ilivalidator-Anleitung

Überblick

Ilivalidator ist ein in Java erstelltes Programm, das eine Interlis-Transferdatei (itf oder xtf) gemäss einem Interlis-Modell (ili) überprüft.

Es bestehen u.a. folgende Konfigurationsmöglichkeiten:

- einzelne Prüfungen ein oder auszuschalten
- eigene Fehlermeldungen inkl. Attributwerte zu definieren
- zusätzliche Bedingung zu definieren
- zusätzliche INTERLIS-Funktionen zu implementieren
- Modellnamen zu setzen

Zusätzlich umfasst der IliValidator Hilfsfunktionen betrf. Daten (z.B. Kataloge) in einem Repository.

Log-Meldungen

Die Log-Meldungen sollen dem Benutzer zeigen, was das Programm macht. Am Anfang erscheinen Angaben zur Programm-Version. Falls das Programm ohne Fehler durchläuft, wird das am Ende ausgegeben.:

```
Info: ilivalidator-1.0.0
...
Info: compile models...
...
Info: ...validation done
```

Bei einem Fehler wird das am Ende des Programms vermerkt. Der eigentliche Fehler wird aber in der Regel schon früher ausgegeben.:

```
Info: ilivalidator-1.0.0
...
Info: compile models...
...
Error: DM01.Bodenbedeckung.BoFlaeche_Geometrie: intersection tids 48, 48
...
Error: ...validation failed
```

Laufzeitanforderungen

Das Programm setzt Java 1.6 voraus.

Zur Validierung wird RAM benötigt. Für eine typische Transferdatei sollten ca. 2 GB RAM ausreichen. Am Anfang des Logs steht, wieviel RAM (heapspace) dem Programm zur Verfügung steht. Sollte das Programm mit einer Heapspace Fehlermeldung abbrechen, kann mittels Java-Option versucht werden, mehr RAM bereitzustellen (Für 3 GB z.B. `java -Xmx3072m -jar ilivalidator.jar data.xtf`). Grundsätzlich ist nicht die Grösse der Datei kritisch, sondern andere Dinge z.B. wieviele Objekte miteinander in Beziehung stehen, oder wieviele Objekte bei UNIQUE Bedingungen geprüft werden müssen, aus wievielen Rändern die Polygone bestehen, usw.

Validierung in anderen Programmen

Der ilivalidator wird auch von anderen Programmen verwendet (z.B. ili2fgdb). Für die Validierung wird im ilivalidator und im anderen Programm (z.B. ili2fgdb) der selbe Code verwendet. Der gemeisame Nenner ist iox-ili. Man muss also die Version von iox-ili vergleichen, um allenfalls die Validierung aufeinander abstimmen zu können (am Anfang des Logs zeigen beide Programme auch die Version von iox-ili, und sonst steht es normalerweise im changelog.txt des jeweiligen Programms.) z.B.

- ili2fgdb-4.4.5 benutzt iox-ili-1.21.4
- ilivalidator-1.11.9 benutzt iox-ili-1.21.4

Grundsätzlich sollten die Daten natürlich gültig sein gegenüber der Spezifikation (also dem INTERLIS-Referenzhandbuch und dem Minimalen Geodatenmodell).

Lizenz

GNU Lesser General Public License

Funktionsweise

In den folgenden Abschnitten wird die Funktionsweise anhand einzelner Anwendungsfälle beispielhaft beschrieben. Die detaillierte Beschreibung einzelner Funktionen ist im Kapitel „Referenz“ zu finden.

Je nach Betriebssystem kann das Programm auch einfach durch Doppelklick mit linker Maustaste auf `` `ilivalidator.jar` `` gestartet werden.

Beispiele

Fall 1

Es wird eine INTERLIS 1-Datei validiert/geprüft.

```
java -jar ilivalidator.jar path/to/data.itf
```

Fall 2

Es wird eine INTERLIS 2-Datei inkl. den Referenzen auf den Katalog validiert/geprüft.

```
java -jar ilivalidator.jar --allObjectsAccessible ilidata:catalogDatasetId path/to/data.itf
```

Fall 3

Es wird eine INTERLIS 2-Datei mit einer spezifischen Konfiguration validiert/geprüft.

```
java -jar ilivalidator.jar --config config.ini path/to/data.xtf
```

In der Datei config.ini wird definiert, welche Prüfungen gar nicht durchzuführen oder bei Nichterfüllen nur als Warnung zu melden sind.

