UNIVERSITÄT ZU LÜBECK

Information Systems

CS4130-KP06

Prof. Dr. Sylvia Melzer

SoSe2026

From Research to Structured Data with Python

Information Systems

What You Will Learn

- How structured research data can be transformed into analyzable formats using Python-based workflows
- To understand the concept of Databasing on Demand (DBoD) and how it enables the dynamic creation of data-driven information systems
- How XML-based data such as EpiDoc can be transformed into tabular formats like CSV using XSLT and Python, and how these transformations shape the resulting data model
- How such pipelines turn archived data into explorable information that can be searched, analyzed, and interpreted
- To reflect on the role of transformation as a methodological step, and how data structures influence analysis and knowledge production

Scenario: Designing an Information System

- In this lecture, we consider a typical scenario in digital humanities research
- A group of researchers works with heterogeneous data such as EpiDoc and TEI
- The data is stored in repositories, but not directly usable for analysis
- Researchers need a system that allows them to search, filter, and explore the data
- The challenge is to transform encoded data into structured and analyzable formats
- This lecture shows how such a system can be designed using transformation pipelines

EpiDoc Viewer

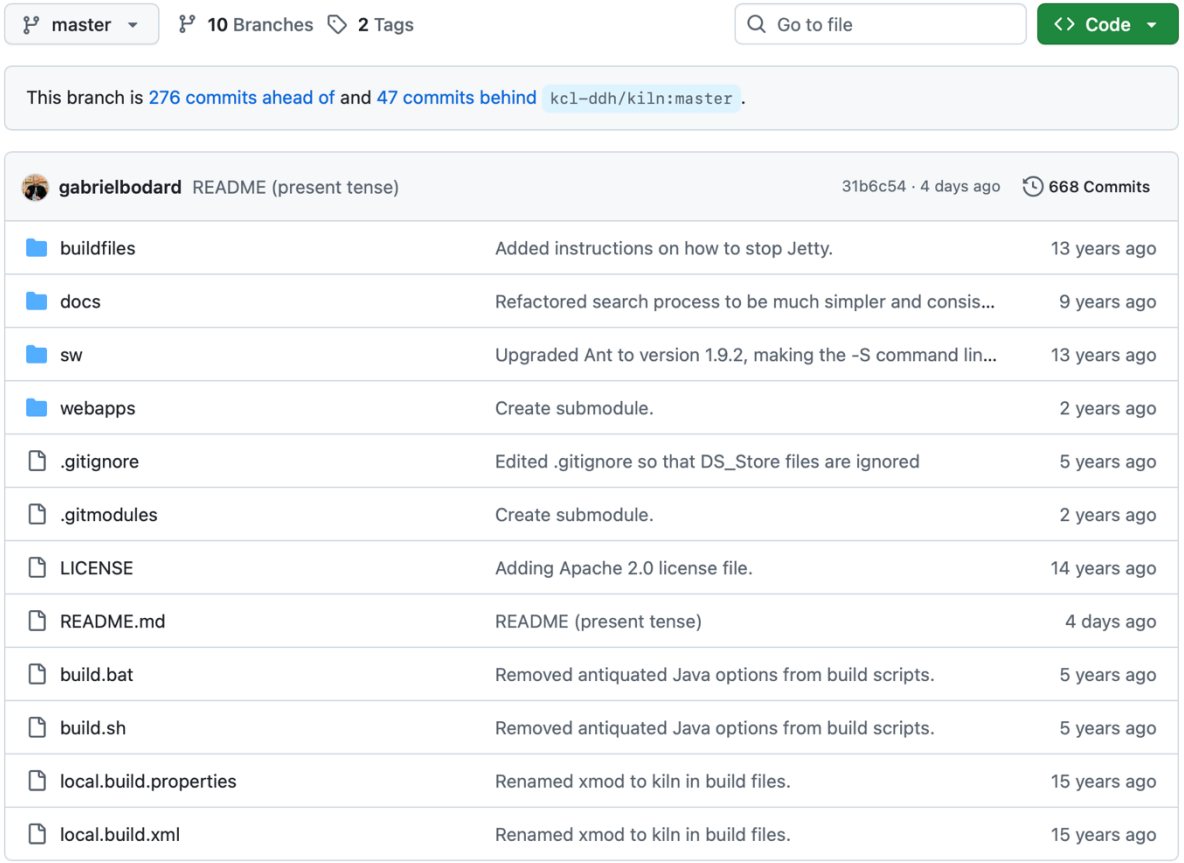
Core EpiDoc Components

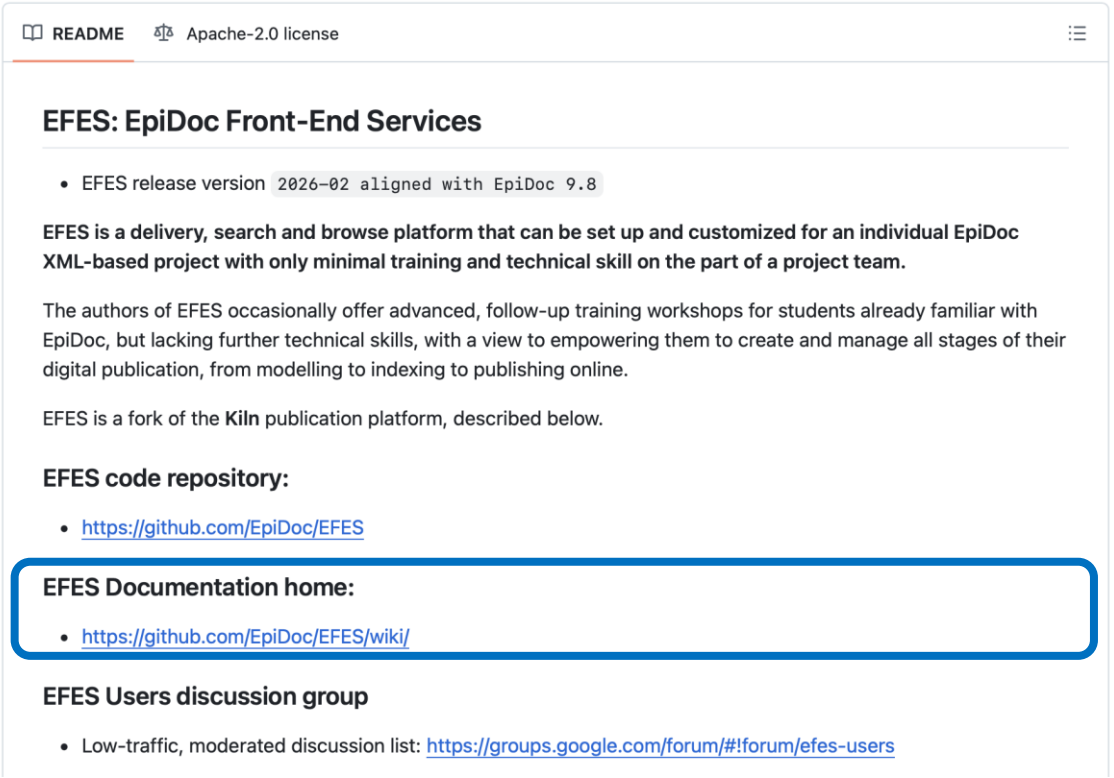
- EpiDoc Guidelines: recommendations for markup of transcription, descriptive features, etc. (About)
- EpiDoc Schema: the TEI-derived schema against which EpiDoc XML should be validated
- EpiDoc Reference Stylesheets: for transforming EpiDoc XML to HTML, text or ODF
- EFES: a low-barrier EpiDoc publication platform

https://epidoc.stoa.org

EFES (EpiDoc Front End Services)

- https://github.com/EpiDoc/EFES





EFES

Home

Gabriel Bodard edited this page on Apr 8, 2024 · 36 revisions

EpiDoc Front-End Services (EFES) is a free, easy to use, highly customisable platform for the online publication of ancient texts in EpiDoc XML, funded by the Andrew W. Mellon Foundation in 2017–18. EFES allows the creation of multiple indices, search and browse interface, geographical visualisation, and integration with linked open data.

