DI-Extractor  
*Detaillierte Projektplanung und Programmbeschreibung*

**Inhaltsverzeichnis**

[Zusammenfassung 1](#_Toc484957211)

[1 Planung (Stand 2012) 2](#_Toc484957212)

[1.1 Konzept 2](#_Toc484957213)

[1.2 Auswahl SIP 2](#_Toc484957214)

[1.3 Metadata.xml -> xIsadg.xml 3](#_Toc484957215)

[1.4 xIsadg Vererbung und Aggregieren prüfen 3](#_Toc484957216)

[1.5 Vererbte und aggregierte Werte erzeugen 4](#_Toc484957217)

[2 Umsetzung als WebAPP (2013) 5](#_Toc484957218)

[2.1 Aufbau der WebAPP 5](#_Toc484957219)

[2.2 Aufbau der XSL-Transaktion 7](#_Toc484957220)

# Zusammenfassung

Der Schnittstelle zur *Descriptive Information* wurde innerhalb der KOST Rahmenspezifikation für die digitale Archivierung (bentō) eine grosse Bedeutung für die Standardisierung zugebilligt. Ziel der Spezifikation dieser Schnittstelle ist es, *Descriptive Information* aus dem Ingest-Prozess an ein Findmittel oder Archivinformationssystem (AIS) hersteller- und produkteunabhängig übergeben zu können.

Um die Einführung/Nutzung dieser Schnittstelle zu fördern, entwickelt die KOST einen *Descriptive Information Extraktor*, der aus einem abgelieferten BAR/bentō SIP die *Descriptive Information* extrahieren, gemäss der bentō-DI-Spezifi­kation, Version 1.6[[1]](#footnote-1) speichern und in einem zweiten Schritt an ein ausgewähltes AIS übergeben kann.

Dabei ist zu berücksichtigen, dass diese Schnittstelle nicht nur die I017-Metadaten des BAR/bentō SIPs abdeckt, sondern die gesamten ISAD(G) Metadaten. Es können also auch andere, nicht auf dem BAR/bentō SIP basierende Ingest Prozesse diese Schnittstelle nutzen.

# Planung (Stand 2012)

## Konzept

Die Lösung basiert auf einzelnen Programmmodulen, die von einem Command/Shell Script aufgerufen werden. Gemeinsame Datenbasis ist dabei die zu konvertierende Datei. Die Ergebnisse der Konvertierung und der Prüfungen werden in einer Log Datei festgehalten.

Der Vorteil dieser sehr losen Koppelung der einzelnen Schritte, *Extrahieren, Transformieren, XML-Validieren, Vererbung und Aggregieren Prüfen, Werte erzeugen und Findmittelimport*, ist der, dass für die einzelnen Module die jeweils adäquate technische Lösung eingesetzt werden kann und dass die Module auch problemlos in eine fremde Gesamtlösung integriert werden können.

**SIP metadata  
lesen/validieren**

**Transformation  
zu xIsadg**

**Vererbung prüfen**

**Aggregieren prüfen**

**explizite   
Werteweitergabe  
(Werte erzeugen)**

**implizite   
Werteweitergabe  
(Werte löschen)**

**Findmittel  
Importdatei  
erzeugen**

***Initialisierung***

***DI-Export***

*DI-Extractor: Programmstrukturplan(vereinfacht)*

## Auswahl SIP

GUI basierte Auswahl des SIPs.  
*(Umsetzung: NSIS, Nullsoft Scriptable Install System)*

### metadata.xml auslesen

### metadata.xml lesen und kopieren

Die Datei *metadata.xml* aus dem gewählten SIP oder SIP in gezippter Form auslesen und in das Verarbeitungsverzeichnis kopieren.

*(Umsetzung: unzip oder 7z)*

### metadata.xml validieren

Die ausgelesene Datei *metadata.xml* mit einem im DI-Extractor integrierten, bzw. verfügbaren Schema validieren.

*(Umsetzung: XSD und xmllint)*

## Metadata.xml -> xIsadg.xml

### XSLT Transformation

Mittels XSLT Transformation wird die BAR/bentō SIPDatei *metadata.xml* in eine *Descriptive Information* Datei vom Typ *xIsadg.xml* umgewandelt.