Fall 4

Es wird eine INTERLIS 2-Datei validiert/geprüft, wobei die Fehlermeldungen in eine Text-Datei geschrieben werden.

```
java -jar ilivalidator.jar --log result.log path/to/data.xtf
```

Die Fehlermeldungen inkl. Warnungen werden in die Datei result.log geschrieben.

Fall 5

Es wird eine INTERLIS 2-Datei validiert/geprüft, wobei die Fehlermeldungen als Daten in eine Xtf-Datei geschrieben werden.

```
java -jar ilivalidator.jar --xtflog result.xtf path/to/data.xtf
```

Die Fehlermeldungen inkl. Warnungen werden in die Datei result.xtf geschrieben. Die Datei result.xtf entspricht dem Modell IliVErrors und kann als normale INTERLIS 2-Transferdatei importiert werden. Dadurch können die Fehler visualisiert werden.

Fall 6

Es erscheint eine Bildschirmmaske, mit deren Hilfe die zu validierende Datei ausgewählt und die Validierung gestartet werden kann.

```
java -jar ilivalidator.jar
```

Fall 7

Es wird eine INTERLIS 2-Datei validiert/geprüft. Wobei spezifische Modelle gesetzt werden. Dazu wird der Pfad zu den spezifischen Modellen gesetzt.

```
java -jar ilivalidator.jar --models modelname1;modelname2 --modeldir path/to/data path/to/data.xtf
```

Fall 8

Es werden alle Dateien (ITF und XTF) im gegebenen Repository geprüft/validiert.

```
java -jar ilivalidator.jar --check-repo-data http://models.geo.admin.ch
```

Fall 9

Es werden alle Dateien (ITF und XTF) im gegebenen Verzeichnis `folder` analysiert und dann ein neues `newIlidata.xml` mit den entsprechenden Metadaten erstellt.

```
java -jar ilivalidator.jar --createIliData --ilidata newIlidata.xml --repos folder
```

Fall 10

Es werden alle Dateien (ITF und XTF) gemäss Dateiliste `files.txt` im Repository `http://models.geo.admin.ch` analysiert und dann ein neues `newIlidata.xml` mit den entsprechenden Metadaten erstellt.

```
java -jar ilivalidator.jar --createIliData --ilidata newIlidata.xml --repos http://models.geo.admin.ch --srcfiles
files.txt
```

Fall 11

Es wird die gegebene Datei `newVersionOfData.xml` (ITF oder XTF) analysiert, und dann das ilidata.xml aus dem gegebenen Repository `http://models.geo.admin.ch` mit einem neuen Eintrag für den Datensatz mit der ID `datasetId` aktualisiert. Die neue Version des ilidata.xml wird in die Datei `updatedIlidata.xml` geschrieben und muss durch den Benutzer ins Repository übertragen werden.

```
java -jar ilivalidator.jar --updateIliData --ilidata updatedIlidata.xml --repos http://models.geo.admin.ch --datasetId
datasetId newVersionOfData.xml
```

Referenz

In den folgenden Abschnitten werden einzelne Aspekte detailliert, aber isoliert, beschrieben. Die Funktionsweise als Ganzes wird anhand einzelner Anwendungsfälle beispielhaft im Kapitel „Funktionsweise“ (weiter oben) beschrieben.

Aufruf-Syntax

```
java -jar ilivalidator.jar [Options] [file]
```

`file` kann auch die Form `ilidata:DatesetId` oder `ilidata:BasketId` haben, dann wird die entsprechende Datei aus den Repositories benutzt.

Ohne Kommandozeilenargumente erscheint die Bildschirmmaske, mit deren Hilfe die zu validierende Datei ausgewählt und die Validierung gestartet werden kann.

