PDS VERSION 1.3

Patron Directory Services
CONFIDENTIAL INFORMATION

The information herein is the property of Ex Libris Ltd. or its affiliates and any misuse or abuse will result in economic loss. DO NOT COPY UNLESS YOU HAVE BEEN GIVEN SPECIFIC WRITTEN AUTHORIZATION FROM EX LIBRIS LTD.

This document is provided for limited and restricted purposes in accordance with a binding contract with Ex Libris Ltd. or an affiliate. The information herein includes trade secrets and is confidential.

DISCLAIMER

The information in this document will be subject to periodic change and updating. Please confirm that you have the most current documentation. There are no warranties of any kind, express or implied, provided in this documentation, other than those expressly agreed upon in the applicable Ex Libris contract.

Any references in this document to non-Ex Libris Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this Ex Libris product and Ex Libris has no liability for materials on those Web sites.

Copyright Ex Libris Limited, 2007. All rights reserved.
Documentation produced February 2007
Document version 1.7
Web address: http://www.exlibrisgroup.com
# Table of Contents

1 OVERVIEW - WHAT IS PDS? 6

2 PDS CONFIGURATION 7
   2.1 PDS Directory tree 7
   2.2 Main PDS Configuration Table: tab_service.<institute> 7
   2.3 PDS Login Page Handling 8
      2.3.1 The load-login request 9
      2.3.2 Local login 10
      2.3.3 Remote Login 10
      2.3.4 Configuring LOAD_LOGIN 11
      2.3.5 Configuring REMOTE_LOGIN 13
      2.3.6 Example of a full Remote login Configuration 14
      2.3.7 Optional security enhancement for remote_login 14
   2.4 User Authentication 15
      2.4.1 Configuring the AUTHENTICATE Service 15
      2.4.2 Local Authentication 15
      2.4.3 Remote Authentication 16
   2.5 User Attributes Retrieval (BOR_INFO) 17
      2.5.1 BOR_INFO service within the tab_service table 17
      2.5.2 Configuration of BOR_INFO 18
   2.6 Logout 19
      2.6.1 Redirection after Logout 19
   2.7 Institute Display 19
      2.7.1 Configuring Institution Display Name 19
      2.7.2 Character Conversion 20
      2.7.3 Sort Order of Institutes 20

3 ADVANCED TOPICS 22
   3.1 Session Cookies 22
   3.2 Single Sign-On Configuration for Ex Libris Products 22
      3.2.1 Overview 22
      3.2.2 The sso_conf file setup 22
      3.2.3 Configuring Single Sign-On 23
      3.2.4 Configuring Single Sign-Off 23
      3.2.5 Calling System Setup 24
      3.2.6 Hints to configuring SSO 25
      3.2.7 Default SSO institute. 25
   3.3 Remote Single Sign-On (SSO) 25
      3.3.1 What is the difference between Remote Login and Remote SSO? 25
      3.3.2 Setting up Remote SSO 26
      3.3.3 Configuring LOAD-SSO 26
      3.3.4 Configuring REMOTE_SSO 28

© Ex Libris Ltd., 2007  
Patron Directory Services version 1.3  
Last Update: February 13, 2007
3.4 CAS Authentication
3.4.1 Create the CAS user and tables
3.4.2 Create the restricted CAS directory
3.4.3 Add the CAS configuration to httpd.conf

3.5 iChain SSO, Authentication, BOR_INFO and LOGOUT

3.6 Multiple tab_service.institute files

4 IMPLEMENTING SHIBBOLETH 1.32 IN EX LIBRIS PRODUCTS

4.1 Prerequisites
4.1.1 Hints for installing the Shibboleth service provider
4.1.2 Testing that the above prerequisites are met

4.2 Additional Apache configuration for Ex Libris products
4.2.1 Adding a symbolic link in /apache/htdocs directory
4.2.2 Changing Apache configuration files
4.2.3 Modify apache startup scripts (optional)
4.2.4 Modify metaib_start

4.3 Implementing Shibboleth with Ex Libris Products

4.4 Shibboleth LOGIN workflow

4.5 The Shibboleth LOGOUT workflow

4.6 The Shibboleth SSO workflow

4.7 Configuring Shibboleth as a WAYF

5 PDS CUSTOMIZATION

5.1.1 PDS HTML Directory: html_form
5.1.2 The login page
5.1.3 Customized institution login screen
5.1.4 PDS Direct access
5.1.5 PDS Direct access: The remote-and-local-list page instead of the login page
5.1.6 PDS Direct access: main-menu-logged-out
5.1.7 PDS Direct access: main-menu-logged-on
5.1.8 Error messages from the PDS
5.1.9 PDS debug mode.

6 PDS PROGRAMS

6.1 Authentication against X-servers

6.2 LDAP Services
6.2.1 LDAP Configuration file
6.2.2 LDAP Configuration file examples
6.2.3 Attributes mapping and defaults

6.3 Remote CGI Hook
6.3.1 Remote CGI Hook: Setup
6.3.2 Remote CGI Hook interface
6.3.3 Remote CGI program
Content and target audience

This document describes the interaction between a calling application, MetaLib, DigiTool, or ALEPH and the Patron Directory Service (PDS), a component that facilitates user authentication and login to the calling application. The document provides an in-depth technical understanding of the PDS and how it can be customized to suit differing authentication needs.

Given its technical content, the primary audience for this document is likely to be systems administrators or staff with technical knowledge of authentication arrangements at the institution or consortium.

Based on typical needs and to help facilitate rapid setup, a step-by-step guide to getting PDS working with a calling application is provided in Appendix A – How to set up PDS on page 63.

Note

The screen capture examples in the document use the Ex Libris PDS screens. MetaLib, DigiTool, and ALEPH all have their own customized screens with their own logos.

1 Overview - What is PDS?

The Patron Directory Service (PDS) is a “back-end” Web component that facilitates user authentication and login to a calling application. The PDS is part of the standard calling application package, but it is a distinct and separate component. All Primo, MetaLib Version 4, and DigiTool Version 3.5 installations work with PDS 1.3. From version 18.01, ALEPH can be configured to use PDS 1.3 as its authentication agent. The PDS does not have a user database of its own. Rather, it can be configured to work against the institution’s local authentication server and user database, such as an LDAP directory service. The PDS can receive user information (user ID and additional user details) via a standard XML output from the institution’s local user database and relay this information to the calling application for the creation/update of individual user records.

In addition to facilitating user authentication and login to the calling application, the PDS provides shared user authentication and Single Sign-On (SSO) capabilities among the Ex Libris suite of products (DigiTool, MetaLib, and ALEPH). The PDS can check if a user trying to access a given product has already been authenticated for access to another Ex Libris product and – if the user has – the PDS logs the user in without requesting a username and password. Similarly, a user logging out from one Ex Libris application can be automatically logged out from all Ex Libris applications working with the same PDS, as well as being logged out of the PDS itself (Single Sign-Off). An alternative model is to configure the PDS to prompt the user logging out of one product with a list of the other products the user is logged into, allowing the user to select the other applications from which the user wishes to log out.

The PDS can be configured against the iChain, Shibboleth and CAS SSO and authentication systems. The LOAD_SSO / REMOTE_SSO mechanism is in place to enable the institution to add its own SSO hook. This mechanism is described in section 3.3 Remote Single Sign-On (SSO) on page 25.
2 PDS Configuration

2.1 PDS Directory tree
To access the root directory of the PDS, enter the following command:

```bash
>>pdsroot
```

Under this root directory the following PDS directories can be found:

- **conf_table** – Contains the PDS main configuration tab_service.institute files. Most of the PDS’s configurations are performed in this directory. These include LDAP configurations, customized attribute mapping files etc.
- **program** – Contains the PDS program files & modules.
- **html_form** – Containing all of the HTML files used by the PDS and customized versions of the different PDS screens.
- **service_proc** – Contains internal PDS Perl routines used by the PDS for communication with different information sources.
- **pds_files** – Stores all active session cookies.
- **pds_proc** – Containing PDS internal utilities.

