Programming XSLT stylesheets for Adlib and Axiell

Axiell ALM Netherlands BV

Contents

Introduction

Underneath the surface of the graphical user interface of your Adlib application or Axiell Collections application, records are handled by the software in XML format, which basically is a hierarchically structured text format. Normally, you won't encounter the XML itself, but you'll have to know about it if you want to start using XSLT to create output formats or to edit presentation formats for Adlib Office Connect, for your Adlib Internet Server web application or for web browser boxes on application screens, to name but a few. XSLT is a stylesheet language (itself in XML format) to "transform" an XML document to some other document; this may be an XML document with the same structure but with changes made to the data in it, or it can be a differently structured XML document, or an HTML document, a PDF, or some other text file.
XML and XSLT are third-party programming technologies, so to properly learn all about them we recommend studying other sources than the document before you. In this manual though, you'll get a quick introduction followed by actual examples applicable to the Adlib or Axiell software, enough to get you started properly.
Some functionality requires Adlib 7.1 or higher.

1 An introduction to XML and XSLT

1.1 What is XML

XML (eXtentible Markup Language) provides a means of hierarchically structuring data in a text file. In contradiction to HTML (which is intended to lay out text or data for display in web browsers), it does not offer layout instructions. Other than a few PIs (Processing Instructions providing metadata about the document for the processor, enclosed in `<?  ?>`) at the start of the file, the only language it contains consists of tags, the name of which can be anything the maker of the XML document desired. Every separate piece of data must be enclosed by a start and end tag, formatted like `<tag>data</tag>`, together called an element or node, which may be spread over different lines. The following is an example of a simple yet complete XML document (not Adlib XML in any way though), although no title has been specified for the third book:

```
<?xml version="1.0" ?>
<!-- my comment -->
<booklist>
  <book isbn="901234567">
    <author>Hesse, Herman</author>
    <author>Claus, Hugo</author>
    <title>Siddharta</title>
  </book>
  <book>
    <author>Wolkers, Jan</author>
    <title>Terug naar Oegstgeest</title>
    <publisher>Summer &amp; Köning</publisher>
  </book>
  <book>
    <author>Austen, Jane</author>
  </book>
</booklist>
```

Every XML document has to start with: `<?xml version="1.0" ?>` or `<?xml version="2.0" ?>` to tell the processor which XML version is implemented in this document. Optionally, you could also mention here the Unicode representation in which this file has been saved, e.g.: `<?xml version="1.0" encoding="UTF-8"?>`

1.1.1 XML document requirements

There are some further rules to putting together an XML document.

1. Each XML document can only have one root tag. In the example above this is `booklist`.

2. Tags must have sensible names, so that others can easily understand the document, and for the sake of interoperability. If it contains Adlib data, those names need not be the same as Adlib field names per se.

3. Every element must be closed. A start tag without an end tag corrupts an XML document. If there is no data between a start and end tag, this may be indicated by a single combined tag to open and close at once: `<tag/>`.
   Note that there's a difference between an empty tag, for instance `<title></title>` and no title tags at all, which is relevant to XSLT stylesheets processing an XML document.

4. Tags are nested. In the example you can see that the authors Hesse and Claus are nested within the first `book` tag, and that the `book` tags are nested within the root tag `booklist`. This nesting is crucial to keeping data together, like it is in Adlib records.

5. The increasing indentation (whitespace) in front of nested tags, as shown in the example, is not strictly necessary, but it keeps the document readable.
   Visual Studio is handy for writing and editing XML docs, because it suggests end tags and adds coloring, but you can edit an XML doc in any text editor, as long as you save the file in Unicode UTF-8 representation (if you want to use the XML file in Adlib).

6. A tag may have attributes. An attribute is metadata included in a start tag, and it's purpose is to describe something about the data in the current element or all elements nested in it. It should be in the format `<tag attribute="value">`. In our example there is one attribute for the tag book: `<book isbn="901234567">`. This may not be a good example because ISBN is a field in an Adlib record, and is not really considered metadata in there. But the maker of the document decides what is metadata and what isn't. The language of records could also be specified this way, for example: `<booklist language="en-us">`. Every start tag may have zero, one or more attributes; attributes should be separated by a space. And every attribute must have a unique name, specified by the maker of the document, and it cannot contain spaces. The double quotes around the value are mandatory, a value in between isn't. Note that double quotes come in different varieties, but should be the straight version, as follows: ", not ‟ or ″ as created by MS Word for example.

7. A few characters have to be "escaped" (meaning: replaced) when used in the data itself, because they are reserved characters to the XML language. These are:

Character	**Escape sequence to use in data**
<	&lt;
>	&gt;
‛	&apos;
‟	&quot;
&	&amp;

   In our example we see an illustration of this: `<publisher>Summer &amp; Köning</publisher>`. Note that other special characters in data, for instance á or € don't need to be escaped because this is a Unicode file.

8. XML tags and attributes are case-sensitive. So `<Author>` is not the same as `<author>`.

The easiest way of checking whether an XML document doesn't contain any errors is by double-clicking it in Windows Explorer. If the file opens normally in Internet Explorer, there are no XML syntax errors. However if there *are* syntax errors, then the file doesn't open and IE displays an error message. So, Internet Explorer can validate an XML document.

1.1.2 How XML documents may be structured

If you were to write an XML document yourself, you would in principle be free to structure it any way you wanted. But if XML documents must be exchangeable between diverse software programs, you'll probably want the XML to adhere to some rules. Because you'll not only have software producing XML but also software to do something with the XML input (like the Adlib Internet Server web application or Adlib Office Connect for example), and therefor the hierarchy in the XML file must be what the software expects it to be.

For Adlib you usually won't write XML documents manually: they are produced internally by Adlib as the (intermediate and/or end) result of an export or print job or as the search result from wwwopac. So the software will by default produce XML documents, which adhere to earlier specified rules for Adlib XML files, together forming the so-called Adlib XML schema. Whenever third-party software produces XML documents to be processed by Adlib software at some point, it must also comply to this schema.

There are two methods to specify rules to which XML files must adhere, via a DTD (Document Type Definition) or XSD (XML Schema Definition).

- DTD is an old-fashioned way, although the EAD (Encoded Archival Description) still uses it. A disadvantage of the DTD is its syntax, which allows for files to become unreadable because of their complexity.
- XSD, the XML Schema Definition is DTD's successor. It is an XML file itself.

Adlib XML is formatted according to the *adlibXML.xsd* (which can be viewed in full at http://www.adlibsoft.com/adlibXML.xsd). The most important thing you need to know about Adlib XML is that only the XML tags of the three highest levels have been defined, namely: `adlibXML` (root tag), `recordlist` (may occur only once), `record` (may occur indefinitly). Further, there is a `diagnostic` tag on the level of the `recordlist`, which contains metadata about the search, such as the elapsed time and the number of records that were found. The fields in the records are contained within each record element and have the English field name by which they are declared in the database *.inf* file. The structure of an Adlib record itself is not defined in the schema definition because this differs per database and XML type.

1.1.3 Available XML types

Within the Adlib XML schema, different XML types are possible, mainly separated into *unstructured* and *grouped* XML, but grouped XML still has variations. XML is either produced by *adlwin.exe* (which runs your Adlib Museum, Library and Archive applications for Windows) or by *wwwopac.ashx* (API) (which processes Adlib data for the Internet Server web application or Adlib Office Connect).

■ Unstructured XML from wwwopac.ashx or adlwin.exe

Unstructured XML can be produced by *wwwopac.ashx* or be exported by *adlwin.exe*, if requested so.

Unstructured XML has a flat structure: all fields and their occurrences are immediate children of the `record` element. If a field has multiple occurrences, then all those occurrences are listed directly underneath each other.

- Of a multilingual field (if present), only the value of the currently searched data language is returned by *wwwopac.ashx*, and it is returned directly in the field node, without language or invariancy attributes. *Adlwin.exe* on the other hand, will export all language values.

- Of an enumerative field, *wwwopac.ashx* only returns the neutral value, directly in the enumerative field node, unless you specify the presentation language with the `language` parameter followed by a standard language code without square brackets around it, in which case only the relevant translation is returned, directly in the enumerative field node. *Adlwin.exe* on the other hand, returns the neutral value as an attribute to the field node while all user interface translations are listed inside the field element, in a `text` subnode per language.

Below you can see an abbreviated example of an *adlwin.exe* export of a Dutch record with repeated `object_name` nodes and a `record_type` enumerative field, to unstructured XML:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<adlibXML xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.adlibsoft.com/
adlibXML.xsd">
<recordList>
  <record>
    <priref>139</priref>
    <institution.name>Nederlands Textielmuseum</institution.name>
    <description>Gordijnstof met voorstelling van kookboeken.
     Verticaal ruggen van boeken in verschillende breedten en
     kleuren. Op de ruggen titels van kookboeken.</description>
    <institution.place/>
    <production.date.end>1960</production.date.end>
    <reproduction.format/>
    <reproduction.reference>06243.jpg</reproduction.reference>
    <object_number>00216243</object_number>
    <object_name>interieurtextiel</object_name>
    <object_name>raambedekking</object_name>
    <object_name>gordijnstof</object_name>
    <title>Gordijnstof met kookboeken</title>
    <record_type option="OBJECT" value="OBJECT">
      <text language="0">single object</text>
      <text language="1">enkelvoudig object</text>
      <text language="2">objet individuel</text>
      <text language="3">Einzelnes Objekt</text>
    </record_type>
  </record>
</recordList>
<diagnostic>
  <xmltype>Unstructured</xmltype>
  <hits>1</hits>
  <dbname>collect</dbname>
  <dsname>intern</dsname>
</diagnostic>
</adlibXML>
```

■ Grouped XML as produced by wwwopac.ashx

Grouped XML is hierarchically structured XML: fields may be a direct child of the `record` element, or when a field group name has been defined in the data dictionary, a child of a group element with the name of the group. In this case the group element is a child of the `record` element. There are differences between the grouped XML of records retrieved for brief display and that of a single record retrieved for detailed display. The rest of this paragraph considers the grouped XML retrieved for detailed display; examples of both types of XML can be studied at http://api.adlibsoft.com/site/api/functions/search by first adding `&xmltype=grouped` to the example queries, if needed, and then executing them.

If at least one of the fields in a field group has multiple occurrences, then the entire field group is repeated as many times. Empty occurrences of fields in a field group are retrieved as well. The main advantage of the grouped type over the unstructured one is that it becomes easier to process repeated occurrences of grouped fields, using XSLT. In unstructured Adlib XML, all fields and field occurrences are just listed in one long list inside the `<record>` node, whilst in grouped Adlib XML, fields are grouped within a field group node (if a relevant field group exists in the data dictionary) and that field group node is repeated for each field group occurrence.

- Of a multilingual field (if present), all language values are returned as `value` subnodes of the field node; the language code and invariancy flag per language value are returned as attributes to the value nodes.
- Of an enumerative field, both the neutral value and all available translations of the enumerative value are returned, in `value` subnodes underneath the enumerative field node; the presentation languages are attributes to the `value` nodes, and are indicated by an Adlib language number, not by their language code. The presentation `language` parameter does not apply to the `grouped` XML output type.

A partial example of grouped *wwwopac.ashx* output of a single record retrieved in detail:

```
<?xml version="1.0" encoding="UTF-8"?>
<adlibXML>
  <recordList>
    <record selected="False" modification="2012-05-31T11:11:27"
     created="2007-02-07T14:40:36" priref="10">
      <acquisition.date>1816</acquisition.date>
      <administration_name>PDP</administration_name>
      <content.person.name>Venus</content.person.name>
      <content.person.name>Cupid</content.person.name>
```

```xml
<content.person.name.type>
  <value lang="neutral">PERSON</value>
  <value lang="0">Person</value>
  <value lang="1">persoon</value>
  <value lang="2">personne</value>
  <value lang="3">Person</value>
  <value lang="4">شخص إسم</value>
  <value lang="6">πρόσωπο</value>
</content.person.name.type>
<creator.role.lref>2</creator.role.lref>
<Dimension>
  <dimension.value>118.1</dimension.value>
  <dimension.type>height</dimension.type>
  <dimension.type.lref>6</dimension.type.lref>
  <dimension.unit>cm</dimension.unit>
  <dimension.unit.lref>8</dimension.unit.lref>
</Dimension>
<Dimension>
  <dimension.value>208.9</dimension.value>
  <dimension.type>width</dimension.type>
  <dimension.type.lref>7</dimension.type.lref>
  <dimension.unit>cm</dimension.unit>
  <dimension.unit.lref>8</dimension.unit.lref>
</Dimension>
<institution.name>The Fitzwilliam Museum</institution.name>
<institution.name.lref>4</institution.name.lref>
<institution.place/>
<Material>
  <material.part>medium</material.part>
  <material>oil paint</material>
</Material>
<Material>
  <material.part>support</material.part>
  <material>canvas</material>
</Material>
<object_category>painting</object_category>
<object_category.lref>1</object_category.lref>
<object_number>109</object_number>
<priref>10</priref>
<Production>
  <creator>Palma, Jacopo il Vecchio</creator>
  <creator.date_of_birth/>
  <creator.date_of_death/>
  <creator.history/>
  <creator.qualifier/>
  <creator.role>painter</creator.role>
  <production.notes/>
  <production.place/>
</Production>
<Title>
  <title>
    <value lang="el-GR" invariant="false">Venus and
     Cupid</value>
  </title>
</Title>
```

```
    </record>
  </recordList>
  <diagnostic>
    <hits>1</hits>
    <xmltype>Grouped</xmltype>
    <first_item>1</first_item>
  </diagnostic>
</adlibXML>
```

- In the grouped *wwwopac.ashx* output, the record priref is an attribute of the `<record>` node, but appears as a separate node as well.
- Up to and including *wwwopac.ashx* version 3.6.1173.0, if in the grouped *wwwopac.ashx* output an accessible field to be retrieved was part of a data dictionary field group, then all fields from the field group would be retrieved, even if they were empty. In later versions, only the available fields set in *adlibweb.xml* will be retrieved.
- In the grouped *wwwopac.ashx* output, the names of the subnodes of a linked field are the names of the linked field in the primary database (which are the target fields for any merged-in fields).
- In the grouped *wwwopac.ashx* output, the linkref field has its own subnode underneath the linked field, containing the actual linkref.

■ Grouped XML as produced by adlwin.exe

Your *adlwin.exe* application, like Adlib Museum or Library, can export to grouped XML as well as to unstructured XML. And internally, you can use record data in grouped XML format to create a display format for a web browser box for or to build an output format, using XSLT.

Grouped XML is hierarchically structured XML: fields may be a direct child of the `record` element, or when a field group name has been defined in the data dictionary, a child of a group element with the name of the group. In this case the group element is a child of the `record` element. Unlike the grouped XML produced by *wwwopac.ashx*, there is no difference between the grouped XML of multiple records exported or printed and that of a single record being exported, printed or displayed in a web browser box.

If at least one of the fields in a field group has multiple occurrences, then the entire field group is repeated as many times. Empty occurrences of fields in a field group are retrieved as well. The main advantage of the grouped type over the unstructured one is that it becomes easier to process repeated occurrences of grouped fields, using XSLT. In unstructured Adlib XML, all fields and field occurrences are just listed in one long list inside the `<record>` node, whilst in grouped

Adlib XML, fields are grouped within a field group node (if a relevant field group exists in the data dictionary) and that field group node is repeated for each field group occurrence.

- Of multilingual fields (if present), all language values are returned in repetitions of the field node itself or in repetitions of the linked-to field if it concerns a multilingual linked field; the language code and possibly the invariancy flag per language value are returned as attributes to the relevant field nodes.

  ```
  <object_name linkref="187" linkfield="term"
   linkreffield="broader_term.lref" linkdb=" C:\Adlib
   Software\Model application 4.2 NL\data++thesau">
      <term occurrence="1" lang="nl-NL">Ansichtkaart</term>
      <term occurrence="1" lang="en-GB" invariant="true">
       Postcard</term>
  </object_name>
  <Title>
      <title.type />
      <title occurrence="1" lang="de-DE">Köln</title>
      <title.notes />
  </Title>
  ```

- Of an enumerative field, the neutral value is returned as an attribute to the field node while all user interface translations are listed inside the field element, in a `text` subnode per interface language

- The record priref appears only as a separate element.

- The names of the subnodes of a linked field (like `object_name`) are the names of the linked-to field and any merged-in fields from the linked database. For example: the `<term>` node underneath `<object_name>` refers to the linked-to `term` field in the linked database THESAU.

- The `linkref`, `linkfield`, `linkreffield` and `linkdb` of a linked field are attributes to the linked field element.

A partial example of a Dutch record with repeated `object_name` nodes and a `record_type` enumerative field can be seen below:

```
<?xml version="1.0" encoding="UTF-8" ?>
<adlibXML xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://www.adlibsoft.com/
 adlibXML.xsd">
  <recordList>
   <record>
     <priref>139</priref>
     <current_location linkref="529" linkfield="location"
      linkreffield="2a" linkdb="C:\Adlib Software\Model application
      4.2 NL\data+Location">
        <location>4.10.03</location>
        <location.type />
```

```
</current_location>
<current_location.type />
<current_location.lref>529</current_location.lref>
<Description>
   <description.name />
   <description.date />
   <description>Gordijnstof met voorstelling van kookboeken.
    Verticaal ruggen van boeken in verschillende breedten en
    kleuren. Op de ruggen titels van kookboeken.</description>
 </Description>
 <institution.name linkref="150" linkfield="name"
  linkreffield="language.lref" linkdb="C:\Adlib Software\Model
  application 4.2 NL\data+people">
   <name>Nederlands Textielmuseum</name>
   <address.place />
   <institution_code />
 </institution.name>
 <institution.place />
 <Production_date>
   <production.date.end>1960</production.date.end>
   <production.date.start.prec />
   <production.date.start>1950</production.date.start>
   <production.date.end.prec />
 </Production_date>
 <production.date.notes>ca.</production.date.notes>
 <Reproduction>
   <reproduction.format />
   <reproduction.reference linkref="180"
    linkfield="reference_number" linkreffield="fn"
    linkdb="C:\Adlib Software\Model application 4.2
    NL\data+photo">
     <format />
     <reference_number>06243.jpg</reference_number>
     <production_date />
     <reproduction_type />
     <creator />
   </reproduction.reference>
   <reproduction.date />
   <reproduction.notes />
   <reproduction.type />
   <reproduction.creator />
   <reproduction.reference.lref>180
    </reproduction.reference.lref>
 </Reproduction>
 <Object_name>
   <object_name.authority linkref="0" linkfield="term"
    linkreffield="lx" linkdb="C:\Adlib Software\Model
    application 4.2 NL\data+thesau" />
   <object_name linkref="411" linkfield="term"
    linkreffield="broader_term.lref" linkdb=
    "C:\Adlib Software\Model application 4.2 NL\data+thesau">
     <term>interieurtextiel</term>
   </object_name>
   <object_name.notes />
```