Der Rückgabewert ist wie folgt:

> - 0 Validierung ok, keine Fehler festgestellt
> - !0 Validierung nicht ok, Fehler festgestellt

Optionen:

Option	Beschreibung

`--config filename`	Konfiguriert die Datenprüfung mit Hilfe einer INI-Datei. `filename` kann auch die Form `ilidat` Datei aus den Repositories benutzt. Der Eintrag im ilidata.xml soll mit folgenden Kategorien m
	
	`<categories>`
	`  <DatasetIdx16.Code_>`
	`  <value>http://codes.interlis.ch/type/ilivalidatorconfig</value> <!-- Hinweis, dass`
	`  </DatasetIdx16.Code_>`
	`  <DatasetIdx16.Code_>`
	`  <value>http://codes.interlis.ch/model/Simple23</value> <!-- Hinweis auf des ili-Mo`
	`  </DatasetIdx16.Code_>`
	`</categories>`

`--metaConfig filename`	Konfiguriert den Validator mit Hilfe einer INI-Datei. `filename` kann auch die Form `ilidata:Da` aus den Repositories benutzt. Der Eintrag im ilidata.xml soll mit folgenden Kategorien markiert
	
	`<categories>`
	`  <DatasetIdx16.Code_>`
	`  <value>http://codes.interlis.ch/type/metaconfig</value> <!-- Hinweis, dass es eine`
	`  </DatasetIdx16.Code_>`
	`  <DatasetIdx16.Code_>`
	`  <value>http://codes.interlis.ch/model/Simple23</value> <!-- Hinweis auf des ili-Mo`
	`  </DatasetIdx16.Code_>`
	`</categories>`

`--forceTypeValidation`	Ignoriert die Konfiguration der Typprüfung aus der INI-Datei, d.h. es kann nur die Multiplizität
`--disableAreaValidation`	Schaltet die AREA Topologieprüfung aus (XTF).
`--disableConstraintValidation`	Schaltet die Constraint prüfung aus.
`--allObjectsAccessible`	Mit der Option nimmt der Validator an, dass er Zugriff auf alle Objekte hat. D.h. es wird z.B. au Objekte geprüft.
`--multiplicityOff`	Schaltet die Prüfung der Multiplizität generell aus.
`--singlePass`	Schaltet alle Prüfungen aus, die nicht unmittelbar beim Ersten Lesen der Objekte ausgeführt w
`--skipPolygonBuilding`	Schaltet die Bildung der Polygone aus (nur ITF).
`--allowItfAreaHoles`	Lässt bei ITF AREA Attributen innere Ränder zu, die keinem Objekt zugeordnet sind.
`--models modelnames`	Setzt spezifische Modellnamen, welche sich innerhalb von ili-Dateien befinden. Mehrere Mode Das Setzen des Pfades, der zu den Modellen führt, muss mittels '--modeldir path' angegeben v
`--modeldir path`	Dateipfade, die Modell-Dateien (ili-Dateien) enthalten. Mehrere Pfade können durch Semikolo Repositories möglich. Default ist %ITF_DIR;http://models.interlis.ch/;%JAR_DIR/ilimodels %ITF_DIR ist ein Platzhalter für das Verzeichnis mit der Transferdatei. %JAR_DIR ist ein Platzhalter für das Verzeichnis des ilivalidator Programms (ilivalidator.jar Date

Option	Beschreibung
	Der erste Modellname (Hauptmodell), zu dem ili2db die ili-Datei sucht, ist nicht von der INTER Reihenfolge nach einer ili-Datei gesucht: zuerst INTERLIS 2.3, dann 1.0 und zuletzt 2.2. Beim Auflösen eines IMPORTs wird die INTERLIS Sprachversion des Hauptmodells berücksichti ili2.3 unterschieden wird.
`--check-repo-data repositoryUrl`	Es werden alle Daten (ITF und XTF) im gegebenen Repository geprüft/validiert. (Alle aktuellen I
`--createIliData --ilidata ilidata.xml --repos repository`	Es werden alle Daten (ITF und XTF) im gegebenen Folder/Repository analysiert und dann ein n erstellt. Wenn `repository` ein remote Repository bezeichnet, muss mit `--srcfiles` die Liste d
`--srcfiles files.txt`	Liste mit relativen Dateipfaden (relativ zum gegebenen Folder/Repository). Ein Pfad pro Zeile.
`--updateIliData --ilidata updatedIlidata.xml --repos repository --dataset datasetId newVersionOfData.xml`	Es wird die gegebene Datei `newVersionOfData.xml` (ITF oder XTF) analysiert, und dann das ilid mit einem neuen Eintrag für den Datensatz mit der ID `datasetId` aktualisiert. Die neue Versio geschrieben und muss durch den Benutzer ins Repository übertragen werden.
`--logtime`	Ergänzt die log-Meldungen in der Log-Datei mit Zeitstempeln.
`--log filename`	Schreibt die log-Meldungen in eine Text-Datei.
`--xtflog filename`	Schreibt die log-Meldungen in eine INTERLIS 2-Datei. Die Datei result.xtf entspricht dem Mode
`--plugins folder`	Verzeichnis mit JAR-Dateien, die Zusatzfunktionen enthalten. Die Zusatzfunktionen müssen da `ch.interlis.iox_j.validator.InterlisFunction` implementieren, und der Name der Java-Klas
`--proxy host`	Proxy Server für den Zugriff auf Modell Repositories
`--proxyPort port`	Proxy Port für den Zugriff auf Modell Repositories
`--gui`	Es erscheint eine Bildschirmmaske, mit deren Hilfe die zu validierende Datei ausgewählt und d Modell-Dateien und die Proxyeinstellungen werden aus der Datei $HOME/.ilivalidator gelesen.
`--verbose`	Schreibt detailiertere validierungs log-Meldungen.
`--trace`	Erzeugt zusätzliche Log-Meldungen (wichtig für Programm-Fehleranalysen)
`--help`	Zeigt einen kurzen Hilfetext an.
`--version`	Zeigt die Version des Programmes an.

