Anleitung zu IOX-ILI

Diese Anleitung beschreibt die für INTERLIS 1 und INTERLIS 2 spezifische Realisierung der IOX Schnittstellen. Für IOX im Allgemeinen beachten sie bitte die Anleitung zu 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. IOX entkoppelt die Applikation vom Format. IOX hat aber nicht die Abbildung der internen auf die externe Datenstruktur zum Ziel.
Da IOX nur Schnittstellen definiert, sind für ein konkretes Transferformat spezifische Realisierungen dieser Schnittstellen erforderlich. Diese Anleitung beschreibt solche Realisierungen für INTERLIS 1 und INTERLIS 2.
Die wichtigsten Typen von IOX-ILI sind:

ch.interlis.iom_j.itf.ItfWriter	INTERLIS 1 Daten schreiben
ch.interlis.iom_j.itf.ItfReader	INTERLIS 1 Daten lesen
ch.interlis.iom_j.xtf.XtfWriter	INTERLIS 2 Daten schreiben
ch.interlis.iom_j.xtf.XtfReader	INTERLIS 2 Daten lesen

INTERLIS-Modelle lesen

Die INTERLIS Reader und Writer benötigen für die korrekte Funktionsweise das zu den Daten passende INTERLIS-Datenmodell. Um das Datenmodell zu Lesen, wird der INTERLIS-Compiler benötigt.
Die wichtigsten Typen und Funktionen sind:

ch.interlis.ili2c.metamodel.TransferDescription	Wurzel der Datenstruktur, die INTERLIS-Modelle repräsentiert
ch.interlis.ili2c.config.Configuration	aktuelle Dateien und Optionen für einen Compiler-Lauf
ch.interlis.ili2c.Ili2c.runCompiler()	Modelle kompilieren; liefert eine TransferDescription
ch.interlis.ilirepository.IliManager.getConfig()	Funktion um ili-Dateien zu ermitteln und evtl. downloaden (wenn Modell-Namen gegeben sind); liefert eine Configuration
ch.interlis.ilirepository.IliManager.getConfigWithFiles()	Funktion um ili-Dateien zu ermitteln und evtl. downloaden (wenn ili-Dateinamen z.T. gegeben sind); liefert eine Configuration
ch.interlis.ili2c.Main.compileIliFiles()	Funktion um ili-Dateien zu Lesen (wenn Dateinamen gegeben sind); ohne download von Dateien

ch.interlis.ili2c.Main.compileIliModels()	Funktion um ili-Modelle zu Lesen (wenn Modell-Namen gegeben sind); ohne download von Dateien

Modelle lesen mit Verwendung von Modell-Ablagen

Möchte man die INTERLIS-Modelle aus dem Internet beziehen, sind folgende Schritte notwendig:

- Zugriff auf Modell-Ablage initialisieren
- Liste der Wurzel-Ablagen definieren
- Liste der Modelldateien ermitteln (inkl. download der Modell-Dateien)
- Modell-Dateien kompilieren

Man verwendet z.B. den folgenden Code, um das Modell/die Modelle zu lesen:

```
import ch.interlis.ili2c.Ili2c;
import ch.interlis.ili2c.config.Configuration;
import ch.interlis.ili2c.metamodel.TransferDescription;
// create repository manager
IliManager manager=new IliManager();
// set list of repositories to search
String repositories[]=new String[]{"http://models.interlis.ch/"};
manager.setRepositories(repositories);
// get list of ili-files for a given model
ArrayList modelNames=new ArrayList();
modelNames.add("DM01");
Configuration config=manager.getConfig(modelNames,1.0); // 1.0=ili 1
modell
// compile models
TransferDescritpion td=Ili2c.runCompiler(config);
```

Stellt der Compiler im Modell Fehler fest (z.B. falsche Syntax), werden diese via EhiLogger ausgegeben. Die Anwendung des EhiLogger's ist in einer getrennten Anleitung beschrieben.