```
      <object_name.type linkref="0" linkfield="term"
       linkreffield="lw" linkdb="C:\Adlib Software\Model
       application 4.2 NL\data+thesau" />
      <object_name.lref>411</object_name.lref>
      <object_name.type.lref />
      <object_name.authority.lref />
    </Object_name>
    <Object_name>
      <object_name.authority linkref="0" linkfield="term"
       linkreffield="lx" linkdb="C:\Adlib Software\Model
       application 4.2 NL\data+thesau" />
      <object_name linkref="442" linkfield="term"
       linkreffield="broader_term.lref" linkdb="C:\Adlib
       Software\Model application 4.2 NL\data+thesau">
        <term>raambedekking</term>
      </object_name>
      <object_name.notes />
      <object_name.type linkref="0" linkfield="term"
       linkreffield="lw" linkdb="C:\Adlib Software\Model
       application 4.2 NL\data+thesau" />
      <object_name.lref>442</object_name.lref>
      <object_name.type.lref />
      <object_name.authority.lref />
    </Object_name>
    <Object_name>
      <object_name.authority linkref="0" linkfield="term"
       linkreffield="lx" linkdb="C:\Adlib Software\Model
       application 4.2 NL\data+thesau" />
      <object_name linkref="443" linkfield="term"
       linkreffield="broader_term.lref" linkdb="C:\Adlib
       Software\Model application 4.2 NL\data+thesau">
        <term>gordijnstof</term>
      </object_name>
      <object_name.notes />
      <object_name.type linkref="0" linkfield="term"
       linkreffield="lw" linkdb="C:\Adlib Software\Model
       application 4.2 NL\data+thesau" />
      <object_name.lref>443</object_name.lref>
      <object_name.type.lref />
      <object_name.authority.lref />
    </Object_name>
    <record_access.owner>Administrator</record_access.owner>
    <Title>
      <title.type />
      <title>Gordijnstof met kookboeken</title>
      <title.notes />
    </Title>
    <record_type option="OBJECT" value="OBJECT">
      <text language="0">single object</text>
      <text language="1">enkelvoudig object</text>
      <text language="2">objet individuel</text>
      <text language="3">Einzelnes Objekt</text>
    </record_type>
  </record>
</recordList>
```

```
  <diagnostic>
    <xmltype>Grouped</xmltype>
    <hits>1</hits>
    <dbname>collect</dbname>
    <dsname>intern</dsname>
  </diagnostic>
</adlibXML>
```

As mentioned in the paragraph above, the main advantage of the grouped type over the unstructured one is that it becomes easier to process repeated occurrences of grouped fields, using XSLT. In unstructured Adlib XML, all fields and field occurrences are just listed in one long list inside the `<record>` node, whilst in grouped Adlib XML, fields are grouped within a field group node (if a relevant field group exists in the data dictionary) and that field group node is repeated for each field group occurrence.

Simply export one or more records to the grouped XML format from within your Adlib application and open the resulting file in Internet Explorer to study the result and learn more about Adlib grouped XML as generated by *adlwin.exe*.

1.2 What is XSLT

XSL(T) stands for eXtensible Stylesheet Language Transformations. It is a pattern-based language and has characteristics of programming languages as well, which you use to “transform” an XML document to some other document; this may be an XML document with the same structure but with changes made to the data in it, or it can be a differently structured XML document, or an HTML document, a PDF, CSV or some other text file. During transformation, the data from the original XML document can also be processed in other ways.

Adlib internally represents records as XML and when you execute an XSLT export format or output format or display it through a web browser box on a record detail screen, this XML is passed on to the associated stylesheet which converts the XML to the desired format: this target format would need to be HTML if it concerns an output (print) format or display format, or any desired format (XML, HTML, plain text, etc.) if it concerns an export format. As XML-to-XML stylesheets, it allows third-party XML export or search results to be tranformed to XML that Adlib can work with, or vice versa. As XML-to-HTML stylesheets, it allows Adlib XML, like produced by *wwwopac.ashx* and internally by *adlwin.exe* to be transformed into fully laid out pages presentable like web pages in Internet Explorer or in a web browser box or to be printed with a nice layout.

Originally XSLT was just named XSL, as it was thought to primarily function as layout language to produce HTML output, but as it turned out that it could be used for other transformations as well, the “T” was added. For stylesheet names it is irrelevant whether you use the extension *.xsl* or *.xslt*: there is no functional difference.

You can apply a stylesheet to an XML document either:

- programmatically via the settings file of a web application like the Adlib Internet Server of Adlib Office Connect;
- by linking the stylesheet to your Adlib application as an output (print) format, an export format or web browser box display format, using Adlib Designer;
- by hardcoding a reference to the stylesheet in the XML document.

In all cases you need a “transformation engine” to do the actual transforming and produce output. Luckily, such an engine is by default part of Internet Explorer, Firefox, the .Net platform, and MSXML.
If you hardcoded a reference to a (XML-to-HTML) stylesheet in an XML document, then all you have to do to apply the transformation is double-click the XML document in your Windows Explorer: it will open as an HTML page in Internet Explorer (although you cannot view or save the actual HTML code, since it has been generated dynamically). Note that a so-called XML-parser only reads XML, it is not necessarily linked to a tranformation engine.

1.2.1 A bare stylesheet

Each stylesheet starts with the following:

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

and ends with: `</xsl:stylesheet>`

XSLT 3.0 and earlier versions are supported from Collections 1.14: only XSLT 1.0 was supported before that, in Adlib and Collections, not XSLT 2.0. Of XML, both version 1.0 and 2.0 are supported, and XSLT 1.0 can be used in an XML 2.0 document if needed.

The header may contain a third line to specify the type of output this stylesheet will generate. For HTML this is: `<xsl:output method="html"/>`

In between you specify the actual patterns. XSLT has a syntax similar to XML, with PIs (as above), and `<namespace:name>output </namespace:name>` elements. The namespace you always use is: `xsl`.

The `names` are XSLT keywords or functions, since the `xsl` Name Space applies. XSLT is also case-sensitive.

1.2.2 XPath and templates

Suppose we have the following XML document (not Adlib XML):

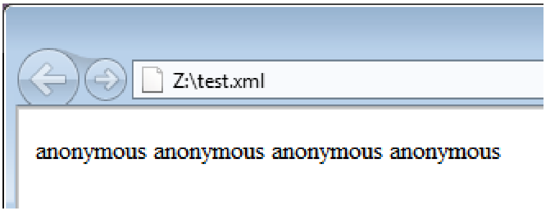
```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="books.xslt"?>

<!-- my comment -->
<booklist>
  <book isbn="901234567">
    <author>Hesse, Herman</author>
    <author>Claus, Hugo</author>
    <title>Siddharta</title>
  </book>
  <book>
    <author>Wolkers, Jan</author>
    <title>Terug naar Oegstgeest</title>
    <publisher>Summer &amp; Köning</publisher>
  </book>
  <book>
    <author>Austen, Jane</author>
  </book>
</booklist>
```

A reference to a stylesheet called *books.xslt* we're about to create (in the same folder), is hardcoded in the XML document, as you can see, so it will be transformed through the stylesheet by Internet Explorer as soon as you open it.

XPath is similar to a path in the folder structure in Windows, but it applies to an XML document. For example, the XPath of any author in this document is */booklist/book/author*. This is relevant for the templates in your stylesheet. In XSLT, templates are the basis for the intended transformation: they contain the functions and text or HTML code to be applied respectively added to XML elements which you consider to be a pattern. A very simple example of a stylesheet *books.xslt* for this XML file might clarify this:

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

        <xsl:template match="/booklist">
          <xsl:apply-templates select="//author"/>
        </xsl:template>
        <xsl:template match="author">
          anonymous
        </xsl:template>

</xsl:stylesheet>
```

Two templates have been defined in here. What the transformation engine does, is it looks for template matches which it can apply, from the root of all XPaths. Whether it can apply a template depends on whether the XPath node to match is accessible from the root node. In the example above we intend to look for every occurrence of an `<author>` element in the XML file and replace it's content by the text "anonymous". From the root node the */booklist* node is accessible, but from there the author node is only available if we precede it by "//": this means the author node can occur anywhere in an XPath. The result of this stylesheet applied to the example XML file is the following:



```
anonymous anonymous anonymous anonymous
```

If we were to leave out "//" the match could not be made, and applying the stylesheet would result in an empty page. But if you know at what level in an XPath the author node occurs you may also point directly to it, in our case via:

```
<xsl:template match="/booklist">
  <xsl:apply-templates select="book/author"/>
</xsl:template>
```

From the result you can deduce how the transformation works. There are two templates, but the `author` template cannot be matched from the root of XPath, the */booklist* can be matched though. So the transformation process enters into this template for instructions about how to transform the */booklist* node of the XML file, and this node also becomes the current XPath level. From this node we want to explicitily call the `author` template, which we do with: `apply-templates select="<relative Xpath to desired template>"`. So from the */booklist* node we can access the `author` template by selecting either *book/author* or *//author*.

And although we only call the `author` template once, it is automatically applied to all author elements in the XML file, at the selected XPath level: */author* elements placed directly underneath the */booklist* node

for example, would not be matched.
In the displayed result we can also see that the titles and publisher from the XML file have been ignored; this is because we haven't specified templates for these elements yet.

By the way, if the XSLT file does exist (in the same folder), but has no templates specified, then the "default" template is used to lay out the XML to HTML, which results in a string of plain text.

1.2.3 Extending the stylesheet to produce proper HMTL

Until now, our transformations have not produced proper HTML documents. Luckily, Internet Explorer isn't very strict about this, so the transformed XML could still be displayed. But it is good practice to always adhere to the rules of the document type you are transforming to. So let's extend our stylesheet to make proper HTML.

An empty HTML file may look as follows:

```
<html>

<head>
<title>My title for this page</title>
</head>

<body>

</body>

</html>
```

Actual content will be placed between the `<body>` tags. A simple piece of content may be:

```
<p>This is one line of <i>text</i>.</p>
```

The word "text" will be displayed in italics.

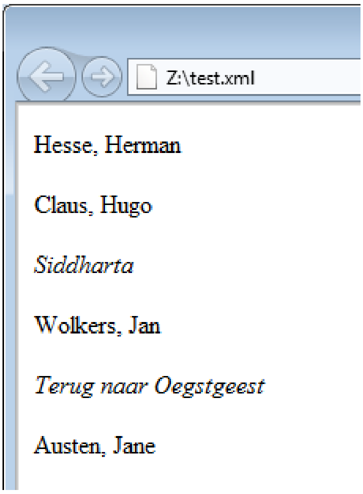
Extending our XSLT stylesheet could for example result in the following:

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

<xsl:template match="/booklist/book">
  <html>
    <head>
      <title>My title for this page</title>
    </head>
    <body>
      <xsl:apply-templates select="author"/>
      <xsl:apply-templates select="title"/>
    </body>
  </html>
</xsl:template>

<xsl:template match="author">
  <p>
    <xsl:value-of select="."/>
  </p>
</xsl:template>

<xsl:template match="title">
  <p><i>
    <xsl:value-of select="."/>
  </i></p>
</xsl:template>

</xsl:stylesheet>
```

Note a couple of things:

- A template for the *title* node has been added.
- The XPaths to the *author* and *title* nodes are handled a little differently here. The base match now takes place on */booklist/book*.
- Instead of replacing author names by “anonymous”, we display the value contained in the *author* node in the XML file, and the actual titles.
- We have added HTML tags in different places to make the output proper HTML.

The result is as follows:



Hesse, Herman

Claus, Hugo

Siddharta

Wolkers, Jan

Terug naar Oegstgeest

Austen, Jane

This illustrates the order in which the templates have been applied. Per *book*-match, to all authors the `author` template is applied, then to all titles the `title` template. And every author and title is placed on a new line, because the HTML `<p>`-tags are in the `author` and `title` templates.

1.2.4 Using CSS stylesheets

In HTML pages you have the option to refer to a CSS (Cascading Style Sheet), although this is in no way a requirement. In a CSS you can assign font types and character layout styles to HTML structural elements (like the body of the page or tables) and to so-called layout classes which you specify yourself. The advantage of doing this in a CSS instead of just hardcoded in the HTML itself (like in the example above for the italic layout of the title), is that it is much more efficient and faster to adjust the definition of a style once, than to re-apply the adjusted style everywhere in the HTML. However, if you don't need reusable layout styles and you don't mind applying all layout through HTML tags, then you might as well leave CSS out of the equation altogether.

An example of a simple CSS is the following. Save this file as *mystyle.css* in the same folder.

```
BODY
{
  color: blue;
  background-color: lightyellow;
  font-family: Verdana, Arial, Helvetica, sans-serif;
  font-size: 85%;
}

TABLE
{
  color: blue;
}

.title
{
  font-style: italic;
  text-decoration: underline;
}
```

Note a couple of things:

- `title` is a new class, `BODY` and `TABLE` are HTML structural elements. (The `TABLE` style will be used later on.)
- The several font types summed up behind `font-family`, indicate the priority in which these are applied. If the computer of the user doesn't have the Verdana type installed, Arial will be used, etc.

- Instead of colour names, you can also use the hexadecimal RGB (Red Green Blue) notation of colours, e.g. #DDDDDD (grey), or #ffff99 (yellow).

In an HTML document you link to a CSS in the `<head>` section:

```
<link type="text/css" href="mystyle.css" rel="stylesheet"/>
```

So in our XSLT stylesheet, where we build up an HTML page, we can do exactly the same, as can be seen in the further extended XML document:

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

<xsl:template match="/booklist/book">
  <html>
    <head>
      <link type="text/css" href="mystyle.css" rel="stylesheet"/>
      <title>My title for this page</title>
    </head>
    <body>
      <xsl:apply-templates select="author"/>
      <xsl:apply-templates select="title"/>
    </body>
  </html>
</xsl:template>

<xsl:template match="author">
  <p>
    <xsl:value-of select="."/>
  </p>
</xsl:template>

<xsl:template match="title">
  <p>
    <div class="title">
      <xsl:value-of select="."/>
    </div>
  </p>
</xsl:template>

</xsl:stylesheet>
```

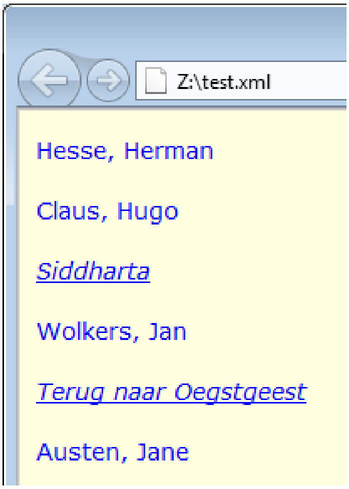
Instead of storing the CSS code in its own file, you can also choose to include it in the XSLT stylesheet itself, in between HTML `<style type="text/css">` and `</style>` tags in the `<head>` section:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/booklist/book">
    <html>
      <head>
        <link type="text/css" href="mystyle.css" rel="stylesheet"/>
        <title>My title for this page</title>
        <style type="text/css">
          BODY
          {
          color: blue;
          background-color: lightyellow;
          font-family: Verdana, Arial, Helvetica, sans-serif;
          font-size: 85%;
          }

          TABLE
          {
          color: blue;
          }

          .title
          {
          font-style: italic;
          text-decoration: underline;
          }
        </style>
      </head>
      <body>
        <xsl:apply-templates select="author"/>
        <xsl:apply-templates select="title"/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="author">
    <p>
      <xsl:value-of select="."/>
    </p>
  </xsl:template>
  <xsl:template match="title">
    <p>
      <div class="title">
        <xsl:value-of select="."/>
      </div>
    </p>
  </xsl:template>
</xsl:stylesheet>
```

Without the CSS styles, you can obtain a similar result by including HTML layout tags and attributes in the XSLT templates, as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/booklist/book">
    <html>
      <head>
        <title>My title for this page</title>
      </head>
      <body bgcolor="lightyellow">
        <font face="verdana" color="blue" size="3">
          <xsl:apply-templates select="author"/>
          <xsl:apply-templates select="title"/>
        </font>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="author">
    <p>
      <xsl:value-of select="."/>
    </p>
  </xsl:template>

  <xsl:template match="title">
    <p>
      <u><i><xsl:value-of select="."/></i></u>
    </p>
  </xsl:template>

</xsl:stylesheet>
```

The result of either transformation now looks as follows:



Hesse, Herman

Claus, Hugo

Siddharta

Wolkers, Jan

Terug naar Oegstgeest

Austen, Jane

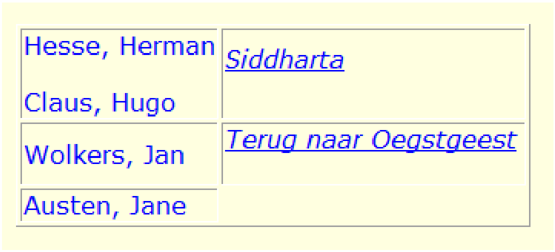
1.2.5 Applying HTML tables

Now let’s try to put this in a nice table, using CSS. Again, we use standard HTML tags to accomplish this. The template and the location therein in which you place these tags matters of course. After all, do you want a table around each *author*, around each *book*, or just one for the entire *booklist*?

To get one table around all books, we have to change the first template to match */booklist*, add a template for a `book`, move the base HTML to that new template, and call the `book` template within HTML `<table>` tags from within the first template and call the `author` and `title` templates within table cells and rows:

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

<xsl:template match="/booklist">
  <html>
    <head>
      <link type="text/css" href="mystyle.css" rel="stylesheet"/>
      <title>My title for this page</title>
    </head>
    <body>
      <table border="1">
        <xsl:apply-templates select="book"/>
      </table>
    </body>
  </html>
</xsl:template>

<xsl:template match="book">
  <tr>
    <td>
      <xsl:apply-templates select="author"/>
    </td>
    <td>
      <xsl:apply-templates select="title"/>
    </td>
  </tr>
</xsl:template>

<xsl:template match="author">
  <p>
    <xsl:value-of select="."/>
  </p>
</xsl:template>

<xsl:template match="title">
  <p>
    <div class="title">
      <xsl:value-of select="."/>
    </div>
  </p>
</xsl:template>

</xsl:stylesheet>
```

The result looks like this:



Hesse, Herman Claus, Hugo	*Siddharta*
Wolkers, Jan	*Terug naar Oegstgeest*
Austen, Jane	

Note the empty cel in the right bottom corner, due to the lack of a title for the author Jane Austen. The cell didn’t even get a border because the title template wasn’t applied here.

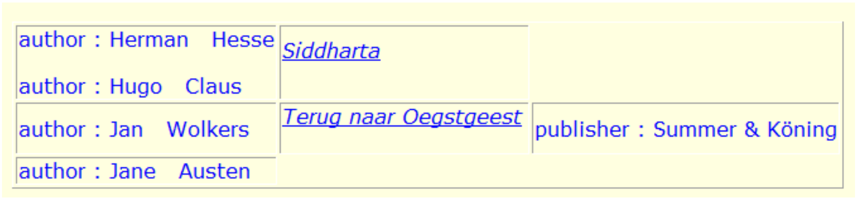
1.2.6 Functions, variables and parameters in XPath

In XSLT you can use variables but you can assign a value to it only once. So you cannot use incremental counters, or string variables which you build up piece by piece. Nor are there normal loop constructions. (The solution here is recursive programming: calling the current template from within the template, with parameters, but that is beyond the scope of this documentation.)
Let’s extend the XSLT stylesheet we’ve been working on with some basic functionality, to finish this introduction:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
<xsl:template match="/booklist">
  <html>
    <head>
     <link type="text/css" href="mystyle.css" rel="stylesheet"/>
     <title>My books list</title>
    </head>
    <body>
      <table border="1">
        <xsl:apply-templates select="book"/>
      </table>
    </body>
  </html>
</xsl:template>
```

```
<xsl:template match="book">
  <tr>
    <td><xsl:apply-templates select="author"/></td>
    <td><xsl:apply-templates select="title"/></td>
    <td><xsl:apply-templates select="publisher"/></td>
  </tr>
</xsl:template>

<xsl:template match="author | publisher">
  <p>
    <xsl:value-of select="name()"/>
    <xsl:variable name="name">
      <xsl:choose>
        <xsl:when test="contains(., ',')">
          <xsl:value-of select="substring-after(., ',')"/>
          &#xa0;
          <xsl:value-of select="substring-before(., ',')"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="."/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:variable>
    <xsl:call-template name="printTheName">
      <xsl:with-param name="nameParameter" select="$name"/>
    </xsl:call-template>
  </p>
</xsl:template>

<xsl:template name="printTheName">
  <xsl:param name="nameParameter"/>
  :
  <xsl:value-of select="$nameParameter"/>
</xsl:template>

<xsl:template match="title">
  <p>
    <div class="title">
      <xsl:value-of select="."/>
    </div>
  </p>
</xsl:template>

</xsl:stylesheet>
```

The result looks as follows:



author : Herman Hesse author : Hugo Claus	*Siddharta*	
author : Jan Wolkers author : Jane Austen	*Terug naar Oegstgeest*	publisher : Summer & Köning

The first thing we may notice is that the publisher is now displayed as well. To this end we've changed the author template so that it applies to publishers too. This is done in:

```
<xsl:template match="author | publisher">
```

And in the `book` template we of course have to apply the `publisher` template as well:

```
<td><xsl:apply-templates select="publisher"/></td>
```

Then we may notice that there is "fixed" text displayed in front of authors and publishers, namely "author :" and "publisher :". "author" and "publisher" are the names of the current XPath nodes, which you include in the output via:

```
<xsl:value-of select="name()"/>
```

In the `printTheName` template the colon is added.