About the project and team

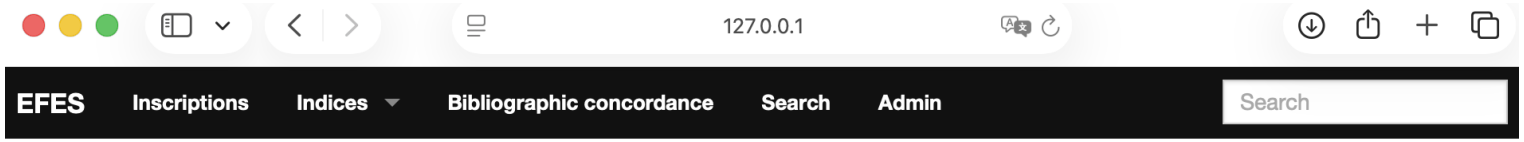
- If you want to download EFES and use it for your project, we recommend you read the **User Guide** (quickstart and detailed tutorial).
- If you're developping on EFES, supporting or hosting an EFES project, take a look at the **Technical Documentation**.

EFES

Quickstart

1. Download or clone the EFES code from the GitHub repository.
2. Open a terminal window/command line and go to the directory where EFES is installed.
3. Run the command build.sh (Mac OS X/Linux) or build.bat (Windows), and leave the Terminal window open.
4. Open a browser and go to http://127.0.0.1:9999/. It should display a welcome page together with some very basic navigation.
5. Store project XML content (EpiDoc files) in the folder **/webapps/ROOT/content/xml/epidoc/**.
6. Index newly added files through the Admin panel on the site.
7. View HTML versions of your XML files at http://127.0.0.1:9999/en/inscriptions/.

EFES



Welcome to EFES!

Now that you have EFES up and running, it's time to start building your project. Start with one of the options listed below and go from there.

Add your files

Put your EpiDoc XML files in webapps/ROOT/content/xml/epidoc and go to the admin and index them for searching. Then see how they are displayed (Inscriptions).

Customise the templates

Change the look of this site by modifying the templates in webapps/ROOT/assets/templates. The template for this page is home.xml, and it builds on the base template base.xml.

Read the documentation

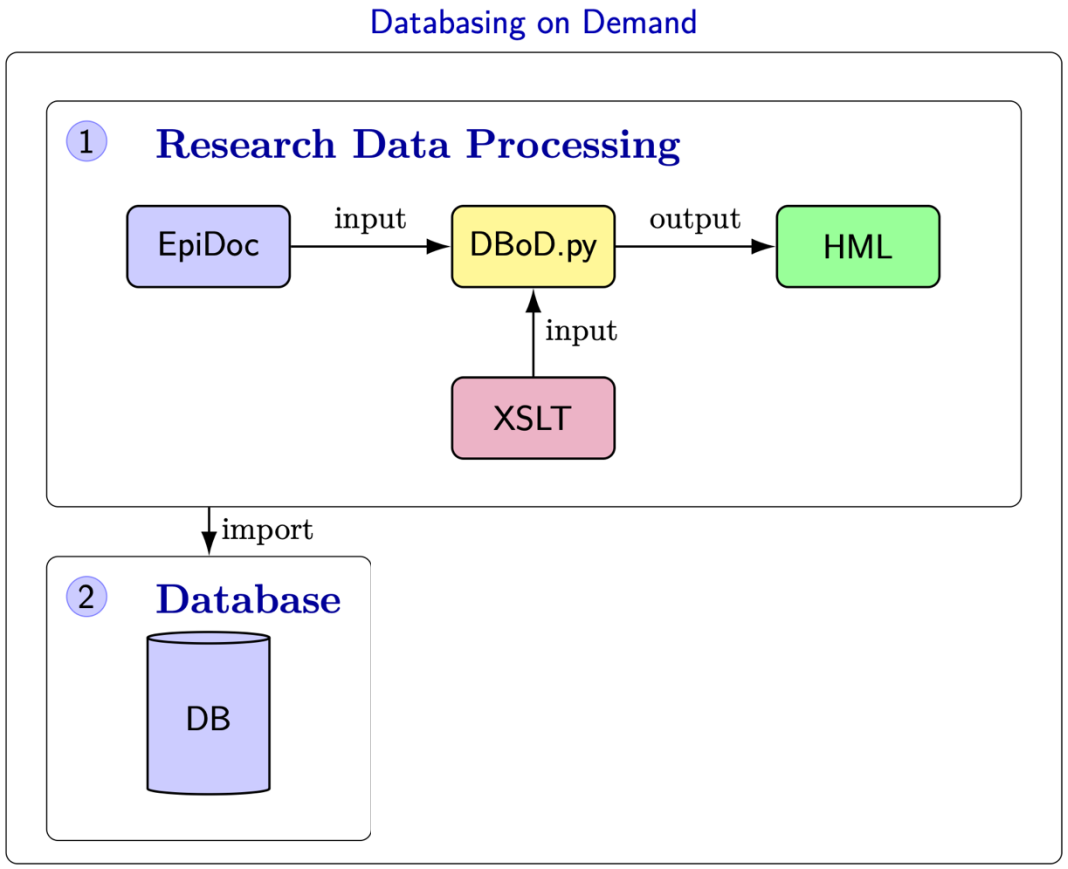
The documentation and User Guide for EFES can be found on the project's GitHub Wiki pages. Documentation for Kiln, the platform that EFES is based on, can be found both online, and in source form, in the docs directory of your installation. It explains how you can modify everything about your project.

What are the advantages/ disadvantages of EFES?

From Data to Database (Project: EDAK)

Databasing on demand (DBoD)

- (1) multiple EpiDoc files are transformed into a single HML file via the DBoD.py script
- and XSLT stylesheets following the Leiden Conventions*
- (2) the HML file is imported into a Heurist database instance



*Leiden Conventions: https://www.epigraphik.uni-hamburg.de/content/misc/diacritics.xml

From EpiDoc to an Information System

EpiDoc.XML

Encoded, hierarchical research data

↓

XSLT

Extraction and Transformation

↓

Python

Processing (aggregation and orchestration)

↓

CSV / HML

new data model (tabular dataset / harmonized XML dataset)

↓

Database (e.g., Heurist)