2.2 Main PDS Configuration Table: tab_service.<institute>
The tab_service.institute table in the ./pds/conf_table directory is the central configuration table in the PDS. There is one table for each institution. The table defines the services required from the PDS for that institution.

Each table is composed of sections and every section always contains the following four lines to which others can be added:

```plaintext
[Service Name]
program     =
params      =
[END]
```

An explanation of each line is given below.

**[SERVICE NAME]**
The service requested from the PDS. The following services are available:
o AUTHENTICATE – performs user authentication
o BOR_INFO – gets user attributes (name, affiliation, etc.)
o REDIRECT_LOGOUT – redirects to a different page after logout
o LOAD_LOGIN – redirects user to remote login page
o REMOTE_LOGIN – handles remote login response
o LOAD_SSO – redirects user to remote SSO System
o REMOTE_SSO – handles the remote SSO callback
o BOR_VERIFICATION - gets user password (used for EZPROXY-SSO)

program
The program used. Specifies the appropriate PDS program to use for the task at hand.
The programs reside in the ./pds/service_proc directory.

params
Parameters for the program or the name of a configuration file that is used to store parameters.

[END]
Added to each section to confirm the end of the service section.

As will become clear later, the setup for a given institution requires the configuration of several PDS services (AUTHENTICATE and then BOR_INFO to retrieve user information, for example). This means that there are multiple sections in the tab_service table for a single institution.

Note
An example of a fairly typical setup is given in Appendix A – How to set up PDS on page 63 of this document.

2.3 PDS Login Page Handling
When a user elects to log in, the load-login request is sent from the calling system to the PDS. The PDS then displays the appropriate login page. An installation of PDS can present a variety of login pages to the user, according to the information sent from the calling system in conjunction with the PDS configuration.

Each institution can have one of two types of login pages:

Local login page – The page is presented by the PDS based on the calling system parameter in the URL and is the default login option. Other local login pages available to the user are the PDS default screen or an institution login screen. This page has a combo
box containing a list of all the local institutions. If there is an institution parameter on the URL, it is selected from the list in the combo box.

![Ex Libris local login page](image)

**Figure 1: Ex Libris local login page**

**Remote login page** - The PDS redirects the user to a login page that is not part of the PDS and can be located on any server.

Most times a local login page is displayed. The remote option is only used when the user’s institution has configured the use of a remote login page, so that there is a LOAD_LOGIN section in the institutes tab_service. Otherwise, a local login page is used.

When the PDS has several institutions some may use local login pages and some remote login pages. A user is presented with a list of institutions from which to select. Upon selecting an institution from the list, the appropriate login page is displayed.

![Institute selection screen](image)

**Figure 2: An institute selection screen**

### 2.3.1 The load-login request

The request for a login page is sent using a URL such as the following:
This URL is sent from the calling system to PDS. It is constructed from several parts.

- The server address and port
  
  http://www.exlibris.com:8331/

- The call to PDS to load a login page
  
  pds?func=load-login

- An optional parameter specifying the institute in the following form
  
  &institute=HUJI

- An optional parameter specifying the calling-system
  
  &calling_system=aleph
  &url=http://www.exlibris.com...

If an institute parameter is sent in the URL then there are two possible login pages, local or remote. If the institute is local, a local login page is displayed. If it is remote, the remote login page is displayed.

If an institute parameter is not sent on the URL, then PDS checks all the tab_service files. If all of them are local, the local login page is displayed. If any are remote, the institute list page is displayed.

### 2.3.2 Local login

The “AUTHENTICATE” functionality is called from the local login process. It looks for an AUTHENTICATE section in the tab_service.institute file. For more information on configuring the AUTHENTICATE service see 2.4 User Authentication on page 15.

The login process adds the username and password sent from the login page to the parameters in the params line of the AUTHENTICATE section. It activates the authentication program and if authentication succeeds, a PDS session and pds handle are created. PDS proceeds to fetch user attributes, after which the user is redirected to the calling system using the callback_url parameter in the login page URL.

### 2.3.3 Remote Login

PDS supports a Remote Login option, in which the PDS redirects the login request to a remote login page, typically presented by a central authentication server that the institution wishes to use instead of the PDS.

In this flow, the PDS must identify where to direct the user and must pass parameters to the remote system. Following authentication, the remote system must pass parameters back to the PDS to enable the PDS to complete its tasks and log the user into the calling application. These two steps are referred to as LOAD-LOGIN and REMOTE-LOGIN and are described in more detail below. Sites not wishing to deploy the Remote Login option can skip this section and move on to 3.1 Session Cookies on page 22.

When the user is authenticated, PDS creates a “pds_handle” for the user and sets a cookie with a session key in the user’s browser. The cookie is named PDS_HANDLE.
In addition to the PDS cookie, the calling application creates its own session that is valid as long as the user’s calling application session is active. The calling application checks the session before passing a login request to the PDS. This means that if the calling application’s session is valid, the user is logged in directly – not via the PDS.

The PDS cookie is deleted from the user’s browser when the user logs off or when the user closes the browser. When a user logs out of the calling application, it sends PDS a logout request and the User’s PDS handle is deleted.

PDS has a cleanup process that removes PDS handles from the disk if they are older than 24 hours. See 3.2 Single Sign-On Configuration for Ex Libris Products on page 22.

It is necessary to configure two separate services in ./pds/conf_table/tab_service.institute for Remote Login. The LOAD-LOGIN service handles the communication from the PDS when redirecting the login request to the remote login program. The REMOTE-LOGIN service handles the communication from the remote login program when it sends the required callback parameters to the PDS.

The following is a schematic diagram of the REMOTELOGIN/LOADLOGIN process.

![Diagram of REMOTELOGIN/LOADLOGIN process]

**Figure 3: Remote Login schema**

### 2.3.4 Configuring LOADLOGIN

The “LOADLOGIN” functionality is called from the calling application when a user requests a login page and PDS is configured to redirect to a remote login page.

**The redirect takes place in the following circumstances:**

An institution parameter has been passed to the PDS when LOADLOGIN is requested and the relevant tab_service.institute is configured with LOADLOGIN

**OR**

There is only one tab_service.institute table in the directory ./pds/conf_table and that table is configured with a LOADLOGIN service.
Following is an example of the configuration needed to be made in the `tab_service.<institute>` file:

**Example configuration of LOAD_LOGIN:**

```plaintext
[LOAD_LOGIN]
program = remote_load_login.pl
params  = load-login-cityuniv
[END]
```

**Note**

In the `.pds/conf_table/tab_service.institute` table, the LOAD_LOGIN service needs to be defined **instead** of the AUTHENTICATE service.

The remote_load_login.pl program used is a generic program that places standard parameters into an HTML page with the redirect request. The params line points to the HTML file which contains the actual redirection to the URL of the remote login screen. The Web page (HTML file) is the value of the params parameter in the above example (named load-login-cityuniv) and should be placed either in the `.pds/html_form` global or in the appropriate `.pds/html-form/calling_system-xxx` directory.

**Example of HTML file configured to support LOAD_LOGIN:**

```html
<!-- login-redirect -->
<html>
<head>
<title>Identification</title>
<include>meta-tags</include>
<script language=JavaScript>
 function redirect()
 {  
  var url = "https://authserver/cgi/login.asp?" +
    "fixed_par1=F1&" +
    "fixed_par2=F2&" +
    "pds_handle=$0100&"+
    "calling_system =$0200&"+
    "institute=$0300&"+
    "url=$0400";
  top.location = url;
  }
</script>
</head>
<body onload="javascript:redirect()">
</body>
</html>
```

The standard parameters that should be included in the Web page with the redirect are as follows:

- Institution code
- PDS handle (which remains unpopulated)
- Calling system
• Back-link URL (originating URL the user came from. This must be the last parameter.)