But the most important change is the reversal of surname and first name. Within the `<xsl:variable name="name">` node we switch substrings. The output generated by the `select` statements is put in the `name` variable automatically simply because this output is created within the `variable` node.

In the `choose` node we have a sort of IF-THEN-ELSE, implemented here as `when` and `otherwise`. `<xsl:when test="contains(., ',')">` means if the current XML node content contains a comma, then execute:

```
<xsl:value-of select="substring-after(., ',')"/>
&#xa0;
<xsl:value-of select="substring-before(., ',')"/>
```

First the current content substring behind the comma is send to output (the first name), then a space is inserted in the output (`&#xa0`), then the last name is extracted and placed behind the first name and the single space. Note that functions are always put in the "value" part of a `select` statement.

If the author name or publisher name contains no comma, then no switch can be performed, so the `otherwise` part is executed: the entire node content is send to output (here, to the `name` variable).

Then the `printTheName` template is called with a parameter. The parameter `nameParameter` is filled with the value from the `name` variable; the `$` in front of `name` retrieves the value.

In the `printTheName` template first the parameter is declared. Then, in `<xsl:value-of select="$nameParameter"/>` the value from `nameParame-`

`ter`, which was assigned when this template was called, is send to output (the HTML file, not the `name` variable).
Note that variables are local within a template, so the above illustrates how to pass on the value from a variable to another template.

A simpler solution might have been to output the name variable from the `author | publisher` template directly, without needing the `printTheName` template at all:

```
:<xsl:value-of select="$name"/>
```

Further note that `apply-templates` is used to apply the named template to all elements with this name in the XML file, while `call-template` calls a template which has no matching XML node.

■ Adlib and Axiell Collections parameters

When Adlib (*adlwin.exe*) generates XML for output or display which will be formatted by an XSLT stylesheet, it passes a number of parameters (aka system variables) to the stylesheet. You can use these parameters and the values contained in them to enhance the functionality of your XSLT stylesheets. The available parameters are the following: namely:

- **`ui_language`** – the current user interface language as referenced in Adlib. For example, English is 0, while Dutch is 1. This parameter can be used in output/export formats and in presentation formats for web browser boxes.
- **`data_language`** – the currently selected data language as an IETF language tag. Examples of these IETF language codes are: `'en-GB'`, `'en-US'`, `'nl-NL'`, `'de-DE'`, `'fr-FR'`. This parameter can be used in output/export formats and in presentation formats for web browser boxes.
- **`background_color`** – the background color of the screen as a hexadecimal HTML colour code (#rrggbb). This parameter can only be used in web browser boxes.
- **`retrievalPath`** - will contain the path or URL as set in the Adlib Designer *Retrieval path* option of an image field in the data dictionary. This parameter can be used in output/export formats and in presentation formats for web browser boxes.
- **`thumbnailRetrievalPath`** - will contain the path or URL as set in the Adlib Designer *Thumbnail retrieval path* option of an image field in the data dictionary. This parameter can be used in output/export formats and in presentation formats for web browser boxes.

- **`baseURL`** - will contain the path to the current Adlib application folder (the folder containing the *adlib.pbk* file). This parameter can be used in output/export formats and in presentation formats for web browser boxes.
- **`userName`** - the login name of the current Adlib user. This parameter can be used in output/export formats and in presentation formats for web browser boxes.
- **`language`** – the IETF language tag of the current user interface language of Microsoft Office, e.g. '`en-US'` or '`nl-NL'`. This parameter is generated by the Adlib Office Connect plugin and can only be used in Adlib Office Connect presentation formats.

In Axiell Collections however, currently only four of these parameters are available, one of them even implemented differently, namely:

- **`ui_language`** – the current user interface language as it is active in Axiell Collections. Contrary to the adlwin implementation, here the parameter contains a standard two-letter language code, like `en` for English, `nl` for Dutch, `fr` for French, `de` for German etc. This parameter can be used in output formats.
- **`data_language`** – the currently selected data language as an IETF language tag. Examples of these IETF language codes are: `'en-GB'`, `'en-US'`, `'nl-NL'`, `'de-DE'`, `'fr-FR'`. This parameter can be used in output formats.
- **`retrievalPath`** - will contain the path or URL as set in the Adlib Designer *Retrieval path* option of an image field in the data dictionary. This parameter can be used in output formats if the path is a full URL. (So you need an image web server to allow printing of images.)
- **`thumbnailRetrievalPath`** - will contain the path or URL as set in the Adlib Designer *Thumbnail retrieval path* option of an image field in the data dictionary. This parameter can be used in output formats if the path is a full URL. (So you need an image web server to allow printing of images.)

To use the parameters in a stylesheet, declare them as a regular XSLT parameter without a default value (because it will be overwritten anyway) somewhere in the file, for example:

```
<xsl:param name="data_language”></xsl:param>
<xsl:param name="ui_language"></xsl:param>
```

It's up to you to choose which ones to use in your stylesheets. More information about these parameters and examples of their application can be found in chapters 2.2, 2.2.2 , 4.1.1 and 5.2.1 .

1.2.7 Getting an example of the generated Adlib XML

Whenever you're about to create an XSLT stylesheet for Adlib or Collections data you need to know what the generated XML looks like. This is especially relevant since currently (December 2017), Adlib grouped XML and unstructured XML are different from the grouped and unstructured XML generated by Collections… In the following chapters you'll find an explanation of the different types of XML you can expect and many examples, but if you're still unsure and there's no obvious way to view the generated XML (like in Adloan), you may create the following very small stylesheet to output the actual generated XML without transforming it to anything else, giving you a good example to work with:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ver-
sion="1.0">
  <xsl:template match="/">
    <textarea rows="70" cols="90">
      <xsl:copy-of select="/" />
    </textarea>
  </xsl:template>
</xsl:stylesheet>
```

or, using a different method, to output the generated XML within the base `<adlibXML>` node, as part of an empty HTML document:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ver-
sion="1.0">
  <xsl:output method="html" />
  <xsl:template match="/adlibXML">
    <html>
      <head>
        <title>Get Axiell Collections output XML</title>
      </head>
      <body>
        <xmp>
          <xsl:copy-of select ="*"/>
        </xmp>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

1.2.8 Best practice

You should always start by creating a template (match) for the XPath root node, in the example above this is */booklist*. This is your main template from which you should select other templates when appropriate.

You could arrange all other templates by size, the smallest ones at the bottom of the stylesheet, increasing in size towards the top. But a grouping based on "procedural" templates versus non-procedural ones would also make sense.
Template sizes should be kept as small as possible. Preferably, any template should be able to fit on your screen entirely. You can achieve this by optimizing any functionality.

For transformation to HTML, it is recommended to use a CSS to specify character layout styles like fonts and colours. This keeps you XSLT stylesheet more clean, and changes in layout styles are easier to implement in a CSS stylesheet anyway.

Comment your XSLT stylesheet as much as possible, with: `<!-- my comments -->` Comments cannot be nested, so if you want to "comment out" a large piece of code which already has comments in it, use a `when test="0"` around it.

1.2.9 Other uses of XML and XSLT

Through a so-called gateway it's possible to restructure queries made in Adlib to fit the syntax of third-party database software. The gateway then accesses such a database over the internet, for instance via HTTP or through SRU. When the search result comes back as XML, it is probably not Adlib XML. However, by using XSLT stylesheets in the gateway, it is possible to transform the foreign XML to Adlib XML, which is then send back to the Adlib application where the data is ready to be derived into the Adlib database. This way, foreign databases can be accessed as if they were "friendly" Adlib databases.

1.2.10 More information

For more information about XPath, see a third-party manual or the internet, for example: http://www.w3.org/TR/xpath

2 Creating output formats

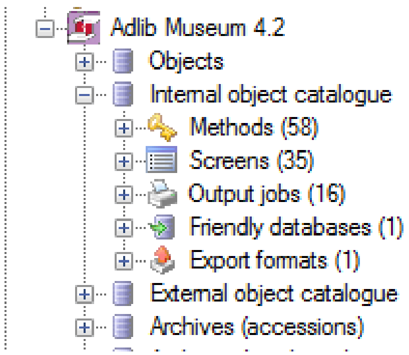
2.1 Grouped XML for XSLT export/output formats

When you have marked one or more records in Adlib, you can choose to print them via a standard or custom output format (available via the *File > Output formats* menu) or to export them to a standard or custom export format (available as export types through the *Export wizard* in Adlib). In Axiell Collections you can use the printer icon in the top toolbar, to print either all records from the result set, all marked records or just the currently selected record.

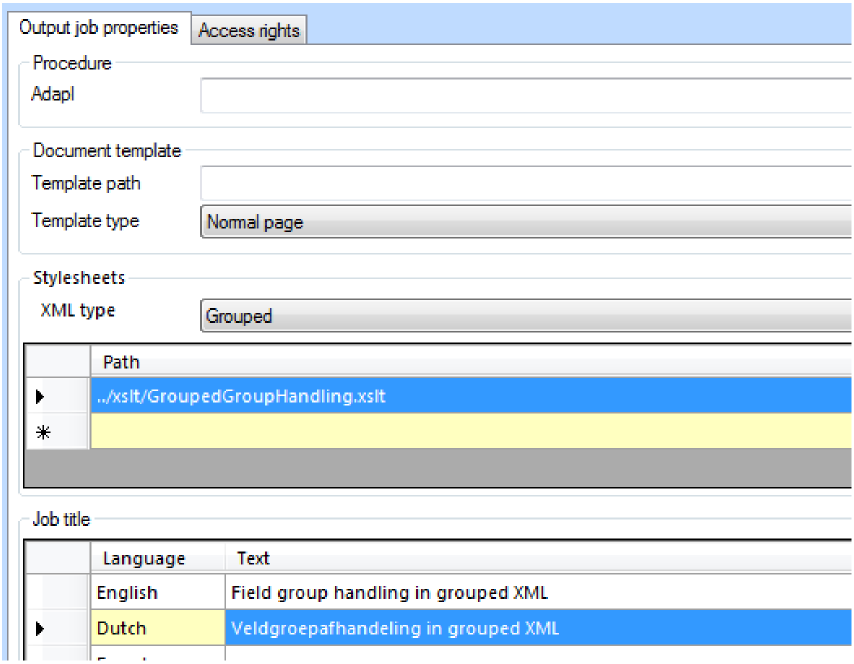
One way to create a custom output format or export format is to build an appropriate XSLT stylesheet. Adlib, as well as Axiell Collections, internally processes records as XML and when you execute an XSLT export format or XSLT output format, this XML is passed on to the stylesheet which converts the XML to the desired format: this target format would need to be HTML if it concerns an output format, or any desired format (XML, HTML, plain text, etc.) if it concerns an export format.
Until Adlib 7.1, the source Adlib XML format passed on to XSLT stylesheets for printing and exporting was of the *unstructured* type by default. Since this type has its drawbacks, Adlib 7.1 (and higher) is capable of passing on Adlib XML of the *grouped* type as well. Axiell Collections can generate either unstructured XML or grouped XML, although currently (December 2017) Adlib for Windows and Axiell Collections do not produce the same unstructured and grouped XML, unfortunately, so you can't create an XSLT output format that works in both environments. Which XML type must be generated by Adlib or Collections, can be set per XSLT *Output job* or *Export format* via Adlib Designer 7.1 or higher.

2.1.1 Setting the XML type in Designer

Output jobs (aka output formats or print formats) as well as *Export formats* (not to be confused with *Export jobs* which consist of the properties of an actual export procedure that can be run from within the *Export job editor*), are registered per data source (like the *Internal object catalogue* for example) underneath an application definition (like that of *Adlib Museum 4.2*) in the Application browser of Adlib Designer.



The XML type for an output job can be set in the *XML type* option on the *Output job properties* tab of a selected output job. See the De-signer Help for more information about setting up output jobs.)



Procedure
Adapl

Document template
Template path
Template type: Normal page

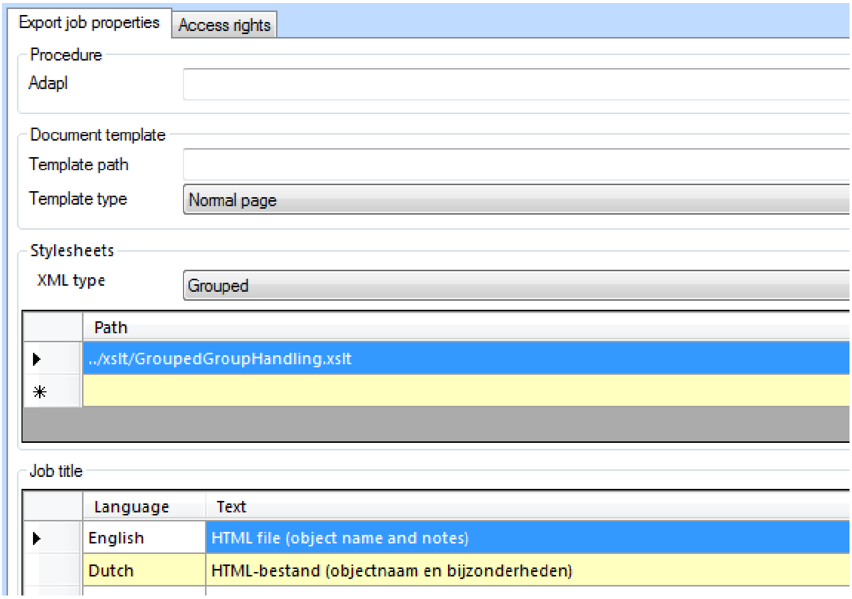
Stylesheets
XML type: Grouped

	Path
▸	../xslt/GroupedGroupHandling.xslt
✳	

Job title

	Language	Text
	English	Field group handling in grouped XML
▸	Dutch	Veldgroepafhandeling in grouped XML

The XML type for an export format can be set in the *XML type* option on the (inaptly named) *Export job properties* tab of a selected export format. See the Designer Help for more information about setting up export formats.)



Procedure

Adapl	

Document template

Template path	
Template type	Normal page

Stylesheets

XML type	Grouped

	Path
▶	../xslt/GroupedGroupHandling.xslt
✳	

Job title

	Language	Text
▶	English	HTML file (object name and notes)
	Dutch	HTML-bestand (objectnaam en bijzonderheden)

As mentioned, earlier created export formats and output formats always had to be based on unstructured XML. From Designer 7.1, unstructured XML will always be assumed for pre-existing formats so you won't have to change anything to your existing export formats and output formats, nor to your existing XSLT stylesheets. For new XSLT export formats and output formats, the option will default to *Grouped* though.

2.1.2 Advantages of grouped XML for use in stylesheets

The main advantage of the grouped type over the unstructured one is that it becomes easier to process repeated occurrences of grouped fields. In unstructured Adlib XML, all fields and field occurrences are just listed in one long list inside the `<record>` node, whilst in grouped Adlib XML, fields are grouped within a field group node (if a relevant field group exists in the data dictionary) and that field group node is repeated for each field group occurrence.
Whenever you create an XSLT stylesheet for unstructured Adlib XML, which must be able to collect field data per field group occurrence, you have no choice but to always count the "position" of every processed field occurrence because that's the only way to retrieve the other fields from the same position. In grouped Adlib XML on the other hand, there's no need for such a workaround because every field group occurrence is contained within its own field group node. Matching an XSLT template to a field group node automatically provides

access to all grouped fields with the same occurrence number (in other words: at the same position).

2.1.3 Examples

Suppose you wish to create an output format based on an XSLT stylesheet, to print the object name(s) and the notes pertaining to the object name, of a museum object. The `object_name` and `object_name.notes` fields, as specified in the data dictionary of the *Collect* database, are part of a field group called `Object_name`. Because of this grouping you can repeat these two fields (and the others belonging to the group) as a group in the Adlib record. When you print these group repetitions, you will want to keep them grouped of course: you don't want a list of all object names followed by a list of all notes.

For unstructured Adlib XML you would have to tackle this problem as follows:

```
<?xml version="1.0" encoding="utf-8"?>
  <xsl:stylesheet xmlns:xsl=http://www.w3.org/1999/XSL/Transform
   version="1.0">
    <xsl:output method="html" />
    <xsl:template match="/adlibXML">
      <html>
        <head>
          <meta http-equiv="X-UA-Compatible" content="IE=edge" />
          <title>Field group handling for unstructured XML</title>
        </head>
        <body>
          <xsl:apply-templates select="recordList/record"/>
        </body>
      </html>
    </xsl:template>

    <xsl:template match="record">
      <xsl:apply-templates select="object_name"/>
    </xsl:template>

    <xsl:template match="object_name">
      <xsl:variable name="pos" select="position()" />
      <p>
        <xsl:value-of select="."/>
        <br/>
        <xsl:apply-templates select="../object_name.notes[$pos]"/>
        <br/>
      </p>
    </xsl:template>

    <xsl:template match="object_name.notes">
         <xsl:value-of select="."/>
    </xsl:template>

  </xsl:stylesheet>
```

For grouped Adlib XML on the other hand, you could code this as shown below:

```
<?xml version="1.0" encoding="utf-8"?>
  <xsl:stylesheet xmlns:xsl=http://www.w3.org/1999/XSL/Transform
   version="1.0">
    <xsl:output method="html" />
    <xsl:template match="/adlibXML">
      <html>
        <head>
          <meta http-equiv="X-UA-Compatible" content="IE=edge" />
          <title>Field group handling for grouped XML</title>
        </head>
        <body>
          <xsl:apply-templates select="recordList/record"/>
        </body>
      </html>
    </xsl:template>

    <xsl:template match="record">
      <xsl:apply-templates select="Object_name"/>
    </xsl:template>

    <xsl:template match="Object_name">
      <p>
        <xsl:apply-templates select="object_name"/>
        <br/>
        <xsl:apply-templates select="object_name.notes"/>
        <br/>
      </p>
    </xsl:template>

    <xsl:template match="object_name">
      <xsl:value-of select="term"/>
    </xsl:template>

    <xsl:template match="object_name.notes">
         <xsl:value-of select="."/>
    </xsl:template>

  </xsl:stylesheet>
```

The output of either stylesheet is structured like this:

object name in field group occurrence 1 of record 1
object name notes in field group occurrence 1 of record 1

object name in field group occurrence 2 of record 1
object name notes in field group occurrence 2 of record 1

object name in field group occurrence 3 of record 1
object name notes in field group occurrence 3 of record 1

object name in field group occurrence 1 of record 2
object name notes in field group occurrence 1 of record 2

object name in field group occurrence 2 of record 2
object name notes in field group occurrence 2 of record 2

…

2.2 Printing images via an XSLT stylesheet

The image reference (aka reproduction reference) in your Adlib records does usually not consist of a full path to an image file. In 3.4 applications for example, it is a relative path by default, like *..\images\BM0034.jpg* (relative to the application folder). In 4.2 applications on the other hand, by default a storage/retrieval path has been set for the image field, so that only the image file name is present in the image reference field. Since the HTML output we would like to generate requires a URL to retrieve an image, we need to find a way to combine the image file name from the image reference field with the URL to the images folder (a file system path won't do). A wwwopac.ashx call to an image server, as is often used as storage/retrieval path for image fields in modern Adlib applications and Axiell Collections is exactly what we need. To get this base URL in your stylesheet, Adlib (adlwin.exe-based) applications offer more automation than Axiell Collections currently does, so if you're creating a stylesheet which must function in both enviroments you'll have to use the base method, which is to hard code the base URL in your stylesheet and concatenate it with the image file name, like in the following example (for grouped XML):

