Anleitung zu IOX

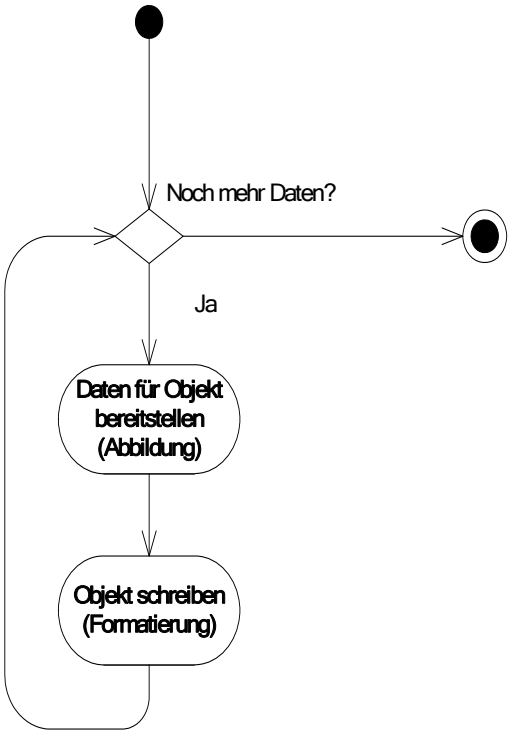
Diese Anleitung beschreibt die Benutzung von IOX.

Konzept

Das Hauptanliegen von IOX ist, eine Applikation vom Transferformat zu entkoppeln, so dass das Transferformat geändert werden kann, ohne dass dies Auswirkungen auf den Code der Applikation hat.
Im Allgemeinen funktioniert eine Export-Schnittstelle nach folgendem Muster:

- Eine interne Datenstruktur wird durchlaufen
- Für ein externes Datenelement werden die Daten in der internen Datenstruktur zusammen gesucht (Abbildung der internen Datenstruktur auf die externe Datenstruktur)
- Das externe Datenelement wird geschrieben (Formatierung)

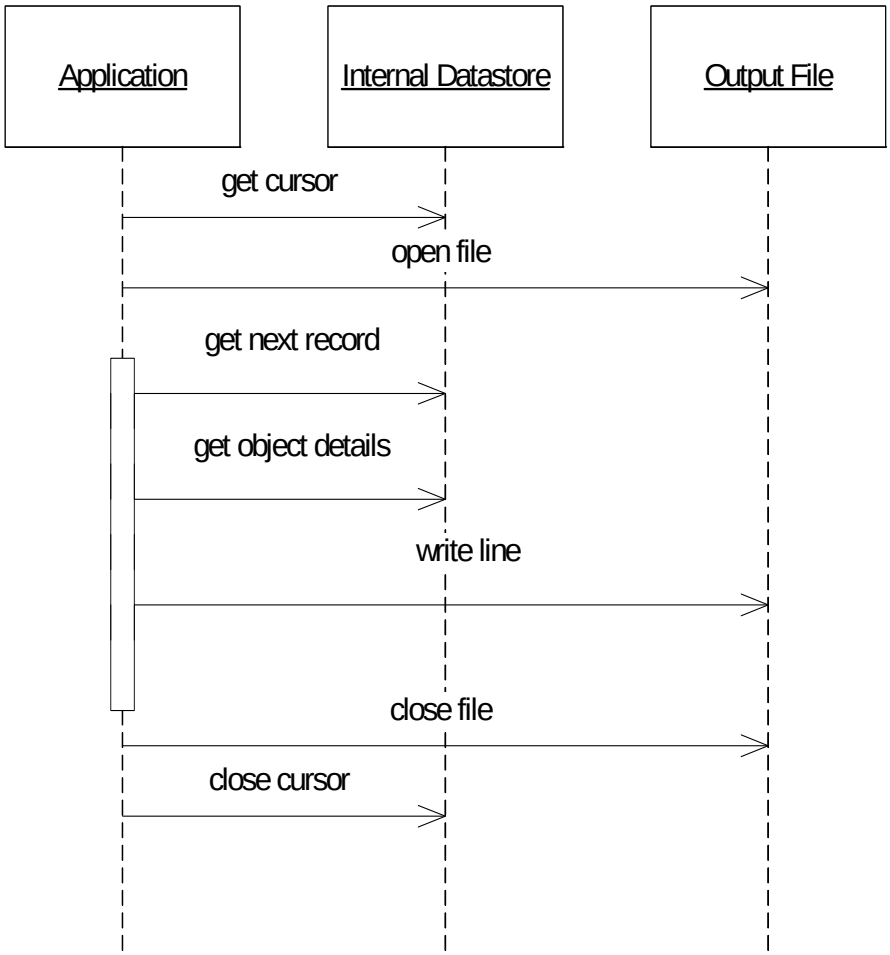
Eine Import-Schnittstelle funktioniert in der selben Art. Das Format muss decodiert werden, und die externen Datenelemente müssen auf die Internen abgebildet werden.