*(Umsetzung: XSLT und xalan)*

### xIsadg Schemavalidierung

Die erzeugte Datei *xIsadg.xml* mit dem *xIsadg.xsd* Schema validieren.

*(Umsetzung: xIsadg.xsd* und *xmllint)*

## xIsadg Vererbung und Aggregieren prüfen

Diese Prüfung kann nicht mit XML-Schema-Mitteln realisiert werden. Eine Lösung für diese Prüfung ist aber prozedural in Java oder mit Hilfe einer XSLT-Transformation möglich.

* Eine prozedurale Lösung parst den XML DOM Baum und sucht darin die notwendigen Werte für Vererbung oder Aggregieren.
* Eine Transformation liefert den Wert *true* oder *false*, je nach dem ob ein verlangter Wert gefunden wurde. Ein Script prüft prozedural die einzelnen Ergebnisse.

### Vererbung prüfen

Nach einem *top-down* zu vererbenden Wert suchen. Der Wert muss mindestens einmal gesetzt sein, kann aber tiefer in der Hierarchie überschrieben werden. Die Position des zu vererbenden Wertes in der Hierarchie der Verzeichnungseinheiten (VE), Root oder VE[[2]](#footnote-2), ist vom Wertetyp abhängig.

Eine automatische Prüfung beim Fall "Überschreiben von vererbten Werten" ist nur sehr bedingt möglich.[[3]](#footnote-3)

*(angestrebte Umsetzung: XSLT und xalan)*

### Aggregieren prüfen

Pro Element mit der Eigenschaft Aggregieren nach einem *bottom-up* zu generierenden Wert suchen. Die Position des zu aggregierenden Wertes in der Hierarchie, Blatt oder VE[[4]](#footnote-4), ist vom Wertetyp abhängig. In der aktuellen bentō-DI-Spezifikation 1.6, können alle Werte sowohl Blatt als auch ein VE Werte sein. Aggregierte VE Werte vererben sich ansonsten *top-down*.

*(angestrebte Umsetzung: XSLT und xalan)*

### Aggregierungsregeln prüfen

Sind bei Elementen, für die aggregieren gilt, auf verschiedenen hierarchischen Ebenen bereits Werte vorhanden, ist neben der Existenzprüfung auch die richtige Implementierung der Aggregierungsregeln zu prüfen[[5]](#footnote-5). Die Prüfung kann nur prozedural erfolgen.

*(mögliche Umsetzung: Java mit dom4j)*

## Vererbte und aggregierte Werte erzeugen

Vererbte und aggregierte Werte können gemäss der bentō-DI-Spezifikation 1.6 sowohl implizit als auch explizit übergeben werden. Implizit heisst hier, der Wert taucht nur einmal auf und wird erst beim Import ins AIS vererbt oder aggregiert. Explizit bedeutet, dass ein Wert für jedes Element in der Verzeichnungshierarchie generiert wird, das Element also quasi obligatorisch gesetzt wird.

Dafür werden zwei unterschiedliche Prozesse implementiert, die optional angewählt werden können:

* Für alle Elemente, für welche Vererbung oder Aggregieren gilt, wird der jeweils richtige Wert erzeugt.
* Elemente, für die Vererbung oder Aggregieren gilt, werden bereinigt, sodass die gleichen Werte, egal ob aggregiert oder vererbt, nicht mehrfach vorkommen.

Die explizite Wertegenerierung ist programmtechnisch einfacher realisierbar; beide Lösungen setzen aber eine prozedurale Programmierung voraus.

### Explizite Wertegenerierung

Die explizite Wertegenerierung kann notwendig sein, wenn der anschliessende Übernahmeprozess in ein Findmittel die implizit geltenden Werte nicht generieren kann. Sind alle Werte explizit gesetzt, kann die Korrektheit der Daten auch in diesem Bereich mit XML Schema geprüft werden. Es entsteht aber eine unschöne Datenredundanz auf jeder hierarchischen Verzeichnungsebene.

*(Mögliche Umsetzung: Java mit dom4j)*

### Implizite Werteweitergabe

Werden explizite Werte entfernt, müssen speziell die Aggregierungsregeln genau beachtet, bzw. implementiert sein, damit keine Information unwiederbringlich verschwindet. Es wird bei diesem Prozess aber unschöne und unnötige Datenredundanz entfernt.

