# Table of Contents

1 **OVERVIEW OF PRINTING IN ALEPH** ................................................................. 4  
   1.1 A New Print Setup? ...................................................................................... 4  
   1.2 Why Customize Templates? ......................................................................... 4  
   1.3 What is XML? .............................................................................................. 4  
   1.4 What is XSL? ................................................................................................ 5  
   1.5 How Does the ALEPH Printing Mechanism Work? .................................... 6  
   1.6 XSLT and ALEPH ........................................................................................ 7  
   1.7 For More Information ................................................................................... 7  

2 **XML IN ALEPH** ................................................................................................. 8  
   2.1 The ALEPH Translation Mechanism ......................................................... 12  
      2.1.1 Summary of XML in ALEPH ........................................................................ 15  

3 **XSL IN ALEPH** .................................................................................................. 16  
   3.1 Structure of a Template - acq-s-order-slip.xsl ............................................ 16  
   3.2 Master Sections. .......................................................................................... 20  
      3.2.1 Calling Header Function (Template) .......................................................... 20  
      3.2.2 Calling Section-0n Functions .................................................................. 21  
      3.2.3 Calling Signature Function ..................................................................... 22  
      3.2.4 Functions in the Master Section ............................................................ 23  
   3.3 Data Sections .............................................................................................. 23  
      3.3.1 Header Section........................................................................................ 24  
      3.3.2 Section-01 Data Section ........................................................................ 24  
      3.3.3 Section-02 (Grid) Data Section .............................................................. 26  
      3.3.4 Signature Section (Section-03 (Free)) ..................................................... 26  
   3.4 Key ALEPH XSL Concepts........................................................................ 26  
      3.4.1 Recurrent Blocks .................................................................................. 26  
      3.4.2 Blocks With Parameters ........................................................................ 27  
      3.4.3 Salutation .............................................................................................. 27  
      3.4.4 Regular Data ......................................................................................... 27  
      3.4.5 Grid Sections ....................................................................................... 28  
      3.4.6 Differences between Free and Split Sections ............................................. 28  
      3.4.7 Signature .............................................................................................. 29  

4 **CUSTOMIZING XSL TEMPLATES** .................................................................... 30  
   4.1 Adding Fields.............................................................................................. 30  
   4.2 Removing Fields ......................................................................................... 31  
   4.3 Changing the Order of Fields ...................................................................... 32  
   4.4 Moving Specific Fields in a Report ............................................................. 33  
   4.5 Changing the Basic Layout ........................................................................ 38  
      4.5.1 Free ↔ Split ......................................................................................... 38  

© Ex Libris Ltd., 2002, 2003  
Customizing Printouts (XML and XSL)  
Version 16  
Updated: August 13, 2003
4.6 Further Customization ................................................................. 39
  4.6.1 Recurring Blocks ................................................................. 39
  4.6.2 Free / Split ................................................................. 39
  4.6.3 Grid ................................................................. 40
  4.6.4 Plain XSLs ................................................................. 40

5 MORE INFORMATION ABOUT ALEPH TEMPLATES .................. 42
  5.1.1 Template Packaging .......................................................... 42
  5.1.2 Mail XSLs ................................................................. 42
  5.1.3 Hard-coded Data in XSLs ................................................... 42

5.2 Viewing Your Changes .......................................................... 43
  5.2.1 Viewing Changes using the GUI Print Configurations .......... 43
  5.2.2 Print History ............................................................... 45

6 REFERENCE .............................................................................. 47
  6.1 XSLT Elements in ALEPH ..................................................... 47
    6.1.1 concat() ........................................................................ 47
    6.1.2 position() ...................................................................... 47
    6.1.3 substring-before () ......................................................... 47
    6.1.4 substring-after () ............................................................ 47
    6.1.5 xsl-attribute ............................................................... 47
    6.1.6 xsl: call-template .......................................................... 47
    6.1.7 xsl:element ............................................................... 47
    6.1.8 xsl:for-each ............................................................... 48
    6.1.9 xsl:if ........................................................................ 49
    6.1.10 xsl:include ............................................................... 49
    6.1.11 xsl:param ............................................................... 49
    6.1.12 xsl:template ............................................................. 49
    6.1.13 xsl:template match .................................................... 49
    6.1.14 xsl:text ............................................................... 49
    6.1.15 xsl:variable ............................................................. 49
    6.1.16 xsl: with-param ......................................................... 50
  6.2 ALEPH XSL Functions .......................................................... 50
    6.2.1 Header functions .......................................................... 50
    6.2.2 Address functions ......................................................... 50
    6.2.3 Free Functions ............................................................. 51
    6.2.4 Grid functions ............................................................. 56
    6.2.5 Split functions ............................................................. 58
  6.3 Special Templates ................................................................. 60
    6.3.1 Currency-report .......................................................... 60
    6.3.2 Order-info ............................................................... 60
    6.3.3 Catalog-records-columnar ............................................. 60
    6.3.4 Loan receipt .............................................................. 60
1 Overview of Printing in ALEPH

This document is intended primarily for system librarians. It explains the ALEPH printing process and shows how printouts can be customized.

1.1 A New Print Setup?

In ALEPH 500™, from version 15, the printing mechanism is based on the XML (eXtensible Markup Language) and XSL (eXtensible Style Language) standards.

This printing setup contains a number of features that allow per-site tailoring and control:

- Multi-tier architecture integrating:
  - Data retrieval (XML)
  - Data translation (XML)
  - Data formatting (XSL)
- Input file (XML) containing all potential data from any relevant Oracle Z table. Adding data to the display is easy
- Option to execute external programs
- Built-in support of UTF-8
- Incorporation of fast-growing standards (XML/XSL), so help and support are readily available
- Support of include files (XSL), so common blocks can be contained in common functions
- Print history

1.2 Why Customize Templates?

ALEPH reports and letters are based on formatted templates, designed by the Ex Libris development team. Depending on your library setup and user base, you may want to:

- Add or change text
- Translate text into other languages
- Add or delete fields and columns
- Change fonts, styles, and color schemes
- Add graphics and logos

This document contains information on the XML and XSL templates and elements used in the ALEPH print mechanism. For modifications, the information provided in Chapter 4, Customizing XSL Templates should be sufficient.

1.3 What is XML?

XML is a standard for data description optimized for delivery via the Web, from server to client, or even from server to server. It looks similar to HTML as it contains
elements with tags and text between them. The difference is that an XML data file contains tags which contain semantic information, but which do not contain any presentation information (font, colors, spacing, and so on). This is contained in the XSL style sheet. A style sheet is basically a statement of display rules, specifying the display attributes of elements in the source code.

The separation of content and styling information enables the same data file to be displayed in different ways. It also enables users to view the data file according to their preferences and abilities, just by modifying the style sheet.

An example taken from an ALEPH XML template (acq-s-order-slip.xml):

```
<z72-vendor-address-occ1>AMERICAN METEOROLOGICAL SOCIETY</z72-vendor-address-occ1>
<z72-vendor-address-occ2>45 BEACON STREET</z72-vendor-address-occ2>
<z72-vendor-address-occ3>BOSTON, MA 02108</z72-vendor-address-occ3>
```

The data represents different parts of the vendor's address. Tags such as `<z72-vendor-address-occ1>`, `<z72-vendor-address-occ2>`, `<z72-vendor-address-occ3>` and so on, are self-describing. That is, they refer to the type of data contained within, and do not contain styling information.

### 1.4 What is XSL?

XSL is a style sheet language that takes an XML file as input and can display it in many different ways. We refer here to HTML output.

A possible XSL section for the XML example above might be:

```
<xsl:template name="vendor-address">
  <xsl:call-template name="table-open-full"/>
  <tr><td width="70%"><xsl:value-of select="//z72-vendor-address-occ1"/></td></tr>
  <tr><td width="70%"><xsl:value-of select="//z72-vendor-address-occ2"/></td></tr>
  <tr><td width="70%"><xsl:value-of select="//z72-vendor-address-occ3"/></td></tr>
  <xsl:call-template name="table-close"/>
```

As you can see, the XSL contains a combination of HTML tags, containing display information, and field tags, matching field tags in the XML file, containing descriptive information.
If we combine this XSL example with the XML example, we get the following HTML:

```html
<tr>
    <td width="70%">AMERICAN METEOROLOGICAL SOCIETY</td>
</tr>
<tr>
    <td width="70%">45 BEACON STREET</td>
</tr>
<tr>
    <td width="70%">BOSTON, MA 02108</td>
</tr>
```

The resulting printout looks like this:

<table>
<thead>
<tr>
<th>AMERICAN METEOROLOGICAL SOCIETY</th>
</tr>
</thead>
<tbody>
<tr>
<td>45 BEACON STREET</td>
</tr>
<tr>
<td>BOSTON, MA 02108</td>
</tr>
</tbody>
</table>

This is a very simple example. We will see how XML and XSL are used in the ALEPH printing mechanism and look at a number of different examples.

1.5 How Does the ALEPH Printing Mechanism Work?

A print request (batch or online) creates a data file in raw XML format. The data file includes all the data fields of all the records that have been retrieved for the printed output. If produced via batch services, the file is stored in the library's /print directory. Otherwise the files are saved under the module's /files directory. Each data field is identified by an XML tag.

For each report and letter in ALEPH, a print template is defined. This template determines:

- Which fields of the XML file are part of the report or letter
- How these fields are displayed

The print templates are XSL files which are located in the bibliographic library's /form_<lng> directory. For example, the print template for the Acquisitions Order Slip is acq-s-order-slip.xsl. The XSL files themselves contain little or no styling information so they are not stylesheets in the conventional sense - rather, they contain tags which call functions contained in global XSL files.

These global XSL files are in general use in ALEPH reports and letters, and they are referred to by all the specific XSL templates (for example, funcs.xsl, funcs-address.xsl, funcs-bib-info.xsl and so on). They contain definitions for the rendering of common report blocks such as the standard salutations (greetings), signatures, sublibrary addresses, patron addresses, bibliographic information, and so on.

The XSL data files are then packaged and transferred to the workstation. This is similar to the way cataloging templates and tables are packaged and transferred.
The XSL print templates generate HTML pages from the XML data. This occurs on your workstation, using an XSL processor included in your ALEPH package. The resulting HTML page is print output, suitable for both online printing from a GUI module, and printing through the Job Manager (for printouts produced by batch services).