Modelle lesen ohne Verwendung von Modell-Ablagen

Kennt man den Dateinamen der INTERLIS-Modelldatei, verwendet man den folgenden Code, um das Modell zu lesen:

```
String iliFile=”dm01av.ili”;
ArrayList ilifiles=new ArrayList();
ArrayList ilidirs=new ArrayList();
ilifiles.add(iliFile);
TransferDescription td=ch.interlis.ili2c.Main.compileIliFiles(
                              ilifiles, ilidirs, null);
if(td==null){
  System.err.println("ili-compiler failed");
}
```

Handelt es sich um ein INTERLIS 2-Modell, werden evtl. andere Modelle importiert (z.B. das Modell „Units“). Die Funktionen suchen automatisch in den gegebenen Verzeichnissen (ilidirs) nach diesen Modellen.
Kennt man den Modellnamen des INTERLIS-Modells, muss man auch das Verzeichnis kennen, indem sich die INTERLIS-Modelldateien befinden. Dann verwendet man den folgenden Code, um das Modell zu lesen:

```
String iliModel=”DM01AV”;
String iliDir=”c:/interlis/modelle”;
ArrayList ilimodels=new ArrayList();
```

```
ilimodels.add(iliModel);
ArrayList ilidirs=new ArrayList();
ilidirs.add(iliDir);
TransferDescription td=ch.interlis.ili2c.Main.compileIliModels(
                                ilimodels, ilidirs, null);
if(td==null){
  System.err.println("ili-compler failed");
}
```

INTERLIS 1

Für INTERLIS 1 stehen die folgenden Typen zur Verfügung:

ch.interlis.iom_j.itf.ItfWriter	INTERLIS 1 Daten schreiben
ch.interlis.iom_j.itf.ItfReader	INTERLIS 1 Daten lesen

Ein INTERLIS 1 Reader wird mit folgendem Code erzeugt (wobei transferDescription das vorgängig gelesene Modell repräsentiert):

```
ItfReader ioxReader=null;
try{
  // open itf file
  ioxReader=new ItfReader(new java.io.File(itfFileName));
  ioxReader.setModel(transferDescription);
  ...

}catch(IoxException ex){
  EhiLogger.logError(ex);
}finally{
  if(ioxReader!=null){
    try{
      ioxReader.close();
    }catch(IoxException ex){
      EhiLogger.logError(ex);
    }
    ioxReader=null;
  }
}
```

Der Aufruf von setModel() kann bis zum ersten StartBasketEvent verzögert werden.
Ein INTERLIS 1 Writer wird mit folgendem Code erzeugt (wobei transferDescription das vorgängig gelesene Modell repräsentiert):

```
ItfWriter ioxWriter=null;
try{
  // create itf file
  ioxWriter=new ItfWriter(
              new java.io.File(itfFileName)
             ,transferDescription
             );
  ...

}catch(IoxException ex){
  EhiLogger.logError(ex);
}finally{
  if(ioxWriter!=null){
    try{
      ioxWriter.close();
    }catch(IoxException ex){
      EhiLogger.logError(ex);
    }
    ioxWriter=null;
  }
}
```

Bei Aufzählungen wird die Codierung gemäss INTERLIS 2 verwendet, d.h. es findet innerhalb des Readers eine Abbildung des numerischen Codes gem. INTERLIS 1 auf den Elementnamen gem. INTERLIS 2 statt. Und entsprechend beim Writer die umgekehrte Abbildung.
ACHTUNG: Die Baskets (Instanzen der TOPICs) und Objekte (Instanzen der TABLEs) müssen in der richtigen Reihenfolge geschrieben werden. Der ItfWriter ordnet sie nicht!
ACHTUNG: Der ItfReader führt keine Flächenbildung durch, d.h. die Geometrie für SURFACE und AREA Attribute wird als Linie in Objekten der Linientabelle geliefert (ItfReader). Entsprechend erwartet auch der ItfWriter diese Linientabellen.

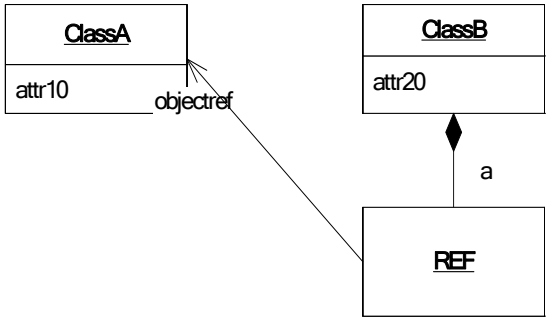
Referenzattribute

Referenzen werden mit IOX als Teil eines strukturierten Werts repräsentiert.
Zum Beispiel ist das folgende konzeptionelle Datenmodell in INTERLIS 1 gegeben:

```
TABLE ClassA =
  attr10 : TEXT*20;
NO IDENT
END ClassA;

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

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();
}
```