Data storage and querying

↓

Viewer / Exploration Interface

Exploration and analysis

Leiden Conventions

Diacritics

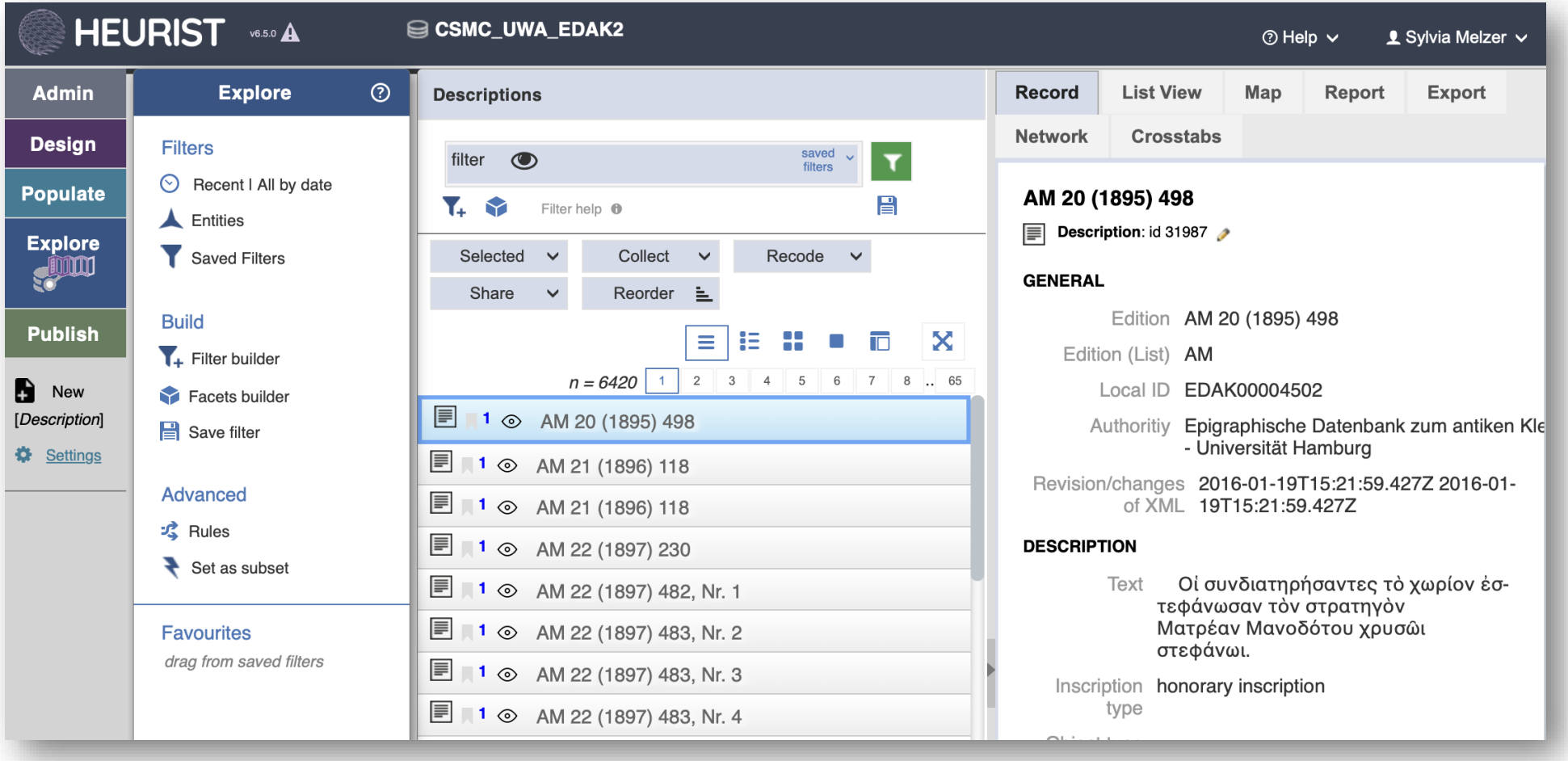
The diacritics are taken from the relevant edition of the texts and therefore are not coherent in every case. They all follow the Leiden Conventions.

Sign	Explanation
[αβ]	Letters restored by the editor
[—]	Missing letters of unknown number, not restorable
{α}	Letters considered superfluous by the editor
<α>	Letters omitted erroneously by the text and restored by the editor
(αβ)	Abbreviation in the text, expanded by the editor
[[αβ]]	Letters or spaces deliberately erased in antiquity
[[—]]	Unknown number of letters erased
α̣	Letters of which sufficient traces remain on the stone but not enough to exclude other possible readings
α̲β̲	Letters seen by an earlier editor but no longer visible on the stone
α͡β	Ligature
(v.) / v. (vv.) / v.v. (vvv.) / v.v.v.	Latin *vacat*; one, two or three uninscribed letter-space(s)
(vac.) vac. vacat	Latin *vacat*; uninscribed space
(sic)	Indicates grammatical or orthographical particularities

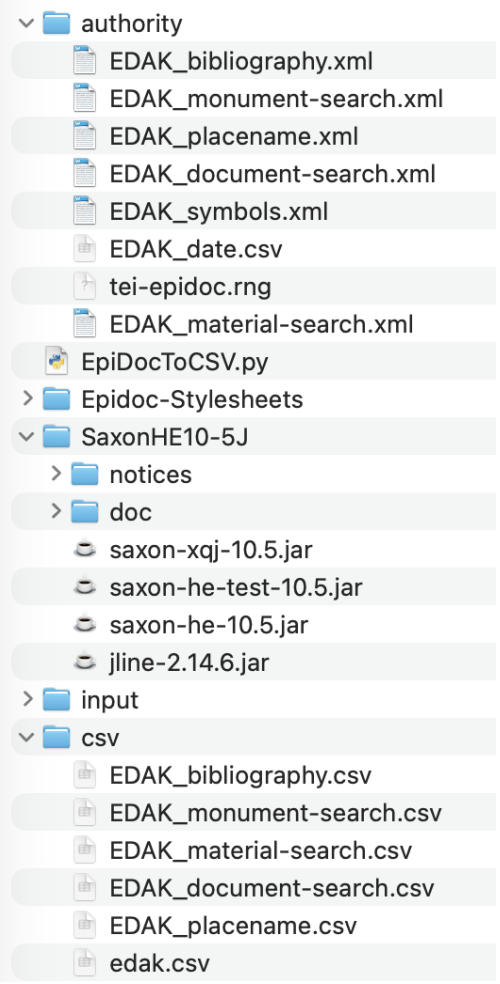
Conversion: Leiden Conventions to XML

https://www.youtube.com/watch?v=Bcr0a-obD8I

From Data to Database (Project: EDAK)



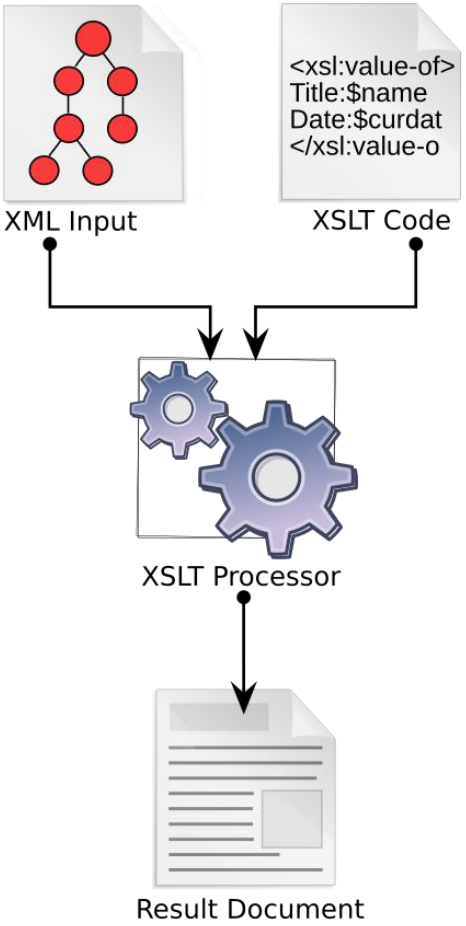
From Data to Database



- **authority** contains the data required across all documents
- **Epidoc-Stylesheets** are a collection of XSLT 2.0 scripts designed to transform EpiDoc-encoded XML documents into readable formats like HTML and plain tex
- **SaxonHE10-5J** refers to the Java Edition 10.5 of Saxon-HE (Home Edition), a popular open-source processor for XSLT, XQuery, and Xpath. This version enables the transformation of XML documents and is often available as a JAR file
- **input** contains the EpiDoc files which are used as input data
- **CSV** contains the output files for further usage

XSL, XSLT, XQuery, XPath

- **XSL** (eXtensible Stylesheet Language) is a styling language for XML
- **XSLT** (XSL Transformations): Transformation for example from EpiDoc to HTML
  https://en.wikipedia.org/wiki/XSLT
- **XQuery** (XML Query) is a query language and functional programming language designed to query and transform collections of structured and unstructured data, primarily in the form of XML
  https://en.wikipedia.org/wiki/XQuery
- **XPath** (XML Path Language) is an expression language designed to support the query or transformation of XML documents
  https://en.wikipedia.org/wiki/XPath