*(Mögliche Umsetzung: Java mit dom4j)*

### ~~xIsadg -> Findmittelimport~~

Als Findmittel Importfile wird nur noch *xIsadg unterstützt*

~~Optional: Umwandlung von~~ *~~Descriptive Information~~* ~~Daten in~~ *~~xIsadg~~* ~~Format in Importformate bekannter Findmittel/Archivinformationssysteme. Das Zielformat der Transformation ist wählbar (scopeArchiv, CMISTAR, Augias etc.)~~

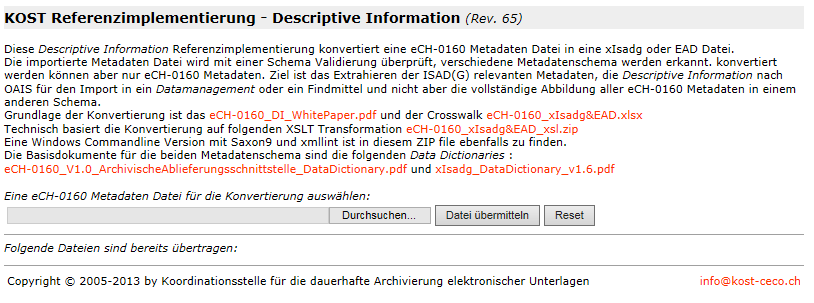
### ~~xIsadg -> scope Übernahme File~~

~~Metadatentransformation mit XSLT von xIsadg zu scope Übernahme Assistent Format.~~

*~~(Umsetzung: XSLT und xalan)~~*

# Umsetzung als WebAPP (2013)

Die Umsetzung erfolgt 2013 als PHP basierte WebAPP mit einer zugrundeliegenden XSL Transformation ohne aggregieren oder vererben:

  
[*http://kost-ceco.ch/isadg/app/*](http://kost-ceco.ch/isadg/app/)

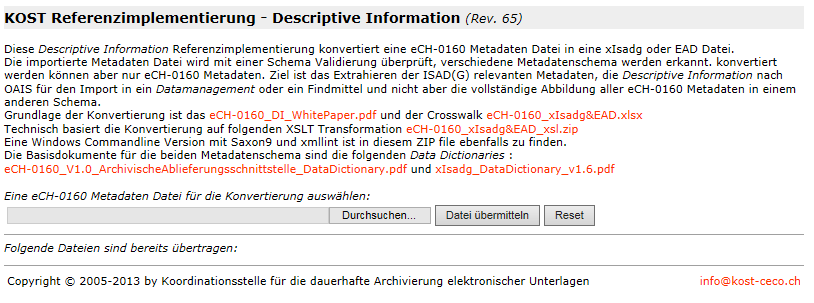
## Aufbau der WebAPP

***ingest.css***

steuert das Layout des Webformulars

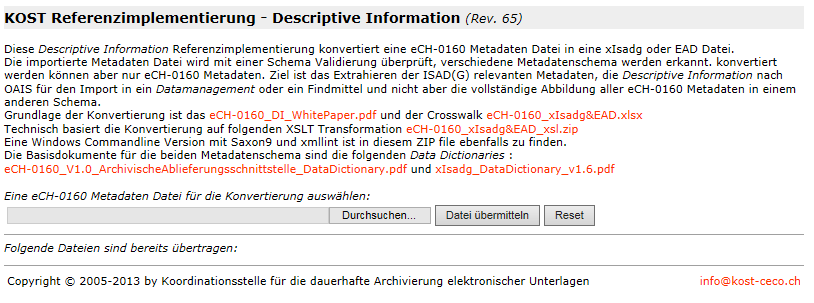
***helptext.php***

Enthält die einführenden Erläuterungen



***copyright.php***

Enthält die Copyright Zeile



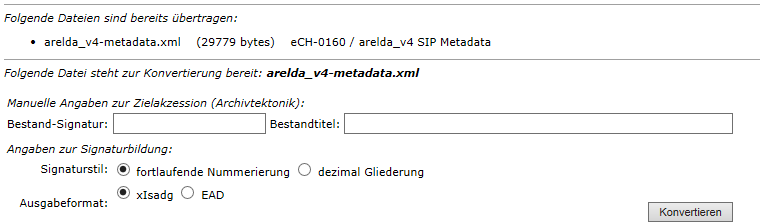
* ***index.php* = *input.php***
* Generiert eine eindeutige Benutzerkennung für eine *Session*
* Legt ein temporäres Benutzerverzeichnis für eine Konvertierung an
* Schreibt Benutzerkennung und *Timestamp* für diese Transaktion in $wdir/usr.txt
* Leitet zum Webformular *ingest.php* weiter