In principle, you can maintain XSL print templates (translate, add / remove fields, and so on) without actually knowing XSL, and rely on the patterns found in ALEPH’s default XSL files.

1.6 XSLT and ALEPH

The XSLT (eXtensible Style Language Transformations) portion of XSL makes it possible for one XML document to be transformed into another according to an XSL Style sheet. ALEPH templates use a very small set of XSLT elements. These are briefly described in section 6.1 XSLT Elements in ALEPH.

1.7 For More Information

XML, XSL and XSLT are complex and evolving languages and ALEPH only uses a small number of elements. For more information on these languages, we recommend the following books:

*XML By Example by Benoit Marchal. December 1999 Que Press Inc.; ISBN 0789722429*

We do not recommend that you visit the W3C (World Wide Web Consortium) Web site (www.w3.org). W3C is the independent body that sets the standards for XML and XSL. Their Web site is targeted to advanced XML users.
2 XML in ALEPH

The data for printouts in ALEPH is packaged in XML files. The structure of these files is very simple: at the beginning of the file, there are several tags for general information such as the name of the template, the language category, and so on:

```xml
<?xml version="1.0"?>
<printout>
<form-name>acq-s-order-slip</form-name>
<form-language>eng</form-language>
<form-format>01</form-format>

This is the opening part of each XML file. printout is a tag, which encloses all the XML. The form-name, form-language, and form-format fields help the system to locate the relevant XSL stylesheet.

Next comes the actual data for the query (a search question which tells ALEPH which information to retrieve). The data is contained in one or more section tags depending on the specific query:

```xml
<section-01>
General information of the form.
Usually appears only once in a printout.
</section-01>

These sections can be repeated.

The data can be transformed (for example, 20020415 into 15/04/2002) before the XML is created, according to special user-defined rules (for more details see the ALEPH Translation Mechanism section (2.1)).

Every print output has its own XSL file that defines which tag contents are included, and how they appear.

Single Section XML Files
Some queries, such as the acquisitions arrival slip, contain information for just one item. The data for this query will have just one section. It looks like this:

```xml
<form-name>acq-arrival-slip</form-name>
<form-language>eng</form-language>
<form-format>00</form-format>
<section-01>
<sub-library-address-1-occ1>Archives Department</sub-library-address-1-occ1>
<sub-library-address-1-occ2>Lincoln Library</sub-library-address-1-occ2>
<sub-library-address-1-occ3>808 Log Lane</sub-library-address-1-occ3>
<sub-library-address-1-occ4>Chicago, IL 60614</sub-library-address-1-occ4>
<sub-library-address-1-occ5>thechoice@exlibris-usa.com</sub-library-address-1-occ5>
<sub-library-address-1-occ6>Tel# 173.404.5527</sub-library-address-1-occ6>
```
Multiple Section XML Files

Some queries have data that refers to multiple items. For example, the Patron Circulation Summary, accessed from the Circulation GUI, contains the following structure:

- **Section-01** – once for patron details
- **Section-02** – repeated for loans
- **Section-03** – repeated for holds

This type of query generates a very large XML file, composed of data and data tags from all of the relevant Oracle Z tables. Here are a few samples of the XML file generated when the query is run:

```xml
<section-01> containing the patron details:

.../
<z302-id>0245</z302-id>
<z302-proxy-for-id></z302-proxy-for-id>
<z302-primary-id></z302-primary-id>
<z302-name-key>alex johns 0245</z302-name-key>
<z302-open-date>23/08/2000</z302-open-date>
<z302-update-date>23/07/2002</z302-update-date>
<z302-con-lng>ENG</z302-con-lng>
<z302-alpha>L</z302-alpha>
<z302-name>Alex, Johns</z302-name>
<z302-title>Dr.</z302-title>
.../

The first **<section-02> containing details of a loan**

.../
<bib-info>Journal of sounds</bib-info>
<z36-doc-number>000007592</z36-doc-number>
<z36-item-sequence>000030</z36-item-sequence>
<z36-id>0245</z36-id>
<z36-number>000003680</z36-number>
<z36-material>ISSUE</z36-material>
<z36-sub-library>Law Library</z36-sub-library>
<z36-status>Active</z36-status>
<z36-loan-date>14/07/2002</z36-loan-date>
.../
The second <section-02> containing details of a second loan
.../

After that, the first <section-03>, containing details of a hold request:

.../

And then a second <section-03>, containing details of another hold request:

.../

And finally, a third <section-03>

.../
The resulting HTML file, after XSL processing, looks like this:

```
23/07/2002
Bor-list-00

Patron Circulation Summary

Dr. Alex Johns
119 Station Road
Hayes, Middlesex, UK
UB3 4BX
Alex.Johns@exlibris.co.uk
Patron ID: 0245

Loans

<table>
<thead>
<tr>
<th>Bib Info</th>
<th>Due Date</th>
<th>Description</th>
<th>Sublibrary</th>
<th>Collection</th>
<th>Item status</th>
<th>Call No. 1</th>
<th>Barcode</th>
<th>Proxy ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Journal of sounds</td>
<td>17/07/2002</td>
<td>2000 1 1</td>
<td>Law Library</td>
<td>One Day Loan</td>
<td>* 7 5 9 2 - 3 0 *</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The List of Requests/Smith, K.</td>
<td>12/09/2002</td>
<td></td>
<td>Lincoln Library</td>
<td>ILL Home Loan</td>
<td>* 5 8 6 9 - 1 *</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Holds

<table>
<thead>
<tr>
<th>Bib Info</th>
<th>Description</th>
<th>Sublibrary</th>
<th>Collection</th>
<th>Call No. 1</th>
<th>Open Date</th>
<th>End Request Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lectures in Computer Science</td>
<td>Archives Library Köln</td>
<td>General</td>
<td>17/06/2002</td>
<td>17/06/2003</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hawking, S. W. (Stephen W.) : A Brief History of Time: From the Big Bang to Black Holes /Stephen W. Hawking ; Introduction by Carl Sagan ; Illustrations by Ron Miller</td>
<td>Science Library</td>
<td>Depository</td>
<td>02/08/2001</td>
<td>02/08/2002</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
2.1 The ALEPH Translation Mechanism

The ALEPH translation mechanism is responsible for changing the data from database data to meaningful formatted data.

<table>
<thead>
<tr>
<th>Database Data</th>
<th>Printform Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>20020415</td>
<td>15/04/2002</td>
</tr>
<tr>
<td>Y</td>
<td>Yes</td>
</tr>
<tr>
<td>000430</td>
<td>430</td>
</tr>
</tbody>
</table>

In xxx01/form_<lng> there are two types of files:

- xxx[-nn].xsl – the XSL print templates. These are explained in Chapter 3, XSL in ALEPH.
- global.trn and [form name].trn (for example, serials-claim-report.trn). These are ALEPH tables which contain data relating to the ALEPH translation rules. Here is an extract from the global.trn ALEPH table:

<table>
<thead>
<tr>
<th>!tag</th>
<th>type</th>
<th>parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>!general values</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.../...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>z68-material-type</td>
<td>pc-tab-exp s acq_order_material</td>
<td></td>
</tr>
<tr>
<td>z68-method-of-aquisition</td>
<td>pc-tab-exp s acq_order_method</td>
<td></td>
</tr>
<tr>
<td>z68-no-units</td>
<td>integer s</td>
<td></td>
</tr>
<tr>
<td>z68-open-date</td>
<td>date s</td>
<td></td>
</tr>
<tr>
<td>z68-order-date</td>
<td>date s</td>
<td></td>
</tr>
<tr>
<td>z68-order-delivery-type</td>
<td>pc-tab-exp s acq_order_delivery</td>
<td></td>
</tr>
<tr>
<td>z68-order-group</td>
<td>pc-tab-exp s acq_order_group</td>
<td></td>
</tr>
<tr>
<td>z68-order-status</td>
<td>pc-tab-exp s acq_order_status</td>
<td></td>
</tr>
<tr>
<td>z68-order-status-date</td>
<td>date s</td>
<td></td>
</tr>
<tr>
<td>z68-order-type</td>
<td>text s m=monograph s=serial</td>
<td></td>
</tr>
<tr>
<td>o=standing</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Col. 1: field name - XML tag

Col. 2: data type - ALEPH translation method

Col. 3: single / multiple – use either S or M

Col. 4: (additional) parameters

M is used in Col. 3 when data is composed of several codes delimited by spaces, and each is less than 100 characters.
Example:

<p-currency-name>usd gbp jpy</p-currency-name>

The ALEPH translation table has:

<table>
<thead>
<tr>
<th>!tag</th>
<th>type</th>
<th>parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-currency-name</td>
<td>currency</td>
<td>m</td>
</tr>
</tbody>
</table>

And the result is a comma-delimited list of ALEPH translations:

<p-currency-name>us dollar, british pound, japanese yen</p-currency-name>

First, the mechanism searches for ALEPH translation rules in the specific <form name> trn ALEPH table. If a rule is found for the specific field, it is translated using this rule. If not, the mechanism starts to browse the global.trn file. If a rule is found, the data is translated using this rule. If not, then the original data are inserted into the XML file.

The special ALEPH translation keyword NONE indicates no ALEPH translation whatsoever.

The sequence of the translation file depends on the information contained in the standard header of any XML template. For example:

<form-name>acq-s-order-slip</form-name>
<form-language>GER</form-language>
<form-format>01</form-format>

The form number (<form-format>) can be any 2-digit number. If the xxxx-nn.xsl is not found, the mechanism searches for xxxx.xsl.

The language (<form-language>) can be any defined language. If the directory is not found, the default is form_eng.

The ALEPH xxx.trn file is searched in the following order:

/usm01/form_ger/acq-s-order-slip-01.trn (format of the form)
/usm01/form_ger/acq-s-order-slip.trn (name of the master version of the form)
/usm01/form_ger/global.trn (file containing global trn definitions)

If the translation file is not found, the system will search again based on $control_lng.