Tabellendefinition (Modelldaten) zu einem Objekt ermitteln

Die Objekte (IomObject) enthalten mit Absicht keinen direkten Verweis auf die entsprechenden Klassendefinitionen wie sie vom Compiler gelesen wurden. Die Zuordnung muss darum über eine Abbildungstabelle erfolgen.

```
TransferDescription td=ch.interlis.ili2c.Main.compileIliModels(...);
// Abbildungstabelle aufbauen
Map tag2type=ch.interlis.iom_j.itf.ModelUtilities.getTagMap(td);
...
IoxEvent event;
...
IomObject iomObj=((ObjectEvent)event).getIomObject();
```

```
// qualifizierter Klassenname abbilden auf Klassendefinition
AbstractClassDef tableDef=tag2type.get(iomObject.getobjecttag());
```

Attribute (Modelldaten) zu einer Tabelle ermitteln

Welche Attribute eine Tabelle hat, ergibt sich aus dem Modell.

Der Compiler übersetzt INTERLIS 1 Modelle intern in ein INTERLIS 2 Modell, d.h. aus einem INTERLIS 1-Referenzattribut entsteht somit eine Assoziation. Würde man das INTERLIS 1-Modell gem. den INTERLIS 2-Regeln transferieren, würde diese Assoziation bei der Klasse wo das INTERLIS 1-Referenzattribut definiert ist, eingebettet, d.h. die Liste der INTERLIS 1-Attribute umfasst die Attribute und Rollen. getAttributesAndRoles() liefert allerdings für INTERLIS 1 nicht die richtige Reihenfolge! Darum gibt es eine Hilfsfunktion getIli1AttrList().

Die Liste der Attribute zu einer INTERLIS 1-Tabelle kann somit mit folgendem Code durchlaufen werden:

```
AbstractClassDef tableDef;
...
ArrayList attrs=ModelUtilities.getIli1AttrList(tableDef);
Iterator attri=attrs.iterator();
while(attri.hasNext()){
  ViewableTransferElement obj=(ViewableTransferElement)attri.next();
  if (obj.obj instanceof AttributeDef) {
    AttributeDef attr = (AttributeDef) obj.obj;
    ...
  }
  if(obj.obj instanceof RoleDef){
    RoleDef role = (RoleDef) obj.obj;
    if(obj.embedded){
      // a role of an embedded association
      ...
    }else if(!((AssociationDef)tableDef).isLightweight()){
      // a role in an association table
      ...
    }else{
      // a role in an association table; but embedded association
      // skip it
    }
  }
}
```

INTERLIS 2

Für INTERLIS 2 stehen die folgenden Typen zur Verfügung:

ch.interlis.iom_j.xtf.XtfWriter	INTERLIS 2 Daten schreiben
ch.interlis.iom_j.xtf.XtfReader	INTERLIS 2 Daten lesen

Ein INTERLIS 2 Reader wird mit folgendem Code erzeugt:

```
XtfReader ioxReader=null;
try{
  // open xtf file
  ioxReader=new XtfReader(new java.io.File(xtfFileName));
  ...

}catch(IoxException ex){
  EhiLogger.logError(ex);
}finally{
  if(ioxReader!=null){
    try{
      ioxReader.close();
    }catch(IoxException ex){
      EhiLogger.logError(ex);
    }
```

```
    ioxReader=null;
  }
}
```

Der XtfReader benötigt kein Datenmodell.
Ein INTERLIS 2 Writer wird mit folgendem Code erzeugt (wobei transferDescription das vorgängig gelesene Modell repräsentiert):

```
XtfWriter ioxWriter=null;
try{
  // create xtf file
  ioxWriter=new XtfWriter(
             new java.io.File(itfFileName)
            ,transferDescription
            );
  ...

}catch(IoxException ex){
  EhiLogger.logError(ex);
}finally{
  if(ioxWriter!=null){
    try{
      ioxWriter.close();
    }catch(IoxException ex){
      EhiLogger.logError(ex);
    }
    ioxWriter=null;
  }
}
```

Assoziationen

Referenzen werden mit IOX als Teil eines strukturierten Werts repräsentiert. Je nach Art der Assoziation ist das sich ergebende Objektmodell unterschiedlich. Die Abbildung folgt dabei 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).