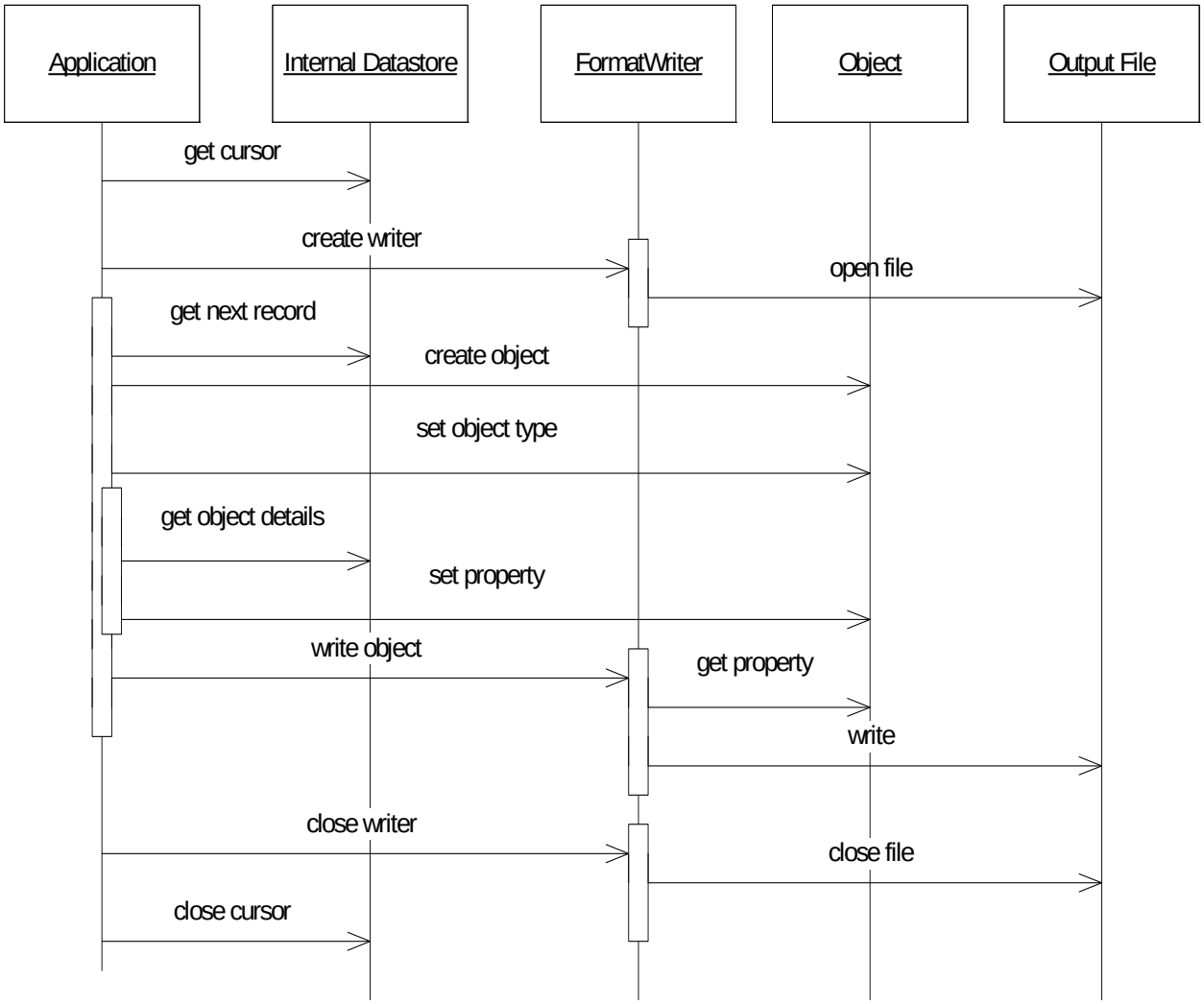
Es ist zu beachten, dass der Export aus zwei Schritten besteht:

- die Abbildung der internen auf die externe Datenstruktur
- die Formatierung

Typischerweise wird das wie folgt realisiert:



Eine solche Struktur hat den Nachteil, dass die Formatierung in der Applikation realisiert ist (Zeile „write line“). Wenn sich die Formatierung ändert (z.B. von CSV nach XML) muss die Applikation geändert werden.
Besser ist darum die folgende Struktur:



Hierbei erzeugt die Applikation formatneutrale Transferelemente (Object) und übergibt diese an ein Modul FormatWriter. Das Modul FormatWriter schreibt dann diese formatneutralen Transferelemente in die konkrete Datei (entsprechend dem jeweiligen Transferformat). Soll die Applikation nun ein anderes Transferformat schreiben, genügt es das Modul FormatWriter auzutauschen.
IOX definiert ein API für FormatWriter und Transferelemente. Die wichtigsten Typen sind:

ch.interlis.iom.IomObject	repräsentiert ein formatneutrales Transferelement
ch.interlis.iox.IoxWriter	Formatneutrale Schnittstelle um Daten zu Schreiben
ch.interlis.iox.IoxReader	Formatneutrale Schnittstelle um Daten zu Lesen
ch.interlis.iox.IoxEvent	Basis für Dinge die aus einem IoxReader „gelesen“ oder in einen IoxWriter „geschrieben“ werden können (z.B. kapselt die Spezialisierung ObjectEvent ein IomObject). Repräsentiert ein Element im Kontrollfluss zwischen der Applikation und einem IoxReader/IoxWriter.

IOX entkoppelt die Applikation vom Format. IOX hat aber nicht die Abbildung der internen auf die externe Datenstruktur zum Ziel.

Daten Lesen

Um Daten zu Lesen, muss ein IoxReader erzeugt werden, und dann werden einzelne IoxEvent’s gelesen.

Wie ein IoxReader erzeugt wird, ist abhängig vom jeweiligen Transferformat. Je nach Implementierung sind auch zusätzlich Konfigurationsschritte notwendig. Beachten Sie dazu die formatspezifische Dokumentation.

```
IoxReader reader=...;
IoxEvent event;
do{
    event=reader.read();
    if(event instanceof StartTransferEvent){
        ...
    }else if(event instanceof StartBasketEvent){
        ...
    }else if(event instanceof ObjectEvent){
        ...
    }else if(event instanceof EndBasketEvent){
        ...
    }else if(event instanceof EndTransferEvent){
        ...
    }
}while(!(event instanceof EndTransferEvent));
```

Das wichtigste Steuerflusselement (IoxEvent) ist ObjectEvent. Damit erhält man Zugriff auf ein Transferelement. Um z.B. eine Liste aller Objekte auszugeben, würde man folgenden Code benutzen:

```
...
}else if(event instanceof ObjectEvent){
  IomObject iomObj=((ObjectEvent)event).getIomObject();
  String aclass=iomObj.getobjecttag();
  String oid=iomObj.getobjectoid();
  out.println("Object "+aclass+" "+oid);
}...
```

Daten Schreiben

Um Daten zu Schreiben, muss ein IoxWriter erzeugt werden, und dann werden einzelne IoxEvent's geschrieben.
Wie ein IoxWriter erzeugt wird, ist abhängig vom jeweiligen Transferformat. Je nach Implementierung sind auch zusätzlich Konfigurationsschritte notwendig. Beachten Sie dazu die formatspezifische Dokumentation.

```
IoxWriter writer=...;
writer.write(StartTransferEvent);
...
while(more baskets){
    writer.write(StartBasketEvent);
    ...
    while(more objects){
        ...
        writer.write(ObjectEvent);
        ...
    }
    ...
    writer.write(EndBasketEvent);
}
writer.write(EndTransferEvent);
```