2.3.5 Configuring REMOTE_LOGIN

This service supports the asynchronous callback of the remote login system to PDS. It is mandatory to have a callback function in PDS to handle the return of the remote login. The section in the tab_service.institute file is called REMOTE_LOGIN and the standard program is called remote_login_gen_1.pl.

The program retrieves the parameters passed back from the server, analyzes them, updates the PDS session, and then redirects the user back to the calling application.

For example:

```
.program = remote_login_gen_1.pl
.params  = check,1800
```

The standard parameters that must be passed back with the remote_logins are:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>user ID</td>
<td>id=0123456</td>
</tr>
<tr>
<td>calling system</td>
<td>calling_system=metalib or aleph or digitool</td>
</tr>
<tr>
<td>Institution code</td>
<td>Institute=HUJI</td>
</tr>
<tr>
<td>PDS handle</td>
<td>pds_handle (This parameter must be sent back as received – it is currently empty as it is reserved for future development)</td>
</tr>
<tr>
<td>Back-link URL to calling application</td>
<td>url=http://.....</td>
</tr>
</tbody>
</table>

The Back-link URL must remain the last parameter as everything that follows it is considered part of the back-link.

**Example of REMOTE_LOGIN input:**

```
http://exlserver:port/pds?func=remote-login&
calling_system=(aleph or digitool or metalib)&
institute=TESTINST&
pds_handle=&
ID=04523&
url=http://exlserver:8999/V/7VNSL9REEIPD-00001?func=quick-1
```

In the above example the remote CGI script returns a parameter “ID=04523” in addition to the calling system, institute, and URL parameters which it received from the PDS. This means the user was authenticated and the remote_login_gen_1.pl program creates a Z311 file on the disk and return a pds_handle to the calling application.
2.3.6 Example of a full Remote login Configuration

The following example shows the addition of a new institution named “BUSINESS”.

This institution uses the remote login, in which PDS redirects the login request to a remote authentication server where it authenticates the user. The BOR_INFO services are defined via a remote CGI script.

[LOAD_LOGIN]
program = remote_load_login.pl
params  = load-login-business-pin
[END]

[REMOTE_LOGIN]
program = remote_login_business.pl
params  = check,1800
[END]

[BOR_INFO]
program = remote_cgi_hook.pl
params  = GET,host:port,cgi-bin/rp_business.pl
[END]

2.3.7 Optional security enhancement for remote_login

If the params = check then the following security enhancement is activated.

A token is added to protect against the following risks:

- playback: user saves the return URL from the login page and uses it to bypass login at a later date.
- False identity: user alters the ID parameter in the return URL to impersonate another user.

The token is 40 characters in length and contains the following:

- 8 characters - hex version of the time when the token was generated.
- 32 characters - MD5 hash of the time, the user ID, and a salt value.

The token is created and sent from the user defined authentication cgi.

&PDS_HANDLE=43f1b446d434e1b963c02196b9610bb910a491b1

To activate the check in the remote_login_gen_1.pl, add to the parameters the value check and value of the time difference to allow in seconds:

[REMOTE_LOGIN]
program = remote_login_gen_1.pl
params  = check,1800
[END]

The code added to remote_login_gen_1.pl does the following:

- Turns the hex time back into a decimal and compares it with the current time. If the difference between the two is greater than the second value in the parameters, it determines that the URL is being "played back".
- Constructs an MD5 of the time (from the token), the user ID and the salt and compares it with the MD5 passed in the token. If any single character of the token has been altered (for example, the user attempted to alter the timestamp portion),
or if the user manually changed the PDS borrower ID value (that is, the current security hole), then does not allow the login.

The same changes were made in the tab_service for the LOAD_SSO/REMOTE_SSO

[REMOTE_SSO]

program = remote_sso_gen_1.pl
params = check,600

[END]

This is the code to encode the token

```perl
my $salt       = 'sadjkhaksjdhjkhkasfhkfjhjkh sdfkhasdfkh sdf';
my $time       = time( );
my $hex_time   = sprintf("%08x",$time);
my $check      = $hex_time.md5_hex( $time, $id, $salt );
```

And in the HTML:

```
"PDS_HANDLE=$check&"
```

2.4 User Authentication

2.4.1 Configuring the AUTHENTICATE Service

The AUTHENTICATE service is responsible for performing the user authentication. PDS can be configured to authenticate users using the calling application’s own user database (called ‘local’ authentication) or via remote authentication servers or both. The authentication process begins when the login request is submitted. Several authentication programs can be configured and the PDS attempts to authenticate the user with the first listed authentication program. If the authentication fails, it tries the next authentication program. If authentication fails in all of the defined services defined the user is redirected back to the login page.

The available authentication programs and examples of their configurations can be found in section 6 PDS Programs on page 47.

2.4.2 Local Authentication

PDS can authenticate the user against the calling applications’ own user database. It uses a Perl interface to activate the calling application’s X-Server and validate the user name and password entered on the login screen.

The parameters configured specify

- Connection information for contacting the calling application X-server such as the server address and port to access when calling the X-Server.
- The N parameter instructs the Perl interface not to use a secure connection.
- Two parameters, service_user and service_password, are the username and password used by PDS to obtain permission to use X-Server services. The default user and password are pds, pds. It is recommended to change the default
password, in which case the username and password must be included in the params line.

**Important Note**

The PDS user and password must be changed in the calling application to authorize PDS to use the relevant X-Server services. Refer to product documentation to determine how to change them.

Some examples of how to configure PDS for local authentication services are listed below:

**AUTHENTICATE via the calling application MetaLib**

```plaintext
[AUTHENTICATE]
program = metalib_x_server.pl
params  = www.exlibris.com,8331,BOR-AUTH,N,service_user,service_password
[END]
```

**AUTHENTICATE via the calling application DigiTool**

```plaintext
[AUTHENTICATE]
program = digitool.pl
params   = www.exlibris.com,8331,op=BOR_AUTH,DAT01,service_user,service_password,N
[END]
```

**AUTHENTICATE via the calling application ALEPH**

```plaintext
[AUTHENTICATE]
program = aleph.pl
params   = www.exlibris.com,8331,BOR-AUTH,USM50,N,service_user,service_password
[END]
```

In the examples above, a different program is used to communicate with each calling system.

**Note**

This document refers to the x_server.pl program as a generic term for one of the metalib_x_server.pl, aleph.pl or digitool.pl programs.

### 2.4.3 Remote Authentication

PDS can authenticate the user against a remote authentication source. For example, by using an LDAP server or a custom remote CGI hook, the user can write the user’s own custom authentication.

You can configure PDS so that it works with both remote and local authentication services. This means that users can be authenticated via their remote authentication server or via the calling application’s user database. PDS attempts to authenticate the user with the first listed authentication program. If authentication fails, it tries the second authentication program. The remote authentication service (for example, remote CGI...
hook, LDAP or ALEPH) should be listed first in tab_service.institute, and then the local authentication service using the calling application’s X-Server.

**Example 1: AUTHENTICATE via CGI script and via the calling application**

```
[AUTHENTICATE]
program  =  remote_cgi_hook.pl
params   =  GET,www.exlibris.com:8331,aleph-cgi/remote_cgi_hook
program  =  x_server.pl
params   =  hostname,8331,BOR-AUTH,N,pds,pds
(END)
```

In the above example, when the authentication routine is activated, it uses the remote_cgi_hook.pl program to check the user name and password entered against the external authentication server. If the external authentication server does not validate the user, PDS tries to validate the user using the calling application’s X-Server.