ALEPH translation rules can contain more than one instruction. For example, you can choose to translate a decimal number in two steps:

1. Delete leading zeros
2. Delete the + sign for positive numbers
What Can be Translated?
The values listed here refer to the ALEPH translation codes in col. 2 (data type).

add-sign – adds a minus (-) or a plus (+) depending on the credit/debit value (c, d).
cash-status/cash-desc – returns the relevant text for cash status and description (for example, paid, not paid) using the cash_status_heading parameter in error_<lng>.
collection – returns the full name of the item collection, using tab40.<lng>.
credit-txt/debit-txt – replaces signs with text. Adds text specified in the parameter for sums proceeded by + or -.
currency – returns the currency text as defined in the Z83 currency table.
date – retrieves a date string and separates the year, month, day with / No parameter required.
decimal – formats decimal numbers by deleting leading zeros before the decimal number, and deleting zeros after the decimal point.
del-sign – removes + or – from prefix.
integer – formats non-decimal numbers by suppressing leading zeros.
ITEM-STA – returns the item status using tab15.<lng>.
LOAN-STA – returns the loan status including due date, transit, lost, recall, and so on.
LOCATION/LOCATION_B – formats call numbers.
NONE – returns the original value, no parameter required.
OBJ-CODE – returns the Budget object code text as defined.
in pc_tab_exp_field.<lng> or other values as needed.
pc-tab-exp – translates a code value to text, using pc_tab_expand_field.<lng>.
SIGN1 – provides accounting notation for signed numbers:
- number is translated to (number). For example, -2013 becomes (2013).
+ number is translated to number. For example, +2013 becomes 2013.
SIGN2 – provides accounting notation for signed numbers:
- number is kept as –number. For example, -2013 stays as –2013.
+ number is translated to number. For example, +2013 becomes 2013.
SUB-LIB – returns the sublibrary name as defined in tab_sub_library.<lng>.
TEXT – Free text translation. Replaces the code with the next word coming immediately after the = sign, and puts in the param-column the codes and the values combined with the = sign. If you want to put more than one code, separate them with one or more blanks (for example, A=abc B=def). You can put up to 100 codes in the parameters column. Each parameter can be a maximum of 100 characters long. If the replaced text is more than one word, combine the words with the _ sign (for example, hello_world).

TIME – separates hours, minutes, seconds with /. No parameter required.

Z30-MATER – returns the item material type, using tab25. <lng>.

The translate_on / translate_off Mechanism

To use the translate_on / translate_off mechanism, from the command prompt, enter as appropriate:

```plaintext
>>translate_on
```

- The translate_off command switches the translation mechanism off.
- The translate_on command switches the translation mechanism on.

The translate_on / translate_off mechanism is useful when analyzing problems or checking real data. When using the translate_on / translate_off mechanism, you must restart the servers/batch queue.

2.1.1 Summary of XML in ALEPH

- Each section with the same name contains the same type of data (that is, the same tags).
- The values (tag contents) differ according to the item.
- Most XML files contain one to three different sections (that is, section-01 to section-03).
- The highest possible section number available is 07.
- A data translating mechanism handles format and display related issues.
3 XSL in ALEPH

An XSL file in ALEPH consists of a set of templates. Each template matches a set of elements in the source XML file and then extracts the contents of the matched elements to a resulting HTML file. The resulting HTML file forms the basis for the printout.

3.1 Structure of a Template - acq-s-order-slip.xsl

acq-s-order-slip.xsl is a basic version of an XSL template suited for a serial order slip letter. In this chapter, we will examine it in detail. First, let us take a look at a printout:

---

15/07/2002
acq-s-order-slip-01

Acquisitions Order Slip

Acquisitions Unit 1
Manag. Building
Ex Libris University
777 Biblio Byway
Chicago, IL 60614
thechoice@exlibris-usa.com
Tel# 773.404.5327

AMERICAN METEOROLOGICAL SOCIETY
45 BEACON STREET
BOSTON, MA 02108
Dear Sir/Madam,

We would like to place an order for the following items:

47 p. ; 23 cm.

Order Number: 593
No. of Items: 3
Sublibrary: Education Library

Order Date: 15/07/2002
Claim Date: 04/08/2002
Unit Price: 25.00
Total Price: 75.00
Currency: US Dollar

Trans. Type: Encumbrance
Budget Number: MAR-2002
Open Date: 15/07/2002
Currency: US Dollar
Original Sum: 75.00
Local Sum: 15.00

Sincerely,
Acquisitions Department
Its underlying XSL looks like this:

```xml
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:include href="functs.xsl"/>

    <xsl:template match="/">
        <xsl:call-template name="header"/>
        <!--section-01 (SPLIT)-->
        <xsl:for-each select="//section-01">
            <xsl:call-template name="section-01"/>
        </xsl:for-each>
        <!--section-02 (GRID)-->
        <xsl:for-each select="//section-02">
            <xsl:if test="position() = 1">
                <xsl:call-template name="section-02">
                    <xsl:with-param name="header" select="'header'"/>
                </xsl:call-template>
            </xsl:if>
            <xsl:call-template name="section-02"/>
        </xsl:for-each>
        <!--section-03 (FREE)-->
        <xsl:for-each select="//section-03">
            <xsl:call-template name="section-03"/>
        </xsl:for-each>
        <xsl:call-template name="signature"/>
    </xsl:template>

    <!-- START DATA -->
    <xsl:template name="header">
        <xsl:call-template name="header-gen">
            <xsl:with-param name="title" select="'Acquisitions Order
Slip'"/>
        </xsl:call-template>
    </xsl:template>

    <!--SECTION-01 (SPLIT)-->
    <xsl:template name="section-01">
        <xsl:call-template name="sublib-address"/>
        <xsl:call-template name="vendor-address"/>
        <xsl:call-template name="generic-line">
            <xsl:with-param name="line" select="'salutation_string##We
would like to place an order for the following items:'"/>
        </xsl:call-template>
        <xsl:call-template name="salutation-end"/>
        <xsl:call-template name="bib-info-hdr"/>
        <xsl:call-template name="table-split-open"/>
        <xsl:call-template name="display-gen-split">
            <xsl:with-param name="label" select="'Order Number:'"/>
            <xsl:with-param name="value" select="./z68-order-number"/>
        </xsl:call-template>
        <xsl:call-template name="display-gen-split">
            <xsl:with-param name="label" select="'Vendor Ref. No:'"/>
            <xsl:with-param name="value" select="./z68-vendor-reference-no"/>
        </xsl:call-template>
        <xsl:call-template name="display-gen-split">
            <xsl:with-param name="label" select="'No. of Items:'"/>
            <xsl:with-param name="value" select="./z68-no-units"/>
        </xsl:call-template>
    </xsl:template>
</xsl:stylesheet>
```
<!--SECTION-02 (GRID)-->
Although the code takes up several pages, it is simply a long set of logical rules, translated from top to bottom.

Most ALEPH XSL files have the same structure as `acq-s-order-slip.xsl`:

- **Master sections** – contain the functions ensuring that every data section is displayed.
- **Data sections** – contain the functions that display the data of the current visited section.
3.2 Master Sections

In the Master section of `acq-s-order-slip.xsl`, the code consists of three basic elements:

- Calling header function (template)
- Calling section-0n functions
- Calling signature function

```xml
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:include href="funcs.xsl"/>
  <xsl:template match="/
    <xsl:call-template name="header"/>
  </xsl:template>

  <!--section-01 (SPLIT)-->  
  <xsl:for-each select="//section-01">
    <xsl:call-template name="section-01"/>
  </xsl:for-each>

  <!--section-02 (GRID)-->  
  <xsl:for-each select="//section-02">
    <xsl:if test="position() = 1">
    <xsl:call-template name="section-02"/>  
    <xsl:with-param name="header" select="'header'"/>
    </xsl:call-template>
  </xsl:if>
  <xsl:call-template name="section-02"/>

  <!--section-03 (FREE)-->  
  <xsl:for-each select="//section-03">
  <xsl:call-template name="section-03"/>
  </xsl:for-each>

  <xsl:call-template name="signature"/>
</xsl:template>
</xsl:stylesheet>
```

The different elements that make up the master sections are explained in the following sections.

3.2.1 Calling Header Function (Template)

The lines which make up the Calling Header Function always appear at the top of the XSL stylesheet. Although this example is taken from `acq-s-order-slip.xsl`, they are in general use in ALEPH XSL files.

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

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

</xsl:stylesheet>
```

These lines define XSL version 1.0 as the language on which this file is based.

```xml
<xsl:include href="funcs.xsl"/>
```

This include line is the first command of all ALEPH XSLs, and ensures that all functions contained in the global `funcs.xsl` file (which contains all general functions) can be used in this file.
These two lines point to the header template, located further down at the beginning of the data sections.

### 3.2.2 Calling Section-0n Functions

The lines which make up the Calling section-0n functions always appear underneath the Calling header function. The structure of these lines varies from file to file. These lines ensure that the data sections, consisting of section-<nn> templates, are displayed.

A section-<nn> template is a repeatable part of each XML; however, a section-01 usually appears just once. There are three basic formats or layouts in which a given section-<nn> can be displayed: **Free; Grid; Split.** Data sections can be in one of three types of layout:

**Split** - the **Split** layout presents data in two columns (in the acq-s-order-slip.xsl template, section-01 is a Split layout):

<table>
<thead>
<tr>
<th>Order Number:</th>
<th>593</th>
<th>Order Date:</th>
<th>15/07/2002</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Items:</td>
<td>3</td>
<td>Claim Date:</td>
<td>04/08/2002</td>
</tr>
<tr>
<td>Sublibrary:</td>
<td>Education Library</td>
<td>Unit Price:</td>
<td>25.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Total Price:</td>
<td>75.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Currency:</td>
<td>US Dollar</td>
</tr>
</tbody>
</table>

**Grid** - the **Grid** layout shows data in row-and-column format with grid lines separating rows and columns (in the acq-s-order-slip.xsl template, section-02 is a Grid layout):

<table>
<thead>
<tr>
<th>Trans. Type:</th>
<th>Budget Number:</th>
<th>Open Date:</th>
<th>Currency:</th>
<th>Original Sum:</th>
<th>Local Sum:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encumbrance</td>
<td>MAR-2002</td>
<td>15/07/2002</td>
<td>US Dollar</td>
<td>75.00</td>
<td>15.00</td>
</tr>
</tbody>
</table>

**Free** - the **Free** layout presents each data field on a separate line, going down the page (in the acq-s-order-slip.xsl template, section-03, the signature section, is a Free layout).

Sincerely,
Acquisitions Department

The following example is also taken from acq-s-order-slip.xsl:

```xml
<!--section-01 (split)-->   
<xsl:for-each select="/section-01">   
  <xsl:call-template name="section-01"/>
</xsl:for-each>
```
In this loop we call a template, section-01, which is located under the header template in the data sections. In this case, section-01 is a Split section.