Das formatneutrale Transferelement

Um Daten formatneutral repräsentieren, verwendet IOX das Interface IomObject. Dabei gilt:

- Jedes Objekt entspricht einem Typ
- Jedes Objekt kann eine Identität haben (für Referenzen)
- Jedes Objekt hat eine Reihe von Eigenschaften
- Eine Eigenschaft hat keinen, einen oder mehrere Wert(e)
- Ein Wert ist primitiv (String) oder strukturiert (IomObject; IomObject ist also eine rekursive Struktur)

Die wichtigsten Funktionen sind:

String getobjecttag()	Typ des Objektes
String getobjectoid()	Identität des Objektes
int getattrcount()	Anzahl Attribute/Eigenschaften
int getattrvaluecount(String attrname)	Anzahl Werte zu einem Attribut
String getattrvalue(String attrname)	Wert des Attributs (falls primitv)
IomObject getattrobj(String attrname,int idx)	Wert des Attributs (falls strukturiert)

Um festzustellen, ob ein Attribut primitiv oder strukturiert ist, soll folgender Code verwendet werden:

```
if(obj.getattrvalue(„attrname“)==null){
     IomObject struct=obj.getattrobj(„attrname“,0);
}
```

Details

Steuerfluss

Zwischen Applikation und IoxReader bzw. IoxWriter werden nicht direkt IomObject's ausgetauscht, sondern IoxEvent's. Damit soll eine Trennung der Steuerung von den eigentlichen Nutz-Daten erreicht werden.
IOX definiert die folgenden IoxEvent's:

StartTransferEvent	kennzeichnet den Anfang einer Transferdatei
StartBasketEvent	kennzeichnet den Anfang eines Behälters
ObjectEvent	kennzeichnet ein Datenelement
EndBasketEvent	kennzeichnet das Ende eines Behälters
EndTransferEvent	kennzeichnet das Ende einer Transferdatei

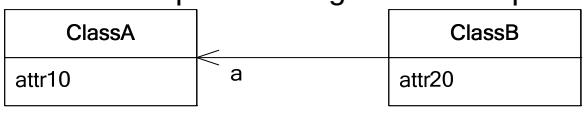
Die Abfolge der IoxEvent's ist durch IOX wie folgt genau definiert (in EBNF Syntax). Jeder IoxReader/IoxWriter muss diese Reihenfolge einhalten bzw. unterstützen:

```
StartTransferEvent
    {StartBasketEvent
        {ObjectEvent}
    EndBasketEvent}
EndTransferEvent
```

Was ein Datenelement oder ein Behälter ist, ist abhängig vom jeweiligen Transferformat. Beachten Sie dazu die formatspezifische Dokumentation.

Referenzen

Referenzen werden mit IOX als Teil eines strukturierten Werts repräsentiert.
Ist zum Beispiel das folgende konzeptionelle Datenmodell gegeben:



In INTERLIS 1 würde das entsprechende Modell wie folgt aussehen:

```
TABLE ClassA =
  attr10 : TEXT*20;
```

```
NO IDENT
END ClassA;

TABLE ClassB =
  attr20 : TEXT*20;
  a : -> ClassA;
NO IDENT
END ClassB;
```

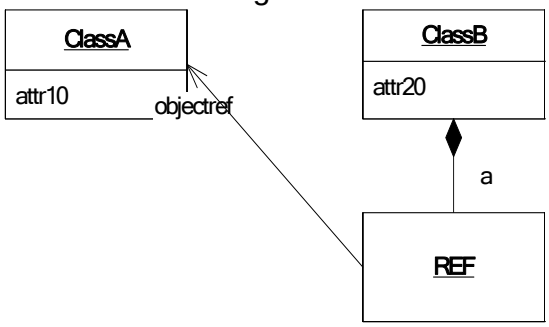
In INTERLIS 2 würde das entsprechende Modell wie folgt aussehen:

```
CLASS ClassA =
  attr10 : TEXT*20;
END ClassA;

CLASS ClassB =
  attr20 : TEXT*20;
END ClassB;

ASSOCIATION =
  a -- {1} ClassA;
  b -- {0..*} ClassB;
END;
```

Verweist nun ein Objekt der Klasse ClassB auf ein Objekt der Klasse ClassA, ergibt dies in IOX die folgende Struktur von Instanzen.