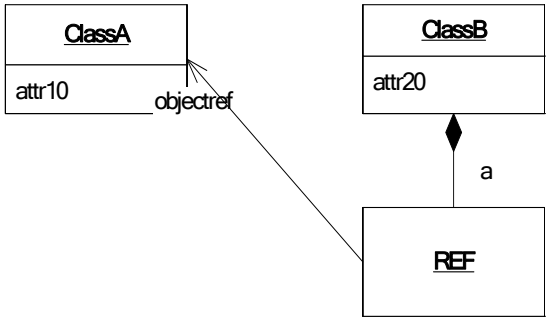
Für eine eingebettete Beziehung würde das konzeptionelle Datenmodell in INTERLIS 2 wie folgt aussehen:

```
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();
}
```

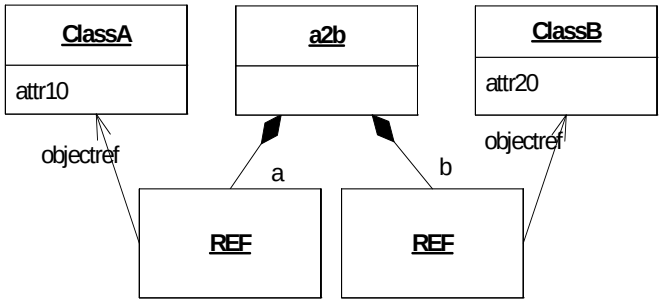
Für eine nicht eingebettet Beziehung würde das Modell wie folgt aussehen:

```
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();
}
```

Geometrien

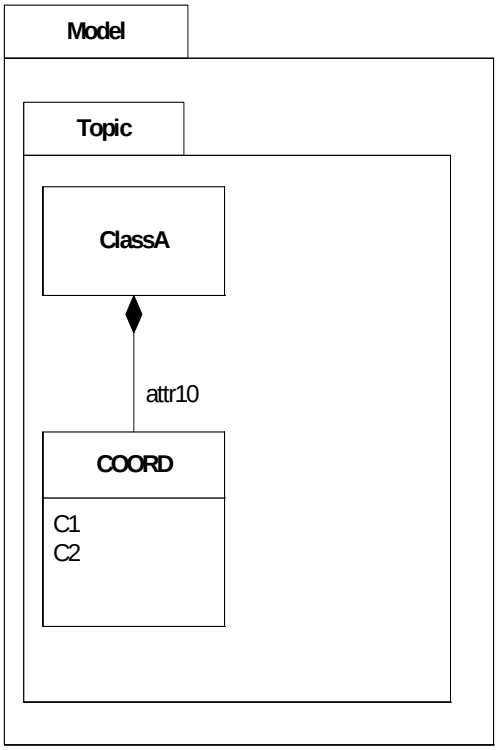
COORD

Um COORD Geometrien zu repräsentieren, verwenden die INTERLIS Reader/Writer die im Folgenden dargestellte Struktur (IomObject ohne Identität).
Beispiel INTERLIS 1 Model:

```
TABLE ClassA =
  attr10 : COORD2 1.00 100.0 9.99 999.9;
NO IDENT
END ClassA;
```

Beispiel INTERLIS 2 Model:

```
CLASS ClassA =
  attr10 : COORD 1.00 .. 9.99, 100.0 .. 999.9;
END ClassA;
```



Die Namen Model, Topic, ClassA und attr10 ergeben sich aus dem Datenmodell. Die übrigen Namen COORD, C1, C2 ergeben sich aus dem Datentyp COORD und dessen Abbildung durch die ItfReader/ItfWriter/XtfReader/XtfWriter.
Gegeben ein IomObject der Klasse ClassA, kann mit folgendem Code auf die Koordinatenwerte des Attributes attr10 zugegriffen werden:

```
if(objClassA.getattrvalue("attr10")==null){
  IomObject coord= objClassA.getattrobj("attr10",0);
  String c1=coord.getattrvalue("C1");
  String c2=coord.getattrvalue("C2");
}
```

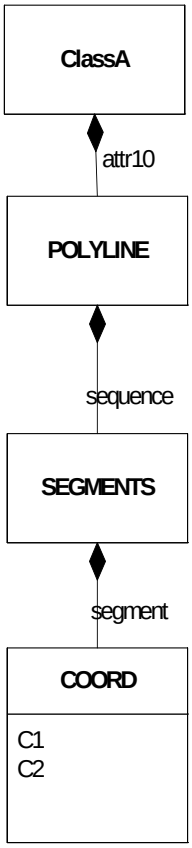
POLYLINE

Um POLYLINE Geometrien zu repräsentieren, verwenden die INTERLIS Reader/Writer die im Folgenden dargestellten Strukturen (IomObject ohne Identität).
Beispiel INTERLIS 1 Model:

```
DOMAIN Lkoord = COORD2 1.00 100.0 9.99 999.9;
TABLE ClassA =
  attr10 : POLYLINE WITH (STRAIGHTS) VERTEX Lkoord;
NO IDENT
END ClassA;
```