```
<xsl:template match="Reproduction">
  <xsl:variable name="imageFileName">
      <xsl:value-of select="reproduction.reference"/>
  </xsl:variable>
  <xsl:variable name="imagePath">
      <xsl:text>http://ourserver.com/images/wwwopac.ashx?
command=getcontent&amp;server=images&amp;value=</xsl:text>
      <xsl:value-of select="$imageFileName"/>
  </xsl:variable>
  <p>
      <img border="0" width="340" src="{$imagePath}" />
  </p>
</xsl:template>
```

As you can see, this template matches the `Reproduction` field group. It fills a new `imageFileName` variable with the (first) linked image file name from the `reproduction.reference` field. Next, another new variable named `imagePath` is created and filled with the base URL to our image server after which the image file name is pasted behind it.

And finally the contents of the `imagePath` variable is used as the `src` attribute of the HTML `img` tag (to retrieve the image in the resulting HTML page).

For an enterprise solution, in which images for the different branches are stored in their own folders, you can still use a single image server: with the `<folderMappingsList>` settings (introduced in October 2016) in adlibweb.xml, you can specify these different folders. In such case you need to extend your wwwopac.ashx call with the `folderId` parameter which must be assigned the record number of the currently processed, linked media record. The above example can then be adapted to the following:

```
<xsl:template match="Reproduction">
  <xsl:variable name="imageFileName">
      <xsl:value-of select="reproduction.reference"/>
  </xsl:variable>
  <xsl:variable name="imageRecordLref">
      <xsl:value-of select="reproduction.reference.lref"/>
  </xsl:variable>
  <xsl:variable name="imagePath">
      <xsl:text>http://ourserver.com/images/wwwopac.ashx?
command=getcontent&amp;server=images&amp;value=</xsl:text>
      <xsl:value-of select="$imageFileName"/>
      <xsl:text>&amp;folderId=</xsl:text>
      <xsl:value-of select="$imageRecordLref"/>
    </xsl:variable>
    <p>
      <img border="0" width="340" src="{$imagePath}" />
    </p>
</xsl:template>
```

However, if you're creating an XSLT output format just for use with Adlib applications, you may not have to hard code the base URL in your stylesheet. Prior to Adlib 7.1, when you had to create an XSLT stylesheet to print images referenced this way in Adlib records, you had to hard-code the path or URL to your images folder in your stylesheet and combine it with the image reference from records, to be able to provide the HTML output with full paths to image files. From Adlib 7.1 though, you'll no longer have to hard-code a path to the images folder into your XSLT stylesheets. Instead, three new Adlib parameters (system variables) are available for use in XSLT stylesheets: `retrievalPath`, `thumbnailRetrievalPath` and `baseUrl`. When you print selected records from Adlib using an XSLT output format, Adlib will pass the relevant path in the appropriate parameters to the stylesheet:

- **`retrievalPath`**: will contain the path or URL as set in the Adlib Designer *Retrieval path* option of an image field in the data dictionary. (See the Designer Help for more information about this option.) If the option has not been set, the path set in the *Storage path* option above it will be used instead. If neither has been set, as is often the case in 3.4 applications and older, the parameter will be empty.

- **`thumbnailRetrievalPath`**: will contain the path or URL as set in the Adlib Designer *Thumbnail retrieval path* option of an image field in the data dictionary. If the option has not been set, the parameter will be empty.
- **`baseURL`**: will contain the path to the current Adlib application folder (the folder containing the *adlib.pbk* file).

It's up to you to choose which ones to use in your stylesheets. Stylesheets for 3.4 applications or older will usually only require the `baseURL` parameter, while newer applications require as least the `retrievalPath` and possibly the `baseURL`.
If you don't want users to be able to print your high resolution images and you have thumbnail images available in a separate folder set up in the Adlib Designer *Thumbnail retrieval path* option of an image field in the data dictionary, then you may use the `thumbnailRetrievalPath` parameter instead of the `retrievalPath` parameter.

2.2.1 Example Adlib output formats

Below you can see two examples of complete XSLT stylesheets (made for *unstructured* XML) for printing some object record data plus a linked image from within Adlib. (You can also download them here.) From each record, the object number, the title, the creator(s) (maximally two) and the object name(s) (maximally three) and only the first linked image will be printed. Exactly five records should fit on a single A4 page: images will be scaled to a fixed height.

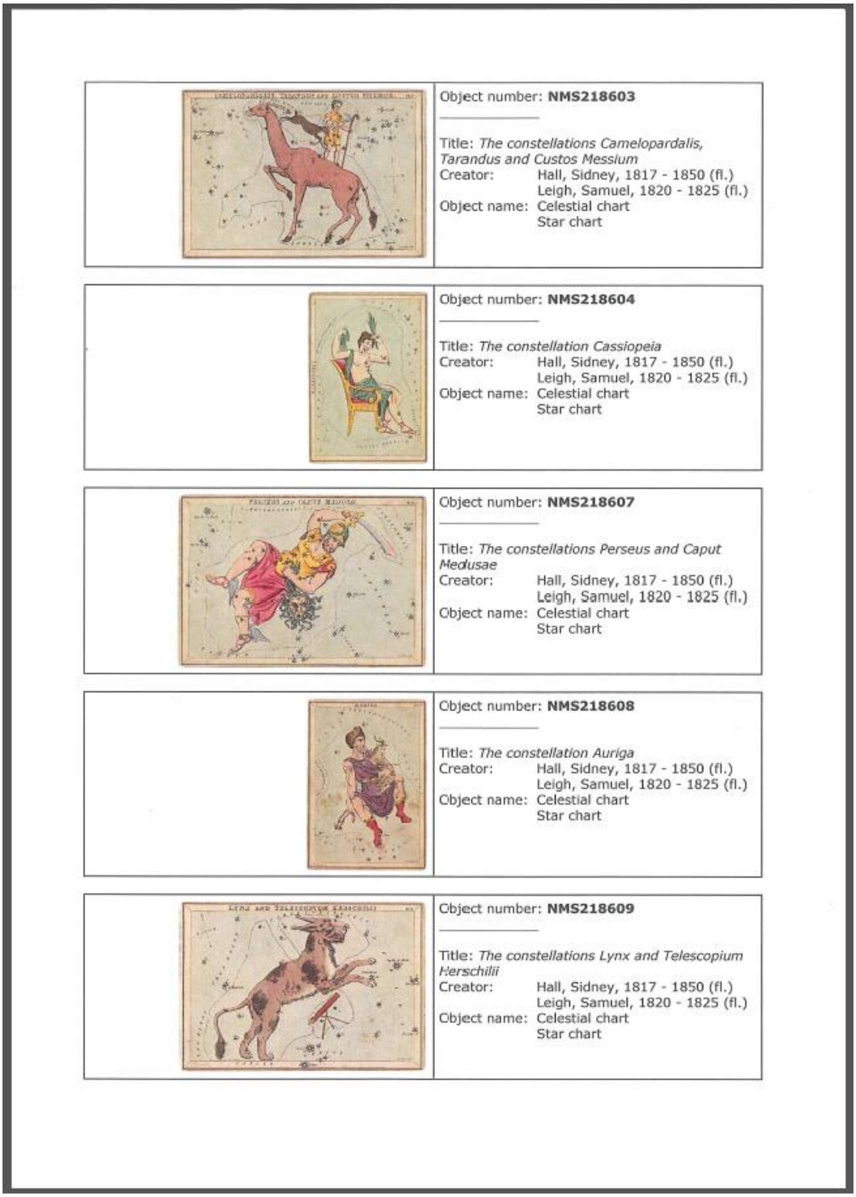
First a code example for a 4.2 model application in which the *Storage path* option for the *reproduction.reference* image field (tag *FN* in *Collect*) has been set to a relative path (like `../images/%data%`):

```
<?xml version="1.0" encoding="utf-8"?>
  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
    <xsl:param name="retrievalPath"/>
    <xsl:param name="baseUrl"/>
    <xsl:output method="html" />
    <xsl:template match="/adlibXML">
      <html>
        <head>
          <meta http-equiv="X-UA-Compatible" content="IE=edge" />
          <title>List of objects</title>
          <style type="text/css">
            .text
            {
```

```
            font-family: Verdana;
            font-size: x-small;
            }
            .table
            {
            border: solid 1px black;
            border-collapse: collapse;
            }
          </style>
        </head>
        <body>
            <xsl:apply-templates select="recordList/record"/>
        </body>
      </html>

    </xsl:template>


   <xsl:template match="record">
      <table width="700px" border="1" cellspacing="0"
             cellpadding="5" class="table">
        <tr valign="top" border="0">
          <td align="right" width="350">
          <xsl:apply-templates select="reproduction.reference[1]"/>
          </td>
          <td valign="top" class="text">
            <p>
              Object number:
              <b>
                <xsl:apply-templates select="object_number"/>
              </b>
              ____________ <br/><br/>
              Title:
              <xsl:apply-templates select="title"/>
              <table cellspacing="0" cellpadding="0">
                <tr valign="top">
                  <td width="100" class="text">Creator:</td>
                  <td class="text">
                    <xsl:apply-templates select="creator"/>
                  </td>
                </tr>
                <tr valign="top">
                  <td width="100" class="text">Object name:</td>
                  <td class="text">
                     <xsl:apply-templates select="object_name"/>
                  </td>
                </tr>
              </table>
            </p>
          </td>
        </tr>
      </table>
      <br/>
      <xsl:if test="position() mod 5 = 0">
        <p style="page-break-before:always" />
      </xsl:if>
    </xsl:template>

    <xsl:template match="title">
        <i>
          <xsl:value-of select="."/>
        </i>
      <br/>
    </xsl:template>

    <xsl:template match="object_name">
      <xsl:if test="position() &lt; 4">
        <xsl:value-of select="."/>
        <br/>
      </xsl:if>
    </xsl:template>


    <xsl:template match="creator">
      <xsl:if test="position() &lt; 3">
        <xsl:value-of select="."/>
        <br/>
      </xsl:if>
    </xsl:template>

    <xsl:template match="object_number">
        <xsl:value-of select="."/>
      <br/>
    </xsl:template>

    <xsl:template match="reproduction.reference">
        <xsl:variable name="imagePath">
          <xsl:call-template name="replace-string">
            <xsl:with-param name="text" select="$retrievalPath"/>
            <xsl:with-param name="replace" select="'%data%'"/>
            <xsl:with-param name="with" select="."/>
          </xsl:call-template>
        </xsl:variable>
      <p>
      <img border="0" height="180" src="{$baseUrl}\{$imagePath}" />
      </p>
      <p>
      </p>
    </xsl:template>

    <xsl:template name="replace-string">
      <xsl:param name="text"/>
      <xsl:param name="replace"/>
      <xsl:param name="with"/>
      <xsl:choose>
        <xsl:when test="contains($text,$replace)">
          <xsl:value-of select="substring-before($text,$replace)"/>
          <xsl:value-of select="$with"/>
          <xsl:call-template name="replace-string">
```

```
            <xsl:with-param name="text"
             select="substring-after($text,$replace)"/>
            <xsl:with-param name="replace" select="$replace"/>
            <xsl:with-param name="with" select="$with"/>
          </xsl:call-template>
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="$text"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:template>

  </xsl:stylesheet>
```

The result of printing to this stylesheet (or the following) will look something like this:



Object number: **NMS218603**

Title: *The constellations Camelopardalis, Tarandus and Custos Messium*
Creator: Hall, Sidney, 1817 - 1850 (fl.)
Leigh, Samuel, 1820 - 1825 (fl.)
Object name: Celestial chart
Star chart

Object number: **NMS218604**

Title: *The constellation Cassiopeia*
Creator: Hall, Sidney, 1817 - 1850 (fl.)
Leigh, Samuel, 1820 - 1825 (fl.)
Object name: Celestial chart
Star chart

Object number: **NMS218607**

Title: *The constellations Perseus and Caput Medusae*
Creator: Hall, Sidney, 1817 - 1850 (fl.)
Leigh, Samuel, 1820 - 1825 (fl.)
Object name: Celestial chart
Star chart

Object number: **NMS218608**

Title: *The constellation Auriga*
Creator: Hall, Sidney, 1817 - 1850 (fl.)
Leigh, Samuel, 1820 - 1825 (fl.)
Object name: Celestial chart
Star chart

Object number: **NMS218609**

Title: *The constellations Lynx and Telescopium Herschilii*
Creator: Hall, Sidney, 1817 - 1850 (fl.)
Leigh, Samuel, 1820 - 1825 (fl.)
Object name: Celestial chart
Star chart

Some comments about the code:

- Note the declaration of the Adlib parameters `retrievalPath` and `baseURL` at the top. You'll have to add these declarations to your stylesheets yourself.
- The `<style>` node specifies two CSS styles, one for the text to be printed and one for the some properties of the border of each table in which a record is to be printed.
  Note that if you choose to set a background colour for the table or one of its cells, it won't be printed by default, because of a default page setting for not printing background colours, in Internet Explorer.
- A record will be printed in a table within a table. The outer one has one row of two cells. The right cell will contain the object number, the title and another two by two table. The first column of the inner table will hold the labels for *Creator:* and *Object name:* whilst the second will contain possible multiple occurrences of these fields. The left cell of the main table will contain the image itself.
- To control the space occupied on the page by each printed record, the height of the image is fixed to 180 pixels: it will be scaled automatically while maintaining the aspect ratio. If too much text would be printed - this can happen if you add fields – the height of a single table will expand and five records will no longer fit on a page, breaking up your printout. To apply to most cases, this example stylesheet limits the text by allowing maximally two creator names and three object names to be printed: see the `<xsl:if test…` nodes in the `object_name` and `creator` templates. Of linked images only the first one will be printed, which is achieved by calling the `reproduction.reference` template only for the first occurrence (indicated by the `[1]` behind it).
- The page break is forced at the end of the `record` template, after every five records. The test "`position() mod 5 = 0`" divides the sequential number of the record in the selection by 5 and becomes `true` if zero remains behind the decimal. (If you were to replace 5 by 4 for example, the test becomes `true` after every 4 records.) The page break will then be forced by the CSS style "`page-break-before:always`" which is assigned to the HTML `<p>` tag in this case. (Note that you can't use this style within tables.)
- The `reproduction.reference` template, in combination with the `replace-string` template, handles the image printing. The `replace-string` template first replaces the `%data%` string in the retrieval path with the actual image reference from the record. As-

suming the retrieval path is a relative path, the resulting path is passed to the `imagePath` variable in the `reproduction.reference` template. Then an HTML `img` element is created in which the image is assigned its maximum height and its source path consisting of the `baseURL` concatenated with the `imagePath`. Note that adding a relative path like *..\images\img45.jpg* to a base path like *C:\Adlib\Museum* is perfectly well allowed. This would automatically form the following path: *C:\Adlib\images\img45.jpg*.
If your retrieval or storage path is a full URL or UNC path, you must not add the `baseURL` to it: then the `imagePath` is all you need.

Next an example for a 3.4 model application (again for unstructured XML, with identical resulting output) in which no *Storage path* or *Retrieval path* has been set and records contain a relative path to an image (like *../images/img2349.jpg*) in the *reproduction.identifier_URL* image field (tag *B1* in *Collect*). (The direction of the slashes in the image reference doesn't matter.)
The only difference here is how the full path to the image file is put together. We actually just need to add the image reference from the record to the `baseUrl`, and that's it (see the `reproduction.identifier_URL` template).

```
<?xml version="1.0" encoding="utf-8"?>
  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
    <xsl:param name="baseUrl"/>
    <xsl:output method="html" />
    <xsl:template match="/adlibXML">
      <html>
        <head>
          <meta http-equiv="X-UA-Compatible" content="IE=edge" />
          <title>List of objects</title>
          <style type="text/css">
            .text
            {
            font-family: Verdana;
            font-size: x-small;
            }
            .table
            {
            border: solid 1px black;
            border-collapse: collapse;
            }
          </style>
        </head>
        <body>
            <xsl:apply-templates select="recordList/record"/>
        </body>
      </html>
```

```
</xsl:template>

<xsl:template match="record">
  <table width="700px" border="1" cellspacing="0"
   cellpadding="5" class="table">
    <tr valign="top" border="0">
      <td align="right" width="350">
        <xsl:apply-templates
         select="reproduction.identifier_URL[1]"/>
      </td>
      <td valign="top" class="text">
        <p>
          Object number:
          <b>
            <xsl:apply-templates select="object_number"/>
          </b>
          ____________ <br/><br/>
          Title:
          <xsl:apply-templates select="title"/>
          <table cellspacing="0" cellpadding="0">
            <tr valign="top">
              <td width="100" class="text">Creator:</td>
              <td class="text">
                <xsl:apply-templates select="creator"/>
              </td>
            </tr>
            <tr valign="top">
              <td width="100" class="text">Object name:</td>
              <td class="text">
                 <xsl:apply-templates select="object_name"/>
              </td>
            </tr>
          </table>
        </p>
      </td>
    </tr>
  </table>
  <br/>
  <xsl:if test="position() mod 5 = 0">
    <p style="page-break-before:always" />
  </xsl:if>
</xsl:template>

<xsl:template match="title">
    <i>
      <xsl:value-of select="."/>
    </i>
  <br/>
</xsl:template>

<xsl:template match="object_name">
  <xsl:if test="position() &lt; 4">
    <xsl:value-of select="."/>
    <br/>
  </xsl:if>
```

```
    </xsl:template>
    <xsl:template match="creator">
      <xsl:if test="position() &lt; 3">
        <xsl:value-of select="."/>
        <br/>
      </xsl:if>
    </xsl:template>

    <xsl:template match="object_number">
      <xsl:value-of select="."/>
      <br/>
    </xsl:template>

    <xsl:template match="reproduction.identifier_URL">
      <p>
       <img border="0" height="180" src="{$baseUrl}\{.}" />
      </p>
      <p>
      </p>
    </xsl:template>