Dreftymac at English Wikipedia

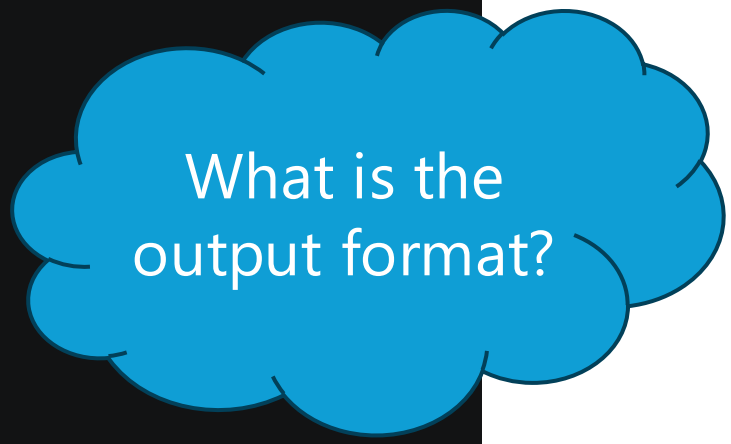
Example: XML→XSLT→XML

```xml
<?xml version="1.0" ?>
<persons>
  <person username="JS1">
    <name>John</name>
    <family-name>Smith</family-name>
  </person>
  <person username="MI1">
    <name>Morka</name>
    <family-name>Ismincius</family-name>
  </person>
</persons>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="xml" indent="yes"/>

  <xsl:template match="/persons">
    <root>
      <xsl:apply-templates select="person"/>
    </root>
  </xsl:template>

  <xsl:template match="person">
    <name username="{@username}">
      <xsl:value-of select="name" />
    </name>
  </xsl:template>

</xsl:stylesheet>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <name username="JS1">John</name>
  <name username="MI1">Morka</name>
</root>
```

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- $Id$ -->
3  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4      xmlns:t="http://www.tei-c.org/ns/1.0"
5     xmlns:xs="http://www.w3.org/2001/XMLSchema"
6     exclude-result-prefixes="t" version="2.0">
7     <xsl:output method="html" encoding="UTF-8" indent="no"/>
8     <!--xsl:output method="html" encoding="UTF-8"/-->
9
10    <xsl:include href="global-varsandparams.xsl"/>
11
12    <!-- html related stylesheets, these may import tei{element} stylesheets if
13    <xsl:include href="htm-teiab.xsl"/>
14    <xsl:include href="htm-teiaddanddel.xsl"/>
15    <xsl:include href="htm-teiapp.xsl"/>
16    <xsl:include href="htm-teidiv.xsl"/>
17    <xsl:include href="htm-teidivedition.xsl"/>
18    <xsl:include href="htm-teidivapparatus.xsl"/>
19    <xsl:include href="htm-teiforeign.xsl"/>
20    <xsl:include href="htm-teifigure.xsl"/>
21    <xsl:include href="htm-teig.xsl"/>
22    <xsl:include href="htm-teigap.xsl"/>
23    <xsl:include href="htm-teihead.xsl"/>
24    <xsl:include href="htm-teihi.xsl"/>
25    <xsl:include href="htm-teilb.xsl"/>
```



```
81      <!-- HTML FILE -->
82      <xsl:template match="/">
83    <!--
84    Title| Object_type| Object_description| Date| Region| Place| PlaceInPeripleo| Findspot| Location_today| Edition
85    -->
86        <xsl:choose>
87          <xsl:when test="$edn-structure = 'edak'">
88                  <xsl:call-template name="edak-structure">
89                      <xsl:with-param name="parm-internal-app-style" select="$internal-app-style" tunnel="yes"/>
90                      <xsl:with-param name="parm-external-app-style" select="$external-app-style" tunnel="yes"/>
91                      <xsl:with-param name="parm-edn-structure" select="$edn-structure" tunnel="yes"/>
92                      <xsl:with-param name="parm-edition-type" select="$edition-type" tunnel="yes"/>
93                      <xsl:with-param name="parm-hgv-gloss" select="$hgv-gloss" tunnel="yes"/>
94                      <xsl:with-param name="parm-leiden-style" select="$leiden-style" tunnel="yes"/>
95                      <xsl:with-param name="parm-line-inc" select="$line-inc" tunnel="yes" as="xs:double"/>
96                      <xsl:with-param name="parm-verse-lines" select="$verse-lines" tunnel="yes"/>
97                      <xsl:with-param name="parm-css-loc" select="$css-loc" tunnel="yes"/>
98              <!--<xsl:with-param name="parm-bib" select="$bibliography" tunnel="yes"/>--><!--bib-short-->
99              <!--<xsl:with-param name="parm-bibloc" select="$localbibl" tunnel="yes"/>-->
100               </xsl:call-template>
```

htm-tpl-struct-epidoc_edak.xsl U

Users > smelzer > Desktop > DBoD-EDAK > Epidoc-Stylesheets > htm-tpl-struct-epidoc_edak.xsl

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- $Id: htm-tpl-struct-edak.xsl 2561 2017-04-04 11:24:24Z gabrielbodard $ -->
3  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
4                  xmlns:t="http://www.tei-c.org/ns/1.0" exclude-result-prefixes="t"
5                  version="2.0">
6     <!-- Contains named templates for EDAK file structure -->
7
8     <xsl:template name="edak-body-structure">
9
10 <!--Inscription type (Category)-->
11 <xsl:if test="//t:profileDesc//t:textClass/t:catRef/@target">
12    <xsl:value-of select="substring-after(//t:profileDesc//t:textClass/t:catRef/@target, '#')"/
13 </xsl:if>
14 <xsl:value-of select="'|'"/>
15
16 <!--Object_type-->
17 <xsl:if test="//t:support/t:objectType/@ref">
18    <xsl:value-of select="substring-after(//t:support/t:objectType/@ref, '#')"/>
19 </xsl:if>
20 <xsl:value-of select="'|'"/>
```

```
142    <!-- Edited text output -->
143    <xsl:variable name="edtxt">
144       <xsl:apply-templates select="//t:div[@type='edition']"/>
145    </xsl:variable>
146    <xsl:apply-templates select="$edtxt" mode="sqbrackets"/>
147    <xsl:value-of select="'|'"/>
148
149    <!-- Apparatus text output -->
150    <div>
151       <xsl:variable name="apptxt">
152          <xsl:apply-templates select="//t:div[@type='apparatus'
153       </xsl:variable>
154       <xsl:apply-templates select="$apptxt" mode="sqbrackets"/>
155    </div>
156    <xsl:value-of select="'|'"/>
```

DBoD: calling EpiDoc-Stylesheets

```python
import subprocess
from lxml import etree


def transform(input, output):
    xslt = "./Epidoc-Stylesheets/start-edition_epidoc.xsl"

    subprocess.call([
        "java",
        "-jar",
        "./SaxonHE10-5J/saxon-he-10.5.jar",
        f"-s:{input}",
        f"-xsl:{xslt}",
        f"-o:{output}"
    ])
```

Why Python?

- XSLT extracts structure
- Python orchestrates workflows
- Python aggregates multiple files
- Python enables automation and scalability
- Python validates the structure

DBoD: validating the EpiDoc files

```python
def validate_epidoc(xml_file_path, rng_file_path, fail_folder="fail_schema"):
    """…
    try:
        # Create the fail_folder if it doesn't already exist
        if not os.path.exists(fail_folder):
            os.makedirs(fail_folder)

        # Load the Relax NG schema
        with open(rng_file_path, 'r') as rng_file:
            relaxng_doc = etree.parse(rng_file)
            relaxng = etree.RelaxNG(relaxng_doc)

        # Load the EPIDOC XML file
        with open(xml_file_path, 'r') as xml_file:
            xml_doc = etree.parse(xml_file)

        # Validate the XML file
        is_valid = relaxng.validate(xml_doc)

        if is_valid:
            print(f"The file '{xml_file_path}' is valid.")
            return True
        else:
            print(f"The file '{xml_file_path}' is not valid.")
```