Meta-Konfiguration

In der Meta-Konfigurationsdatei werden die folgenden Parameter unterstützt (hier nicht aufgeführte Kommandozeilenargument werden in der Meta-Konfiguration nicht unterstützt).

Konfiguration	Beispiel	Beschreibung
baseConfig	`[CONFIGURATION]` `baseConfig=ilidata:DatesetId`	Basis-Meta-Konfiguration, auf der die aktuelle Meta-Konfiguration aufbaut. Statt `ilidata:DatesetId` kann auch die Form `file:/localfile` benutzt werden, dann wird die entsprechende lokale Datei benutzt. Mehrere Basiskonfigurationen werden mit einem Strichpunkt ";" getrennt.
org.interlis2.validator.config	`[CONFIGURATION]` `org.interlis2.validator.config=ilidata:DatesetId`	

Konfiguration	Beispiel	Beschreibung
		Validierungs-Konfiguration, die benutzt werden soll. Statt `ilidata:DatesetId` kann auch die Form `file:/localfile` benutzt werden, dann wird die entsprechende lokale Datei benutzt. Mehrere Validierungs-Konfigurationen werden mit einem Strichpunkt ";" getrennt.
ch.interlis.referenceData	`[CONFIGURATION]` `ch.interlis.referenceData=ilidata:DatesetId`	Basis-Daten (z.B. Kataloge), die benutzt werden sollen. Statt `ilidata:DatesetId` kann auch die Form `file:/localfile` benutzt werden, dann wird die entsprechende lokale Datei benutzt. Mehrere Basis-Daten werden mit einem Strichpunkt ";" getrennt.
models	`[ch.ehi.ilivalidator]` `models=Simple23`	Entspricht dem Kommandozeilenargument `--models`
config	`[ch.ehi.ilivalidator]` `config=ilidata:DatesetId`	Entspricht dem Kommandozeilenargument `--config`
forceTypeValidation	`[ch.ehi.ilivalidator]` `forceTypeValidation=true`	Entspricht dem Kommandozeilenargument `--forceTypeValidation`
disableAreaValidation	`[ch.ehi.ilivalidator]` `disableAreaValidation=true`	Entspricht dem Kommandozeilenargument `--disableAreaValidation`
disableConstraintValidation	`[ch.ehi.ilivalidator]` `disableConstraintValidation=true`	Entspricht dem Kommandozeilenargument `--disableConstraintValidation`
multiplicityOff	`[ch.ehi.ilivalidator]` `multiplicityOff=true`	Entspricht dem Kommandozeilenargument `--multiplicityOff`
allObjectsAccessible	`[ch.ehi.ilivalidator]` `allObjectsAccessible=true`	Entspricht dem Kommandozeilenargument `--allObjectsAccessible`
allowItfAreaHoles	`[ch.ehi.ilivalidator]` `allowItfAreaHoles=true`	Entspricht dem Kommandozeilenargument `--allowItfAreaHoles`
skipPolygonBuilding	`[ch.ehi.ilivalidator]` `skipPolygonBuilding=true`	Entspricht dem Kommandozeilenargument `--skipPolygonBuilding`

Konfiguration

Die einzelnen Prüfungen können direkt im Modell über Metaaatribute konfiguriert werden oder in einer getrennten Konfigurations-Datei, so dass keine Änderung der ili-Datei notwendig ist.