**Example 2: AUTHENTICATE via LDAP and the calling application**

```
[AUTHENTICATE]
program = ldap.pl
params  = ldap_prn.conf
program = x_server.pl
params  = hostname,8331,BOR-AUTH,N,pds,pds
(END)
```

When this authentication routine is activated, it uses the ldap.pl program and the ldap_prn.conf file to check the user name and password entered against the LDAP Server. If the LDAP server does not validate the user, PDS tries to validate the user using the calling application’s X-Server.

2.5 User Attributes Retrieval (BOR_INFO)

2.5.1 BOR_INFO service within the tab_service table

After a successful authentication, the “BOR_INFO” functionality is called by the calling application. The PDS_HANDLE is no longer GUEST but a long stream of numbers. The BOR_INFO functionality is usually called after a successful login with the AUTHENTICATE functionality. It can also take place in one of the following cases:

- A remote login using the LOAD_LOGIN/REMOTE_LOGIN functionalities.
- A successful LOAD_SSO/REMOTE_SSO function call establishing the user’s logged-in status.
- After an SSO function call where the user has an authenticated session in another Ex Libris application.

The BOR_INFO service handles retrieving user attributes, processing them and delivering them to the calling system. The PDS can be configured to obtain user attributes either from the calling application’s user database or from remote user directories. Attributes retrieved are handled in an XML format, mapped and normalized. The normalization of XML tags makes it possible for PDS to receive and send attributes with tag names containing the user table’s prefix (Z303 for ALEPH, Z311 for DigiTool and Z312 for MetaLib) as well as sending them without these prefixes. Throughout this document, both...
tag formats are used. The attribute mapping process and configuration are described in section 7 on page 56 of this guide.

PDS handles formatting of the attributes but the calling application is responsible for interpreting these attributes and granting appropriate authorizations in the calling application.

2.5.2 Configuration of BOR_INFO
The BOR_INFO configuration is made in the [BOR_INFO] section of the tab_service.institute file. This section is a mandatory part of the PDS configuration file and must exist. Different types of user directories are configured in different ways which may require sending different parameters. Refer to Appendix A – How to set up PDS on page 63 for information on configuring different BOR_INFO services. Following are several examples to illustrate the BOR_INFO workflow.

Example 1: BOR_INFO via the calling application’s X-server
The following is an example of a configuration when information is gathered using x-server:

```
[BOR_INFO]
program = x_server.pl
params = hostname,8331,op=BOR-INFO,N,pds,pds
[END]
```

In this example, PDS is configured to obtain user information with a calling application X-Server.

Following the program name, the program parameters include in this case the server hostname, port, the requested service, secure connection flag, and the username and password to the X-Server services. In other cases the params line can include other parameters or the name of a configuration file containing all the different parameters used.

Example 2: BOR_INFO via CGI script and calling application

```
[BOR_INFO]
program = remote_cgi_hook.pl
params = GET,www.cityuniv.com:8331,cgi-bin/rpa_cla.pl
program = x_server.pl
params = www.cityuniv.com,8331,BOR-INFO,N,pds,pds
[END]
```

In the above example when the BOR_INFO section in the tab_services.institute is activated, it uses the remote_cgi_hook.pl program to retrieve user information from the external authentication Server. If PDS cannot retrieve user information from the remote CGI script, it tries to obtain user information using one of the Ex Libris X Servers.

Following the remote CGI program name, the method used, server hostname, port, and program name of the remote CGI program should be entered as input parameters.
Following the x_server.pl program name, the server hostname, port, and requested service, secure connection flag, and X-Server username and password should be entered as input parameters for the program.

2.6 Logout

The “LOGOUT” function is called from the calling application when the user decides to sign-out. The LOGOUT removes the PDS_HANDLE from the disk and expires the PDS cookie.

2.6.1 Redirection after Logout

You can redirect a user to a specific calling application HTML page or some other URL after the user has logged out. This is an optional service.

Example 1: REDIRECT_LOGOUT

[REDIRECT_LOGOUT]
params = Y/?func=find
[END]

In the above example, when a user logs out from the calling application, the user is redirected to the “Find” module as specified in the params link. When redirecting to an html page within the application, there is no need to enter the hostname and port.

Example 2: REDIRECT_LOGOUT

[REDIRECT_LOGOUT]
params = http://www.cnn.com
[END]

In this example, when an authenticated user logs out from the calling application, the user is redirected to http://www.cnn.com as specified in the params field.

2.7 Institute Display

2.7.1 Configuring Institution Display Name

The drop down list in the PDS login screen takes its data from the desc field in the [INSTITUTE_DISPLAY] section of the tab_service.institute file.

For example:

[INSTITUTE_DISPLAY]
code = CITYUNIV
lang = ENG
desc = City University
[END]

In this example, the PDS is configured to display the institution name in the drop-down menu of the login page as “City University”. The drop-down menu is sorted alphabetically according to the values entered in the desc field of all tab_service.inst files.
Note

You can display the institution name in one language only.

2.7.2 Character Conversion

Institute names can be displayed in many languages using extended characters. When using special characters, add a **character conversion** definition above the string to be converted. PDS uses the appropriate character conversion routines to convert the local character set to UTF8.

Example of character conversion definition and the corresponding screen output:

```
[INSTITUTE_DISPLAY]
code      = METALIB
lang      = ENG
CHARACTER_CONVERSION=8859_8_TO_UTF
desc      = נאסיבַת טמאטליב
[END]
```

![Example of Institute name in Hebrew](image)

*Figure 4: An example of an Institute name in Hebrew*

An alternative to using the character conversion is to code the strings in the tab_service.institute file in &string; or &#Unicode; HTML notation which is then properly displayed without any conversion, for example desc = &Aring;bo Akademi displays as Åbo Akademi.

2.7.3 Sort Order of Institutes

The default order of the institute names in the combo box is alephabetic. To change the order there is an optional sort_key = rank parameter. To display City University first in the combo box, add a sort_key = 1 parameter. For example:

```
[INSTITUTE_DISPLAY]
code      = CITYUNIV
lang      = ENG
desc      = City University
sort_key  = 1
```
3  Advanced Topics

3.1  Session Cookies

When the user is authenticated, PDS creates a “pds_handle” for the user and sets a cookie with a session key in the user’s browser. The cookie is named PDS_HANDLE.

In addition to the PDS cookie, the calling application creates its own session that is valid as long as the user’s calling application session is active. The calling application checks the session before passing a login request to the PDS. This means that if the calling application’s session is valid, the user is logged in directly – not via the PDS.

The PDS cookie is deleted from the user’s browser when the user logs off or when the user closes the browser. When a user logs out of the calling application, it sends PDS a logout request and the User’s PDS handle is deleted.

PDS has a cleanup process that removes the PDS handles from the disk if they are older than 24 hours.

3.2  Single Sign-On Configuration for Ex Libris Products

3.2.1  Overview

PDS can be configured to provide Single Sign-On services between the following Ex Libris products: MetaLib, DigiTool and ALEPH.

The following sections describe the configuration steps necessary to configure the calling systems to share the PDS and to provide Single Sign-On (SSO) across Ex Libris products.

The underlying concept is that the calling systems (MetaLib / DigiTool / ALEPH) all share the same PDS by using the same PDS hostname and PDS port number definitions. In addition, the PDS has a configuration file to define SSO logon and logout behavior.

When setting up the SSO between different IP addresses, the hosts-allow table in the Apache conf directory must be updated (see the section on X-Server Security on page 61).

When there is more then one tab_service.institute file in the conf_table directory the sso_conf needs to contain a section to define a default SSO institute.