DBoD: start

```python
if __name__ == "__main__":

    print('----------------------------------------')
    print('Databasing on Demand: EpiDoc-to-CSV')
    print('Version: ' + str(_version) + '                ')
    print('----------------------------------------')

    # Überprüfe und verschiebe fehlerhafte XML-Dateien
    result = check_and_move_invalid_xml(_path, _fail_path)

    if result == False:
        result = check_xml_declaration(_path_authority)

    if result == False:
        data = parse_xml_to_csv_placenames(_path_authority+"EDAK_placename.xml", "csv/EDAK_placename.csv")
        print("Created: csv/EDAK_placename.csv")

        parse_xml_to_csv_edak(_path_authority+"EDAK_document-search.xml","csv/EDAK_document-search.csv")
        print("Created: csv/EDAK_document-search.csv")

        parse_material_xml_to_csv(_path_authority+"EDAK_material-search.xml","csv/EDAK_material-search.csv")
        print("Created: csv/EDAK_material-search.csv")

        parse_monument_xml_to_csv(_path_authority+"EDAK_monument-search.xml","csv/EDAK_monument-search.csv")
        print("Created: csv/EDAK_monument-search.csv")

        data_biblList = parse_bibliography_xml_to_csv(_path_authority+"EDAK_bibliography.xml","csv/EDAK_bibliog
        print("Created: csv/EDAK_bibliography.csv")
```

DBoD: delimiter |

```python
def check_and_move_invalid_xml(directory, fail_directory):
    missing_declaration_found = False
    # Erstelle den Ordner "fail", falls er noch nicht existiert
    if not os.path.exists(fail_directory):
        os.makedirs(fail_directory)

    # Durchlaufe alle Dateien im angegebenen Ordner
    for filename in os.listdir(directory):
        # Überprüfen, ob die Datei eine XML-Datei ist
        if filename.endswith('.xml'):
            filepath = os.path.join(directory, filename)
            # Öffne die Datei im "with"-Block, damit sie nach dem Lesen automatisch geschlossen wird
            with open(filepath, 'r', encoding='utf-8') as file:
                # Dateiinhalt lesen
                content = file.read()
            # Überprüfen, ob das "|" Zeichen vorhanden ist
            if '|' in content:
                missing_declaration_found = True
                # Wenn das "|" Zeichen gefunden wird, verschiebe die Datei in den fail-Ordner
                new_filepath = os.path.join(fail_directory, filename)
                try:
                    shutil.move(filepath, new_filepath)
                    print(f'{filename} enthält ein "|" Zeichen und wurde in den Ordner "{fail_directory}"
                except PermissionError as e:
                    print(f"Fehler beim Verschieben von {filename}: {e}")
    return missing_declaration_found
```

DBoD: output files

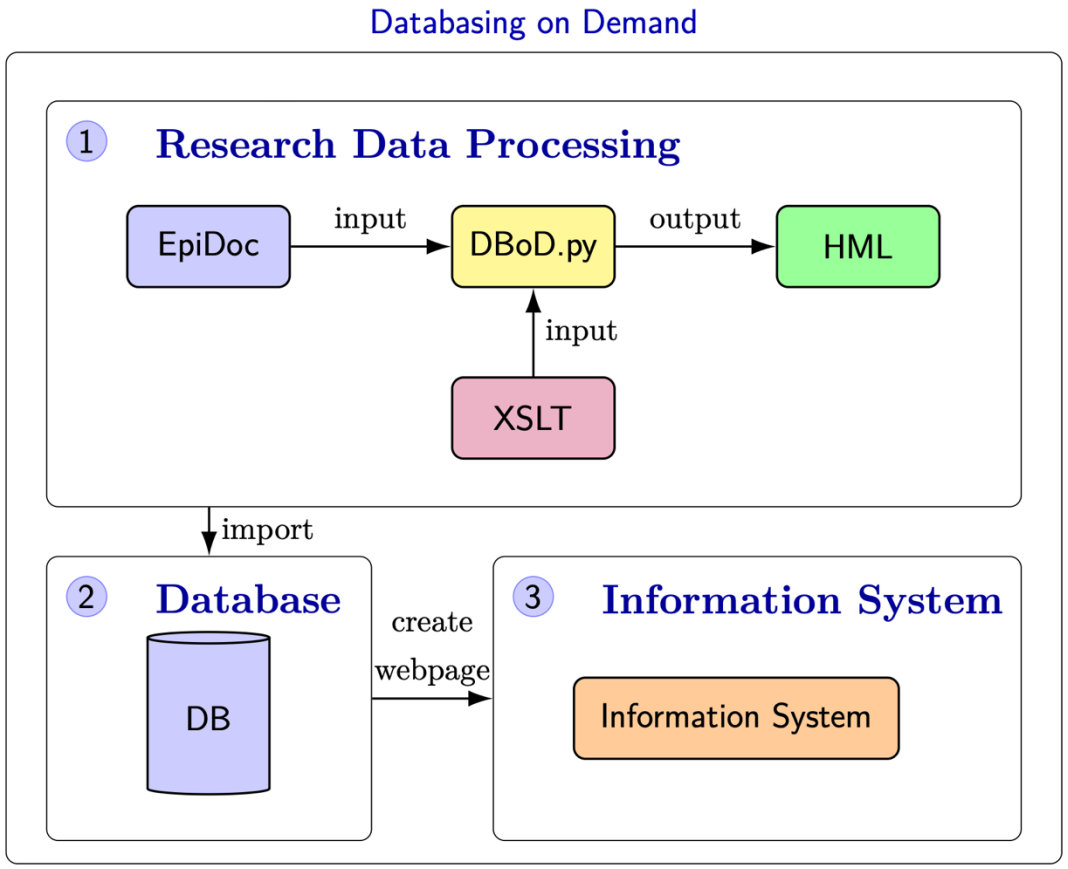
```
headers = ['ID', 'placename_de', 'placename_en', 'reference', 'location']
write_csv('csv/region.csv', headers, region_data)
write_csv('csv/settlement.csv', headers, settlement_data)
write_csv('csv/findspot.csv', headers, findspot_data)

# Create output file (CSV)
output_csv_file = "csv\\edak.csv"
column_names = ["Edition", "Edition (List)", "Inscription Type", "Object Type", "Object Description
with open(output_csv_file, 'w', newline='', encoding='utf-8') as csvfile:
    csvwriter = csv.writer(csvfile, delimiter=',')
    # Write column names
    csvwriter.writerow(column_names)
    # Write data
    for row in all_data:
        csvwriter.writerow(row)
```

DBoD: From EpiDoc to CSV

- The transformation defines a new data model
- Data from different parts of the XML document is combined into this single row
- Each row in the CSV represents one analytical unit (e.g., inscription, place, date)
- This model enables comparison across multiple inscriptions