Beispiel INTERLIS 2 Model:

```
DOMAIN Lkoord = COORD 1.00 .. 9.99, 100.0 .. 999.9;
CLASS ClassA =
  attr10 : POLYLINE WITH (STRAIGHTS) VERTEX Lkoord;
END ClassA;
```



Die Namen ClassA und attr10 ergeben sich aus dem Datenmodell. Die übrigen Namen (POLYLINE, sequence, SEGMENTS, ...) ergeben sich aus dem Datentyp POLYLINE und dessen Abbildung durch die ItfReader/ItfWriter/XtfReader/XtfWriter. Gegeben ein IomObject der Klasse ClassA, kann mit folgendem Code auf die Kurvenstücke des Attributes attr10 zugegriffen werden:

```
IomObject polylineObj=objClassA.getattrobj("attr10",0);
boolean clipped
    = polylineObj.getobjectconsistency()==IomConstants.IOM_INCOMPLETE;
for(int sequencei=0;
    sequencei<polylineObj.getattrvaluecount("sequence");
    sequencei++){
  if(!clipped && sequencei>0){
    // an unclipped polyline should have only one sequence element
    throw new IllegalArgumentException();
  }
  IomObject sequence=polylineObj.getattrobj("sequence",sequencei);
  for(int segmenti=0;
      segmenti<sequence.getattrvaluecount("segment");
      segmenti++){
    IomObject segment=sequence.getattrobj("segment",segmenti);
    if(segment.getobjecttag().equals("COORD")){
      // COORD
      ...
    }else if(segment.getobjecttag().equals("ARC")){
      // ARC
      ...
    }else{
      // custum line form
      ...
    }

  }
```

```
}
```

Mit der Funktion getobjectconsistency() kann abgefragt werden, ob es sich um eine vollständige oder unterteilte Linie handelt. Eine unterteilte Linie entsteht, wenn nur ein Ausschnitt aus einem Datensatz transferiert wird. Ob ein Basket vollständig ist oder nur ein Ausschnitt, lässt sich mit der Funktion getConsistency() auf dem StartBasketEvent überprüfen.
Die Objektstruktur entspricht aus zwei Gründen nicht 1:1 der Elementstruktur in XML.

1. die Elementnamen sind z.T. sowohl Rollenname als auch Typname.
2. eine Schachtelung der Elementstruktur entfällt bei einer vollständigen Linie (bei einer die nicht wegen Ausschnittbildung unterteilt wurde) zugunsten von weniger Markup. Die Objektstruktur dagegen ist immer identisch. Bei einer vollständigen Linie gibt es genau ein Objekt SEGMENTS, bei einer Unterteilten Mehrere.

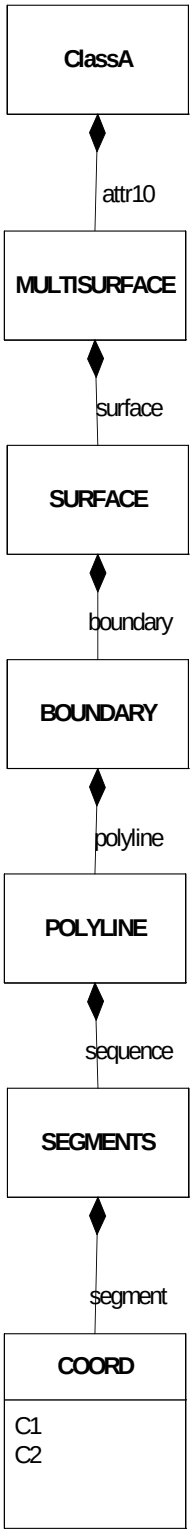
SURFACE/AREA

Um SURFACE bzw. AREA Geometrien zu repräsentieren, verwenden der INTERLIS 2 Reader/Writer die im Folgenden dargestellten Strukturen (IomObject ohne Identität). SURFACE und AREA werden nicht unterschieden, d.h. dieser Unterscheid ergibt sich nur aus dem Modell.
ACHTUNG: Der ItfReader führt keine Flächenbildung durch, d.h. die Geometrie für SURFACE und AREA Attribute wird als Linie in Objekten der Linientabelle geliefert (ItfReader). Entsprechend erwartet auch der ItfWriter diese Linientabellen. Beachten Sie darum das vorgängige Kapitel zum Thema POLYLINE.
Beispiel INTERLIS 2 Model für SURFACE:

```
DOMAIN Lkoord = COORD 1.00 .. 9.99, 100.0 .. 999.9;
CLASS ClassA =
  attr10 : SURFACE WITH (STRAIGHTS,ARCS) VERTEX Lkoord;
END ClassA;
```