```xml
<!--section-02 (grid)-->  
<xsl:for-each select="/section-02">
  <xsl:if test="position() = 1">
    <xsl:call-template name="section-02">
      <xsl:with-param name="header" select="'header'"/>
    </xsl:call-template>
  </xsl:if>
  <xsl:call-template name="section-02"/>
</xsl:for-each>
```

In this loop, a section-02 template, is called. In this case, section-02 is a Grid section.

```xml
<!--section-03 (free)-->  
<xsl:for-each select="/section-03">
  <xsl:call-template name="section-03"/>
</xsl:for-each>
```

In this loop, a section-03 template, is called. In this case, section-03 is a Free section.

### 3.2.3 Calling Signature Function

The lines which make up the Calling signature function always appear after the Calling section-<0n> functions.

```xml
<xsl:call-template name="signature"/>
```

These lines ensure that the signature section, consisting of a signature template, appears at the end of the printout.
3.2.4 Functions in the Master Section

There are 2 XSL elements used in the header section of the XSL, for-each and section-0n:

```
<xs1:for-each select="/section-0n">

for-each instructs the XSL processor to look for every occurrence of the tag “section-0n” and to do something with its contents. Each section is visited in the order of its appearance in the XML file.

<xs1:call-template name="section-0n"/>
```

When a “section-0n” parameter is encountered, a function called “section-0n” is called. The context of this function is the current “section-0n”. These functions form the core of the rest of the XSL file. These functions are examined later on in this document.

Note that grid-open and table-close are functions that are not found in the XSL template. They are utility functions used by most templates and defined in one of several common files used by all templates.

3.3 Data Sections

The data part of the XSL includes all the section-0n functions that refer to actual fields (tags) of the XML file to be displayed.

Most ALEPH XSL templates have a similar structure, namely:

- For the opening part of the XML, display the form data in the header
- For every section-01, display all relevant current section-01 data
- For every section-02, display all relevant current section-02 data

And so on until the highest section-0n of the XML file is reached.

Some basic blocks that appear in many templates, such as sublibrary address, patron address and so on, do not appear with explicit tag names – instead, they specify functions to handle them.

For explanations of XSL functions used in ALEPH XSL templates and not explained here, refer to section 6.2 ALEPH XSL Functions.

The following paragraphs describe in detail the data sections of the acq-s-order-slip.xsl template.
3.3.1 Header Section
These lines define how the header of the printout will look

```xml
<!-- start data -->
<xsl:template name="header">
  <xsl:call-template name="header-gen">
    <xsl:with-param name="title" select="'Acquisitions Order Slip'"/>
  </xsl:call-template>
</xsl:template>
```

header-gen is a function which is taken from the global funcs.xsl template. It takes the general form data from the opening part of the XML file. As you can see, it displays the date, the XML form name and form number, a horizontal line and shows the title in centred alignment.

3.3.2 Section-01 Data Section
This is how the section-01 displays the sublibrary's address and the vendor's address:

```xml
<!--section-01 (Split)-->  
<xsl:template name="section-01">
  <xsl:call-template name="sublib-address"/>
  <xsl:call-template name="vendor-address"/>
  <xsl:call-template name="generic-line"/>
</xsl:template>
```

Acquisitions Unit 1  
Manag. Building  
Ex Libris University  
777 Biblio Byway  
Chicago, IL 60614  
thechoice@exlibris-usa.com  
Tel# 773.404.5327  

AMERICAN METEOROLOGICAL SOCIETY  
45 BEACON STREET  
BOSTON, MA 02108  

Acquisitions Order Slip

15/07/2002  
acq-s-order-slip-01
Next, the salutation, called by `generic-line`. Note the use of `##` for a double line break:

```xml
<xsl:call-template name="generic-line">
  <xsl:with-param name="line" select="'salutation_string##We would like to place an order for the following items:'"/>
</xsl:call-template>
<xsl:call-template name="salutation-end"/>
```

Dear Sir/Madam,

We would like to place an order for the following items:

The bibliographic information concerning the item ordered appears:

```xml
<xsl:call-template name="bib-info-hdr"/>
```


And now the specifics of the order (note that only these lines in the `section-01` are actually in Split format):

```xml
<xsl:call-template name="display-gen-split">
  <xsl:with-param name="label" select="'No. of Items:'"/>
  <xsl:with-param name="value" select="./z68-no-units"/>
</xsl:call-template>
```

<table>
<thead>
<tr>
<th>Order Number:</th>
<th>593</th>
<th>Order Date:</th>
<th>15/07/2002</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Items:</td>
<td>3</td>
<td>Claim Date:</td>
<td>04/08/2002</td>
</tr>
<tr>
<td>Sublibrary:</td>
<td>Education Library</td>
<td>Unit Price:</td>
<td>25.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Total Price:</td>
<td>75.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Currency:</td>
<td>US Dollar</td>
</tr>
</tbody>
</table>
3.3.3 Section-02 (Grid) Data Section

Moving further down, here is a view of part of the section-02 Grid:

```xml
<xsl:call-template name="display-grid-gen">
  <xsl:with-param name="label" select="'Open Date'"/>
  <xsl:with-param name="value" select="./open-date"/>
  <xsl:with-param name="header" select="$header"/>
</xsl:call-template>
```

<table>
<thead>
<tr>
<th>Trans. Type</th>
<th>Budget Number</th>
<th>Open Date</th>
<th>Currency</th>
<th>Original Sum</th>
<th>Local Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encumbrance</td>
<td>MAR-2002</td>
<td>15/07/2002</td>
<td>US Dollar</td>
<td>75.00</td>
<td>15.00</td>
</tr>
</tbody>
</table>

3.3.4 Signature Section (Section-03 (Free))

Finally, the signature lines. In acq-s-order-slip.xsl, these lines are in Free format:

```xml
<xsl:template name="signature">
  <xsl:call-template name="generic-line">
    <xsl:with-param name="line" select="'####Sincerely,##Acquisitions Department'"/>
  </xsl:call-template>
</xsl:template>
```

Sincerely,  
Acquisitions Department

3.4 Key ALEPH XSL Concepts

3.4.1 Recurrent Blocks

The recurrent blocks are:

- sublib-address
- patron-address
- vendor-address
- transfer-address
- bib-info-hdr

The following is an example of a block of data that appears in many templates, always with the same tag name and always displayed in the same way:

```xml
<xsl:call-template name="sublib-address"/>
```

The “sublib-address” function retrieves the relevant tags and adds the necessary HTML tags for the correct display.
3.4.2 Blocks With Parameters

There are others kinds of recurrent blocks: those that need parameters. These are:

header-gen
salutation

The recurrent block, header-gen, (which is explained in 3.3.1 Header Section) requires a printout's title as a parameter (for example, `<xsl:with-param name="title" select="'Acquisitions Order Slip'"/>`, but salutation is more complicated, as it can span many lines and may contain tag data (see following section).

3.4.3 Salutation

salutation can span several lines.

```
<xsl:call-template name="generic-line">
  <xsl:with-param name="line" select='concat("salutation_string##We are sorry to inform you that
the following photocopy, which you requested on ", //z38-open-date,",
cannot be filled at this time. Please inform us by return mail if
you would like us to supply you with the item when available, or
whether you would prefer to cancel the request.")'/>
  <xsl:with-param name="val1" select='//z38-open-date'/>
</xsl:call-template>
```

The actual output is:

Dear Sir/Madam,

We are sorry to inform you that the following photocopy, which you requested on 01/01/2002, cannot be filled at this time. Please inform us by return mail if you would like us to supply you with the item when available, or whether you would prefer to cancel the request.

The generic-line function is responsible for displaying a line of text in a hardcoded style and width. “Dear Sir/Madam” replaces salutation_string. The ## is the prompt for a new line. The text is concatenated with z38-open-date, which is 01/01/2002 in this case.

After all salutation lines have been executed, salutation-close must be called.

3.4.4 Regular Data

Regular data that is to be displayed as label is displayed using another “common” function – “display-gen”, as in:

```
<xsl:call-template name="display-gen">
  <xsl:with-param name="label" select="'Note:'"/>
  <xsl:with-param name="value" select="./z38-note-1"/>
</xsl:call-template>
```

There are several parameters display-gen can get. For details see section 6.2 XSL Functions.
In order to align all values independently of the length of their labels, they must be enclosed in an HTML table. To do this, all display-gen parameters must be enclosed between the lines:

```xml
<xsl:call-template name="table-open"/>
<xsl:call-template name="table-close"/>
```

### 3.4.5 Grid Sections

A section-<0n> template is needed to build a Grid layout. A Grid section uses the display-grid-gen function. Grid data and headers are displayed when the display-grid-gen function is used. An example:

```xml
<xsl:call-template name="display-grid-gen">
  <xsl:with-param name="label" select="'Trans. Type:'"/>
  <xsl:with-param name="value" select="./z601-type"/>
</xsl:call-template>
```

which supports this display:

<table>
<thead>
<tr>
<th>Trans. Type:</th>
<th>Encumbrance</th>
</tr>
</thead>
</table>

This function can contain many different types of parameter. For further details, see chapter 4, Customizing Templates.

### 3.4.6 Differences between Free and Split Sections

The difference between Free and Split layout is whether the data is enclosed in two columns, or simply line-by-line. Some common functions:

**Free:** table-open; table-close

**Split:** table-split-open; table-split-right; table-split-close

In the following example, we can see a “pure” Split sections, but both Free and Split layouts can be mixed in one section – just surround the display-gen with the relevant table- functions. For more details, see chapter 4, Customizing Templates.