3.2.2  The sso_conf file setup

The sso_conf file is a file that exists by default in the ./pds/conf_table/ directory. This file defines the PDS’s behavior upon login and logout in an SSO environment. Following is an example of SSO configurations in the ./pds/conf_table/sso_conf file:

```plaintext
[LOGON]
TYPE1   = digitool,metalib,aleph
[END]

[LOGOUT]
TYPE2   = digitool,metalib,aleph
```
3.2.3 Configuring Single Sign-On

The [LOGON] section contains the configuration of Single Sign-On across Ex Libris products. There is one configuration option:

**TYPE 1: Enable automatic Sign-On**—enables a shared session between the various TYPE1 applications listed.

**Single Sign-On example**

If a user logs into MetaLib, and then links or logs into DigiTool, DigiTool queries PDS to establish if the user has a valid PDS cookie. If the user has a valid PDS cookie, the user is automatically logged into DigiTool. If the ./pds/conf_table/sso_conf is configured like the example given below.

**TYPE 1: Enable Single Sign-On**

This option enables SSO for applications defined for TYPE 1. For example:

```
[LOGON]
TYPE1 = metalib,digitool
[END]
```

For sites with MetaLib only, it is recommended to use TYPE 1 with only MetaLib defined.

3.2.4 Configuring Single Sign-Off

The [LOGON] section contains the configuration of Single Sign-Off across Ex Libris products. There are two configuration options:

**TYPE 1: Enable automatic Sign-Off**—when logging off from one application, the user is automatically logged off from all other applications defined as TYPE 1. In the above example, when the user logs off from MetaLib, he/she is logged off from DigiTool and ALEPH as well.

**TYPE 2: Enable selective Sign-Off**—when a user logs off from MetaLib, for example, a menu of other applications defined as TYPE 2 appears. This allows the user to select the applications the user wants to log off from.

**Single Sign-Off examples**

When the user logs off from MetaLib, the user is logged off from DigiTool as well.

**TYPE 1: Enable automatic Sign-Off**—Meaning: when logging off from MetaLib, the user is automatically logged off from all other applications defined as TYPE 1, for example:

```
[LOGOUT]
TYPE1 = digitool,metalib
[END]
```

When the user logs off from MetaLib, the user is prompted with a menu of other applications defined as TYPE 2 and selects the applications s/he wishes to log off.

**TYPE 2: Enable selective Sign-Off**—to be used only with multiple Ex Libris products.
3.2.5 Calling System Setup

**MetaLib**

**Steps**

1. From the $metalib_conf directory, open the metalib_start file.
2. Change the two variables below to point to the shared PDS instance. For example:
   ```
   setenv PDS_HOST 10.1.235.47
   setenv PDS_HOST _IN 10.1.235.47
   setenv PDS_PORT 8339
   setenv PDS_PORT _IN 8339
   ```
3. Source metalib_start and restart the MetaLib servers (start_w).

**DigiTool**

**Steps**

1. From the $dtle_root directory, open dtl_start.
2. Change the two variables below to point to the shared PDS instance. For example:
   ```
   setenv PDS_HOST 10.1.235.60
   setenv PDS_PORT 8339
   ```
3. In the www_server.conf file, locate the server_pds parameter. If it is hard coded, change the it to use variables from the dtle_start file. The result should be the following:
   ```
   # PDS definitions:
   setenv server_pds "http://${PDS_HOST}:${PDS_PORT}/pds"
   ```
4. Source dtl_start and run UTIL W/3/1 to start the servers.

**ALEPH**

**Steps**

1. From the $alephe_tab directory, open the tab100 table.
2. Set the following two PDS-related variables:
   ```
   PDS-AWARE=Y
   PDS-KEY-TYPE=00
   ```
3. From the $alephe_root directory, open aleph_start.
4. Change the two variables below to point to the shared PDS instance. For example:
   ```
   setenv PDS_HOST 10.1.235.60
   ```
5. In the www_server.conf file, locate the server_pds variable. If it is hard coded, change it to use variables from the aleph_start file. The result should be the following:

```
setenv server_pds "http://${PDS_HOST}:${PDS_PORT}/pds"
```

6. Source aleph_start and run UTIL W/3/1 to start the servers.

### Hints to configuring SSO

For the setup to work, the calling applications need to build a shared cookie. That means the values of PDS_HOST must be consistently either an IP address (for example, 10.1.235.47), or a hostname (for example, RAM47) but not one of each.

#### Default SSO institute.

The LOAD_SSO section in the tab_service.institute file is only to be activated if the on the URL of the /pds?func=sso there is an institute parameter or there is only one tab_service.institute file. The calling system when it calls the func=sso sometimes cannot identify the correct SSO institute. A default institute section in the conf_sso file has been added to supply a default when there isn’t an institute=value parameter on the URL.

For example the section below defines for calling_system metalib the default institute is LAW.

```
[DEFAULT_INSTITUTE]
LAW = metalib
[END]
```

### Remote Single Sign-On (SSO)

This service enables the integration of a calling application, with external Single Sign-On servers. The service addresses a scenario where a user logs into the university’s portal or another central login system and then moves to a calling application. Using this service, a user who authenticated at the SSO system and then accesses the calling application, is seamlessly redirected to the remote SSO system to determine whether the user has already logged in. The remote SSO system then validates the user and redirects the user back to PDS with the correct callback parameters described below, enabling login into the calling application without the user being challenged by a login page.

#### What is the difference between Remote Login and Remote SSO?

The Remote SSO process is similar to the Remote login scenario. However, in the Remote Login scenario, the user must initiate a login request from the calling application in order to activate the Remote Login service. Thus, a user accessing DigiTool, MetaLib, or ALEPH with a Remote Login configuration would access the calling application as a guest user and is not be automatically logged in. However, a user accessing MetaLib, DigiTool, or ALEPH with a Remote SSO configuration would be automatically logged into the calling application provided the user logged into the university system prior to accessing the calling application. Thus, the Remote Login process is activated only when
a user chooses to login from a calling application, whereas the Remote SSO process is always activated at the start of a new calling application Session. The reason for differentiating between both “hooks” is to allow greater flexibility as not all external systems can support Single Sign-On capabilities.

<table>
<thead>
<tr>
<th>Calling Application</th>
<th>Browser</th>
<th>PDS</th>
<th>remote sso</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://login.lib.com:3333/css">http://login.lib.com:3333/css</a></td>
<td>tab_service institute</td>
<td>PDS creates a handle</td>
<td>Remote Hook</td>
</tr>
<tr>
<td>User authenticated by calling application</td>
<td>URL + PDS handle</td>
<td>PDS creates a handle</td>
<td>Remote Hook</td>
</tr>
</tbody>
</table>

**Figure 5: A diagram schema of the REMOTE SSO flow**

**Note**

Both Remote Login and Remote SSO services work ONLY if one of these two conditions is met:

1. There is only one tab_service.institute file or the sso_conf contains a default SSO institute.
2. The calling application sends the user’s institute to PDS with the func=load-login or func=sso request. This enables PDS to identify the user’s affiliation and activate the appropriate services as defined for the user’s institution in tab_service.

**3.3.2 Setting up Remote SSO**

To configure Remote SSO, two services need to be defined – one to handle the redirection of the user’s browser to the external system (LOAD_SSO); the other to handle the callback from the external system (REMOTE_SSO). LOAD_SSO calls an HTML file pointing to the remote script to fetch the user’s ID. REMOTE_SSO handles the return call and add the received user’s ID to the PDS session.

**3.3.3 Configuring LOAD-SSO**

LOAD_SSO calls an HTML file pointing to the remote script to fetch the user’s ID.
The LOAD-SSO program receives the HTML file name in the params line and places standard parameters into the file to create the redirect request. In the above example, the HTML file is called \textit{sso-remote-1}. The file specified must be placed in the ./pds/html_form/global directory.