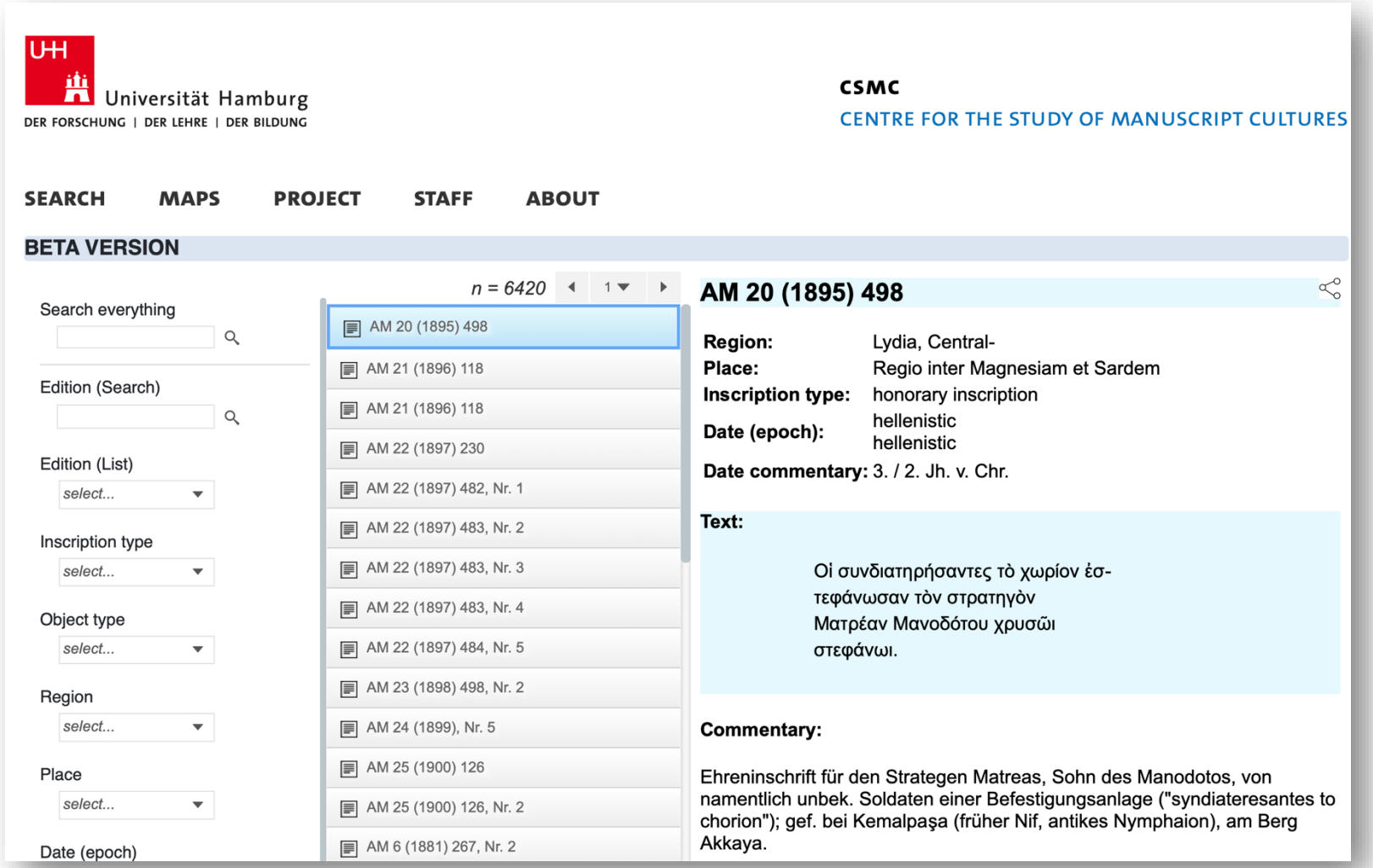
From Data to Information System

- (1) multiple EpiDoc files are transformed into a single HML file via the DBoD.py script
- and XSLT stylesheets following the Leiden Conventions*
- (2) the HML file is imported into a Heurist database instance
- an information system is built on top of the database



*Leiden conventions: https://www.epigraphik.uni-hamburg.de/content/misc/diacritics.xml

From Data to Information System



CSV vs HML: Two Transformation Outputs

CSV: Tabular Data for Analysis

- CSV represents data in a flat, tabular structure where each row corresponds to one analytical unit such as an inscription
- Data from different parts of the XML is combined into a single row, enabling direct comparison across multiple records
- This format is optimized for querying, filtering, and statistical analysis
- CSV is widely supported by analytical tools such as Python, Excel, and databases
- It simplifies complex hierarchical data into a structure that is easy to process computationally

HML: Structured XML for Database Integration

- HML remains an XML-based format but restructures and harmonizes multiple EpiDoc files into a unified representation
- It preserves structural relationships while standardizing elements across documents
- This format is designed for import into the Heurist database system
- HML serves as an intermediate step between raw XML and a fully functional information system
- It supports system integration rather than direct analytical processing

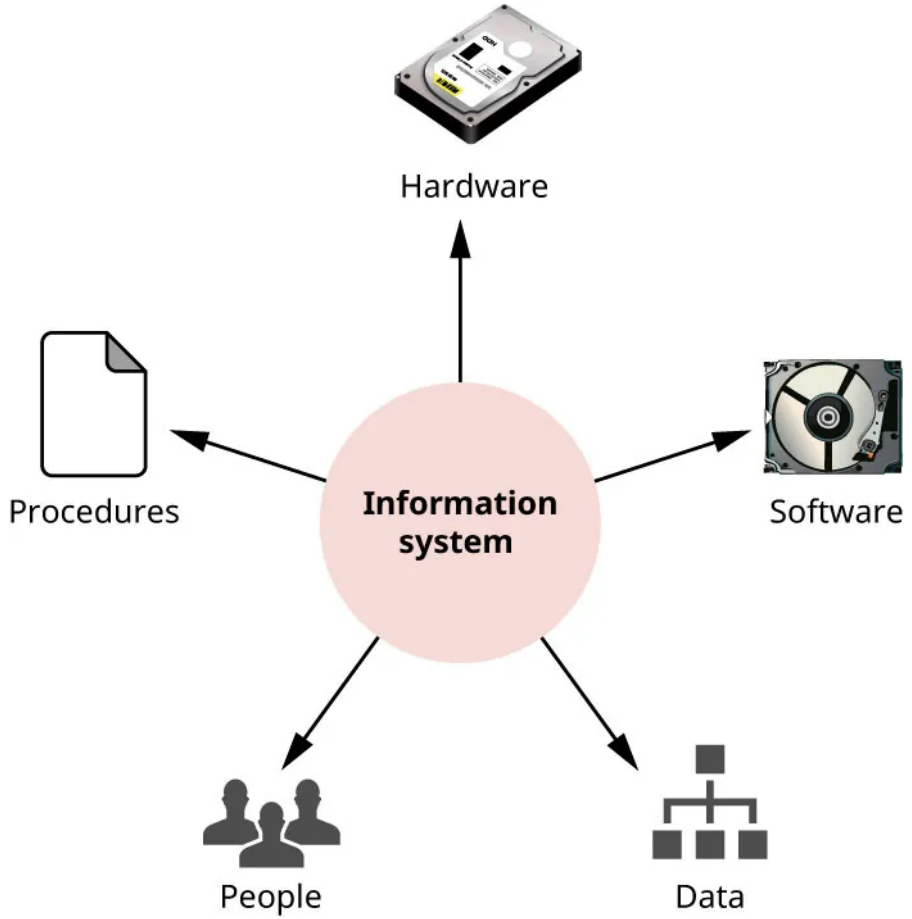
CSV enables analysis, while HML enables system integration!

From EpiDoc to an Information System

EpiDoc.XML
Encoded, hierarchical research data
↓
XSLT
Extraction and Transformation
↓
Python
Processing (aggregation and orchestration)
↓
CSV / HML
new data model (tabular dataset / harmonized XML dataset)
↓
Database (e.g., Heurist)
Data storage and querying
↓
Viewer / Exploration Interface
Exploration and analysis

Task

- Describe the workflow of a humanities researcher who starts with an image of a written artifact. At the end of the project, the researcher uses a) EFES or b) Heurist. How do the processes differ when a) and b) are used?
- Are all components of an information system in place? If so, to what extent? If not, what is missing?



When Structure Becomes a Barrier to Analysis

- XML-based formats such as EpiDoc organize data into deeply nested hierarchical structures
- Relevant information is often spread across multiple levels, making it difficult to extract complete entities
- Database systems such as PostgreSQL can store and query XML data, working with hierarchical structures typically requires complex queries (e.g., XPath or XQuery)
- Analytical workflows, however, are mostly based on tabular data, where each row represents a consistent unit of analysis
- In hierarchical XML, relevant information is often distributed across multiple nested levels, which makes aggregation and comparison difficult
- Even when XML can be processed directly, it is often converted into tabular formats to simplify analysis