```xml
<!--section-03 (split)-->  
<xsl:template name="section-03">
  <xsl:call-template name="table-split-open"/>
  <xsl:call-template name="display-gen">
    <xsl:with-param name="label" select="'additional number 2:'"/>  
    <xsl:with-param name="value" select="./z68-order-number-2"/>  
  </xsl:call-template>
  <xsl:call-template name="display-gen">
    <xsl:with-param name="label" select="'order group:'"/>  
    <xsl:with-param name="value" select="./z68-order-group"/>  
  </xsl:call-template>
  <xsl:call-template name="display-gen">
    <xsl:with-param name="label" select="'isbn/issn:'"/>  
    <xsl:with-param name="value" select="./z68-isbn"/>  
  </xsl:call-template>
  <xsl:call-template name="table-split-right"/>
  <xsl:call-template name="display-gen">
    <xsl:with-param name="label" select="'unit price:'"/>  
    <xsl:with-param name="value" select="./z68-unit-price"/>  
  </xsl:call-template>
</xsl:template>
```
3.4.7 Signature

<xsl:call-template name="signature"/>
: this line appears only where a signature is needed. For example, “Sincerely, XX Department”.
4 Customizing XSL Templates

There are several levels of customization and different ways in which to customize.

The rest of this section shows you how you can customize the default XSL templates by using a simple file editor (vi on Unix or Notepad on Windows).

4.1 Adding Fields

Before adding a new field to the display, check first if this field is included in the XML printout file. To do this, set your GUI interface print default to raw XML, generate the file, and check the resulting XML file.

Let us take, as an example, ill-print-letter.xsl. You want to add z41-sequence (ILL request sequence number) after the currently displayed z303-name.

The relevant area in the XSL print template is:

```xml
<xsl:call-template name="display-gen">
  <xsl:with-param name="label" select="'Patron name:'"/>
  <xsl:with-param name="value" select="./z303-name"/>
</xsl:call-template>

<xsl:call-template name="display-gen">
  <xsl:with-param name="label" select="'Cataloger:'"/>
  <xsl:with-param name="value" select="./z41-cataloger"/>
</xsl:call-template>
```

Copy and paste the bold lines:

```xml
<xsl:call-template name="display-gen">
  <xsl:with-param name="label" select="'Patron name:'"/>
  <xsl:with-param name="value" select="./z303-name"/>
</xsl:call-template>

<xsl:call-template name="display-gen">
  <xsl:with-param name="label" select="'Cataloger:'"/>
  <xsl:with-param name="value" select="./z41-cataloger"/>
</xsl:call-template>
```

then change the pasted Patron name to sequence: (or to whichever label you want), and z303-name to z41-sequence.
The new file looks like this:

```xml
<xsl:call-template name="display-gen">
  <xsl:with-param name="label" select="'Patron name:'"/>
  <xsl:with-param name="value" select="./z303-name"/>
</xsl:call-template>
<xsl:call-template name="display-gen">
  <xsl:with-param name="label" select="'Sequence No.: '"/>
  <xsl:with-param name="value" select="./z41-sequence"/>
</xsl:call-template>
<xsl:call-template name="display-gen">
  <xsl:with-param name="label" select="'Cataloger:'"/>
  <xsl:with-param name="value" select="./z41-cataloger"/>
</xsl:call-template>
```

4.2 Removing Fields

In `ill-print-letter.xsl`, you want to remove the `z303-name` field.

The original is:

```xml
<xsl:call-template name="display-gen">
  <xsl:with-param name="label" select="'Cataloger:'"/>
  <xsl:with-param name="value" select="./z41-cataloger"/>
</xsl:call-template>
```

Delete the bold lines:

```xml
<xsl:call-template name="display-gen">
  <xsl:with-param name="label" select="'Cataloger:'"/>
  <xsl:with-param name="value" select="./z41-cataloger"/>
</xsl:call-template>
```

And the patron name is removed from the HTML:

<table>
<thead>
<tr>
<th>Patron ID:</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patron name:</td>
<td>Smith, John</td>
</tr>
<tr>
<td>Cataloger:</td>
<td>Judith</td>
</tr>
</tbody>
</table>

| Sequence No.: | 100092 |
4.3 Changing the Order of Fields

The printing order is the order in which the fields appear in the XSL file, therefore you only need to change the order in the file. We will use the same example as before:

In `ill-print-letter.xsl` we want to move `z303-name` after `z41-cataloger`.

The original is:

```xml
<xsl:call-template name="display-gen">
    <xsl:with-param name="label" select="'Patron name:'"/>
    <xsl:with-param name="value" select="./z303-name"/>
</xsl:call-template>
<xsl:call-template name="display-gen">
    <xsl:with-param name="label" select="'Cataloger:'"/>
    <xsl:with-param name="value" select="./z41-cataloger"/>
</xsl:call-template>
```

Change it to:

```xml
<xsl:call-template name="display-gen">
    <xsl:with-param name="label" select="'Cataloger:'"/>
    <xsl:with-param name="value" select="./z41-cataloger"/>
</xsl:call-template>
<xsl:call-template name="display-gen">
    <xsl:with-param name="label" select="'Patron name:'"/>
    <xsl:with-param name="value" select="./z303-name"/>
</xsl:call-template>
```
4.4 Moving Specific Fields in a Report

In certain cases you might want to rebuild a report so that one of the fields (for example, in the section-01 of the XSL file) prints out in a different location (by itself and without the rest of the data in section-01). In the printout example below, the Sorted By field appears in the Free section-01, above the Grid section-02:

The underlying XSL looks like this:

```xml
<xsl:stylesheet
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:include href="funcs.xsl"/>
  <xsl:template match="/">
    <xsl:call-template name="header"/>
    <xsl:for-each select="//section-01">
      <xsl:call-template name="section-01"/>
    </xsl:for-each>
    <!--section-02 (GRID)-->  
    <xsl:for-each select="//section-02">
      <xsl:if test="position() = 1">
        <xsl:call-template name="section-02">
          <xsl:with-param name="header" select="'header'"/>
        </xsl:call-template>
      </xsl:if>
      <xsl:call-template name="section-02"/>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```
<!-- START DATA -->
<xsl:template name="header">
  <xsl:call-template name="header-gen">
    <xsl:with-param name="title" select="'Arrived Issues Report'"/>
  </xsl:call-template>
</xsl:template>

<!--SECTION-01 (FREE)-->
<xsl:template name="section-01">
  <xsl:call-template name="table-open"/>
  <xsl:call-template name="display-gen">
    <xsl:with-param name="label" select="'Date From:'"/>
    <xsl:with-param name="value" select="./date-from"/>
  </xsl:call-template>
  <xsl:call-template name="display-gen">
    <xsl:with-param name="label" select="'Date To:'"/>
    <xsl:with-param name="value" select="./date-to"/>
  </xsl:call-template>
  <xsl:call-template name="display-gen">
    <xsl:with-param name="label" select="'Sorted By:'"/>
    <xsl:with-param name="value" select="./sorted-by"/>
  </xsl:call-template>
  <xsl:call-template name="table-close"/>
</xsl:template>

<!--SECTION-02 (GRID)-->
<xsl:template name="section-02">
  <xsl:param name="header"/>
  <xsl:if test="$header!=''">
    <xsl:call-template name="start-grid"/>
  </xsl:if>
  <xsl:call-template name="table-start-row"/>
  <xsl:call-template name="display-grid-gen">
    <xsl:with-param name="label" select="'Title'"/>
    <xsl:with-param name="value" select="./z13-title"/>
    <xsl:with-param name="header" select="$header"/>
  </xsl:call-template>
  <xsl:call-template name="display-grid-gen">
    <xsl:with-param name="label" select="'ISBN/ISSN'"/>
    <xsl:with-param name="value" select="./z13-isbn-issn"/>
    <xsl:with-param name="header" select="$header"/>
  </xsl:call-template>
  <xsl:call-template name="display-grid-gen">
    <xsl:with-param name="label" select="'Imprint'"/>
    <xsl:with-param name="value" select="./z13-imprint"/>
    <xsl:with-param name="header" select="$header"/>
  </xsl:call-template>
  <xsl:call-template name="display-grid-gen">
    <xsl:with-param name="label" select="'Issue Description'"/>
    <xsl:with-param name="value" select="./z30-description"/>
    <xsl:with-param name="header" select="$header"/>
  </xsl:call-template>
  <xsl:call-template name="display-grid-gen">
    <xsl:with-param name="label" select="'Arrival Date'"/>
    <xsl:with-param name="value" select="./z30-arrival-date"/>
    <xsl:with-param name="header" select="$header"/>
  </xsl:call-template>
  <xsl:call-template name="table-end-row"/>
</xsl:template>
</xsl:stylesheet>

End of XSL Template
Let us assume that you want to place the **Sorted By** field after the table:

![Print Preview](image)

In this case you must define a new call function (template). Let us say that you call it `section-01b`.

First, create a new function called `section-01b`.

```xml
<xsl:template name="section-01b">  
<xsl:call-template name="table-open"/>
<xsl:call-template name="display-gen">  
<xsl:with-param name="label" select="'Sorted By:'"/>  
<xsl:with-param name="value" select="./sorted-by"/>
</xsl:call-template>
<xsl:call-template name="table-close"/>
</xsl:template>
```

This function displays the **Sorted By** field, but does not control its location. To define the new location of the **Sorted By** field, paste the following after the `section-02` call-template:

```xml
<xsl:for-each select="//section-01">  
<xsl:call-template name="section-01b"/>
</xsl:for-each>
```

To prevent the **Sort By** field printing in its original location; delete the following lines from template `section-01`:

```xml
<xsl:call-template name="display-gen">  
<xsl:with-param name="label" select="'Sorted By:'"/>  
<xsl:with-param name="value" select="./sorted-by"/>
</xsl:call-template>
```
You have finished your editing. Take a look at your new code (the new lines appear in bold):

```xml
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
    <xsl:include href="funcs.xsl"/>
    <xsl:template match="/">
        <xsl:call-template name="header"/>
        <!--section-01 (FREE)-->  
        <xsl:for-each select="//section-01">
            <xsl:call-template name="section-01"/>
        </xsl:for-each>
        <!--section-02 (GRID)-->  
        <xsl:for-each select="//section-02">
            <xsl:if test="position() = 1">
                <xsl:call-template name="section-02">
                    <xsl:with-param name="header" select="'header'"/>
                </xsl:call-template>
            </xsl:if>
            <xsl:call-template name="section-02"/>
        </xsl:for-each>
        <!--section-01 (FREE)-->  
        <xsl:for-each select="//section-01">
            <xsl:call-template name="section-01b"/>
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

Calling Header function (template)

Calling section-01 functions

Start of new section-01b call function

End of the call functions

Start of Header data

Start of section-01 (Free) data

<table>
<thead>
<tr>
<th>New Section-01b Call</th>
<th>Start of Header Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calling section-01b function</td>
<td>Start of section-01 (Free) data</td>
</tr>
</tbody>
</table>

Report"/>
```
\[\text{XSL template}\]

\[\text{End of XSL template}\]

\[\text{Start of section-02 (Grid) data}\]

\[\text{End of section-02 (Grid) data}\]
The new printout looks like this:

![Arrived Issues Report](image-url)

The **Sorted By** field now appears in the **Free** section-01 below the **Grid** section-02.