Um z.B. bei einem Attribut den Mandatory Check ganz auszuschalten, schreibt man in der ili-Datei:

CLASS Gebaeude =
!!@ ilivalid.multiplicity = off

Art : MANDATORY (...);
Um dieselbe Konfiguration ohne Änderung der ili-Datei vorzunehmen, schreibt man in der INI-Datei:

["Beispiel1.Bodenbedeckung.Gebaeude.Art"]
multiplicity="off"
Zusätzlich erlaubt die INI Datei pauschale Konfigurationen im Abschnitt "PARAMETER". Um z.B. generell alle Prüfungen auszuschalten schreibt man in die INI-Datei:

["PARAMETER"]
validation="off"

INI-Konfigurationsdatei

Beispiel1.ini

INI-Globale Konfigurationen

Konfiguration	Beispiel	Beschreibung
additionalModels	["PARAMETER"] additionalModels="Model1;Modell2"	"Model1" und "Modell2" sind die Namen der Modelle mit Definitionen von zusätzlichen Validierungen (in Form von Interlis Konsistenbedingungen). Mehrere Zusatzmodelle werden mit einem Strichpunkt ";" getrennt.
validation	["PARAMETER"] validation="off"	"off" schaltet generell alle Prüfungen aus. Mögliche Einstellungen sind: "off", "on". DEFAULT ist "on".
areaOverlapValidation	["PARAMETER"] areaOverlapValidation="off"	"off" schaltet die AREA-Topology Prüfung aus. Mögliche Einstellungen sind: "off", "on". DEFAULT ist "on".
constraintValidation	["PARAMETER"] constraintValidation="off"	"off" schaltet alle Prüfungen von Konsistenzbedingungen aus. Mögliche Einstellungen sind: "off", "on". DEFAULT ist "on".
defaultGeometryTypeValidation	["PARAMETER"] defaultGeometryTypeValidation="off"	Der Default-Wert für die Datentypprüfung bei Geometrie-Attributen. Mögliche Einstellungen sind: "warning", "off", "on". DEFAULT ist "on".
allowOnlyMultiplicityReduction	["PARAMETER"] allowOnlyMultiplicityReduction="true"	"true" ignoriert die Konfiguration der Typprüfungen aus der INI-Datei, d.h. es kann nur die Prüfung der Multiplizität konfiguriert werden. Mögliche Einstellungen sind: "true", "false". DEFAULT ist "false".
allObjectsAccessible	["PARAMETER"] allObjectsAccessible="true"	"true" definiert, dass die mitgegebenen Dateien alle Objekte enthalten, d.h. dass alle Referenzen (insb. mit EXTERNAL) auflösbar sind. Mit false können bei Referenzen mit EXTERNAL nicht alle Prüfungen durchgeführt werden. Mögliche Einstellungen sind: "true", "false". DEFAULT ist "false".
multiplicity	["PARAMETER"] multiplicity="off"	"off" schaltet die Multiplizitätsprüfung für alle Attribute und Rollen aus. Mögliche Einstellungen sind: "on", "warning", "off". DEFAULT ist "on".
disableRounding	["PARAMETER"] disableRounding="true"	"true" schaltet das Runden vor der Validierung von numerischen Werten aus (inkl. Koordinaten). Mögliche Einstellungen sind: "true", "false". DEFAULT ist "false".

Konfiguration	Beispiel	Beschreibung
disableAreAreasMessages	["PARAMETER"] disableAreAreasMessages="true"	"true" schaltet die Meldungen bei areAreas() Funktionen aus, d.h. die Funktion gibt keine Meldung aus, und liefert nur via den Funktioneswert, ob die Daten die AREA Bedingung erfüllen, oder nicht. Bei "false" gibt die areAreas() Funktionen zusätzlich zum Funktionswert Meldungen aus, wo die Daten die AREA Bedingung nicht erfüllen. Betrifft: INTERLIS.areAreas(), INTERLIS_ext.areAreas2(), INTERLIS_ext.areaAreas3() Mögliche Einstellungen sind: "true", "false". DEFAULT ist "false".
verifyModelVersion	["PARAMETER"] verifyModelVersion="true"	"true" es wird geprüft, ob die VERSIONs Angabe zum Model in der HEADERSECTION der XTF-Datei mit der Angabe im Modell (.ili-Datei) übereinstimmt. Wenn die Angabe nicht übereinstimmt, erfolt eine Info-Meldung. Mögliche Einstellungen sind: "true", "false". DEFAULT ist "false".