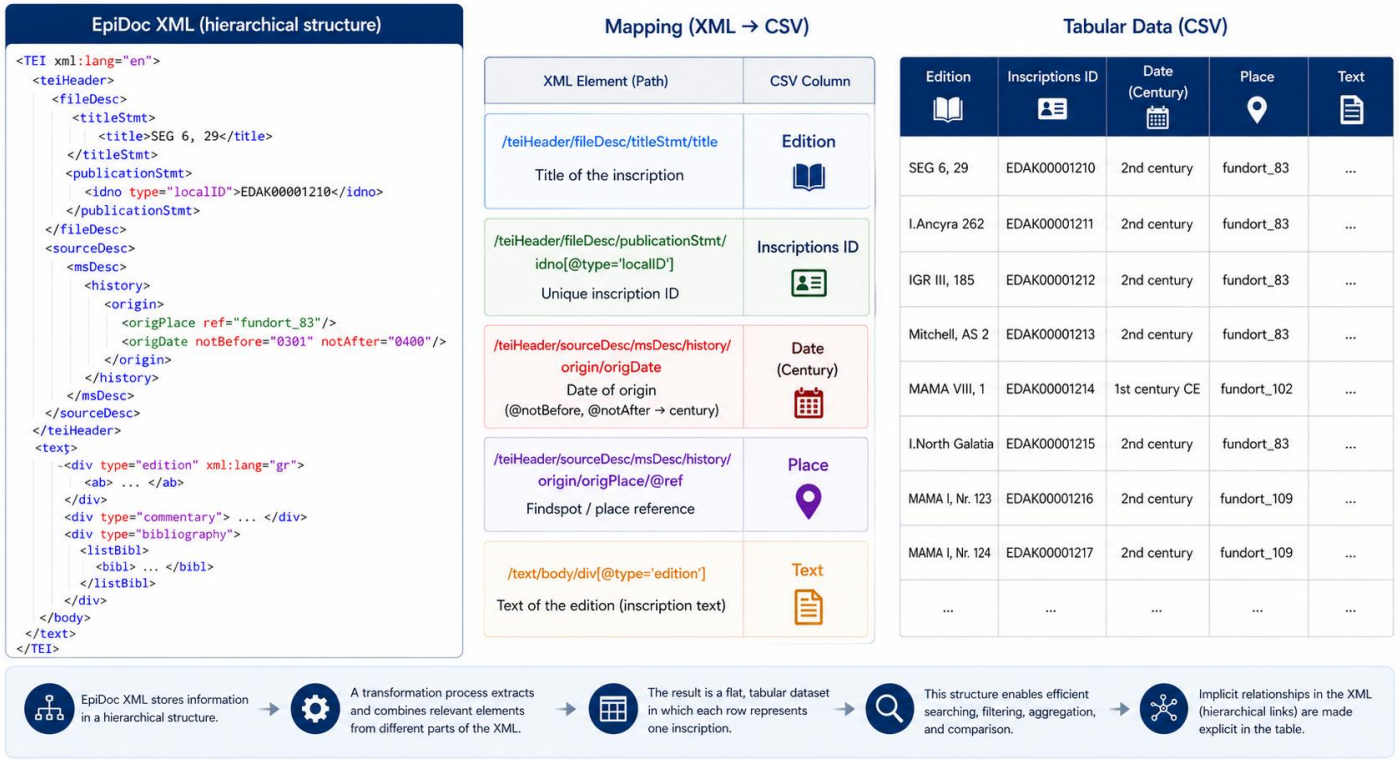
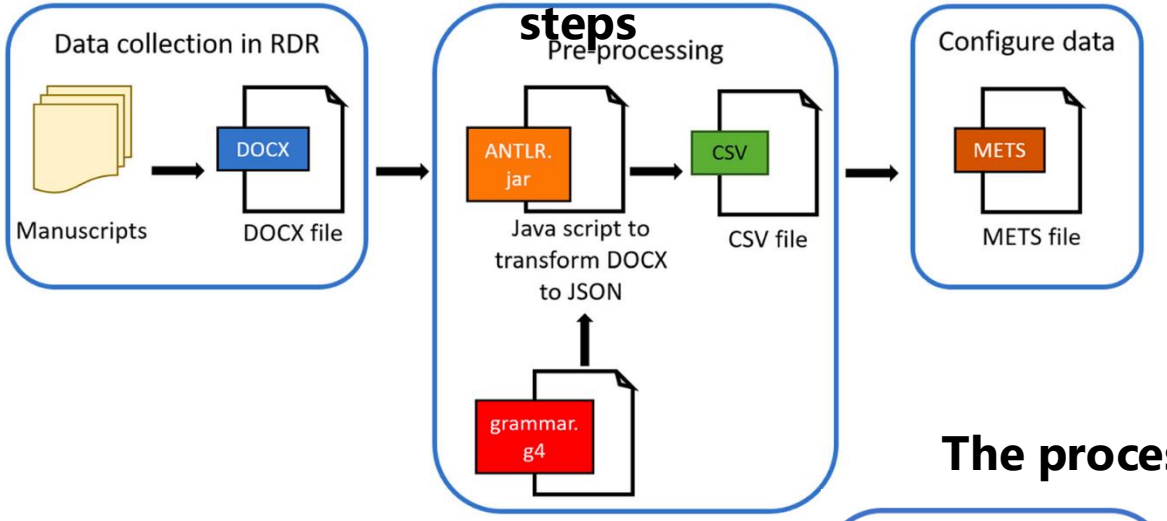


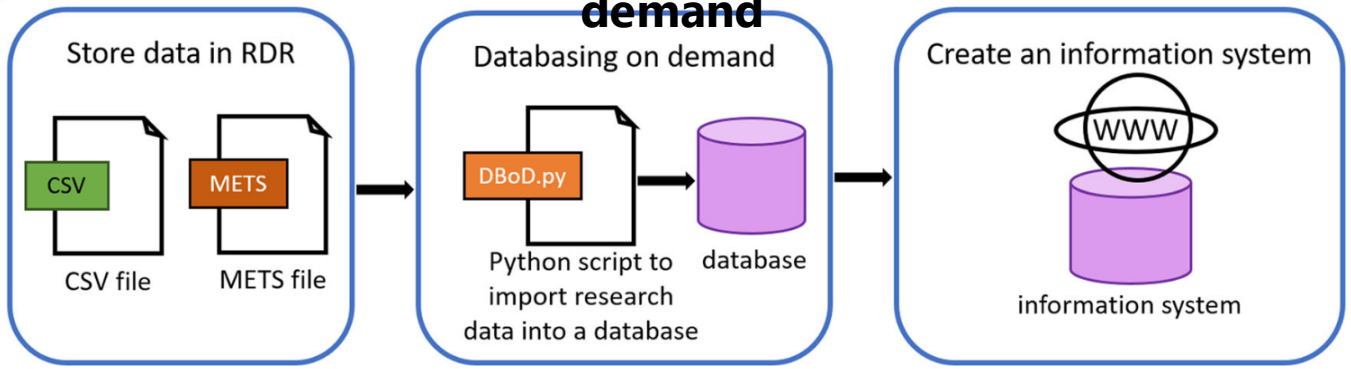
Illustration generated using ChatGPT based on a prompt provided by Sylvia Melzer

From Data to Information Systems

Pre-processing and archiving process with three steps



The process for building information systems on demand



Asselborn, T., Melzer, S., Schiff, S. *et al.* Building sustainable information systems and transformer models on demand. *Humanit Soc Sci Commun* **12**, 216 (2025). https://doi.org/10.1057/s41599-025-04491-x

Metadata Encoding & Transmission Standard (METS)

- Manifest files contain optional metadata (configuration data) for the associated data set and are available in XML format
- METS is a recognized standard that facilitates the exchange of digitized documents among cultural heritage institutions
- It operates as an XML schema specially designed for the generation of digital objects
- A digital object may encompass one or multiple digital files, potentially in varying formats, and can provide an intricate internal structure description
- A METS file consists of seven major sections
- In the example, an excerpt is presented of the fields important for ISoD

METS

Header

```
<mets:metsHdr CREATEDATE="2023-10-16T09:30:00"
              RECORDSTATUS="Complete">
 <mets:agent ROLE ="CREATOR"
             TYPE="INDIVIDUAL">
  <mets:name>
   Konrad Hirschler
  </mets:name>
 </mets:agent>
 <mets:agent ROLE ="ARCHIVIST"
             TYPE="INDIVIDUAL">
  <mets:name>
   Invenio Packager Webapp
  </mets:name>
 </mets:agent>
</mets:metsHdr>
```

Descriptive Metadata

```
<mets:dmdSec ID="DC">
 <mets:mdWrap MIMETYPE ="text/xml"
              MDTYPE="DC"
              LABEL="Dublin Core Metadata">
  <mets:xmlData>
   <dc:dc>
    <dc:creator>
     Konrad Hirschler
    </dc:creator>
   </dc:dc>
  </mets:xmlData>
 </mets:mdWrap>
</mets:dmdSec>
```