Following is a sample \textit{sso-remote-1} file:

```html
<!-- remote sso -redirect -->
<html>
<head>
<title>Identification</title>
<include>meta-tags
<script language=Javascript>
    function redirect()
    {
        var url =   "http://hostname:port/cgi/mysso?" +
                    "pds_handle=$0100&"+
                    "calling_system=$0200&"+
                    "institute=$0300&"+
                    "url=$0400";
        top.location = url;
    }
</script>
</head>
<body onload="javascript:redirect()">
</html>
```

The standard parameters included in the HTML redirection are:

- **Redirect address** – the address of the remote SSO system. This needs to be changed to the location of the remote SSO script. For example:
  ```javascript
  var url = "https://www.hostname.com/pin/default.asp?"
  ```

- **PDS handle** entered in the following syntax:
  ```javascript
  "pds_handle =$0100&"
  ```

- **Calling system** entered in the following syntax:
  ```javascript
  "calling_system =$0200&"
  ```

- **Institution code** entered in the following syntax:
  ```javascript
  "institute =$0300&"
  ```

- **Back-link URL** to calling application (the Web page that the user came from). This must be the last parameter sent. This is entered in the following format:
  ```javascript
  "url =$0400"
  ```

Following is an example of a request sent to the remote SSO system:
This file and link format can be a to suite the link format expected by the remote SSO system. The PDS parameters and parameter names must be included as in the above example.

**Note**

The parameters sent by PDS to the remote system should be passed back as is and **must not** be changed during the processing of the remote script.

### 3.3.4 Configuring REMOTE_SSO

After the LOAD_SSO request is sent to the remote SSO system. The remote system sends a response back to PDS as a URL. PDS takes the ID provided in the URL and update the PDS session accordingly. The PDS then fetches user attributes as defined in the BOR_INFO section and redirect the browser back to the calling application using the returned backlink URL.

An example of REMOTE_SSO configuration:

```
[REMOTE_SSO]
program     = remote_sso_gen_1.pl
params      =
[END]
```

**Note:**

The Remote SSO system must communicate back to the PDS using the remote-sso function with **only the standard five callback parameters** as defined bellow.

**Structure of REMOTE SSO callback URL (via intermediate program)**

The response of the remote SSO system needs to be a URL sent back to PDS using the following URL format:

```
http://<server>:<port>/pds?func=remote-sso
```

The URL passed back to the REMOTE-SSO needs to include five standard parameters. These standard parameters are:

- **Institute code** - The parameter to return is `institute` and it contains the institute code sent in the LOAD-SSO URL.
- **PDS handle** - The parameter to return is `pds_handle` and it is empty.
- **Calling system** - The parameter to return is `calling_system` and it contains the name of the calling system sent in the LOAD-SSO URL.
- **User ID** – The parameter to return is `id` and it contains the user ID if that user has logged in or be empty if the user has not previously logged in.
• Backlink URL to calling application – The parameter to return is `url` with the value as it was sent by PDS. This must be the last parameter sent.

**An example of a REMOTE-SSO URL:**

```
calling_system=exlibris&
institute=TESTINST&
pds_handle=&
id=04523&
url= http://exlserver:8999/Z/7VNSL9REEEIPD-00001?func=quick-8
```

**Note:**

Sites can configure both Remote Login and Remote SSO services to cover two scenarios: The first scenario is when users first access the calling application and then initiate a login request; the second scenario is when users first log in to a central SSO system and then move to the calling application.

### 3.4 CAS Authentication

PDS provides a “hook” for CAS (Central Authentication System). The PDS host's apache needs to be configured to add a restricted resource `/cas` and to add a Perl module which communicates with the CAS server.

**Note:**

The PDS CAS support is still in beta stage. Those who wish to implement CAS SSO with PDS please contact your local support office.

The interface is a mod_perl handler which keeps sessions in the database so there are additional configuration steps needed to activate this option.

#### 3.4.1 Create the CAS user and tables

dlib vir00
$aleph_proc/create_ora_user_b cas

sqlplus cas/cas @cas.sql

```sql
CREATE TABLE cas_sessions(
  id       varchar(32) not null primary key,
  last_accessed  number not null,
  "uid"      varchar(32) not null,
  pgtiou    varchar(64) not null
);

CREATE TABLE cas_pgtiou_to_pgt (      
  pgtiou    varchar(64) not null primary key,
  pgt       varchar(64) not null,
  created   number not null
);
```

© Ex Libris Ltd., 2007  
Patron Directory Services version 1.3  
Last Update: February 13, 2007
3.4.2 Create the restricted CAS directory

```
mkdir htdocs/cas
ln -s $metalib_dev/pds/program/pds_main pds_main
```

3.4.3 Add the CAS configuration to httpd.conf

In the apache httpd.conf file:

```xml
<Location /cas>
    SetHandler perl-script
    PerlResponseHandler ModPerl::Registry
    Options +ExecCGI
    PerlOptions +ParseHeaders

    PerlSetVar isMetalib "true"
    PerlSetVar PDS_USER_NAME "pds_id"
    PerlSetVar PDS_PAGE_KEY "user"
    PerlSetVar PDS_INSTITUTE "castest"

    PerlSetVar CASHost "xxx.xxx.edu"
    PerlSetVar CASPort "443"
    PerlSetVar CASErrorURL "http://xx.xx.xx:8331/error_oracle.html"
    # For Oracle this is the SID
    PerlSetVar CASDatabaseName "dev"
    PerlSetVar CASDatabaseHost "ic-dev.xxx.xx"
    PerlSetVar CASDatabasePort "nnnn"
    # DBI is case sensitive in UNIX environments so be careful.
    PerlSetVar CASDatabaseDriver "Oracle"
    PerlSetVar CASDatabaseUser "xxxx"
    PerlSetVar CASDatabasePass "yyyy"
    # Remember to have a different cookie for each instance
    PerlSetVar CASSessionCookieName "APACHECASORACLE"
    PerlSetVar CASSessionTimeout "1800"
    PerlSetVar CASLogLevel "0"
    PerlSetVar CASRemoveTicket "false"
    # to change the default login page from /cas/login to /sso/login
    # PerlSetVar CASLoginURL /sso/login
    PerlSetVar CASLoginURL /sso/login

    AuthType Apache::AuthCASEXL
    AuthName "CAS"
    PerlAuthenHandler Apache::AuthCASEXL->authenticate
    require valid-user
</Location>
```

Instead of the calling application calling PDS with the URL:

```
http://hostname:port/pds?func=load-login
```

It uses the URL:

```
http://hostname:port/cas/pds_main?func=load-login
```

The /cas in the URL activates the Apache server which redirects the user to a CAS login screen. After the authentication completes the session continues to pds_main?func=load-login.

PDS checks the environment for a HTTP_PDS_ID variable and if it exists, the authentication succeeds and a PDS_HANDLE is returned to the calling application.

In the example below, tab_service.institute uses the cas.pl program to check if the CAS authentication was successful.
Note:
The interface with CAS only provides for AUTHENTICATION and only the ID is stored. BOR_INFO attributes have to be set up using an alternate method, for example, LDAP.

(Optional) REMOTE_LOGOUT to logout of the CAS server:

```
[REMOTE_LOGOUT]
params       =  https://casserver:port/cas/logout
[END]
```

3.5 iChain SSO, Authentication, BOR_INFO and LOGOUT

There are two options for implementing iChain with PDS.

- **A restricted resource /pds_ichain is set up in the http.conf** and /pds_ichain is also defined as a ScriptAlias to pds/program/pds_main. The load-login process calls /pds_ichain instead of /pds. The Apache redirects the user to the iChain login screen and then continues with the load-login PDS functionality.

  iChain passes authentication information in the HTTP headers. User credentials are sent using the HTTP_AUTHORIZATION parameter containing user information for “basic authentication”.