  </xsl:stylesheet>
```

2.2.2 Example Axiell Collections output format

Below you can see an example of a complete XSLT stylesheet (made for *grouped* XML) for printing some object record data plus a linked image from within Axiell Collections.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:output method="html" />
  <xsl:param name="data_language"/>

  <xsl:template match="/adlibXML">
    <html>
      <head>
        <meta http-equiv="X-UA-Compatible" content="IE=edge" />
        <title>Object report</title>
        <style type="text/css">
          .text
          {
          font-family: Verdana;
          font-size: x-small;
          vertical-align: top;
          }
          .longtext
          {
          font-family: Verdana;
          font-size: x-small;
          vertical-align: top;
          white-space: pre-wrap;
          }
          .titletext
          {
          font-family: Verdana;
          font-weight: bold;
          font-size: large;
          }
          .headertext
          {
          font-family: Verdana;
          font-weight: bold;
          font-size: x-small;
          }
          .outertable
          {
          border: solid 1px black;
          border-collapse: collapse;
          }
          .innertable
          {
          border: solid 0px;
          border-collapse: collapse;
          }
        </style>
      </head>
      <body>
        <p class="titletext">
          Object report
        </p>
        <xsl:apply-templates select="recordList/record"/>
      </body>
    </html>

  </xsl:template>

  <xsl:template match="record">
    <table width="700px" border="1" cellspacing="0" cellpadding="5"
           class="outertable">
      <tr valign="top" border="0">
        <td align="right" width="350">
          <xsl:apply-templates select="Reproduction[1]"/>
        </td>
        <td valign="top" class="text">
          <p class="headertext">
            Identification
          </p>
          <table width="340px" border="0" cellspacing="0"
                 cellpadding="0" class="innertable">
            <tr valign="top">
              <td width="100" class="text">Object number:</td>
              <td class="text">
                <xsl:apply-templates select="object_number"/>
              </td>
            </tr>
            <tr valign="top">
              <td width="100" class="text">Title:</td>
              <td class="text">
                <xsl:apply-templates select="Title/title/value
                 [@lang=$data_language]|Title/title/value
                 [@lang='']"/>
              </td>
            </tr>
            <tr valign="top">
              <td width="100" class="text">Object category:</td>
              <td class="text">
                <xsl:apply-templates select="object_category/value
                 [@lang=$data_language]|object_category/value
                 [@lang='']"/>
              </td>
            </tr>
            <tr valign="top">
              <td width="100" class="text">Object name:</td>
              <td class="text">
              <xsl:apply-templates select="Object_name/object_name/
               value[@lang=$data_language]|Object_name/object_name/
               value[@lang='']"/>
              </td>
            </tr>
          </table>

          <p class="headertext">
            Production
          </p>
          <table width="340px" border="0" cellspacing="0"
                 cellpadding="0" class="innertable">
            <tr>
              <td width="100" class="text">Creator:</td>
              <td class="text">
              <xsl:apply-templates select="Production/creator/value
               [@lang=$data_language]|Production/creator/
               value[@lang='']"/>
              </td>
            </tr>
            <tr>
              <td width="100" class="text">Dating:</td>
              <td class="text">
                <xsl:apply-templates select="Production_date"/>
              </td>
            </tr>
            <tr>
              <td width="100" class="text">Production place:</td>
              <td class="text">
                <xsl:apply-templates select="Production/
                 production.place/value[@lang=$data_language]|
                 Production/production.place/value[@lang='']"/>
              </td>
            </tr>
            <tr>
              <td width="100" class="text">Production notes:</td>
              <td class="text">
                <xsl:apply-templates select="Production/
                 production.notes"/>
              </td>
            </tr>
          </table>

          <p class="headertext">
            Object history
          </p>
          <table width="340px" border="0" cellspacing="0"
                 cellpadding="0" class="innertable">
            <tr>
              <td class="text">
                <xsl:apply-templates select="object_history_note"/>
              </td>
            </tr>
          </table>
        </td>
      </tr>
    </table>

    <p class="headertext">
      <text>Description</text>
    </p>
    <p class="longtext">
      <xsl:apply-templates select="Description/description"/>
    </p>

    <table width="700px" border="1" cellspacing="0" cellpadding="5"
           class="outertable">
      <tr valign="top" border="0">
        <td align="left" width="350">
          <table width="340px" border="0" cellspacing="0"
           cellpadding="0" class="innertable">
            <tr valign="top">
              <td width="100" class="text">Accession date:</td>
              <td class="text">
                <xsl:apply-templates select="acquisition.date"/>
              </td>
            </tr>
            <tr valign="top">
              <td width="100" class="text">Accession method:</td>
              <td class="text">
                <xsl:apply-templates select="acquisition.method/
                 value[@lang=$data_language]|
                 acquisition.method/value[@lang='']"/>
              </td>
            </tr>
          </table>
        </td>
        <td valign="top" class="text">
          <table width="340px" border="0" cellspacing="0"
                 cellpadding="0" class="innertable">
            <tr valign="top">
              <td width="100" class="text">Institution:</td>
              <td class="text">
```

```
                <xsl:apply-templates select="institution.name/
                 value[@lang=$data_language]|
                 institution.name/value[@lang='']"/>
              </td>
            </tr>
            <tr valign="top">
              <td width="100" class="text">Collection:</td>
              <td class="text">
               <xsl:apply-templates select="collection/value
               [@lang=$data_language]|collection/value[@lang='']"/>
              </td>
            </tr>
          </table>
        </td>
      </tr>
    </table>

    <xsl:if test="position() mod 1 = 0">
      <p style="page-break-before:always" />
    </xsl:if>
  </xsl:template>

  <xsl:template match="object_number">
    <xsl:value-of select="."/>
  </xsl:template>

  <xsl:template match="Title/title/value">
    <xsl:value-of select="."/>
    <br />
  </xsl:template>

  <xsl:template match="object_category/value">
    <xsl:value-of select="."/>
    <br />
  </xsl:template>

  <xsl:template match="Object_name/object_name/value">
    <xsl:value-of select="."/>
    <br />
  </xsl:template>

  <xsl:template match="Production/creator/value">
    <xsl:value-of select="."/>
    <br />
  </xsl:template>

  <xsl:template match="Production_date">
    <xsl:value-of select="production.date.start"/>
    <text> - </text>
    <xsl:value-of select="production.date.end"/>
    <br />
  </xsl:template>

  <xsl:template match="Production/production.place/value">
    <xsl:value-of select="."/>
    <br />
  </xsl:template>


  <xsl:template match="Production/production.notes">
    <xsl:value-of select="."/>
    <br />
  </xsl:template>

  <xsl:template match="object_history_note">
    <xsl:value-of select="."/>
  </xsl:template>

  <xsl:template match="Description/description">
    <xsl:value-of select="."/>
  </xsl:template>

  <xsl:template match="acquisition.date">
    <xsl:value-of select="."/>
    <br />
  </xsl:template>

  <xsl:template match="acquisition.method/value">
    <xsl:value-of select="."/>
    <br />
  </xsl:template>

  <xsl:template match="institution.name/value">
    <xsl:value-of select="."/>
    <br />
  </xsl:template>

  <xsl:template match="collection/value">
    <xsl:value-of select="."/>
    <br />
  </xsl:template>

  <xsl:template match="Reproduction">
    <xsl:variable name="imageFileName">
      <xsl:value-of select="reproduction.reference"/>
    </xsl:variable>
    <xsl:variable name="imagePath">
      <xsl:text>http://ourserver.com/images/wwwopac.ashx?
command=getcontent&amp;server=images&amp;value=</xsl:text>
      <xsl:value-of select="$imageFileName"/>
    </xsl:variable>
    <p>
      <img border="0" width="340" src="{$imagePath}" />
    </p>
    <p>
    </p>
  </xsl:template>

</xsl:stylesheet>
```

2.3 Accessing the current user name in XSLT

Sometimes you may want to include the name of the current user in your XSLT output, the person who starts the printing or exporting. Therefor a parameter has been added to Adlib 7.1 (not available in Axiell Collections), which you can use in your XSLT stylesheets: `userName`. When an XSLT stylesheet is called from within Adlib, `userName` will automatically contain the (login) name of the current user. By declaring the parameter explicitly in the stylesheet, you can access it in the rest of the code, for example to print it somewhere. An example of a rather rudimentary stylesheet which only prints the user name, is the following:

```
<?xml version="1.0" encoding="utf-8"?>
  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
    <xsl:param name="userName"/>
    <xsl:output method="html" />
    <xsl:template match="/adlibXML">
      <html>
        <head>
          <title>Current user</title>
        </head>
        <body>
          <p>
            <xsl:value-of select="$userName"/>
          </p>
        </body>
      </html>
    </xsl:template>
  </xsl:stylesheet>
```

2.4 Printing barcode labels to a normal printer

This chapter offers two examples of XSLT stylesheets to print barcode labels to a normal printer, the first (made for unstructured XML) prints barcode labels containing object images from within Adlib for Windows, while the second (made for grouped XML) prints simple barcode labels to a normal printer from within Axiell Collections. You can print to paper or label sheets. The examples are really just that, very basic examples to show you how you can build such stylesheets yourself.

■ Stylesheet for unstructured XML, to print from within Adlib

```
<?xml version='1.0' ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:output method="html" />
  <!-- 4,5cm is 127 points       -->
```

```
<!-- ===================================================== -->
<xsl:template match="/">
  <html>
    <style>
      div
      {
      font-family:Arial;
      }
      span.barcode
      {
      font-size:14pt;
      font-family:BC39;
      }
      div.top-row
      {
      width:269pt;
      padding-bottom:10pt;
      }
      div.bc-column
      {
      float:right;
      text-align:right;
      }
      div.transit-logo
      {
      vertical-align:bottom;
      padding-top:13pt;
      }
      div.img-column
      {
      float:left;
      }
      div.img-column img
      {
      height:122pt;
      }
      div.lbl-row
      {
      width:269pt;
      border-style:solid;
      border-width:1px;
      padding:5px;
      margin-bottom:10pt;
      }
    </style>
    <body>
      <xsl:apply-templates select="adlibXML" />
    </body>
  </html>
</xsl:template>
<!-- ===================================================== -->
<xsl:template match="adlibXML">
   <xsl:apply-templates select="recordList/record" />
</xsl:template>
<!-- ===================================================== -->
```

```
<xsl:template match="record">
  <div class="top-row">
    <div class="img-column">
    <xsl:apply-templates select="reproduction.reference[1]"/>
    </div>
    <div class="bc-column">
      <div>
        <xsl:apply-templates select="object_number"
         mode ="barcode"/>
      </div>
      <div class="transit-logo">
        <xsl:call-template name ="transit-logo"/>
      </div>
    </div>
  </div>
  <div style="clear:all"/>
  <div class="lbl-row">
    <xsl:apply-templates select="object_number" mode ="text"/>
    <br/>
    <xsl:apply-templates select="object_name"/>
    <!-- = the content of the object_name node will simply = -->
    <!-- = be included in the output, since we define no = -->
    <!-- = object_name template in this stylesheet. The = -->
    <!-- = same applies to the title and current_location. = -->
    <br/>
    <xsl:apply-templates select="title"/>
    <br/>
    <xsl:apply-templates select="current_location"/>
  </div>
</xsl:template>

<xsl:template match ="reproduction.reference">
  <img>
    <xsl:attribute name="src">
      <!--fill in URL Image Handler or full (UNC) Path-->
      <xsl:text>C:\Adlib\Model 4.2\images\</xsl:text>
      <xsl:value-of select="." />
    </xsl:attribute>
  </img>
</xsl:template>

<xsl:template match ="object_number" mode ="barcode">
  <span class="barcode">
    <!-- Barcode 39 needs a start and stop character * -->
    <xsl:text>*</xsl:text>
    <xsl:value-of select ="."/>
    <xsl:text>*</xsl:text>
  </span>
</xsl:template>

<xsl:template match ="object_number" mode ="text">
    <xsl:value-of select ="."/>
</xsl:template>

<xsl:template name ="transit-logo">
```

```
      <xsl:choose>
        <xsl:when test="starts-with(object_name,'m')">
          <img src="C:\Adlib\Model 4.2\xslt\vrachtwagen.png"
           width="51pt"/>
        </xsl:when>
        <xsl:when test="starts-with(object_name,'r')">
          <img src="C:\Adlib\Model 4.2\xslt\vrachtwagen2.png"
           width="51pt"/>
        </xsl:when>
        <xsl:otherwise >
          <img src="C:\Adlib\Model 4.2\xslt\vrachtwagen3.png"
           width="51pt"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:template>
    <!-- ================================================ -->
  </xsl:stylesheet>
```

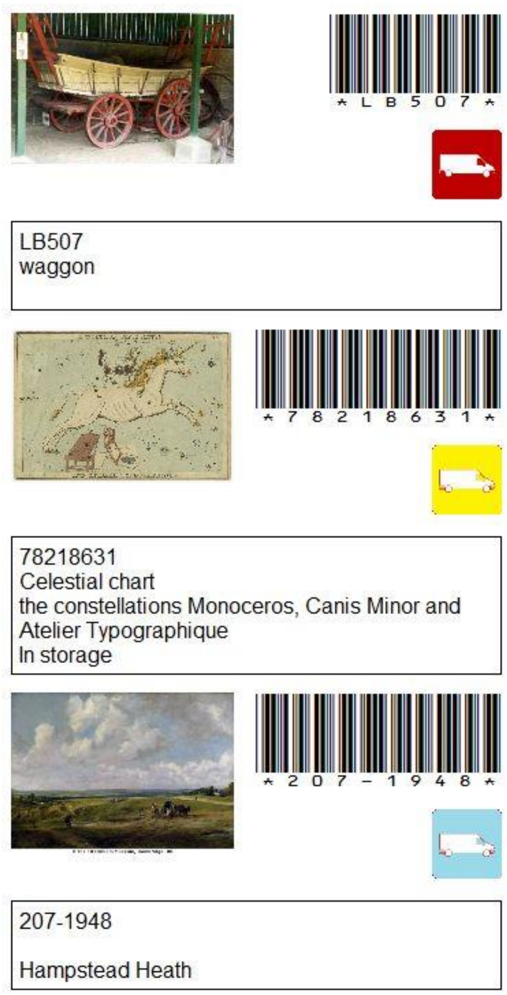
You can change it all you like of course, and to get it working in your Adlib system you do indeed have to make some changes to it.

Proceed as follows:

1. You have to have a barcode font (a Code 39 font for example) installed on the computer from which you want to print. If you haven't got one, you'll have buy such a font first, find a freely available font on the internet or try to install a demo version of a suitable font. (Without a barcode font, the object number itself will be printed on the label instead.) After installing the font, you can find it in the *Fonts* section of your *Control panel*. Right-click it and choose *Properties* to find its *Title*: this is the name to reference the font by later on.

2. Save the XSLT code in a file and move it into an *\xslt* subfolder in your application. Create the folder if it doesn't exist yet and put it on the same level as the other Adlib subfolders (like *\adapls*, *\data* and such).

3. Open the XSLT file in a simple text editor and adjust the fixed paths to the *\images* folder and the *\xslt* folder to match your environment. Also change the name of the referred barcode font (currently `BC39`) to the title you found in step 1. You can also change the conditions to print the three different "vrachtwagen#.png" icons to something else: currently these conditions are rather silly, because they check the first letter of the object name to select an icon. You should also change the icons themselves (to some sensible images that you actually have), come up with different conditions, etc. Feel free to experiment with the stylesheet.



LB507

LB507
waggon

78218631

78218631
Celestial chart
the constellations Monoceros, Canis Minor and
Atelier Typographique
In storage

207-1948

207-1948

Hampstead Heath

4. Setup the XSLT stylesheet as an output format for the *Internal object catalogue*. See the Designer Help for instructions on how to do this.
5. Open your Museum application and mark some object records containing images: more or less square or vertically oriented images work best in this example, as are relatively short object numbers so that the barcode fits on a single line.
6. In the Adlib *File* menu, select your *Output format* to print to it.

■ Stylesheet for grouped XML, to print from within Collections

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

  <xsl:template match="/adlibXML">
    <html>
      <head>
        <title>Object number barcode</title>
        <style type="text/css">
           .text
          {
          font-family: Verdana;
          font-size: large;
          vertical-align: top;
          }
          .titletext
          {
          font-family: Verdana;
          font-weight: bold;
          font-size: large;
          }
          .innertable
          {
          border: solid 0px;
          border-collapse: collapse;
          }
          @font-face {
          font-family: "MyBarcode";
          src: url(https://ourserver.com/free3of9.woff);
          }
          span.barcode
          {
          font-size:48pt;
          font-family: "MyBarcode"
          }
        </style>
      </head>
      <body>
        <xsl:apply-templates select="recordList/record"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="record">
   <table width="500px" border="0" cellspacing="0" cellpadding="5"
    class="innertable">
    <tr valign="top">
     <td align="center" class="text">
      <xsl:apply-templates select="object_number" mode ="barcode"/>
     </td>
    </tr>
    <tr valign="top">
     <td align="center" class="text">
```

```
      <xsl:apply-templates select="object_number" mode ="text"/>
      <p>________________________</p>
     </td>
    </tr>
   </table>
  </xsl:template>

  <xsl:template match ="object_number" mode ="barcode">
    <span class="barcode">
      <!-- Barcode 39 needs a start and stop character * -->
      <xsl:text>*</xsl:text>
      <xsl:value-of select ="."/>
      <xsl:text>*</xsl:text>
    </span>
  </xsl:template>

  <xsl:template match ="object_number" mode ="text">
      <xsl:value-of select ="."/>
  </xsl:template>