***ingest.php***

* Ist kein Benutzererkennung wird *input.php* aufgerufen  
  if ($usr=='') { header ("location: ./input.php"); }
* Arbeitsverzeichnis setzen und Inhalt löschen
* Formular zum Hochladen einer Metadaten Dateie generieren  
  <form enctype="multipart/form-data" action="ingest.php" method="post">
* Hochgeladene Datei ins Arbeitsverzeichnis kopieren
* Hochgeladene und mit verschiedenen Schema validieren Datei in Liste anzeigen [arelda\_v4, arelda\_v3.13.2, xIsadg\_v1.6.1, ead]
* Erkannter Schematyp in der Liste anzeigen und im Falle von Typ arelda\_v4 die Weiterverarbeitung starten -> *maninput.php*

***maninput.php***

* Notwendige manuelle Dateneingabe (Bestand-Signatur / Bestandtitel)
* Zielformat xIsadg oder EAD
* Art der Signaturbildung (fortlaufend 1,2,3… / dezimal 1, 1.1, 1.1.1, 1.2 …)



* Aufruf der Tranformationsfunktion *processingest.php*  
  <form action="processingest.php" method="post">

***processingest.php***

* Variablen für die Transformation setzen: *$collsig, $colltitle, $collstyle, $xschema*
* Referenzdatei für die Signaturnummerierung erstellen (numlist oder null.xml)
* XSLT Konvertierung konfigurieren:  
   $xsl->load('eCH2xIsadg.xsl');  
   $xsl->load('eCH2EAD.xsl');
* XSL Transformation ausführen  
  $xisadg = $proc->transformToXML($xml);
* Alle Dateien im Arbeitsverzeichnis löschen
* Ergebnis als XML Datei ausgeben  
  header('Content-type: application/xml');  
  echo $xisadg;

***class.inputfilter.php***

* Hilfsfunktionen für den sicheren Import von Formulareingaben  
  require\_once("class.inputfilter.php");

## Aufbau der XSL-Transaktion

Die Transformation zu xIsadg startet *mit eCH2xIsadg.xsl*

Die Transformation zu EAD startet mit *eCH2EAD.xsl*

***eCH2xIsadg.xsl***

übernimmt die Parameter *fondtitel, archsig*, und *reffilename*:

<xsl:param name="fondtitle"/>

<xsl:param name="archsig"/>

<xsl:param name="reffilename"/>

<xsl:variable name="reffile" select="document($reffilename)"/>

Importiert die Hilfsfunktionen als named template *xIdate, xIaccess* und *xIreference* :  
 <xsl:include href="xIdate.xsl"/>

<xsl:include href="xIaccess.xsl"/>

<xsl:include href="xIreference.xsl"/>

Weiter die Rekursiven Teiltransformationen:

<!-- Ablieferung - Provenienz -->

<xsl:include href="xIfond.xsl"/>

<!-- Ordnungsystemposition -->

<xsl:include href="xIserie.xsl"/>

<!-- Dossier -->

<xsl:include href="xIfile.xsl"/>

<!-- Dokument -->

<xsl:include href="xIitem.xsl"/>

Der erste Knoten *archivalDescription* wird angelegt:

<xsl:element name="archivalDescription">

<xsl:attribute name="xsi:schemaLocation"><xsl:value-of   
select="$location"/></xsl:attribute>

<xsl:apply-templates select="arelda:ablieferung"/>

<xsl:apply-templates   
select="arelda:ablieferung/arelda:ordnungssystem"/>

</xsl:element>

***xIserie.xsl***

Anschliessend wird in *xIserie.xsl* rekursiv über *arelda:ordnungssystem* itteriert:

<!-- Ordnungsystemposition -->

<xsl:template match="arelda:ordnungssystemposition">

Oder ein Dossier angelegt

<xsl:apply-templates select="arelda:dossier">

<xsl:with-param name="sig" select="$signature"/>

</xsl:apply-templates>

***xIfile.xsl***

Ist für das Anlegen von Dossier zuständig (file engl. = Dossier)

<!-- Ordnungsystemposition -->

<xsl:template match="arelda:dossier">

***xIitem.xsl***

Modelliert ein Dokument:

<!-- Ordnungsystemposition - type GEVER-->

<xsl:template match="arelda:dokument">

Beziehungsweise eine Datei:

<!-- Ordnungsystemposition - type FILE -->

<xsl:template match="arelda:dateiRef">

***xIdate.xsl***

Generiert auf allen Stufen das Element *isad:dates*  
*[ xIfunction.xsl* ist eine Variante von *xIdate.xsl ]*

***xIaccess.xsl***

Generiert auf allen Stufen das Element *isad:accessConditions*

***xIreference.xsl***

Generiert auf allen Stufen die Signatur, bzw. das Element *isad:referenceCode*

<xsl:element name="referenceCode">

Für die *fortlaufende* Nummerierung wird vorgängig eine Liste aller Objektreferenzen erzeugt und diese anschliessend nummeriert:

1. Die Transaktion *createRef.xsl* erzeugt *createRef.xml*

<reference>

<identity>

<referenceCode>sig.1</referenceCode>

<depth>0</depth>

</identity>

<identity>

<title>Erziehung, Bildung, Kultur</title>

<referenceCode>sig.1.1</referenceCode>

<depth>1</depth>

1. Die Transaktion *elementRef.xsl* nummeriert die Objekte fortlaufend und schreibt *numberRef.xml [ numberRef.xsl* ist eine Variante von *elementRef.xsl ]*

<reference>

<\_sig.1>

<referenceNo>1</referenceNo>

<depth>0</depth>

</\_sig.1>

<\_sig.1.1>

<referenceNo>2</referenceNo>

<depth>1</depth>

</\_sig.1.1>

Bei *rekursiver* Nummerierung wird diese Liste der Objektreferenzen durch ein Null File *null.xml* ersetzt und die Signatur fortlaufend rekursiv generiert (sig.1, sig.1.1, etc.)

Anschliessend wird die Datei *elementRef.*xsl in der Funktion xIreference gelesen und nach einem Nummerneintrag *[<referenceNo>2</referenceNo>]* zu einem entsprechende *[referenceCode <referenceCode>sig.1.1</referenceCode>]* gesucht:

<!-- apply template using select on reference file to find number for signature -->

<xsl:apply-templates select="$reffile//reference/\*[name()=$sig]" mode="refNo"/>

<!-- -->

<xsl:template match="\*" mode="refNo">

<xsl:value-of select="referenceNo"/>

</xsl:template>

***eCH2EAD.xsl***

Die Transformation *eCH2EAD.xsl* folgt in Grundzügen der Transformation *eCH2xIsadg.xsl.*

1. Descriptive Information, Data Dictionary & XML Schema Beschreibung, Version 1.6 vom 23.05.2011. <http://kost-ceco.ch/cms/index.php?bento_de>, dort xIsadg\_DataDictionary\_v1.6.pdf [↑](#footnote-ref-1)
2. *1.1 Signatur* ist auf jeder Ebene denkbar, *2.4 Abgebende* Stelle hingegen ist ein Root Element. [↑](#footnote-ref-2)
3. So kann sowohl *2.1 Name der Provenienzstelle* als auch *2.4 Abgebende Stelle* innerhalb einer Ablieferung unterschiedlich sein [↑](#footnote-ref-3)
4. *1.3 Entstehungszeitraum/Laufzeit* kann sowohl ein Blatt Wert wie auch auf Ebene Verzeichnungseinheit gesetzt sein, wenn das in der Regel auch eher unwahrscheinlich ist. [↑](#footnote-ref-4)
5. Beispiel: *1.3 Entstehungszeitraum/Laufzeit* kann auf einer höheren VE Ebene nur einen grösseren Zeitraum als die aggregierten, darunter liegenden umfassen, nicht aber einen kleineren. [↑](#footnote-ref-5)