  When a user first accesses a calling application having this information, PDS reads the header, fetches user attributes in the background and logs the user into the calling application.

  If a user has not previously logged in and wishes to log in, the user is redirected to the iChain login screen where the user’s credentials are entered, iChain authenticates the user and creates the authorization parameter.

  This method of implementation allows for SSO, login and logout. There is no need for external scripts to perform these tasks.

  Apache does not normally send the Authorization header onto clients. To get around this, rewrite rules must be added to the httpd.conf file

  ```
  RewriteEngine On
  RewriteCond %{HTTP:Authorization} ^(.*)
  RewriteRule ^/(.*)$ - [env=HTTP_AUTHORIZATION:%1]
  ```

  The PDS has special programs (ichain.pl and ichain_sso.pl) in place to process information sent from iChain by checking for an HTTP_AUTHORIZATION in the header.

  The user name and password are extracted from the headers and stored so even though iChain does not have attributes if the information exists somewhere it can be accessed with a standard BOR_INFO section in the tab_service.institute.
If a user logs out of iChain the calling application sees the Authorization headers are missing and logs the user out.

If a user logs out of the calling application the optional REDIRECT_LOGOUT section in tab_service can be configured to log the user out of iChain. If the user does not logout of iChain the next user who attempts to login is logged in as the previous user.

```
[LOAD_LOGIN]
program     =  ichain.pl
params      =
[END]

[LOAD_SSO]
program     =  ichain_sso.pl
params      =
[END]

[REDIRECT_LOGOUT]
params       =  http://server:port/ichain/logout
[END]
```

- **Form fill** - recognizing that a login page is sent to the user, iChain intercepts it, enters the user’s credentials and sends the response back to PDS. This is done in the background and is seamless to the user. Upon receiving information, PDS performs the login process and logs the user into the calling application.

SSO functionality is available using the standard hooks LOAD_SSO / REMOTE_SSO and user defined scripts.

### 3.6 Multiple tab_service.institute files

Usually there is a single tab_service.institute file corresponding to every calling system institute. The calling application institute is by default the code from the tab_service.inst file. In a consortia where different AUTHENTICATE methods are needed for the same institute or in a SSO scenario between ALEPH and MetaLib where one tab_service needs to return both the MetaLib Institute and the ALEPH Admin. Library, there is an optional parameter to override the default institute.

The tab_service.cityuniv returns to the calling application institute=LAW in the bor-info XML for calling system MetaLib and CTY50 for calling system ALEPH.

For example:

```
[INSTITUTE_DISPLAY]
code      =  CITYUNIV
lang      =  ENG
desc      =  City University
aleph     =  CTY50
metalib   =  LAW
[END]
```
4 Implementing Shibboleth 1.32 in Ex Libris products

The objective of this chapter is to outline the prerequisites and guidelines for Shibboleth implementation in Ex Libris products via PDS.

Disclaimer

Third party products such as Shibboleth are not under the control of Ex Libris, and Ex Libris is not responsible for any changes or updates to Shibboleth. Furthermore, Ex Libris is not responsible for providing any end user support with respect to Shibboleth. For further information about Shibboleth, refer to the appropriate Web site (http://shibboleth.internet2.edu/)

The following Ex Libris products and versions can interface with Shibboleth Version 1.32:

- MetaLib Version 3.13
- DigiTool Version 3.x
- ALEPH Version 18.x

All the above product versions work with Apache Version 2.

4.1 Prerequisites

In order to implement Shibboleth in the product listed above, you must first set up and configure the appropriate environment outlined below or have access to such an environment:

- A Shibboleth Identity Provider
- Shibboleth Service Provider software built and configured to work with the Shibboleth Identity Provider.

Note

When you set this up, PDS can play the role of a WAYF server. PDS can first present patrons with a list of institutions to choose for login. If a patron chooses a non-Shibboleth institution, PDS presents a login screen and handle the login. If a patron chooses a Shibboleth institution, PDS redirects to a Shibboleth Service Provider that links directly to one Shibboleth institution's Identity Provider. The Service Provider would yield to the identity provider to handle authentication, and would then retrieve the appropriate attributes. The Service Provider would then return these attributes to PDS in a format that PDS can translate. In this scenario, there would be a separate Service Provider for each Identity Provider that would be handled by PDS. Refer to XX on page for instructions on setting up Shibboleth to use PDS as a WAYF server.

- The service provider needs to be located on the same server as PDS
- Configuration of Shibboleth in the product’s Apache server
- URL for performing logout from the identity providers (optional)
4.1.1 Hints for installing the Shibboleth service provider

Ex Libris installs its own Apache on the system. First, you need to make sure it supports dynamic loading of shared modules (mod_so compiled in; check with “httpd -l”). The Apache is compiled against its own OpenSSL libraries. You need to make sure Shibboleth is using the same *shared* libraries (.so!) as Apache, or you see segfaults of the Apache children at startup. Check your system before attempting to compile the Shibboleth software. Here are some useful commands:

$ ldd /exlibris/metalib/m3_1/product/httpd-2.0.xx/bin/httpd
$ /exlibris/…/httpd-2.0.xx/bin httpd -l
$ ldd /exlibris/…/httpd-2.0.xx/modules/mod_ssl.so
$ ldd /path/to/shibboleth-sp/libexec/mod_shib_20.so

You may need to recompile (the supplied OpenSSL and) Apache so it can use the same libraries as Shibboleth.

Add the shared parameter to the ./product_source/install openssl.csh

29c29

< config --prefix=${base_exe}/${product_dir}

>config --prefix=${base_exe}/${product_dir} shared

Change the configure line in the ./product_source/apache2.csh

< ./configure --enable-modules="headers expires so rewrite ssl" --with-ssl=${base_ssl} --prefix=${base_exe}/${product_dir}

> ./configure --enable-modules="headers expires so rewrite ssl" --with-ssl=${base_ssl} --enable-ssl --enable-mods-shared=mod_ssl --prefix=${base_exe}/${product_dir}

4.1.2 Testing that the above prerequisites are met

Log the user into the Shibboleth Identity provider. Using the address bar of your Web browser, type the following http request:

http://<producthost>:<productport>/shib

This should launch a login screen of your Identity Provider. Enter a valid Shibboleth username and password and confirm that you can log into Shibboleth.

Note

It is highly recommended to run the SP on https.

4.2 Additional Apache configuration for Ex Libris products

4.2.1 Adding a symbolic link in /apache/htdocs directory

Create a separate institution subdirectory in the /shib directory under htdocs which has a symbolic link pointing to the pds_main program in the /pds/program/ directory. This
directory is protected by Shibboleth so PDS can consume the Shibboleth attributes after authentication. When running these instructions on an installation located in a slot other than m3_1, please revise the command accordingly and replace m3_1 with the appropriate slot indicator.