</xsl:stylesheet>
```

You can change it all you like of course, and to get it working in Axiell Collections you do indeed have to make some changes to it.

Proceed as follows:

1. You have to have a barcode font in the .*woff* format available on a web server (accessible to users of Axiell Collections via a URL), on the same domain as Collections. If you haven't got a barcode font, you'll have purchase such a font first, find a freely available font on the internet or try to install a demo version of a suitable font. TrueType fonts don't work but can be converted to a .*woff* type font: there are (free) services on the web that can do this for you. You refer to this font in the CSS section of the stylesheet:

   ```
   @font-face {
   font-family: "MyBarcode";
   src: url(https://ourserver.com/free3of9.woff);
   }
   ```
   Replace the URL by your own URL.

   A limitation of this implementation is that the HTML won't be portable as far as the barcode is concerned. When printing with this stylesheet, HTML will be generated. Often you will print this HTML directly and the barcode will be printed too, but if you ever copy the HTML itself to save it in a file or send it by e-mail you'll notice that it can't show the barcode when that HTML is opened again, outside of the Collections session.

2. Save the XSLT code in a file and move it into an *\xslt* subfolder in your application. Create the folder if it doesn't exist yet and put it on the same level as the other Adlib subfolders (like *\adapls*, *\data* and such).

3. Setup the XSLT stylesheet as an output format for the *Internal object catalogue*. See the Designer Help for instructions on how to do this.

4. Open Axiell Collections and mark some object records.

5. Click the *Create a report with a predefined output format* icon in the top toolbar and select your output format to generate the HTML output.

2.5 Creating text labels from HTML fields

An HTML field is a database field meant for long, laid-out text, possibly including images, much like a small and simple web page. Layout can be applied to the text during editing of the record. You could use such a field to create printable text labels to be presented with your museum objects, for example. From within Adlib you can print the contents of such a field to a Word template or with the aid of a custom XSLT stylesheet, whilst keeping the layout intact. Although normally you will only see the laid-out text while you are editing an HTML field, the field contents will actually be stored as (editable) HTML code in the background. From Adlib 6.6.0 you can implement HTML fields in your application.

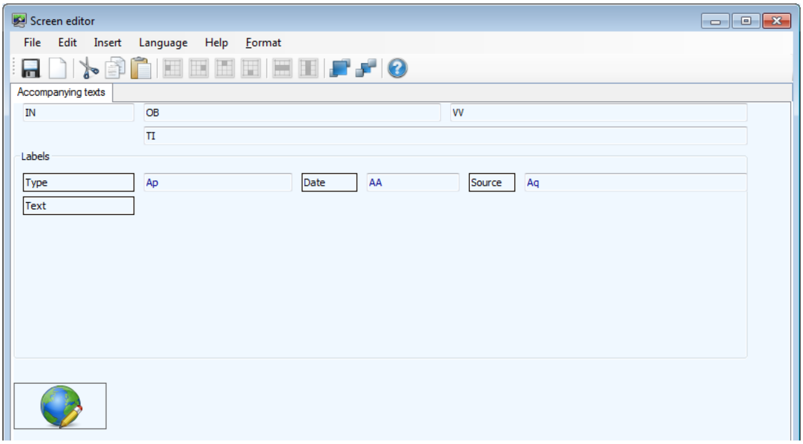
For example: a good spot to implement such a field is the field for label text on the *Accompanying texts* tab of a museum object record (4.2 model application), since it would be nice if you could apply layout to a label text during data entry, and maintain that layout in printouts.
You can execute the following procedure in your own application, even if the field to be converted already contains data: your data won't get lost:

1. Create a backup of your database(s) and applications before you make any changes, just to be safe.

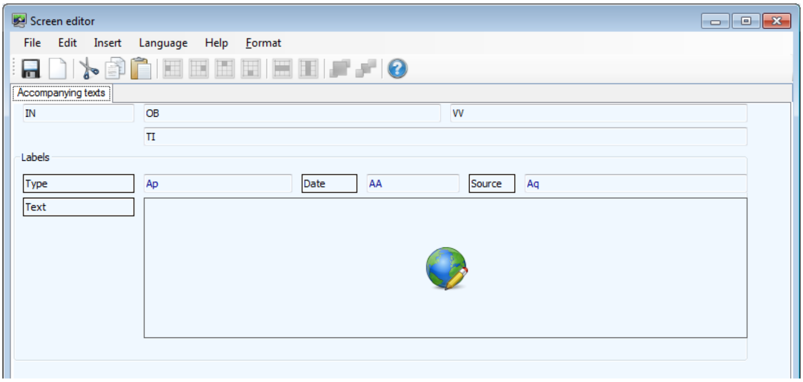
2. Open the *Collect* database in the *Application browser* in Designer, and select the *label.text* field (tag AB).
   If this field is not present in your application, you can of course pick another description-like field of the *Text* data type to convert, or create a new field.

3. Open the *Data type* drop-down list on the *Field properties* tab (which has *Text* currently selected) and choose the *HTML* option.

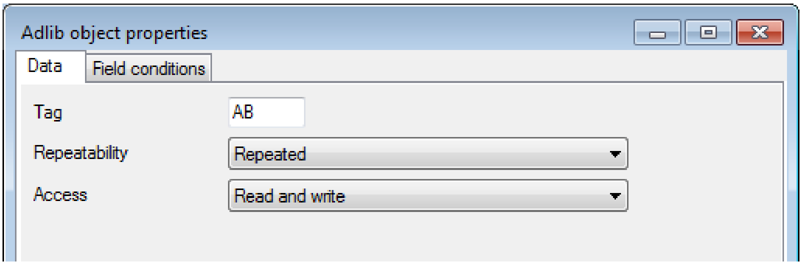
4. Save the change: right-click the *Collect* database and select *Save* in the pop-up menu.
5. In the *Screen editor*, open the *labels.fmt* screen (*Accompanying texts*), or another screen on which you want to place the new HTML field.
6. Drag the lower border of the *Labels* box a few centimetres downwards so that some space is created for the new field. In the menu choose *Insert > HTML field*.



7. Drag the HTML field into the box to the desired location and make it as big as you like. This type of field will not automatically resize as the user types more text in it, but a scroll bar will appear instead. So if there is enough room on the screen, like in this example, then make the HTML field tall enough to allow long text to be visible in its entirety, in most cases. If desired, you can insert a label in front of the HTML field, into which you copy label texts from the already existing *Text* field label.
8. Now select the old *AB* field and choose *Edit > Delete* in the menu. Label and field will be removed and the fields underneath it will move up a line.



9. Right-click the right or left border of the HTML field (where the cursor changes into a double arrow) and choose *Properties* in the pop-up menu. (See that you do not open the properties of the *Box* by accident.) Now enter the *Tag* of the HTML field, `AB` in our example, and set the other properties as desired. In this example, the field should be set to *Repeated* because the old field was repeatable too.

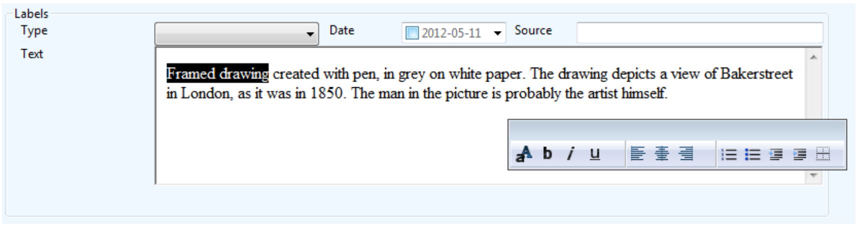


Tag: AB
Repeatability: Repeated
Access: Read and write

10. Save the changes in the screen and you are done.

Restart Adlib to be able to admire the result. In an existing record, the adjusted screen in display mode will present existing HTML field contents as plain text at first. The text has no layout yet and hasn't been stored as HTML code either. Put the record in edit mode to apply layout. Note that existing text in this converted field will only be saved as HTML when you have placed the cursor in this field once and saved the record again.

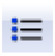
As soon as the cursor is in the HTML field in edit mode, a floating toolbar appears, to lay out the text.



Framed drawing created with pen, in grey on white paper. The drawing depicts a view of Bakerstreet in London, as it was in 1850. The man in the picture is probably the artist himself.

The buttons have the following meaning:

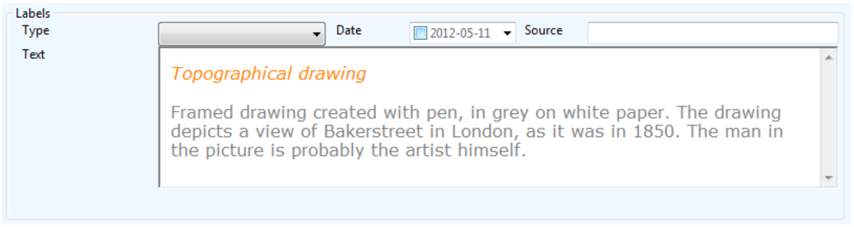
Text layout buttons for HTML fields

	Choose a font type.
	Apply bold layout to text.
	Apply italic layout to text.
	Underline text.
	Align text to the left.
	Centre text.
	Align text to the right.
	Number the paragraphs.
	Place bullets in front of paragraphs.
	Reduce the indentation.
	Enlarge the indentation.
	Insert a horizontal line.

The buttons largely work the same way as they do in Windows text editors. For instance, you must first type text, select it or just a part of it, and then apply a layout to it.
Moreover, you can copy (`Ctrl+C`) formatted text from other Windows applications, like Microsoft Word, and paste (`Ctrl+V`) it in an HTML field, or vice versa, while conserving the layout. However, when copy-

ing from within Word, it is possible that not all layout characteristics will be included, like the colour of text for instance.
For inserting new lines you can use **`Enter`** for a new paragraph, and **`Shift+Enter`** to begin a new line within the same paragraph.

After layout has been applied the text might look as follows, for example:

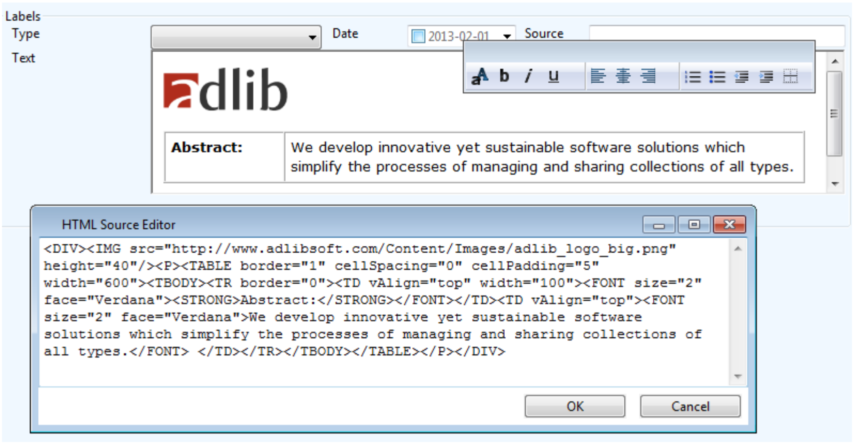


Topographical drawing

Framed drawing created with pen, in grey on white paper. The drawing depicts a view of Bakerstreet in London, as it was in 1850. The man in the picture is probably the artist himself.

From Adlib 7.1, you can edit the HTML code itself in a separate window. If you know how to code HTML, you can use it to include URLs to images or websites, create tables and apply other possibilities of HTML; you just can't use CSS styles though.

Right-click an HTML field in the running application and select *Edit the HTML source* in the pop-up menu which opens. The *HTML Source Editor* window opens. The HTML of already present field content will be shown. Here you can adjust the HTML code directly. It's possible as well to copy (**`Ctrl+C`**) and paste (**`Ctrl+V`**) code from and to this editor, which might be convenient if you prefer to edit your HTML code in a different HTML (web page) editor first. Click *OK* to close the window. The result of your changes can be viewed in the Adlib record.



Abstract:	We develop innovative yet sustainable software solutions which simplify the processes of managing and sharing collections of all types.

```
<DIV><IMG src="http://www.adlibsoft.com/Content/Images/adlib_logo_big.png"
height="40"/><P><TABLE border="1" cellSpacing="0" cellPadding="5"
width="600"><TBODY><TR border="0"><TD vAlign="top" width="100"><FONT size="2"
face="Verdana"><STRONG>Abstract:</STRONG></FONT></TD><TD vAlign="top"><FONT
size="2" face="Verdana">We develop innovative yet sustainable software
solutions which simplify the processes of managing and sharing collections of
all types.</FONT> </TD></TR></TBODY></TABLE></P></DIV>
```

The HTML code in these fields always starts and ends with `<div>` and `</div>` respectively, and has no `<body>` tags or `<head>` section. If you add `<body>` tags or a `<head>` section anyway, then these will be removed as soon as you click *OK*. When you click *OK*, the HTML will also be adjusted automatically to allow it to be saved within the (XML) storage format of the Adlib record. Any indentation and layout of the HTML code itself will be removed as well, so that the code will be displayed as a single paragraph of text next time you open the editor; since such a cluttered presentation makes it hard to edit the code, we recommend to keep the code relatively short.

In the example above you can see that in the HTML field an image (of the Adlib logo) is retrieved and a table has been created, which is only possible here by adjusting the HTML code itself.

2.5.1 Printing, export and wwwopac output

The laid out text in HTML fields can only be printed correctly to Word templates or XSLT stylesheets which have been set up as output formats. In Adlib you can print to output formats and Word templates via the *Print wizard* of course. In principle, you must create these templates or stylesheets yourself, but Word templates do not require any special instructions: you can refer to the field tag normally. An example XSLT stylesheet can be seen below.

If you still print HTML fields via an adapl (also via the interactive *Print wizard* method), then the HTML field contents will be printed as it is stored: the layout won't be visible but the HTML codes will be.

When you export records with HTML fields, the HTML field contents will be extracted as HTML code: the code begins and ends with `<div>` and respectively `</div>`, and has no `<body>` or `<head>` tags. When exporting to XML, the HTML field contents will be produced as HTML within the XML field tags; the same applies to the XML search result of *wwwopac.ashx*. An Adlib Internet Server web application uses XSLT stylesheets to convert the XML output of *wwwopac.ashx* to HTML.
For both printing and web display, XSLT stylesheets need to be coded for HTML fields as follows, like for the `label.text` field in our example:

```
<xsl:template match="label.text">
  <xsl:value-of select="." disable-output-escaping="yes"/>
</xsl:template>
```

A complete example stylesheet (made for unstructured XML) which you can set up as an output format in Adlib is the following:

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

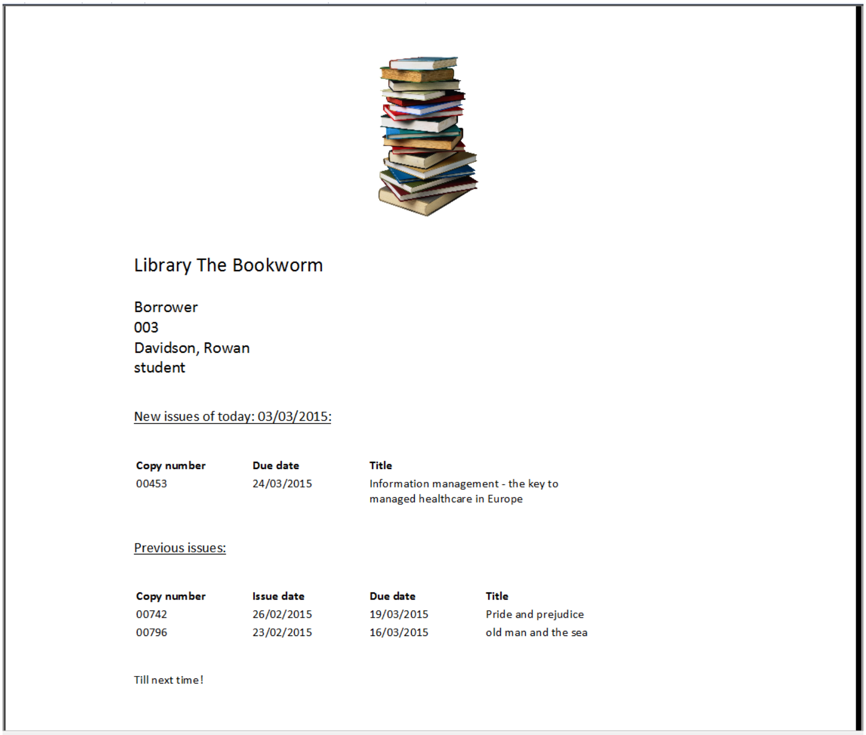
```
    <font face="verdana" color="#808080">
      <xsl:apply-templates select="recordList"/>
    </font>
  </xsl:template>
  <xsl:template match="recordList">
      <xsl:apply-templates select="record"/>
  </xsl:template>

  <xsl:template match="record">
    <html>
     <head>
     </head>
     <body>
       <xsl:apply-templates select="object_number"/>
       <xsl:apply-templates select="label.text"/>
     </body>
    </html>
  </xsl:template>

  <xsl:template match="object_number">
    <p>
      <xsl:text>Object: </xsl:text>
      <xsl:value-of select="."/>
      <xsl:apply-templates select="../object_category"/>
    </p>
  </xsl:template>

  <xsl:template match="object_category">
    <xsl:text> (</xsl:text>
    <xsl:value-of select="."/>
    <xsl:text>)</xsl:text>
  </xsl:template>

  <xsl:template match="label.text">
  <table border="1" cellpadding="10" cellspacing="0"
   style="border-collapse: collapse" bordercolor="#808080"
   width="100%">
   <tr>
    <td width="100%">
       <xsl:value-of select="." disable-output-escaping="yes"/>
    </td>
   </tr>
  </table>
  </xsl:template>
</xsl:stylesheet>
```

2.5.2 Notes

- Font types for the Adlib interface, which you can set via the *View > Change font* menu, have no effect on the font types in HTML fields.
- Existing, filled-in RTF fields cannot be converted to HTML fields.

- After the conversion, hard and soft returns in the relevant text fields have been converted to HTML `<BR/>` tags.
- The first time you print a record with an HTML field to a Word template, Word may ask if you would like to set Word as the standard editor for HTML pages. Answer *No* if you don't want that, and *Yes* if you are fine with it. Either way, the resulting printout may not be okay after that. Then try to print again. Microsoft Word shouldn't pose the question anymore, and the printout must now be correct.

3 Stylesheets for Adloan slips



Library The Bookworm

Borrower
003
Davidson, Rowan
student

New issues of today: 03/03/2015:

Copy number	Due date	Title
00453	24/03/2015	Information management - the key to managed healthcare in Europe

Previous issues:

Copy number	Issue date	Due date	Title
00742	26/02/2015	19/03/2015	Pride and prejudice
00796	23/02/2015	16/03/2015	old man and the sea

Till next time!

From within Adloan Circulation you can print issue slips, return slips and reservation slips for the borrower, to remind him or her about any borrowed, returned or reserved copies.

The contents of these slips (fixed texts and Adlib field references) are determined by special templates in the Adlib *\executables* folder, which you can edit.
In old Adloan versions, these templates were made up by six types of *.rtf* files (you can edit these in WordPad): *islipall#.rtf*, *islipone#.rtf*, *dslipall#.rtf*, *dslipone#.rtf*, *rslipall#.rtf* and *rslipone#.rtf*, in which # stands for an Adlib language number (English = 0, Dutch = 1, French = 2, German = 3, Arabic = 4, Italian = 5, Greek = 6, Portuguese = 7, Russian = 8, Swedish = 9, Hebrew = 10, Danish = 11, Norwegian = 12, Finnish = 13 and Chinese = 14).
However, from Adloan version 7.2.15061.3 or 7.3.15065 six XSLT templates (which have more possibilities than the .rtfs), plus a single XML file containing fixed texts in different translations are used by

Adloan by default.
If you start using a new Adloan version but you have no desire to use the XSLT stylesheets, then simply remove them from the *\executables* folder and store them in a backup folder somewhere else: Adloan will then revert to using the old *.rtf* templates.
If you'd like Adloan to use the XSLT stylesheets, make sure the following files reside in the Adlib *\executables* subfolder:

- **issue-list-all.xsl** (replaces *islipall#.rtf*) to generate the *Overall issue slip.*
- **issue-list-one.xsl** (replaces *islipone#.rtf*) to generate the issue slip for one or more selected copies.
- **return-list-all.xsl** (replaces *dslipall#.rtf*) to generate the *Overall return slip*.
- **return-list-one.xsl** (replaces *dslipone#.rtf*) to generate the return slip for one or more selected copies.
- **reservation-list-all.xsl** (replaces *rslipall#.rtf*) to generate the *Overall reservation slip*.
- **reservation-list-one.xsl** (replaces *rslipone#.rtf*) to generate the reservation slip for one or more selected copies.
- **adloanListTranslations.xml** in which all required translations of the fixed texts in the slips are or can be included, amongst which the name of your institution and possibly a contact e-mail address. An e-mail address will be printed in large underneath the institution name.

You can edit all files in a text editor like Windows Notepad, Notepad++ or Visual Studio, to customize the design and fixed texts, for example, or to add the logo of your institution.

In *adloanListTranslation.xml* for example, replace the fixed text *placeholder_for_institution_name* in language 0 (English) by your own institution name. If you're not always displaying Adloan Circulation in English, you should enter your institution name for those other languages as well: de language numbers are explained in the XML document itself.
An e-mail address for information requests can be entered in the next `<translation>` node instead of *placeholder_for_e-mail_address*. If you don't want any e-mail address to be printed then simply remove the *placeholder_for_e-mail_address* text, but do leave the surrounding XML `<outputtext>` tags where they are. And instead of an e-mail address you might as well enter the URL to your website here.

