» und «`1`» auf eine leere ab, der zweite bildet von einer leeren auf eine aus «`0`» und «`1`» ab, usw.

Tabelle 2-3 Abschnitt «[IAM_MAP/#]»

2.2 XML-Datenformat

TODO