Um auf die Referenz zuzugreifen, ist somit der folgende Code erforderlich:

```
String objectref=null;
IomObject refStruct=classBobj.getattrobj("a",0);
if(refStruct!=null){
  objectref=refStruct.getobjectrefoid();
}
```

Bei INTERLIS 2 Modellen entspricht die Abbildung auf Strukturen den Kodierungsregeln einer Assoziation. Im INTERLIS 2-Referenzhandbuch steht dazu:

> Beziehungen werden auf zwei Arten codiert: eingebettet oder nicht eingebettet. Eine eingebettete Beziehung wird als Sub-Element von einer, an der Assoziation beteiligten, Klasse codiert. Die Instanz einer nicht eingebetteten Beziehung (Link) wird wie eine Instanz einer Klasse codiert.
>
> Beziehungen werden immer eingebettet, ausser
>
> - wenn sie mehr als zwei Rollen haben oder
> - wenn bei beiden (Basis-)Rollen die maximale Kardinalität grösser 1 ist oder
> - wenn für die Beziehung eine OID gefordert wird oder
> - bei gewissen themenübergreifenden Beziehungen (s. unten).
>
> Falls bei einer der beiden (Basis-)Rollen die maximale Kardinalität grösser 1 ist, wird bei der Ziel-Klasse dieser Rolle eingebettet. Wenn diese Ziel-Klasse in einem anderen Topic definiert ist als die (Basis-)Assoziation, kann nicht eingebettet werden.

Falls bei beiden (Basis-)Rollen die maximale Kardinalität kleiner gleich 1 ist, wird bei der Ziel-Klasse der zweiten Rolle eingebettet. Wenn diese Ziel-Klasse in einem anderen Topic definiert ist als die (Basis-)Assoziation und die Ziel-Klasse der ersten Rolle im selben Topic definiert ist wie die (Basis-)Assoziation, wird bei der Ziel-Klasse der ersten Rolle eingebettet (d.h., wenn die Ziel-Klassen der beiden Rollen in einem anderen Topic definiert sind als die (Basis-)Assoziation, kann nicht eingebettet werden).

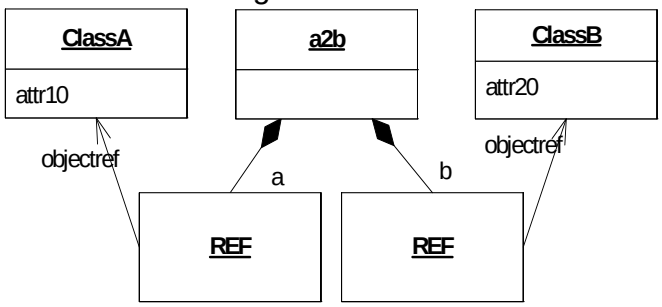
Beim vorgängigen Beispiel handelt es sich somit um eine eingebettete Beziehung. Für eine nicht eingebettet Beziehung würde das Modell wie folgt aussehen:

```
CLASS ClassA =
  attr10 : TEXT*20;
END ClassA;

CLASS ClassB =
  attr20 : TEXT*20;
END ClassB;

ASSOCIATION a2b =
  a -- {0..*} ClassA;
  b -- {0..*} ClassB;
END a2b;
```

Verweist nun ein Objekt der Klasse ClassB auf ein Objekt der Klasse ClassA, ergibt dies in IOX die folgende Struktur von Instanzen.



Die Referenz vom Objekt ClassB zum Objekt ClassA ist also nicht mehr eine Eigenschaft von ClassB, sondern eine Eigenschaft des eigenständigen Objektes a2b.
Um auf die Referenz zuzugreifen, ist somit der folgende Code erforderlich:

```
String objectref=null;
IomObject refStruct=a2bObj.getattrobj("a",0);
if(refStruct!=null){
  objectref=refStruct.getobjectrefoid();
}
```

Fehler

Programmfehler werden von IoxReader oder IoxWriter durch das Auslösen einer IoxException der Applikation gemeldet.
Wie Datenfehler gemeldet werden, ist IoxReader/IoxWriter spezifisch. Beachten Sie dazu die formatspezifische Dokumentation.

Benötigte JAR-Dateien

iox-api.jar	IOX API (Interfaces)