```
<translation name="institution name">
  <outputtext language="0">placeholder_for_institution_name</outputtext>
  <outputtext language="1">ruimte_voor_instellingsnaam</outputtext>
  <outputtext language="2">espace_réservé_pour_nom_d'institution
   </outputtext>
  <outputtext language="3">Platzhalter_für_Institutionsname</outputtext>
  <outputtext language="4">placeholder_for_institution_name</outputtext>
  <outputtext language="5">placeholder_for_institution_name</outputtext>
  <outputtext language="6">placeholder_for_institution_name</outputtext>
</translation>
<translation name="e-mail address">
  <outputtext language="0">placeholder_for_e-mail_address</outputtext>
  <outputtext language="1">placeholder_for_e-mail_address</outputtext>
  <outputtext language="2">placeholder_for_e-mail_address</outputtext>
  <outputtext language="3">placeholder_for_e-mail_address</outputtext>
  <outputtext language="4">placeholder_for_e-mail_address</outputtext>
  <outputtext language="5">placeholder_for_e-mail_address</outputtext>
  <outputtext language="6">placeholder_for_e-mail_address</outputtext>
</translation>
```

The other fixed texts can be adjusted too, but don't change anything in between < and > brackets.

If for any desired language an Adlib language number does exist while no `<outputtext>` lines are present for that language yet, then you may add those lines yourself.

In the .xsl files you don't need to change anything per se, unless you would like to have an image (the logo of your institution for example) inserted at the top of the printout or wish to change the layout of the slips. In the first case you'll only have to edit a single line of code:

1. For 7.3, place the desired image file in the folder one level up from your Adlib \Library loans management subfolder. This is probably your main Adlib folder.

2. Look for the following line of (HTML) code: `<img src="books.jpg" width="150px" heigth="250px"/>`

3. Substitute `books.jpg` by the actual name of your image file.

4. Enter the desired maximum width and height of the image and save the changes.

5. From within Adloan, now print the relevant slip to test your changes.
   If you do not actually want to print anything to the printer you can show a preview on screen by keeping the **`Shift`** key pressed down when selecting the relevant pop-up menu option (like *Overall issue slip*).

6. Execute the above changes in all six XSLT stylesheets if you'd like the changes to become visible in all types of slips.

In the second case, where you'd like to change the layout of the slips you'll have to have some knowledge of CSS and XSLT, as explained in this manual, and you'll have to know about the XML output which Adloan generates.

3.1 Available Adloan XML output

The three transaction types (issues, returns and reservations) basically generate the same XML, with some minor differences.

- The `<type>` node in the `<request>` section of all XML will contain one of the values `issue`, `discharge` or `reserve`, depending on which type of slip the user is printing.
- The `<mode>` node in the `<request>` section of all XML will contain either the value `one` (the XML will contain only a single item) or `all` (the XML will contain all items), depending on which print option the user selected.
- The `<language>` node in the `<request>` section of all XML will contain the current interface language of Adloan Circulation. This will be the language in which the fixed texts from *adloanListTranslation.xml* will be printed.
  Currently available language numbers are: English = 0, Dutch = 1, French = 2, German = 3, Arabic = 4, Italian = 5, Greek = 6, Portuguese = 7, Russian = 8, Swedish = 9, Hebrew = 10, Danish = 11, Norwegian = 12, Finnish = 13 and Chinese = 14.

■ Issues

```
<adlibXML>
  <recordList>
    <record>
      <request>
        <type>issue</type>
        <mode>all</mode>
        <language>1</language>
        <location>main</location>
        <date>29/04/2015</date>
        <time>10:26</time>
      </request>
      <borrower>
        <id>001</id>
        <name>Bourne, Jason</name>
        <category>personel</category>
      </borrower>
      <item>
        <copyId>01421</copyId>
        <catalogueId>38463</catalogueId>
        <title>Lord of the rings: part 1,2,3</title>
        <issueDate>03/03/2015</issueDate>
        <issueTime>10:26</issueTime>
        <dueDate>16/05/2015</dueDate>
        <dueTime>16:00</dueTime>
        <location>main</location>
        <shelfMark>4892.32</shelfMark>
      </item>
      <item>
        <copyId>03801</copyId>
        <catalogueId>92357</catalogueId>
        <title>Steam engines</title>
        <issueDate>01/03/2015</issueDate>
        <issueTime>11:33</issueTime>
        <dueDate>14/05/2015</dueDate>
        <dueTime>16:00</dueTime>
        <location>main</location>
        <shelfMark>1372.88</shelfMark>
      </item>
    </record>
  </recordList>
</adlibXML>
```

Note:

- The XML applies to both *issue-list-all.xsl* and *issue-list-one.xsl*, but the XML for *issue-list-one.xsl* will contain only one `<item>` node whereas the XML for *issue-list-all.xsl* may contain multiple `<item>` nodes. (You can't change the names of these stylesheets.)

■ Returns

```
<adlibXML>
  <recordList>
    <record>
      <request>
        <location>main</location>
        <date>29/04/2015</date>
        <time>10:26</time>
        <type>discharge</type>
        <mode>all</mode>
        <language>1</language>
      </request>
      <borrower>
        <id>001</id>
        <name>Bourne, Jason</name>
        <category>personel</category>
      </borrower>
      <item>
        <copyId>01421</copyId>
        <title>Lord of the rings: part 1,2,3</title>
        <dueDate>16/05/2015</dueDate>
        <dueTime>16:00</dueTime>
        <returnDate>29/04/2015</returnDate>
        <returnTime>10:26</returnTime>
        <location>main</location>
        <shelfMark>4892.32</shelfMark>
      </item>
      <item>
        <copyId>03801</copyId>
        <title>Steam engines</title>
        <dueDate>16/05/2015</dueDate>
        <dueTime>16:00</dueTime>
        <returnDate>29/04/2015</returnDate>
        <returnTime>10:26</returnTime>
        <location>main</location>
        <shelfMark>1372.88</shelfMark>
      </item>
    </record>
  </recordList>
</adlibXML>
```

Note:

- The XML applies to both *return-list-all.xsl* and *return-list-one.xsl*, but the XML for *return-list-one.xsl* will contain only one `<item>` node whereas the XML for *return-list-all.xsl* may contain multiple `<item>` nodes. (You can’t change the names of these stylesheets.)

■ Reservations

```
<adlibXML>
  <recordList>
    <record>
      <request>
        <location>main</location>
        <date>05/03/2015</date>
        <time>10:26</time>
        <type>reserve</type>
        <mode>all</mode>
        <language>1</language>
      </request>
      <borrower>
        <id>001</id>
        <name>Bourne, Jason</name>
        <category>-</category>
      </borrower>
      <item>
        <copyId>01421</copyId>
        <title>Lord of the rings: part 1,2,3</title>
        <reservationDate>16/05/2015</reservationDate>
        <expiryDate>08/06/2015</expiryDate>
      </item>
      <item>
        <copyId>00272</copyId>
        <catalogueId>187</catalogueId>
        <title>Legislation</title>
        <reservationDate>05/03/2015</reservationDate>
        <expiryDate>12/06/2015</expiryDate>
        <location>main</location>
      </item>
    </record>
  </recordList>
</adlibXML>
```

Note:

- The contents of an `<item>` node depends on whether a reservation has been made on copy number or on catalogue number. The example above shows both cases.
- The XML applies to both *reservation-list-all.xsl* and *reservation-list-one.xsl*, but the XML for *reservation-list-one.xsl* will contain only one `<item>` node whereas the XML for *reservation-list-all.xsl* may contain multiple `<item>` nodes. (You can't change the names of these stylesheets.)

3.2 Adloan version differences

Note that this functionality really is supported from Adloan version 7.2.15061.3, but then you won't find the XSLT stylesheets and the XML file in the \executables folder by default. However, you can request those files from our helpdesk if you want.

Further, Adloan 7.2 deviates from 7.3 in that an image of your own to be referred to from within the stylesheets, must be located in the application folder itself, in \Library loans management for example, not the higher folder.

4 A web browser box display format

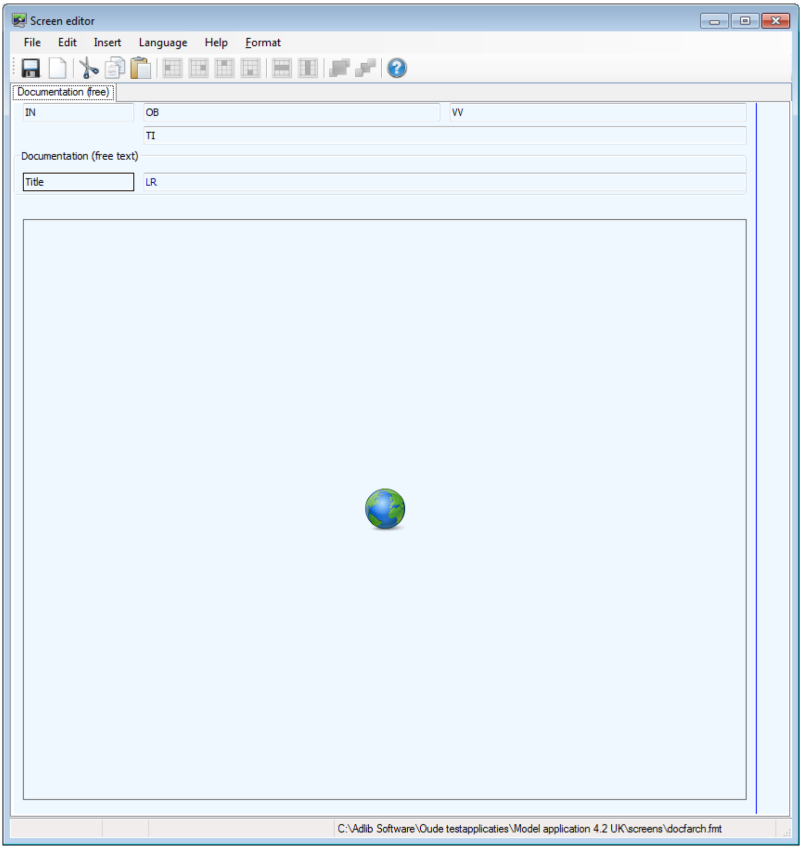
4.1 Web browser box setup

From 6.6.0 you can place a special web browser box on a screen in your application, to display record data as a web page. The box is only meant for display, and possibly printing, but not for editing.
You do need an XSLT stylesheet for this: under the hood of Adlib, all data is processed as XML, and with an XSLT stylesheet that XML can be transformed to HTML, for example, while HTML can be displayed as a web page in a web browser and in a web browser box in Adlib.
Such a web browser box can be useful when you want to present the user of Adlib with a nice presentation of the record in a single box. Or maybe you have a website which displays records in detail, and you would like to have the same presentation available in Adlib as well, so that during record entry a registrar can already see how the record will look on the website. Every time the screen is redrawn during entry or editing, for instance when you switch tabs, the web page display will be updated. You can implement a web browser box as follows:

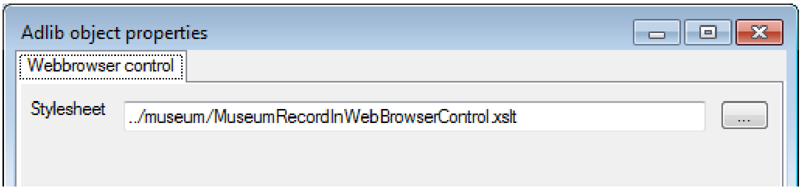
1. Create an XSLT stylesheet to transform the grouped XML of records from a certain database into HTML; see the next paragraph for an example. The example code contains an assumed, fixed relative path to the images folder: you should change this path to the (path to the) folder which actually contains your images. You are of course free to further adjust this stylesheet to your requirements. Place the stylesheet in a suitable location in your Adlib Software folder, maybe the folder with the name of the application, or in a new *\stylesheets* or *\xslt* subfolder.

2. Suppose you would like to be able to view an object record as a web page, then search your object catalogue for a screen with enough empty space to hold the web page presentation (underneath or between the field rows, but not to the right of them), or create a new screen. In this example we use the *docfarch.fmt* screen (*Documentation (free)*). Open the selected screen in the *Screen editor* of Adlib Designer.

3. Click *Insert > Web browser control* in the menu bar. A box with the icon of a globe will be placed on the screen. Drag it to the desired spot and make the box as big as you like by dragging its edges. When the HTML page is shown on this screen in your Adlib application, the box will have this exact size and cannot be adjusted there.

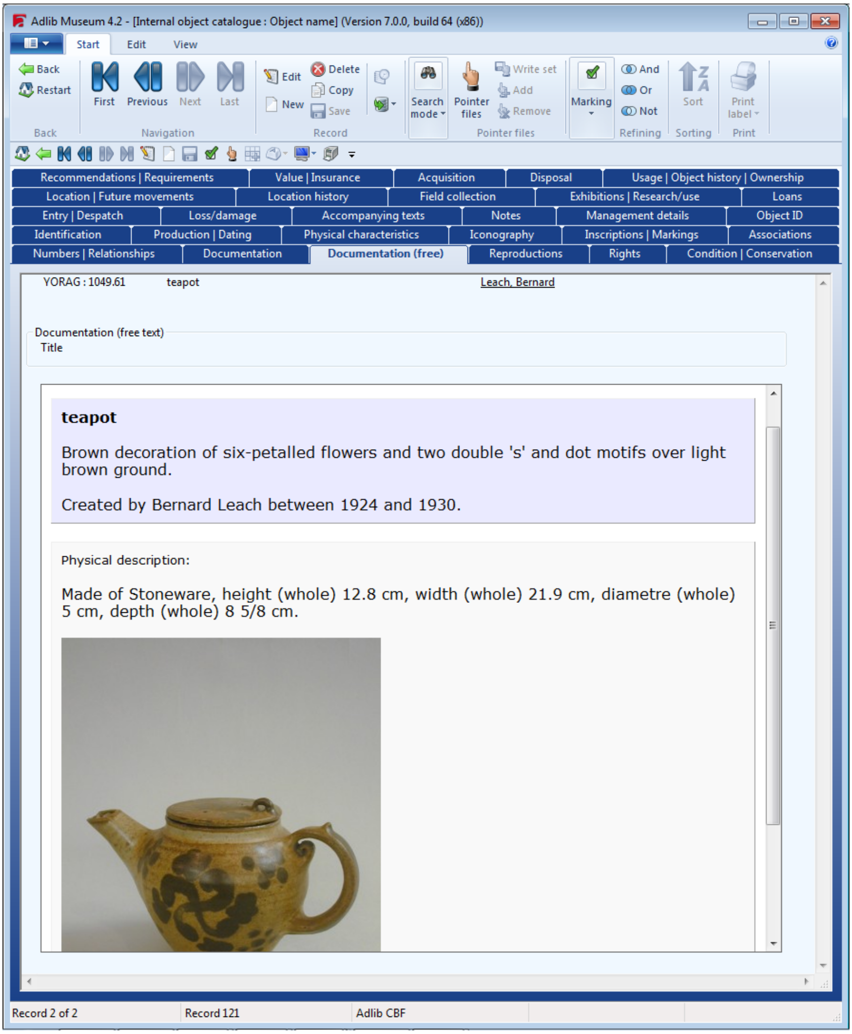

Screen editor
File Edit Insert Language Help Format
Documentation (free)
IN
OB
VV
TI
Documentation (free text)
Title
LR
C:\Adlib Software\Oude testapplicaties\Model application 4.2 UK\screens\docfarch.fmt

4. Right-click the new box and choose *Properties* in the pop-up menu which opens. There's only one option available. Click the … button to search for the desired XSLT stylesheet on your system. The path to the file must be relative to the application folder, and the *.xslt* extension must be present behind the file name.



Stylesheet ../museum/MuseumRecordInWebBrowserControl.xslt

5. Save the changes in the screen and view the result in Adlib by opening a record and switching to the adjusted tab.



YORAG : 1049.61 teapot Leach, Bernard

Documentation (free text)
Title

teapot

Brown decoration of six-petalled flowers and two double 's' and dot motifs over light brown ground.

Created by Bernard Leach between 1924 and 1930.

Physical description:

Made of Stoneware, height (whole) 12.8 cm, width (whole) 21.9 cm, diametre (whole) 5 cm, depth (whole) 8 5/8 cm.

Record 2 of 2 Record 121 Adlib CBF

You can copy the web page display if you want, and paste it in a Word document. Right-click the display and choose *Select all*. Press `Ctrl+C` to copy everything. Switch to your Word document and paste the text (`Ctrl+V`).
If you like, you can even print the web page display directly. Right-click the display and choose *Print…* in the pop-up menu. With the standard Windows print dialog you can then actually start the printing. It is possible that on the printout a page numbering, date and other header and footer information can be seen, and that background colours won't print. This is caused by page settings in Internet

Explorer, because the web browser box uses IE functionality for printing. In Internet Explorer 9 you can change these settings via the menu in the web browser (press **`Alt`**) *File > Page setup* or via the *Tools* button (**`Alt+X`**) *Print > Page setup*. For example, mark the *Print background colours and images* option to be able to print background colours. The headers and footers can be changed here as well.

4.1.1 Examples

Earlier in this manual, we already learned what the grouped XML formgenereated by *adlwin.exe* looks like, but if you're still unsure about it, there's an easy way here to find out how the record XML is structured exactly, after the finishing the above setup. You don't need to have an advanced stylesheet already or even one that works, to be able to view the XML. Right-click the web browser box in your active Adlib application and choose *View > Record XML* in the pop-up menu. The entire contents of the record will be diplayed as XML in Windows NotePad. To obtain a better presentation of the XML, you can save the file with the *.xml* extension, and double-click it in Windows Explorer. The file will open in a program able to display XML properly, like Internet Explorer. Now, the XML structure is clear and you'll be able to customize your stylesheet efficiently.

Via the same pop-up menu you can display the generated HTML as well. Your stylesheet must convert the record XML into HTML and the result can be shown with *View > Page source*. Correct HTML code will be presented only if your stylesheet works properly.

An example of a bilingual* XSLT stylesheet which transforms the grouped XML of *Collect* records into HTML, and can be used for a web browser box on an Adlib screen, is the following:

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

  <xsl:output method="html" encoding="UTF-8" indent="yes"/>
  <xsl:param name="ui_language"></xsl:param>

  <xsl:template match="/adlibXML">
    <html>
      <head>
        <meta http-equiv="X-UA-Compatible" content="IE=edge" />
      </head>
      <body>
        <font face="verdana" size="3">
          <xsl:apply-templates select="recordList"/>
        </font>
      </body>
    </html>
  </xsl:template>
```

```
<xsl:template match="recordList">
  <xsl:apply-templates select="record"/>
</xsl:template>

<xsl:template match="record">
  <p>
    <font face="Verdana" size="2">
      <xsl:apply-templates select="priref"/>
      <xsl:apply-templates select="object_number"/>
    </font>
  </p>
  <xsl:if test="count(Object_name) != 0">
    <p>
      <table border="1" cellpadding="10" style="border-collapse:
             collapse" width="100%">
        <tr>
          <td width="100%" bgcolor="#EAEAFF">
            <p>
              <xsl:apply-templates select="Object_name"/>
              <xsl:apply-templates select="other_name"/>
            </p>
            <xsl:apply-templates select="Description"/>
            <xsl:if test="Production/creator[@linkref] != '0'">
              <p>
                <xsl:apply-templates select="Production"/>
              </p>
            </xsl:if>
          </td>
        </tr>
      </table>
    </p>
  </xsl:if>

  <p>
    <table border="1" cellpadding="10" style="border-collapse:
           collapse" width="100%">
      <tr>
        <td width="100%" bgcolor="#F9F9F9">
          <p>
            <xsl:if test="physical_description != '' or
                    Dimension/dimension.value != ''">
              <font face="Verdana" size="2">
                <xsl:choose>
                  <xsl:when test="$ui_language = 1">
                    <xsl:text>Fysieke beschrijving:</xsl:text>
                  </xsl:when>
                  <xsl:otherwise>
                    <xsl:text>Physical description:</xsl:text>
                  </xsl:otherwise>
                </xsl:choose>
              </font>
              <br/>
              <br/>
            </xsl:if>
```

```
              <xsl:apply-templates select="physical_description"/>
              <xsl:if test="Material/material/term != ''
                      or Dimension/dimension.value != ''">
                <p>
                  <xsl:if test="Material/material/term != ''">
                    <xsl:apply-templates select="Material"/>
                  </xsl:if>
                  <xsl:if test="Dimension/dimension.value != ''">
                    <xsl:if test="Material/material/term != ''">
                      <xsl:text>, </xsl:text>
                    </xsl:if>
                    <xsl:apply-templates select="Dimension"/>
                  </xsl:if>
                  <xsl:text>.</xsl:text>
                </p>
              </xsl:if>
              <xsl:if
          test="Reproduction/reproduction.reference[@linkref]!='0'">
                <p>
                  <xsl:apply-templates
                   select="Reproduction/reproduction.reference"/>
                </p>
              </xsl:if>
            </p>
          </td>
        </tr>
      </table>
    </p>
  </xsl:template>

  <xsl:template match="reproduction.reference">
    <p>
      <img src="..\images\{normalize-space(reference_number)}" />
    </p>
  </xsl:template>

  <xsl:template match="priref">
    <xsl:if test="position() = 1">
      <xsl:text>Record: </xsl:text>
      <xsl:value-of select="."/>
      <br/>
    </xsl:if>
  </xsl:template>

  <xsl:template match="object_number">
    <xsl:text>Object: </xsl:text>
    <xsl:value-of select="."/>
    <xsl:apply-templates select="../object_category"/>
  </xsl:template>

  <xsl:template match="object_category">
    <xsl:text>  (</xsl:text>
    <xsl:value-of select="normalize-space(term)"/>
    <xsl:text>)</xsl:text>
  </xsl:template>
```

```
<xsl:template match="Object_name">
  <strong>
    <xsl:if test="position() &gt; 1">
      <xsl:text> &amp; </xsl:text>
    </xsl:if>
    <xsl:value-of select="object_name/term"/>
  </strong>
</xsl:template>

<xsl:template match="other_name">
  <xsl:text>, other name: </xsl:text>
  <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="physical_description">
  <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="Description">
  <xsl:if test="description != ''">
    <xsl:value-of select="description"/>
  </xsl:if>
</xsl:template>

<xsl:template match="Material">
  <xsl:choose>
    <xsl:when test="position() = 1">
      <xsl:choose>
        <xsl:when test="$ui_language = 1">
          <xsl:text>Gemaakt van </xsl:text>
        </xsl:when>
        <xsl:otherwise>
          <xsl:text>Made of </xsl:text>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:when>
    <xsl:otherwise>
      <xsl:text> &amp; </xsl:text>
    </xsl:otherwise>
  </xsl:choose>
  <xsl:value-of select="material/term"/>
</xsl:template>

<xsl:template match="Dimension">
  <xsl:if test="position() &gt; 1">
    <xsl:text>, </xsl:text>
  </xsl:if>
  <xsl:value-of select="dimension.type/term"/>
  <xsl:if test="dimension.part != ''">
    <xsl:text> (</xsl:text>
    <xsl:value-of select="dimension.part"/>
    <xsl:text>)</xsl:text>
  </xsl:if>
  <xsl:text> </xsl:text>
```

```
    <xsl:value-of select="dimension.value"/>
    <xsl:text> </xsl:text>
    <xsl:value-of select="dimension.unit/term"/>
  </xsl:template>

  <xsl:template match="Production">
    <xsl:variable name="pos" select="position()"/>
    <xsl:choose>
      <xsl:when test="$pos = 1">
        <xsl:choose>
          <xsl:when test="$ui_language = 1">
            <xsl:text>Vervaardigd door </xsl:text>
          </xsl:when>
          <xsl:otherwise>
            <xsl:text>Created by </xsl:text>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:when>
      <xsl:otherwise>
        <xsl:text> &amp; </xsl:text>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:apply-templates select="creator"/>
    <xsl:if test="position() = last()">
      <xsl:apply-templates select="../Production_date[1]"/>
    </xsl:if>
  </xsl:template>

  <xsl:template match="Production_date">
    <xsl:text> </xsl:text>
    <xsl:choose>
      <xsl:when test="(production.date.start!=production.date.end)
                  and (production.date.end != '')">
        <xsl:choose>
          <xsl:when test="$ui_language = 1">
            <xsl:text>tussen </xsl:text>
          </xsl:when>
          <xsl:otherwise>
            <xsl:text>between </xsl:text>
          </xsl:otherwise>
        </xsl:choose>
        <xsl:value-of select="production.date.start"/>
        <xsl:choose>
          <xsl:when test="$ui_language = 1">
            <xsl:text> en </xsl:text>
          </xsl:when>
          <xsl:otherwise>
            <xsl:text> and </xsl:text>
          </xsl:otherwise>
        </xsl:choose>
        <xsl:value-of select="production.date.end"/>
        <xsl:text>.</xsl:text>
      </xsl:when>
      <xsl:otherwise>
        <xsl:text> in </xsl:text>
        <xsl:value-of select="production.date.start"/>
        <xsl:text>.</xsl:text>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <xsl:template match="creator">
    <xsl:choose>
      <xsl:when test="contains(name, ',')">
        <xsl:value-of select="substring-after(name , ',')"/>
        <xsl:text> </xsl:text>
        <xsl:value-of select="substring-before(name , ',')"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="name"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

</xsl:stylesheet>
```