INTERLIS-Metaattribute

Die einzelnen Prüfungen können direkt im Modell über Metaaatribute konfiguriert werden. Metaattribute stehen unmittelbar vor dem Modellelement das sie betreffen und beginnen mit `!!@`. Falls der Wert (rechts von `=`) aus mehreren durch Leerstellen getrennten Wörtern besteht, muss er mit Gänsefüsschen eingerahmt werden (`"..."`).

Beispiel1.ili

Modelelement	Metaattribut	Beschreibung

ClassDef	`ilivalid.keymsg`
	`ilivalid.keymsg_de`
		Zusätzlicher Text für die Objektidentifikation für alle Fehlermeldung die sich auf ein Objekt der diesem Metaattribut folgenden Klasse beziehen. Die TID und Zeilennummer erscheint immer, falls vorhanden. keymsg ist zusätzlich (eine Benutzerdefinierte/verständliche Identifikation). Bei Export aus/Check auf DB ist TID evtl. nicht vorhanden. Bei XML ist die Zeilennummer in der Regel nicht hilfreich. Inkl. Attributwerte in {}. Für irgendeine Sprache bzw. fuer DE.
		
		`!!@ ilivalid.keymsg = "AssNr {AssNr}"`
		`!!@ ilivalid.keymsg_de = "Assekuranz-Nr {AssNr}"`

AttributeDef	`ilivalid.type`	Datentyppruefung ein/ausschalten bzw. nur als Hinweis. z.B. ob eine Zahlenwert innerhalb des Bereichs ist, oder ein Aufzählwert dem Modell entspricht oder die Flächen eine Gebietseinteilung sind usw. Werte sind on/warning/off `!!@ ilivalid.type = off`
AttributeDef	`ilivalid.multiplicity`	Multiplizitätprüfung ein/ausschalten bzw. nur als Hinweis. z.B. ob bei MANDATORY ein Wert vorhanden ist, oder nicht bzw. bei BAG/LIST ob die entsprechende Anzahl Strukturelemente vorhanden ist Werte sind on/warning/off `!!@ ilivalid.multiplicity = warning`
AttributeDef	`ilivalid.requiredIn`	Bei einem Referenz-Attribut oder Struktur-Attribut definieren, dass nur Objekte referenziert werden dürfen, die im Behälter mit der gegebenen BID vorkommen. Wenn das Metaattribut bei einem Struktur-Attribut benutzt wird, muss die Struktur ein Referenzattribut enthalten, und die Restriktion betrifft dann die von diesem Referenz-Attribut referenzierten Objekte.

Modelelement	Metaattribut	Beschreibung
		`!!@ ilivalid.requiredIn = bid1`
RoleDef	`ilivalid.target`	Zielobjekt-Prüfung ein/ausschalten bzw. nur als Hinweis. Prüft ob das referenzierte Objekt vorhanden ist und ob es von der gewünschten Klasse ist. Werte sind on/warning/off `!!@ ilivalid.target = warning`
RoleDef	`ilivalid.multiplicity`	Multiplizitätprüfung ein/ausschalten bzw. nur als Hinweis. Prüfen ob die vom Modell geforderte Anzahl Objekte referenziert wird. Werte sind on/warning/off `!!@ ilivalid.multiplicity = off`
RoleDef	`ilivalid.requiredIn`	Bei einer Rolle definieren, dass nur Objekte referenziert werden dürfen, die im Behälter mit der gegebenen BID vorkommen. `!!@ ilivalid.requiredIn = bid1`
ConstraintDef	`ilivalid.check`	Constraint-Prüfung ein/ausschalten bzw. nur als Hinweis. Prüfen ob die Konsistenzbedingung erfüllt ist oder nicht. Werte sind on/warning/off `!!@ ilivalid.check = warning`
ConstraintDef	`ilivalid.msg` `ilivalid.msg_de`	Meldungstext, falls dieses Constraint nicht erfüllt ist. Wird ergänzt um Objektidentifikation und Name des Constraints. inkl. Attributwerte in {} `!!@ ilivalid.msg_de = "AndereArt muss definiert sein"`
ConstraintDef	`name`	Name des Constraints (ili2.3 oder bei ili2.4 falls constraint kein name hat) Ergänzt die Fehlermeldung (ohne Name wird interne Id des Constraints verwendet) `!!@ name = c1023`