### 4.5 Changing the Basic Layout

In general, this kind of change (from **Grid** to **Free / Split** or the other way around) is not recommended. The principles behind the ALEPH default setup are:

- **Grid** – Repeating section with less than 10 fields
- **Split** – Numerous fields (more than 15) and non-repeatable
- **Free** – Non-repeatable and not as many fields. Also, on rare occasions, repeatable but too many fields for an attractive **Grid**.

Although changing the basic layout is not recommended, it is possible. Here is how it is done:

#### 4.5.1 Free ↔ Split

This is relatively simple as the change is effected on data arranged in one or two columns. The only difference between the two is the table functions that are called.
A **Free** layout is enclosed by:

```xml
<xsl:call-template name="table-open"/>
<xsl:call-template name="table-close"/>
```

To make it a **Split** layout, replace these lines by:

```xml
<xsl:call-template name="table-split-open"/>
<xsl:call-template name="table-split-right"/> (at the split point)
<xsl:call-template name="table-split-close"/>
```

And vice versa to change a **Split** layout to a **Free** layout.

### 4.6 Further Customization

You may want to change fonts, spacing, and so on. To do so, you do not have to change anything in the templates - only in the common functions in the `funcs-*`.xsl common files. Most printouts are affected by a small group of functions.

#### 4.6.1 Recurring Blocks

You can change the fonts and position of `sublib-address`, `vendor-address`, `transfer-address`, `bib-info-hdr` by changing the table definitions in their respective `funcs*xsl` files, and the “td” definition of the rows.

For example, here is the XSL code for the `sublib-address` block from `funcs-address.xsl` which can be modified by editing the settings that appear:

```xml
<xsl:template name="sublib-address">
  <TABLE WIDTH="100%" STYLE="font-size: 9pt; font-family: Arial">
    <tr><td width="70%"></td><td><xsl:value-of select="//sub-library-address-1-oc c1"></td></tr>
    <tr><td width="70%"></td><td><xsl:value-of select="//sub-library-address-1-oc c2"></td></tr>
  </TABLE>
</xsl:template>
```

#### 4.6.2 Free / Split

- The table layout (width, alignment) can be changed globally by editing `table-open` (in `funcs-table.xsl`), or by calling it with parameters from specific templates.
- Edit `table-open` to change the font.
- The spacing between label and value is defined in the following specific functions called by `display-gen`: 
<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>display-not-empty</td>
<td>Prints when value is present</td>
</tr>
<tr>
<td>display-always</td>
<td>Prints whether or not value is present</td>
</tr>
<tr>
<td>display-not-empty-barcode</td>
<td>Prints using barcode font when value is present</td>
</tr>
<tr>
<td>display-always-barcode</td>
<td>Prints using barcode font whether or not value is present</td>
</tr>
<tr>
<td>display-not-empty-right</td>
<td>Prints and right-justifies when value is present</td>
</tr>
<tr>
<td>display-always-right</td>
<td>Prints and right-justifies whether or not value is present</td>
</tr>
</tbody>
</table>

### 4.6.3 Grid

- Same as for **Free** but using the **Grid** functions accessed via `funcs-grid.xsl`:
  - `grid-table` - `grid-open`
- Spacing – by editing the functions called by `display-grid-gen`:

<table>
<thead>
<tr>
<th>Grid-hdr</th>
<th>Prints data in the <strong>Grid</strong> header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid-data-right</td>
<td>Prints and right-justifies data in the <strong>Grid</strong></td>
</tr>
<tr>
<td>Grid-data-barcode</td>
<td>Prints data in the <strong>Grid</strong> in barcode font</td>
</tr>
<tr>
<td>Grid-data-bib-info</td>
<td>Prints data in the bib-info format</td>
</tr>
</tbody>
</table>

### 4.6.4 Plain XSLs

When a printout is sent by e-mail, the HTML file (the printout) is always sent as an attachment. The body of the e-mail message is a standard text located in the client's `\alephcom\tab\<lng>\MailHead.dat` file. For example:

```
Dear Sir/Madam,
The University of Ex Libris Library is sending you a mail message in the attached file.
Sincerely,
Library Administrator
```

However, you can include the printout data, in plain format, in the body of the e-mail message, in addition to the attached HTML file. In this case, the standard text from `MailHead.dat` is not included. This will occur if there is a `plain-*`.xsl` file for the specific printout.

For example, `plain-acq-s-order-slip.xsl` is the plain version of `acq-s-order-slip.xsl`.

The first rows of the XSL file are as follows:

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:include href="plain-funcs.xsl"/>
```
include href points to plain-funcs.xsl instead of funcs.xsl. This is the only difference from the regular HTML-creating XSL files. All functions are implemented in the plain-funcs as well.

Plain XSL Print Templates

plain-bor-list.xsl

plain-funcs.xsl

plain-funcs-grid.xsl     plain-funcs-free.xsl     plain-funcs-split.xsl

You can send any template in a plain format. The Grid or Split attributes are not affected when a template is sent in plain format via e-mail.
5 More Information about ALEPH Templates

5.1.1 Template Packaging
The packaging concept is used to ensure swift distribution of the templates to GUI users.

All templates reside in the form_<lng> directory of a library (for example, ./usm01/form_eng). You can use the path_convert table (in the ./alephe/tab directory) to re-direct the file location and maintain all templates in one directory per language.

UTIL I/6 packs all the XSL templates in the form_<lng> directory into one file. When a client is opened and connects to a library, they are downloaded to the client subject to a date check.

When a template has been changed, run UTIL I/6 to repack the XSL templates. Reconnect to the database in order to access the new templates package.

Note that UTIL I/6 must only be run manually and cannot be scheduled as an automatic job in the job_list table (in the ./alephe/tab directory).

5.1.2 Mail XSLs
There are some XSL stylesheets which have a separate version for printing and for attaching files to outgoing e-mail messages. You can configure different XSL templates for mail by attaching the mail- prefix to the template name, (for example, mail-acq-s-order-slip.xsl).

5.1.3 Hard-coded Data in XSLs
Hard-coded data in XSL templates can appear in labels, generic-lines, signatures, and so on.

Example:

```xml
<xsl:call-template name="display-gen">
  <xsl:with-param name="label" select="Sublibrary:'"/>
  <xsl:with-param name="value" select="./sub-library"/>
</xsl:call-template>
```

The first parameter sent to the display-gen function is the label. This contains the character string: "Sublibrary". This string can include any legal ASCII character. However as defined in the XML-standard by W3C, there are five characters, which must be replaced by another string for correct XML parsing:

<table>
<thead>
<tr>
<th>original data</th>
<th>replace with</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>&amp;</td>
<td>&amp;</td>
</tr>
</tbody>
</table>
The fifth character is ‘,’ which should be replaced by &amp; in a normal XML file. However, in the XSL standard, you cannot use an ampersand (&amp;) as a part of a hard-coded string. We recommend using ‘ instead of ‘.

5.2 Viewing Your Changes

Generally you edit XML and XSL files using a simple file editor such as vi on Unix or Notepad on Windows. In these editing environments, you cannot see how text and graphics are positioned for printing.

5.2.1 Viewing Changes using the GUI Print Configurations

To review your work, open the appropriate ALEPH GUI (for example, Acquisitions for acq-s-order-slip.xsl). Every ALEPH GUI module contains a printer icon at the extreme right of the status bar at the bottom of the screen.

When you right-click the printer icon, a list of available print options appears:

![Screen with print options]

These options are explained below:

**Normal Printing**

Selecting Normal Printing facilitates direct printing, although the actual procedure required to send a print command to the printer varies from template to template. For example, hold-request-letter.xsl can be printed by pressing the Letter button on the Hold Requests window.

Print settings, which control the sequence of events when a print command is initiated, are defined for each template in the print.ini file. For example, sending a print command might cause the printout to be sent directly to the printer, or it might cause the Print setup window to appear, depending on the corresponding settings in print.ini.
Preview
When Preview is selected, clicking the Print command lets you preview the printout:

You can then click Print to print the file.

View Raw XML
When this option is selected, clicking the Print command lets you view the file in raw XML format in an editor (such as NotePad) window:

Browse XML
When this option is selected, clicking the Print command lets you view the file in raw XML format in a Browser window:
This mode can be very useful for debugging.

5.2.2 Print History

All files that are printed from an ALEPH GUI are saved in a "history" section under the operator's profile. The Print History function is used to view and/or re-print all files that were printed from the GUI.

Each module has its own "history". For example, if an online Acquisitions claim was printed, it will appear in the Acquisitions module's print history. However, if an Acquisitions report was produced in an Acquisitions service, and therefore was printed through the Task Manager, it will appear in the Task Manager module's print history.

You can access a module's Print History by selecting the File menu and clicking Print History. The following window opens:

The left pane is a navigation tree that lists all the dates and rounded hours when files were printed. Each day is a separate node, and each rounded hour is a separate sub-node. Highlight the relevant node to list the names of all the files that were produced in that time period. The XML data is stored on the client, and can, therefore, be re-
printed, even after a print template has been changed. The data is stored for a time set in the client’s alephcom.ini configuration, listed under the section [print]:

[print]
savehistorynumberofdays=1

This setting determines how many days the print history should be stored. The cleanup of old history files takes place when the GUI is started.

The upper right pane lists each of these files and their size. Highlight a file on this list to see the beginning of its contents in XML format in the lower right pane.

**Print Configuration**
These options are similar to the options available from the **printer** icon at the extreme right of the status bar at the bottom of the screen and which are explained in the preceding section. Select one of the following and then click the **Go** button:

- Normal Printing
- Preview
- View Raw XML
- Browse XML

**Print Setup**
Select **Y** to have the **Print Setup** window appear before printing. Otherwise select **N**.

**Setup Type**
This field is only enabled when **Normal Printing** is selected in the **Print Configuration** field. Select **P** to print the file, **M** to send it by e-mail, and **B** for both.

**E-mail Address**
Enter the e-mail address here if you are e-mailing the file. If the print file includes an e-mail address, it will be displayed here.

**Refresh**
Click the **Refresh** button to clear the bottom pane.

Using this interface, you can test a re-designed print template (.xsl file). After changing the .xsl file, run UTIL I6 to repackage the .xsl files and then open the client. The new .xsl file will be copied to your client, and when you reprint the file in the history list, it will be printed using the new template. (Note that UTIL I6 must only be run manually and cannot be scheduled as an automatic job in the **job_list** table (in the ./alephe/tab directory)).
6 Reference

6.1 XSLT Elements in ALEPH

ALEPH templates use a very small set of XSLT elements. These are briefly described in this section. If you want to know more about them or about XSL/XSLT in general, it is strongly suggested you consult a book in the subject. There are many such books. The one on which this work is based is:


6.1.1 concat()

Concatenates strings

concat(“aa”, “bb”) returns “aabb”

6.1.2 position()

Current index in context of for-each.

6.1.3 substring-before ()

Substring before an internal substring:

substring-before(“aaa##bbb”, “##”) returns “aaa”

6.1.4 substring-after ()

Substring after an internal substring:

substring-after(“aaa##bbb”, “##”) returns “bbb”

6.1.5 xsl-attribute

See xsl :element below

6.1.6 xsl: call-template

The way to invoke functions:

<xsl:call-template name="section-01"/>

6.1.7 xsl:element

There are two ways to generate HTML elements and their attributes:

1. Writing the XSL code explicitly
2. Using the xsl:element element

For instance to start a table, you need to write something like:

<table width="100%" style="font-family=arial; font-size=9pt">
And, of course, its closing tag: </table>

You could write these lines explicitly in every XSL template, but then if you wanted to change the table definition globally, you would have to do so in every template. One of the guiding principles in ALEPH templates is that all actual HTML displays should be contained in the common functions. For example:

```
<xsl:template name="table-open-example">
    <table width="100%" style="font-family=arial; font-size=9pt">
</xsl:template>
```

The problem is that the XSLT processor accepts `<TABLE` as an XSL element and rejects it as it is not closed. One solution is to use the following:

```
&lt;table width="100%" style="font-family=arial; font-size=9pt"&gt;
```

As in HTML the special characters ‘<’ ‘>’ can be replaced by `&lt;`, `&gt;`. In addition, you must enclose it as follows:

```
<xsl:text disable-output-escaping="yes">
    &lt;table width="100%" style="font-family=arial; font-size=9pt"&gt;
</xsl:text>
```

This instructs XSLT to write it as the required HTML element.

Another option is to use the `xsl:element / xsl:attribute`.

```
<xsl:element name="string"/>
```

produces the output: <string>.

To add attributes, enter the following:

```
<xsl:element name="string">
    <xsl:attribute name="att1">val1</xsl:attribute>
</xsl:element>
```

The output:

```
<string att1="val1">
```

To get back to the example:

```
<xsl:template name="table-open-example">
    <xsl:element name="table">
        <xsl:attribute name="width">100%</xsl:attribute>
        <xsl:attribute name="style">
            font-family=arial; font-size=9pt
        </xsl:attribute>
    </xsl:element>
</xsl:template>
```

6.1.8  `xsl:for-each`

This element is used for reaching all entries with the same name. In ALEPH, these are the section-0n sections such as section-01, section-02, and so on:

```
<xsl:for-each select="/section-01">
    <xsl:call-template name="section-01" />
</xsl:for-each>
```
Note: Each time section-01 is called, it is in the context of the current XML section-01.

6.1.9 xsl:if
This element executes the contained elements only if the test expression returns true (or a filled node set):

```xml
<xsl:for-each select="//section-02">
  <xsl:if test="position() = 1">
    <xsl:call-template name="section-02-first"/>
  </xsl:if>
  <xsl:call-template name="section-02"/>
</xsl:for-each>
```

6.1.10 xsl:include
Used for including other XSL files within an XSL file.

```xml
<xsl:include href="funcs.xsl"/>
```

6.1.11 xsl:param
Defines parameters in functions (invoked using with-param):

```xml
<xsl:template name="display-gen">
  <xsl:with-param name="label"/>
  <xsl:with-param name="value"/>
  <xsl:with-param name="display"/>
  <xsl:with-param name="type"/>
</xsl:template>
```

6.1.12 xsl:template
Used to define functions.

```xml
<xsl:template name="signature">
  (function body)
</xsl:template>
```

6.1.13 xsl:template match
Used in ALEPH only as the beginning of the XSL code. The value of the match attribute is a specific pattern. In the following example, the pattern match="/" is an instruction to match the root node:

```xml
<xsl:template match="/"
```

6.1.14 xsl-text
See xsl :element (section 6.1.7)

6.1.15 xsl:variable
This element is used to declare a local or global variable in a stylesheet, and to give it a value.

```xml
<xsl:variable name="bib" select="/bib-info"/>
```
6.1.16 xsl: with-param
Defines a parameter on a xsl:template or xsl:stylesheet. Also specifies a default value. Each parameter require its own with-param:

```xml
<xsl:call-template name="header-gen">
  <xsl:with-param name="title" select="'Acquisitions Cancel Slip'"/>
</xsl:call-template>
```

6.2 ALEPH XSL Functions

6.2.1 Header functions

**header-gen**

*Function:* Takes the general form-data from the opening part of the XML. It displays the date, the XML form name and form number, a horizontal line and it displays the "title" in centralized alignment.

*Example:*

```xml
<!-- START DATA -->
<xsl:template name="header">
  <xsl:call-template name="header-gen">
    <xsl:with-param name="title" select="'Acquisition Order Slip'"/>
  </xsl:call-template>
</xsl:template>
```

This produces the following display:

```
15/07/2002
acq-s-order-slip-01

Acquisitions Order Slip
```

6.2.2 Address functions

**patron-address**

*Function:* This function displays the fields of the patron-address to the left side of the page.

*Parameters:* None

*Example:* `<xsl:call-template name="patron-address"/>`
sublib-address

**Function:** This function displays the fields of the `sublib-address` to the right side of the page.

**Parameters:** None

**Example:**

```xml
<xsl:call-template name="sublib-address"/>
```

<table>
<thead>
<tr>
<th>Acquisitions Unit 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manag. Building</td>
</tr>
<tr>
<td>Ex Libris University</td>
</tr>
<tr>
<td>777 Biblio Byway</td>
</tr>
<tr>
<td>Chicago, IL 60614</td>
</tr>
<tr>
<td><a href="mailto:thechoice@exlibris-usa.com">thechoice@exlibris-usa.com</a></td>
</tr>
<tr>
<td>Tel# 773.404.5327</td>
</tr>
</tbody>
</table>

transfer-address

**Function:** This function displays the fields of the `transfer-address` to the left side of the page.

**Parameters:** None

**Example:**

```xml
<xsl:call-template name="transfer-address"/>
```

Only in use on one form: `transfer-slip.xsl`

vendor-address

**Function:** This function displays the fields of the `vendor-address` to the left side of the screen.

**Parameters:** None

**Example:**

```xml
<xsl:call-template name="vendor-address"/>
```

<table>
<thead>
<tr>
<th>AMERICAN METEOROLOGICAL SOCIETY</th>
</tr>
</thead>
<tbody>
<tr>
<td>45 BEACON STREET</td>
</tr>
<tr>
<td>BOSTON, MA 02108</td>
</tr>
</tbody>
</table>

6.2.3 Free Functions

**bib-info-free**

**Function:** Displays bibliographic information in a Free layout section, but not in the header. `bib-info-free` is formatted according to the bibliographic library's `edit_paragraph` table; the format is set in the ADM library's `bib-format` table. `bib-info-free` is printed if the value is not empty.

**Parameters:** `<xsl:param name="value"/>`

**Example:**

```xml
<xsl:call-template name="bib-info-free">
  <xsl:with-param name="value" select="node()"/>
</xsl:call-template>
```
bib-info-hdr
Function: Displays the bibliographic information in the header in non-Grid layouts.
Parameters: None
Example:
```xml
<xsl:call-template name="bib-info-hdr"/>
```

We would like to place an order for the following items:
47 p. ; 23 cm.

blank-line
Function: Draws one blank line in non-Grid layout.
Parameters: None.
Example:
```xml
<xsl:call-template name="blank-line"/>
```

blank-line-first
Function: Draws one blank line in non-Grid layout only if the position in the XML file is the first.
Parameters: None.
Example:
```xml
<xsl:call-template name="blank-line-first"/>
```

blank-line-in-free
Function: Draws the number of blank lines defined in "lines" in non-Grid layout. If the parameter is empty, the default is one blank line.
Parameters: `<xsl:param name="lines"/>`
Example:
```xml
<xsl:call-template name="blank-line-in-free">
  <xsl:with-param name="lines" select="'3'"/>
</xsl:call-template>
```

checkbox
Function: Displays the text with the check box. The check box is always unmarked.
Parameters: `<xsl:param name="text"/>`
Example:
```xml
<xsl:call-template name="checkbox">
  <xsl:with-param name="text" select="'Please supply me with the item when available.'"/>
</xsl:call-template>
```
direct-to-patron

**Function:** If the condition-field is equal to the condition-value then it displays 'Send Directly to:' and then the patron-address.

**Parameters:**
- `<xsl:param name="condition-field"/>
- `<xsl:param name="condition-value"/>

**Example:**
```
<xsl:template name="directly-to-patron">
  <xsl:param name="condition-field"/>
  <xsl:param name="condition-value"/>
  <xsl:if test="$condition-field = $condition-value">
    <xsl:call-template name="table-open"/>
    <xsl:call-template name="display-gen">
      <xsl:with-param name="label" select="'Send Directly to:'"/>
      <xsl:with-param name="value" select="''"/>
      <xsl:with-param name="display" select="'always'"/>
    </xsl:call-template>
    <xsl:call-template name="patron-address"/>
    <xsl:call-template name="table-close"/>
  </xsl:if>
</xsl:template>
```

display-gen

**Function:** Displays the label and the value in a Free style.

- If value is empty, the label does not display.
- If display is 'always', then the label displays even if the value is empty.
- You can specify the width parameter. If this is not specified, the default is 20%.
- If type is 'index', then the position in XML is displayed.
- If type is 'right' then the value is right-justified.
- If type is 'barcode', then the character set becomes CarolinaBar-B39-2.5-22x158x720.
- If type is 'check', a check box is displayed. If the value begins with "y" or "Y" then the check box is marked. You must NOT translate this value to other languages. If you do, the system will not identify the correct check box value (just leave it as Y/N).

**Parameters:**
- `<xsl:param name="label"/>
- `<xsl:param name="value"/>
- `<xsl:param name="display"/>
- `<xsl:param name="type"/>
- `<xsl:param name="style"/>
- `<xsl:param name="width"/>

**Example**
```
<xsl:call-template name="display-gen">
  <xsl:with-param name="label" select="'Note:'"/>
  <xsl:with-param name="value" select="/z38-note-1"/>
</xsl:call-template>
```
free-title
**Function:** Displays a bold, underlined title.

**Parameters:**
```xml
<xsl:param name="free-title"/>
```

**Example:**
```xml
<xsl:call-template name="free-title">
  <xsl:with-param name="free-title" select="'Material Requested:'"/>
</xsl:call-template>
```

generic-line

**Function:** Displays the line in the specified style, with the specified width.

- line can be any string.
- If ## is present in the line, it places a line break in the display.
- If salutation_string is found, it is replaced by “Dear Sir/Madam”.
- If width is not specified, it will display in full-width (100%).
- Styles can be bold, underlined, italic (or all of them).

**Parameters:**
```xml
<xsl:param name="line"/>
<xsl:param name="style"/>
<xsl:param name="width"/>
```

**Example:**
```xml
<xsl:call-template name="generic-line">
  <xsl:with-param name="line" select='concat("salutation_string##We are sorry to inform you that the following photocopy, which you requested on ", //z38-open-date ,", cannot be filled at this time. Please inform us by return mail if you would like us to supply you with the item when available, or whether you would prefer to cancel the request.")'/>
</xsl:call-template>
```

The output is:

```
Dear Sir/Madam,

We are sorry to inform you that the following photocopy, which you requested on 01/01/2002, cannot be filled at this time. Please inform us by return mail if you would like us to supply you with the item when available, or whether you would prefer to cancel the request.
```

horizontal-line

**Function:** Draws a horizontal line in non-Grid layout.

**Parameters:** None.

**Example:**
```xml
<xsl:call-template name="horizontal-line ">
```

**requested-by-proxy**
*Function*: If the `condition-field` is not empty, then it displays "Requested By:' and then the `z37-requester-id` value.

*Parameters*:

```xml
<xsl:param name="condition-field"/>
```

*Example*:

```xml
<xsl:call-template name="requested-by-proxy">
  <xsl:with-param name="condition-field" select="./z37-requester-id" />
</xsl:call-template>
```

---

**table-open**
*Function*: Starts a **Free** layout by opening a borderless table.

*Parameters*:
- `width` (optional), default = 50%
- `align`, possible values:
  - `left` (default)
  - `center`
  - `right`

*Example*:

```xml
<xsl:call-template name="table-open"/>
```

*Note*: Every `table-open` element opens a table which must be closed by a `table-close` element.

---

**table-open-full**
*Function*: Starts a **Free** layout by opening a borderless table with `width=100%`.

*Parameters*: None

*Example*:

```xml
<xsl:call-template name="table-open-full"/>
```

---

**table-close**
*Function*: Ends any open table.

*Parameters*: None

*Example*:

```xml
<xsl:call-template name="table-close"/>
```
6.2.4 Grid functions

display-Grid-gen

**Function:** Displays the value in a Grid table.

- The header of the column is label.
- Styles can be bold; underline; italic.
- If the header has the value header, then it only puts grid-label in the very first row of the Grid only.
- The data is displayed in grid-box, so each time ## appears in the data, it displays a line break within the box.
- If type is set to right the data are right-justified.
- If type is check, a check box is displayed. The check box is marked if the value begins with 'y/Y' or if check-value equals value. Otherwise the box is unmarked.
- If type is barcode, then the value appears as a real barcode.
- If type is bib-info, then the value is displayed in bib-info style.
- If type is restart, then the Grid closes, and restarts again.

**Parameters:**

```xml
<xsl:param name="value"/>
<xsl:param name="type"/>
<xsl:param name="label"/>
<xsl:param name="style"/>
<xsl:param name="check-value"/>
```

**Example:**

```xml
<xsl:call-template name="display-grid-gen">
  <xsl:with-param name="label" select="'Budget Number:'"/>
  <xsl:with-param name="value" select="./z601-budget-number"/>
</xsl:call-template>
```

<table>
<thead>
<tr>
<th>Trans. Type</th>
<th>Budget Number</th>
<th>Open Date</th>
<th>Currency</th>
<th>Original Sum</th>
<th>Local Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encumbrance</td>
<td>MAR-2002</td>
<td>15/07/2002</td>
<td>US Dollar</td>
<td>75.00</td>
<td>15.00</td>
</tr>
</tbody>
</table>

---

display-grid-currency

**Function:** Similar to display-grid-gen. For special Grids, when the number of columns is known only at runtime, such as the currency-report template. The regular Grid has a fixed number of columns, and when the data for a column are missing, a blank is displayed. The currency Grid displays only when there are some data to display.

**Parameters:**

```xml
<xsl:param name="label"/>
<xsl:param name="value"/>
<xsl:param name="type"/>
<xsl:param name="display"/>
```

**Example:**

```xml
<xsl:call-template name="display-grid-currency">
  <xsl:with-param name="label" select="'.code-1'"/>
  <xsl:with-param name="value" select="'.data-1'"/>
  <xsl:with-param name="display" select="'.code-1'"/>
</xsl:call-template>
```

<table>
<thead>
<tr>
<th>Trans. Type</th>
<th>Budget Number</th>
<th>Open Date</th>
<th>Currency</th>
<th>Original Sum</th>
<th>Local Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encumbrance</td>
<td>MAR-2002</td>
<td>15/07/2002</td>
<td>US Dollar</td>
<td>75.00</td>
<td>15.00</td>
</tr>
</tbody>
</table>
grid-open
Function: Starts a Grid layout by opening a table with border. Width default = 100%
Parameters:
<xsl:param name="width"/>
<xsl:param name="align"/>
Example:
<xsl:call-template name="grid-open"/>
Note: grid-open / grid-close are always paired.

grid-close
Function: Ends a Grid layout by ending the table.
Parameters: None
Example:
<xsl:call-template name="grid-close"/>
Note: grid-open / grid-close are always paired.

grid-title
Function: Displays a title for the Grid.
Parameters:
<xsl:param name="grid-title"/>
Example:
<xsl:call-template name="grid-title">
  <xsl:with-param name="grid-title" select="'Total sum due to vendor:'"/>
</xsl:call-template>

start-grid
Function: Calls grid-title, and then grid-open. grid-title is the title of the Grid. The width is the width of the Grid table. If ‘nobold’ is defined, then the title does not appear in bold.
Parameters:
<xsl:param name="grid-title"/>
<xsl:param name="width"/>
<xsl:param name="nobold"/>
Example:
<xsl:template name="section-02">
  <xsl:param name="header"/>
  <xsl:if test="$header!=''">
    <xsl:call-template name="start-grid">
      <xsl:with-param name="grid-title" select="'Loans'"/>
    </xsl:call-template>
  </xsl:if>
</xsl:template>

Loans
<table>
<thead>
<tr>
<th>Bib Info</th>
<th>Due Date</th>
<th>Description</th>
<th>Sublibrary</th>
<th>Item status</th>
<th>Call No. 1</th>
<th>Barcode</th>
<th>Proxy ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Journal of sounds</td>
<td>17/07/2002</td>
<td>2000 1 1</td>
<td>Law Library</td>
<td>One Day Loan</td>
<td><em>7592-30</em></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
table-start-row
Function: Before display-grid-gen is called, a new row must be opened in grid-table. Each time you enter a grid-section, you must call this function.
Parameters: None
Example
<xsl:call-template name="table-start-row"/>
Note: table-start-row and table-end-row are always paired.

---


table-end-row
Function: After the call to the last ‘display-grid-gen’, you must close the row in the grid-table. Each time a grid-section is finished, this function must be called.
Parameters: None
Example
<xsl:call-template name="table-end-row"/>
Note: table-start-row and table-end-row are always paired.

---

6.2.5 Split functions
display-gen-split
Function: Displays data in the Split layout if the value is not empty. The label is bold, and then the value is displayed. type, display and style parameters are equivalent in the Free-layout to the display-gen function.
Parameters:
<xsl:param name="label"/>
<xsl:param name="value"/>
<xsl:param name="display"/>
<xsl:param name="type"/>
<xsl:param name="style"/>
Example:
<xsl:call-template name="display-gen-split">
<xsl:with-param name="label" select="'Vendor Ref. No:'"/>
<xsl:with-param name="value" select="./z68-vendor-reference-no"/>
</xsl:call-template>

---

table-split-open
Function: Starts a Split layout.
Parameters: None
Example:
<xsl:call-template name="table-split-open "/>
Note: table-split-open / table-split-right / table-split-close come together.

---

table-split-right
Function: Starts the right hand side of a Split layout.
Parameters: None
Example
<xsl:call-template name="table-split-right "/>
**table-split-close**  
**Function:** Ends a Split layout.  
**Parameters:** None  
**Example**  
```xml  
<xsl:call-template name="table-split-close"/>  
```
6.3 Special Templates

There are a few templates (less than 10) that do not conform to the description above. Their header part has the same structure, but their data part is handled differently.

6.3.1 Currency-report

The basic difference between this template and the “regular” ones is that the number of tags depends on the query – one tag (=Grid column) per currency and an additional tag for the date. In other templates, a Grid has a fixed number of columns, and if there are no data for a column in a specific row, an empty cell is displayed. Here we want the number of columns to depend on the query. To solve this, a special function exists - `display-grid-currency` –, which is like `display-grid-gen`, with an additional parameter – display. It is invoked as follows:

```xml
<xsl:call-template name="display-grid-currency">
  <xsl:with-param name="value" select="./currency-ratio-1"/>
  <xsl:with-param name="display" select="./currency-1"/>
  <xsl:with-param name="type" select="'right'"/>
</xsl:call-template>
```

The column only displays if the tag `<currency-1>` exists.

The XSL file is built with what we assume is the maximum number of currencies that will be queried (15). Each query displays just the requested currencies. If you think you need more than 15 queries, just add more lines.

6.3.2 Order-info

Same problem and solution as in currency-report.

6.3.3 Catalog-records-columnar

Same problem and solution as in currency-report. The maximum number of tags required is six.

6.3.4 Loan receipt

In this XSL file, a Grid is displayed. A note is attached to each row (./z36-note-1) which is displayed beneath the grid-row. This is why the Grid must be restarted for each record. Each record actually displays the labels, the data and the note-1 (in the third row). The call to this section is:

```xml
<!--section-02 (GRID)-->
<xsl:for-each select="/section-02">
  <xsl:call-template name="section-02">
    <xsl:with-param name="header" select="'header'"/>
  </xsl:call-template>
</xsl:for-each>
```