* More Adlib parameters for XSLT stylesheets

When the detail screen containing the web browser control is formatted for display, three parameters (system variables) are passed to the XSLT stylesheet: the currently selected data language, the current user interface language and the background color of the Adlib screen. These parameters must be used as follows:

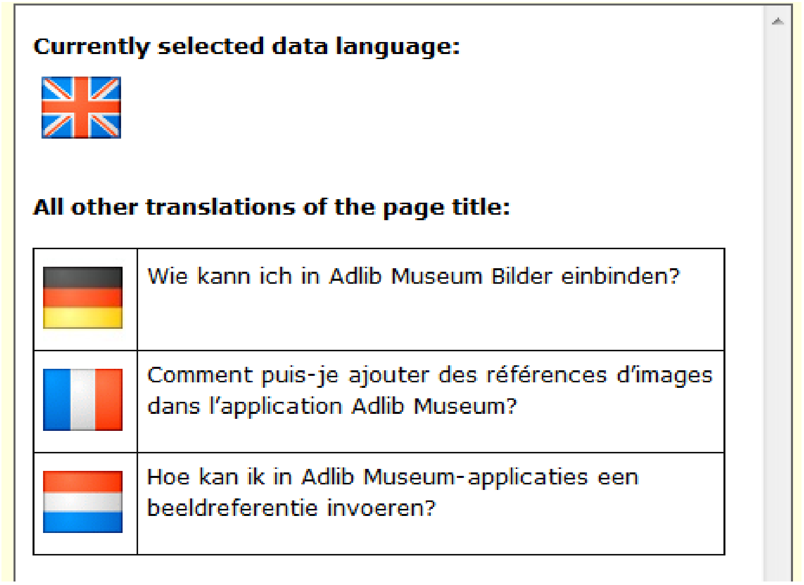
- `ui_language` – the current user interface language as referenced in Adlib. For example, English is 0, while Dutch is 1. You can use this to present fixed texts in the display in the current interface language. See the code of the example stylesheet presented above (*MuseumRecordInWebBrowserControl.xslt*) for a way to use this parameter. This parameter can also be used in output and export formats.

- `data_language` – the currently selected data language as an IETF language tag. Examples of these IETF language codes are: `'en-GB'`, `'en-US'`, `'nl-NL'`, `'de-DE'`, `'fr-FR'`. In an XSLT stylesheet for a multilingual Adlib SQL or Adlib Oracle database you can use this parameter for conditional purposes, for example:

  ```
  <xsl:when test="$data_language = 'nl-NL'">
    <img border="0"  src="flag_netherlands.gif" width="48"
     height="48"/>
  </xsl:when>
  ```

  This parameter can also be used in output and export formats.

In the image below you can observe a possible application of this functionality, for a multilingual page title field. See the example *MultilingualRecordInWebBrowserControl.xslt* stylesheet (shown below) for the code behind this presentation.



Currently selected data language:

All other translations of the page title:

🇩🇪	Wie kann ich in Adlib Museum Bilder einbinden?
🇫🇷	Comment puis-je ajouter des références d'images dans l'application Adlib Museum?
🇳🇱	Hoe kan ik in Adlib Museum-applicaties een beeldreferentie invoeren?

- `background_color` – the background color of the screen as a hexadecimal HTML colour code (#rrggbb). You can use this parameter to provide the HTML page with the same background colour as the Adlib screen, if desired. This parameter cannot be used in output and export formats.

To use the parameters in a stylesheet, declare them as a regular XSLT parameter without a default value (because it will be overwritten anyway) somewhere in the file, for example:

```
<xsl:param name="background_color"></xsl:param>
<xsl:param name="data_language”></xsl:param>
<xsl:param name="ui_language"></xsl:param>
```

The background color can be used like this, for example:

```
<style type="text/css">
  body { background: <xsl:value-of select="$background_color"/>; }
</style>
```

The following *MultilingualRecordInWebBrowserControl.xslt* example XSLT file is primarily meant to show the principle of processing a multilingual field in XSLT:

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

<xsl:output method="html" encoding="UTF-8" indent="yes"/>
<xsl:param name="data_language"></xsl:param>

<xsl:template match="/adlibXML">
 <html>
  <head>
  </head>
  <body>
    <font face="verdana" size="2">
      <xsl:apply-templates select="recordList"/>
    </font>
  </body>
 </html>
</xsl:template>

<xsl:template match="recordList">
  <xsl:apply-templates select="record"/>
</xsl:template>

<xsl:template match="record">
  <b><xsl:text>Currently selected data language:</xsl:text></b>
  <br/>
  <table border="0" cellpadding="5" cellspacing="0"
   style="border-collapse: collapse" bordercolor="#111111"
   width="49">
    <tr><td width="49" valign="top">
      <xsl:choose>
        <xsl:when test="$data_language = 'nl-NL'">
          <img border="0"  src="flag_netherlands.gif" width="48"
           height="48"/>
        </xsl:when>
        <xsl:when test="$data_language = 'en-GB'">
          <img border="0"  src="flag_great_britain.gif" width="48"
           height="48"/>
        </xsl:when>
        <xsl:when test="$data_language = 'fr-FR'">
          <img border="0"  src="flag_france.gif" width="48"
           height="48"/>
        </xsl:when>
        <xsl:when test="$data_language = 'de-DE'">
          <img border="0"  src="flag_germany.gif" width="48"
           height="48"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:text>&lt;missing image&gt;</xsl:text>
        </xsl:otherwise>
      </xsl:choose>
    </td></tr>
  </table>
```

```
    <p>
      <b><xsl:text>All other translations of the page
          title:</xsl:text></b>
      <br/><br/>
      <table border="1" cellpadding="5" cellspacing="0"
       style="border-collapse: collapse" bordercolor="#111111"
       width="400">
        <xsl:apply-templates select="page_title"/>
      </table>
    </p>
  </xsl:template>

  <xsl:template match="page_title">
    <xsl:if test="(@lang='nl-NL' and $data_language != 'nl-NL') or
    (@lang='en-GB' and $data_language != 'en-GB') or
    (@lang='fr-FR' and $data_language != 'fr-FR') or
    (@lang='de-DE' and $data_language != 'de-DE')">
      <tr>
        <td width="49" valign="top">
          <xsl:choose>
            <xsl:when test="@lang='nl-NL'">
              <img border="0"  src="flag_netherlands.gif" width="48"
               height="48"/>
            </xsl:when>
            <xsl:when test="@lang='en-GB'">
              <img border="0"  src="flag_great_britain.gif"
               width="48" height="48"/>
            </xsl:when>
            <xsl:when test="@lang='fr-FR'">
              <img border="0"  src="flag_france.gif" width="48"
               height="48"/>
            </xsl:when>
            <xsl:when test="@lang='de-DE'">
              <img border="0"  src="flag_germany.jpg" width="48"
               height="48"/>
            </xsl:when>
            <xsl:otherwise>
              <xsl:text>&lt;missing image&gt;</xsl:text>
            </xsl:otherwise>
          </xsl:choose>
        </td>
        <td width="351" valign="top">
          <font face="verdana" size="2">
            <xsl:value-of select="."/>
          </font>
        </td>
      </tr>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

4.1.2 Adding hyperlinks to your stylesheet

Every XSLT stylesheet for the current application will contain fixed HTML code. That means you can also add URLs to web pages. Use `target="_blank"` in the reference to have the link opened in your default web browser. If you leave `target="_blank"` out, the web page will open in the Adlib web browser box. Example:

```
<a target="_blank" href="http://www.adlibsoft.com">Adlib</a>
```

4.1.3 Error handling and testing

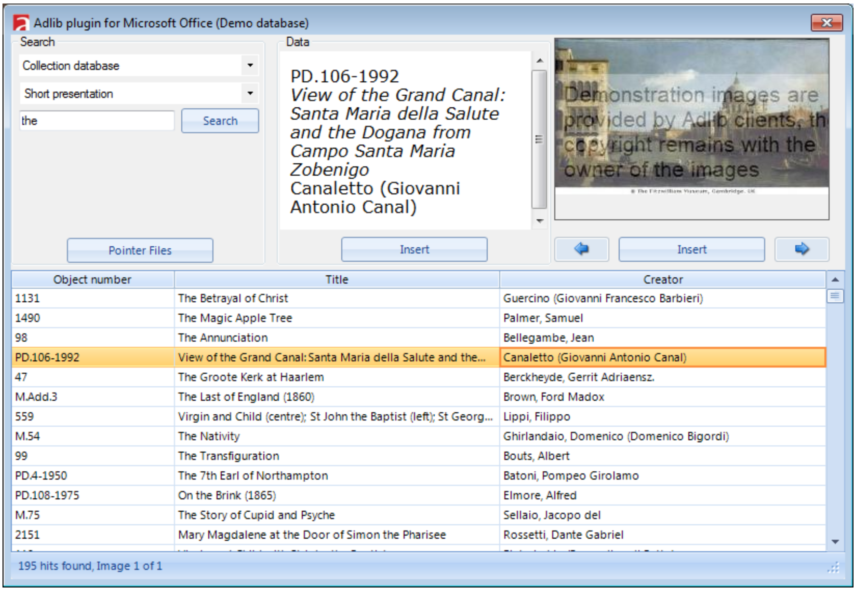
Any errors in the stylesheet will be shown inside of the web browser control with line and column numbers, and an explanation for the failure.

Testing a stylesheet that you are still working on, is very easy. Once you've set it up like explained above, and you are currently showing a record in detailed display with the web browser box, you only need to switch tabs to reload the stylesheet. So after every change in your stylesheet, save it, switch tabs in Adlib, and you can see the effect of your latest adjustment immediately.

5 Adlib Office Connect stylesheets

5.1 The Adlib Office Connect plugin

Adlib Office Connect is a plugin for Microsoft Office 2007 and 2010 (32 bit and 64 bit). In combination with an Adlib *wwwopac.ashx* server, the plugin allows you to search your Adlib SQL database from within Word, PowerPoint or Excel, with a very simple search interface: no knowledge of Adlib applications and their user-interface is required. Selected data from the search result can be copied to the current document with a single mouse click, where the user can change the layout as desired.


Adlib plugin for Microsoft Office (Demo database)
Search
Collection database
Short presentation
the
Search
Pointer Files
Data
Insert
Insert
Demonstration images are provided by Adlib clients, the copyright remains with the owner of the images

PD.106-1992
View of the Grand Canal: Santa Maria della Salute and the Dogana from Campo Santa Maria Zobenigo
Canaletto (Giovanni Antonio Canal)

Object number	Title	Creator
1131	The Betrayal of Christ	Guercino (Giovanni Francesco Barbieri)
1490	The Magic Apple Tree	Palmer, Samuel
98	The Annunciation	Bellegambe, Jean
PD.106-1992	View of the Grand Canal: Santa Maria della Salute and the...	Canaletto (Giovanni Antonio Canal)
47	The Groote Kerk at Haarlem	Berckheyde, Gerrit Adriaensz.
M.Add.3	The Last of England (1860)	Brown, Ford Madox
559	Virgin and Child (centre); St John the Baptist (left); St Georg...	Lippi, Filippo
M.54	The Nativity	Ghirlandaio, Domenico (Domenico Bigordi)
99	The Transfiguration	Bouts, Albert
PD.4-1950	The 7th Earl of Northampton	Batoni, Pompeo Girolamo
PD.108-1975	On the Brink (1865)	Elmore, Alfred
M.75	The Story of Cupid and Psyche	Sellaio, Jacopo del
2151	Mary Magdalene at the Door of Simon the Pharisee	Rossetti, Dante Gabriel

195 hits found, Image 1 of 1

5.2 The standard stylesheets

On the server side of Office Connect, in the folder that contains *wwwopac.ashx*, a number of XSLT stylesheets provide the means by which data retrieved by *wwwopac.ashx* (in grouped XML format) is tranformed to an HTML page. This HTML page is displayed in the *Data* box in the Office Connect plugin and can be copied to the relevant Microsoft Office document by clicking a button.
Each (database-specific) XSLT stylesheet, as can be selected in the second drop-down list in the left upper corner of the plugin window, specifies a layout type of the detailed display of a retrieved record,

excluding the image: the image retrieval and display is not handled by the stylesheets. Once you've retrieved a record via the Office Connect plugin, you can still choose how to present its textual data. By default the user can choose between data in table form and a long or short description*. When you insert the data into a Word document, it will be copied in the layout you chose to present it in.

> * On our demo server (http://demo.adlibsoft.com/officeConnect), for the *Collection* database there is the **Short presentation** showing the object number, object name, title and creator, the **Long presentation** showing the record number, object number, object name, description, creator, production dates and place, material, and dimensions, all preceded by appropriate labels, and there is the **Table display** which shows the institution name, the object number, title, creator and material, which arranges the data in a table with field names and borders.
> Also in our demo application, for the *Library* database there is a **Table display** showing the title, the author, material type, statement of responsibility, the ISBN, publisher, place of publication, pagination, notes, the shelf mark and copy number, which arranges the data in a table with field labels and borders.
> And finally, for the *Collection of the Amsterdam Museum*, there is a **Long presentation** format present, showing the object number, title, creator, object category and object name.

By editing the standard XSLT stylesheets (which can be found on your Office Connect server, in the folder that contains *wwwopac.ashx*) or by creating new ones, you can specify your own data layout. New stylesheets must be placed in the same folder and must be referred to in the `<StyleSheets>` section for the proper database in the *AdlibConnectPreferences.xml* file. As mentioned, do not include any image reference in these stylesheets.

As an example, consider the *simple_object_table.xslt* stylesheet which produces a result similar to:

field	**data**
institution.name	The Fitzwilliam Museum
object_number	PD.44-1999
title	A rider on a rearing horse
creator	Leonardo da Vinci
material	metalpoint
	brown ink
	paper

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

  <xsl:template match="/adlibXML">
    <xsl:apply-templates select="recordList"/>
  </xsl:template>

  <xsl:template match="diagnostic"/>

  <xsl:template match="recordList">
    <font face="verdana">
    <table border="1">
      <tr>
        <td>
          <strong>field</strong></td>
        <td>
          <strong>data</strong>
        </td>
      </tr>
      <xsl:apply-templates select="record"/>
    </table>
    </font>
  </xsl:template>

  <xsl:template match="record">
    <xsl:apply-templates select="institution.name"/>
    <xsl:apply-templates select="object_number"/>
    <xsl:apply-templates select="Title/title"/>
    <xsl:apply-templates select="Production/creator"/>
    <xsl:apply-templates select="Material/material"/>
  </xsl:template>

  <xsl:template match="record//*">
    <tr valign="top">
      <td>
        <xsl:if test="position() = 1">
          <xsl:value-of select="name()"/>
        </xsl:if>
      </td>
      <td>
        <xsl:value-of select="."/>
      </td>
    </tr>
  </xsl:template>

</xsl:stylesheet>
```

- The `<diagnostic>` node is matched, but nothing is currently done with the metadata it contains.
- Each record will be contained in its own table. The first row contains the *field* and *data* labels in separate columns. This is specified in the `recordList` template.

- For each `<record>` node, five field templates are applied which are being matched by the `record//*` template: the `//` XPath expression selects all nodes hierarchically down from the preceding node (`<record>` in this case) that match the selection (`*` in this case, a wildcard representing any possible element node) no matter where they are.
  Each of the five fields (if present in the retrieved data) will then be inserted in its own table row. In front of the first occurrence of each field, the field name will be printed: in the current standard stylesheets, field names (as defined in the data dictionary, usually not identical to screen field labels) in relevant layout types (like the table display) are always inserted in English.

5.2.1 Adding interface language dependent texts

The XSLT stylesheets receive a language parameter (`language`), generated by the Adlib Office Connect plugin, which contains the IETF language tag of the user interface language of Microsoft Office, e.g. `'en-US'` or `'nl-NL'`. This means that the standard stylesheets can be adjusted to be able to print fixed texts in the current user interface language. For example, add the following parameter declaration to *simple_object_table.xslt*:

```
<xsl:param name="language"></xsl:param>
```

Then change the `recordList` template to the following, to make the first column header interface language dependent:

```
<xsl:template match="recordList">
  <font face="verdana">
  <table border="1">
    <tr>
      <td>
       <strong>
        <xsl:choose>
          <xsl:when test="$language = 'nl-NL'">
            <xsl:text>veld</xsl:text>
          </xsl:when>
          <xsl:otherwise>
            <xsl:text>field</xsl:text>
          </xsl:otherwise>
        </xsl:choose>
       </strong>
      </td>
      <td>
        <strong>data</strong>
      </td>
    </tr>
    <xsl:apply-templates select="record"/>
  </table>
  </font>
</xsl:template>
```

Whenever the interface language of Microsoft Office is Dutch, the Dutch column header will be used:

veld	**data**
institution.name	The Fitzwilliam Museum
object_number	PD.44-1999
title	A rider on a rearing horse
creator	Leonardo da Vinci
material	metalpoint
	brown ink
	paper