Wenn ein ConstraintDef keinen expliziten Namen hat, wird für die Referenzierung eine Name aus der interne Id des Constraints erzeugt. Die interne Id ist eine aufsteigende Zahl und beginnt pro Klasse mit 1. Das erste Constraint einer Klasse heisst also `Constraint1` , das Zweite `Constraint2` usw.

Modell IliVErrors

IliVErrors.ili

INTERLIS 1

Das Interlis 1 Modell wird intern in ein Interlis 2 Modell übersetzt. Tabellen werden zu Klassen, Attribute bleiben Attribute. Referenzattribute werden zu Assoziationen. Für die Namen der Assoziation und Rollen gelten folgende Regeln.

Normalerweise ist ein Rollenname der Name des Referenzattributes und der andere ist der Tabellenname, der das Referenzattribut enthält. Und der Assoziationsname ist die Verkettung der beiden (falls dies nicht zu einem Namenskonflikt führt). Zum Beispiel folgendes Interlis 1 Modell:

```
MODEL M =
        TOPIC T =
            TABLE A =
                    AttrA1: TEXT*20;
            END A;
                TABLE B =
                        AttrB1: TEXT*10;
```

```
                        AttrB2: -> A;
                        AttrB3: -> A;
                END B;
        END T.
END M.
```

`AttrB2` wird wie folgt übersetzt:

```
ASSOCIATION BAttrB2 =
        B -- {0..*} B;
        AttrB2 -- {1} A;
END BAttrB2;
```

Somit sind die qualifizierten Namen der Rollen (die sich aus dem Referenzattribut ergeben): `M.T.BAttrB2.B` und `M.T.BAttrB2.AttrB2` .

Wenn ein Namenskonflikt besteht (wie bei `AttrB3` im Beispiel), wird der Name um einen Index (beginnend bei 2 pro Tabelle) verlängert. `AttrB3` führt also zu:

```
ASSOCIATION B2AttrB3 =
  B2 -- {0..*} B;
  AttrB3 -- {1} A;
END B2AttrB3;
```

Somit sind die qualifizierten Namen: `M.T.B2AttrB3.B2` und `M.T.B2AttrB3.AttrB3` .

Die qualifizierten Rollennamen werden auch im Log aufgeführt. z.B.

```
Info: validate target of role ``M.T.BAttrB2.B``...
Info: validate multiplicity of role ``M.T.BAttrB2.B``...
```

Hinweise zu Fehlermeldungen

Intersection overlap

Die Fehlermeldung erscheint, wenn sich zwei Liniensegmente überlappen (also zwei Schnittpunkte haben):

Beispielmeldung:

```
Error: Model.Topic.Class: Intersection overlap 3.2508012350263016E-4, coord1 (2612419.901, 1248771.194), coord2 (2612428.5
```

Das Mass der Überlappung (`overlap 3.2508012350263016E-4`), die beiden Schnittpunkte (`coord1 (2612419.901, 1248771.194), coord2 (2612428.532, 1248767.551)`) und die TIDs/OIDs der betroffenen Objekte (`tids o1, o2`) werden aufgeführt.

Intersection

Die Fehlermeldung erscheint, wenn sich zwei Liniensegmente schneiden (also einen Schnittpunkte haben):

Beispielmeldung:

```
Error: Model.Topic.Class: Intersection coord1 (2612419.220, 1248771.482), tids o1/attrA[1]/flaeche[1], o2/attrA[2]/flaeche
```

Der Schnittpunkte (`coord1 (2612419.220, 1248771.482)`) und die TIDs/OIDs der betroffenen Objekte (`tids o1/attrA[1]/flaeche[1], o2/attrA[2]/flaeche[1]`) werden aufgeführt. In diesem Fall sind die Geometrien innerhalb von Strukturen, darum wird der ganze Pfad vom Objekt bis zur Geometrie aufgeführt (`o1/attrA[1]/flaeche[1]` : im Objekt `o1` das erste Strukturelement des Attributs `attrA` und darin das erste Element von `flaeche`)