Beispiel INTERLIS 2 Model für AREA:

```
DOMAIN Lkoord = COORD 1.00 .. 9.99, 100.0 .. 999.9;
CLASS ClassA =
  attr10 : AREA WITH (STRAIGHTS,ARCS) VERTEX Lkoord;
END ClassA;
```



Die Namen ClassA und attr10 ergeben sich aus dem Datenmodell. Die übrigen Namen (MULTISURFACE, surface, SURFACE, boundary, ...) ergeben sich aus dem Datentyp SURFACE oder AREA und dessen Abbildung durch den XtfReader/XtfWriter.
Gegeben ein IomObject der Klasse ClassA, kann mit folgendem Code auf die Linien der Ränder des Attributes attr10 zugegriffen werden:

```
IomObject surfaceObj=objClassA.getattrobj("attr10",0);
assert(surfaceObj.getobjecttag().equals("MULTISURFACE"));
boolean clipped =
```

```
    surfaceObj.getobjectconsistency()==IomConstants.IOM_INCOMPLETE;
for(int surfacei=0;
    surfacei< surfaceObj.getattrvaluecount("surface");
    surfacei++){
  if(!clipped && surfacei>0){
    // unclipped surface with multi 'surface' elements
    throw new IllegalArgumentException();
  }
  IomObject surface= surfaceObj.getattrobj("surface",surfacei);
  int boundaryc=surface.getattrvaluecount("boundary");
  for(int boundaryi=0;boundaryi<boundaryc;boundaryi++){
    IomObject boundary=surface.getattrobj("boundary",boundaryi);
    if(boundaryi==0){
      // shell
    }else{
      // hole
    }
    for(int polylinei=0;
        polylinei<boundary.getattrvaluecount("polyline");
        polylinei++){
      IomObject polyline=boundary.getattrobj("polyline",polylinei);
      // add line to shell or hole
      ...
    }
    // add shell or hole to surface
    // ....
  }
}
```

Modellname aus Transferdatei ermitteln

Kennt man das Modell nicht, kann man es aus der Transferdatei ermitteln, indem man bis zum ersten Basket liest. Der folgende Code zeigt wie das geht für INTERLIS 1, es funktioniert aber identisch für INTERLIS 2 (mit dem XtfReader statt dem ItfReader).

```
ItfReader ioxReader=null;
try{
  // open itf file
  ioxReader=new ItfReader(new java.io.File(itfFileName));
  // get first basket
  IoxEvent event;
  StartBasketEvent be=null;
  do{
    event=ioxReader.read();
    if(event instanceof StartBasketEvent){
      be=(StartBasketEvent)event;
      break;
    }
    }while(!(event instanceof EndTransferEvent));
    ioxReader.close();
    ioxReader=null;
    // no baskets?
    if(be==null){
      // no models!
      throw new IllegalArgumentException("no baskets in transfer-file");
  }
  String namev[]=be.getType().split("\\.");
  String model=namev[0];
}catch(IoxException ex){
  EhiLogger.logError(ex);
}finally{
```

```
  if(ioxReader!=null){
    try{
      ioxReader.close();
    }catch(IoxException ex){
      EhiLogger.logError(ex);
    }
    ioxReader=null;
  }
}
```

Fehlermeldungen

Bei Datenfehlern die eine Weiterverarbeitung nicht ausschliessen (z.B. falscher Code bei einem Aufzählattribut), wird die Fehlermeldung via EhiLogger ausgegeben.
Bei Datenfehlern (z.B. falsche Folge von TABL, OBJE, ETAB in INTERLIS 1) die eine Weiterverarbeitung behindern/ausschliessen, wird eine IoxException ausgelöst.

Benötigte JAR-Dateien

iox-api.jar	IOX API (Interfaces)
iox-ili.jar	INTERLIS 1+2 Reader/Writer
ili2c.jar	INTERLIS-Compiler (beinhaltet EhiLogger)