METS

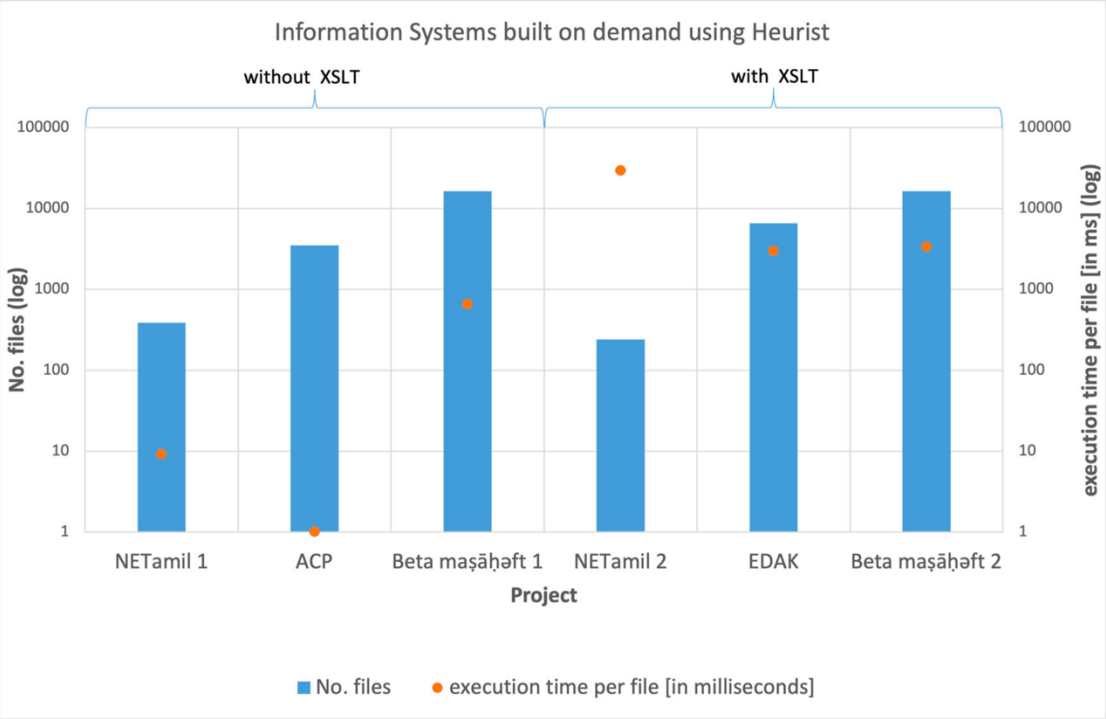
File Section

```
<mets:fileSec>
 <mets:fileGrp ID ="media">
  <mets:file ID="acp.csv"
             MIMETYPE="text/csv"
             SIZE=148826>
   <mets:FLocat LOCTYPE="URL"
                xlink:href="acp.csv">
   </mets:FLocat>
  </mets:file>
 </mets:fileGrp>
</mets:fileSec>
```

Structural Map & Behavior

```
<mets:structMap TYPE ="logical">
 <mets:div TYPE ="root">
  <mets:fptr FILEID="acp"/>
 </mets:div>
</mets:structMap>
```

```
<mets:behaviorSec>
 <mets:behavior ADMID="heurist">
  <mets:mechanism LABEL="isod"/>
 </mets:behavior>
</mets:behaviorSec>
```



Project	File Type	XSLT Used	No. of Files	Execution Time per File [in ms]
NETamil	CSV	no	388	9.2784
ACP	JSON	no	3,516	1.0239
Beta maṣāḥəft 1	TEI	no	16,337	661.0761
NETamil 2	TEI	yes	242	29,752.0661
EDAK	EpiDoc	yes	6,570	3,013.6986
Beta maṣāḥəft 2	TEI	yes	16,337	3,415.5598

Execution time comparison for different projects

- Asselborn, T., Melzer, S., Schiff, S. *et al.* Building sustainable information systems and transformer models on demand. *Humanit Soc Sci Commun* **12**, 216 (2025). https://doi.org/10.1057/s41599-025-04491-x

Differences

EFES

- Document-centric
- XML → HTML
- Viewing
- Preserves structure
- Single document
- Best for reading and publishing

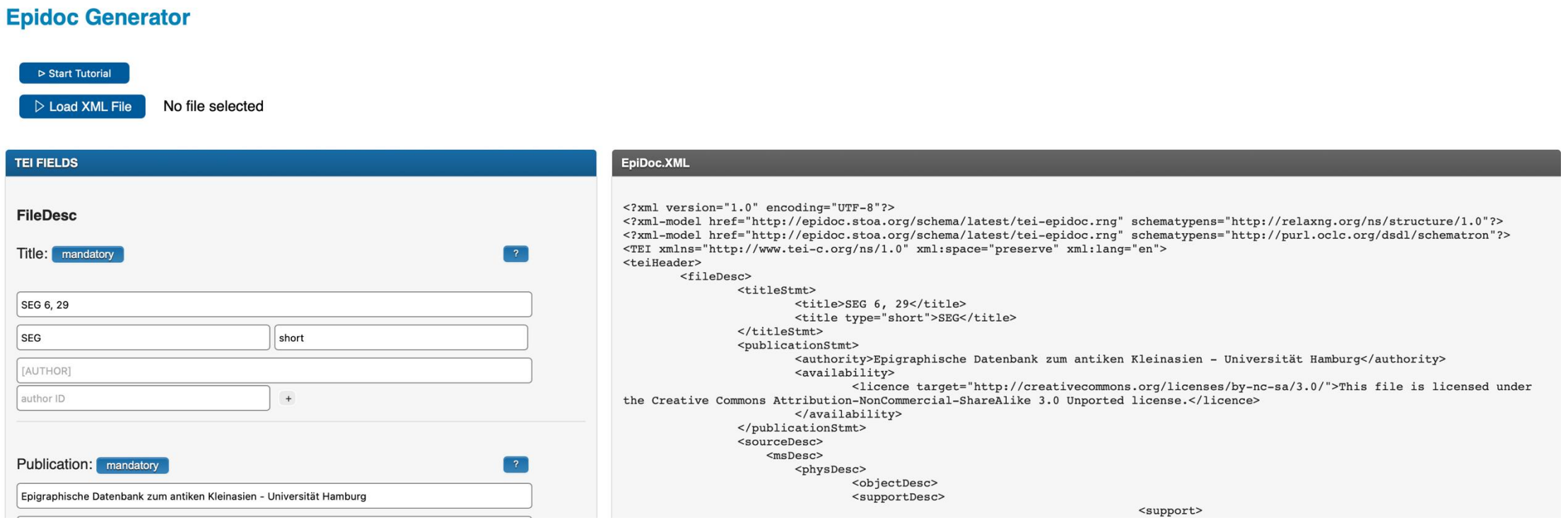
DBoD

- Data-centric
- XML → database
- Analysis
- Restructure data
- Multiple document
- Best for analysis and aggregation

Workflow

- How can we ensure that epigraphers use the same format?
- Currently, epigraphers create the EpiDoc files manually. How can we ensure that epigraphers create the EpiDoc files correctly?

EpiDoc Generator



https://tools.fdm.uni-hamburg.de/mdg/generator_metadata_epidoc.html

Conclusion

- Research data encoded in formats such as EpiDoc is rich in structure but not directly suitable for analysis
- Hierarchical XML structures distribute relevant information across multiple levels, which complicates querying and comparison
- Transformation pipelines using XSLT and Python enable the extraction, restructuring, and aggregation of this data
- DBoD allows the dynamic creation of structured datasets and information systems tailored to specific research questions
- Different transformation approaches serve different purposes: EFES supports document-oriented viewing, while DBoD enables data-driven analysis across multiple documents
- The transformation process defines the resulting data model and therefore directly influences interpretation and knowledge production
- The goal is not only to store data, but to make it explorable, accessible with high data quality, and usable for research
- **Well-structured data is not necessarily analysis-ready data**