Create the symbolic link using the following commands:

```bash
>>apch
>>mkdir shib
>>cd shib/
>>mkdir $inst
>>cd $inst
>>ln -s /exlibris/metalib/m3_1/pds/program/pds_main pds_main
```

To confirm the creation of the symbolic link run the following command:

```bash
>>ls -l
```

The result should look like the following:

```bash
lrwxrwxrwx 1 metalib exlibris 43 Nov 22 08:18 pds_main ->
/exlibris/metalib/m3_1/pds/program/pds_main*
```

### 4.2.2 Changing Apache configuration files

Apache configuration files needs to be changed to regard pds_main as executable program. To do so, open the httpd.conf file or ssl.conf using the following commands. Change httpd.conf with ssl.conf where applicable:

```bash
>>apcc
>>cp httpd.conf httpd.conf.<date>
>>vi httpd.conf
```

Add the following configuration within a VirtualHost context (NOTE: VirtualHost definition must match Host in RequestMap in shibboleth.xml, in order for RequestMap to function properly):

Match the next lines to your setup (certainly change SHIB_ROOT):

```bash
LoadModule mod_shib  SHIB_ROOT/shibboleth/libexec/mod_shib_20.so
ShibSchemaDir SHIB_ROOT /shibboleth/etc/shibboleth
ShibConfig SHIB_ROOT/shibboleth/etc/shibboleth/shibboleth.xml

#for Shibboleth 1.2 only
#<Files *.*rire>
#SetHandler shib-shire-post
#</Files>

<Location /shib>
  AuthType shibboleth
  ShibRequireSession On
  #it might be smart to modify the next line:
  Options ExecCGI FollowSymlinks
  ForceType application/x-httpd-cgi
</Location>
```
4.2.3 Modify apache startup scripts (optional)
Modify apache startup scripts to start/sop shibd (formerly shar) process. Sample
apachectl and apachectl_auto scripts have been distributed for use with shibboleth. They
have been named apachectl.shib and apachectl_auto.shib (for an example of
apachectl.shib, see page 71 and for an example of apachectl_auto.shib, see page 76).

4.2.4 Modify metaib_start
In many cases it is not necessary to add the Shibboleth lib directory to the library search
path, but you are advised to do so anyway. Add the following lines to metaib_start:

```
# set shib env variables
setenv SHIB_CONFIG SHIBROOT/shibboleth/etc/shibboleth/shibboleth.xml
setenv SHIB_SCHEMAS SHIBROOT/shibboleth/etc/shibboleth
setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:SHIBROOT/shibboleth/lib
```

4.3 Implementing Shibboleth with Ex Libris Products
After the above prerequisites have been successfully fulfilled, PDS should be configured
as follows:

1. On the server, enter the following commands (where <inst> is a well-chosen
   value):
   ```
   >> pdsroot
   >> cd conf_table
   >> vi tab_service.<inst>
   ```
   This opens the tab_service.<inst> configuration file.

2. In tab_service.<inst>, configure the following services:
   ```
   [LOAD_LOGIN]
   program = shib.pl
   params = shib.conf
   [END]
   [BOR_INFO]
   program = shib_bor_info.pl
   [END]
   [INSTITUTE_DISPLAY]
   code = INST-CODE
   lang = eng
   desc = INST-DESCRIPTION
   institute = INST-CODE
   [END]
   ```

3. In tab_service.<inst>, set up logout service (optional):
   ```
   [SHIB-LOGOUT]
   params = http://server/logoutscript
   [END]
   ```
The SHIB-LOGOUT configuration needs to include a link to the customer’s identity provider logout URL if using one. If not, PDS only logs out of the service provider and Ex Libris products.

4. In tab_service.<inst>, set up Remote SSO (optional):
   
   ![LOAD_SSO]
   
   program = shib_sso.pl
   params = shib.conf

   [END]

   To complete the optional LOAD_SSO configuration if the _shibsession_ cookie has the institute as a suffix you can add the following lines to sso_conf.

   ![DEFAULT_INSTITUTE]
   
   cookie = _shibsession_(.*)

   [END]

5. Create a shib.conf file under the ./pds/conf_table directory. The following is an example of a shib.conf file. The file must be customized according to your institution's attributes. To be able to save user information (for example, favorites) you need to map at least "z312_source_id" from Shibboleth attribute that contains a unique value. Fields can also be specified to receive default values. Dates can be configured according to context as seen in the following example:

   ![shib_attributes]
   
   REMOTE_USER = z312_source_id
   HTTP_SHIB_PERSON_COMMONNAME = z312_name
   HTTP_SHIB_LIMSBARCODE = z312_source_id
   HTTP_SHIB_BORSTATUS = z312_academic_status

   [attributes_mapping]
   academic_status,ABC = group,STAFF

   [defaults]
   expiry_date, today+1y

6. Add the path to the program directory in the pds_main
   
   use lib "$ENV{metalib_dev}/pds/program";

4.4 Shibboleth LOGIN workflow

Below, find the flow of processes Shibboleth performs when a user logs in.

1. The user logs on and chooses the institute they belong to.

2. In the institute’s tab_service file. The LOAD_LOGIN section calls the program shib.pl. Following is an example of this section:

   ![LOADLOGIN]
   
   program = shib.pl
   params = shib.conf

   [END]
3. The shib.pl program is started. First checks for the existence of the environment parameter, \$ENV{'HTTP_SHIB_APPLICATION_ID'}. If the parameter exists, jump to step 8.

4. If the parameter does not exist, Shibboleth redirects using the redirect-remote-shib HTML. Pointing to the restricted location shibboleth:

   ```
   &server_httpsd/shib/institute/pds_main?func=load-login...
   redirect-remote-shib
   <html>
   <head>
   <title>Redirect remote login - no id exist</title>
   <include>meta-tags

   <script language=Javascript>
       function redirect()
       {
           var url = "&server_httpsd/shib/$0300/pds_main?func=load-login&"
               + "calling_system=$0200&"+
               "institute=$0300&"+
               "PDS_HANDLE=$0100&"+
               "url=$0400";
           top.location = url;
       }
   </script>
   </head>

   <body onload="javascript:redirect()">
   </html>
   ```

5. The browser redirects to the identity provider where the user can log in.

6. After logging in the browser runs pds_main?func=load-login.

7. This function, when initiated, reads the tab_service file. Upon reading the LOAD_LOGIN definition it calls the program shib.pl, which is now be able to find the \$ENV{'HTTP_SHIB_APPLICATION_ID'} parameter.

8. A pds_handle is generated.

9. The shib.pl looks for attributes which are mapped and written to the disk as a PDS cache file in preparation for a bor-info request.

10. The bor_info request fetches information written previously to disk and sends an XML response to the calling system.
4.5 The Shibboleth LOGOUT workflow

1. The user attempts to logout from PDS (or from a PDS calling application) by following a URL similar to the following:

   https://HOSTNAME:PORT/pds?func=logout(&calling_system=metalib&url=.. .)

2. This URL calls the program PDSLogout which checks for the existence of a cookie like _shibsession_. If no such cookie exists, jump to step 6.

3. If the shibsession cookie does exist, PDSLogout expires the cookie and redirects the user to logout from shibboleth using a link similar to the following:

   https://SHIB-HOSTNAME:PORT/shibboleth/logout?url=...

   Note, that at this point, logout is done only from the shibboleth service provider session and not from the PDS session.

4. The identity provider expires its cookie(s) and redirects back to the following URL:

   https://HOSTNAME:PORT/pds?func=logout(%26calling_system=metalib%26ur l=...)

5. The URL calls PDSLogout which again checks for the existence of a cookie like _shibsession_ and does not find one.

6. PDSLogout continues to log out the PDS session and possibly redirects back to the calling_system, depending on how the logout was originally called in step 1 and the backlink sent.

4.6 The Shibboleth SSO workflow

When a user logs into Shibboleth using ALEPH, and then accesses MetaLib, PDS checks if the user has preformed Single-On. If the LOAD_SSO section is defined in the tab_service.INSTITUTE file as described in section 3.3 Remote Single Sign-On (SSO) on page 25, the SSO workflow would be the following:

The shib_sso.pl program looks for the _shibsession_ cookie

If the cookie exists it continues with the following steps

1. A pds_handle is generated

2. The shib_sso.pl looks for attributes which are mapped and written to the disk as a Z312 record in preparation for a bor-info request.

3. The bor_info request fetches information written previously to disk and sends an XML response to the calling system.

4. PDS creates a session and returns the pds_handle to the calling application.

If there is no _shibsession_ cookie the user remains a GUEST.

4.7 Configuring Shibboleth as a WAYF

To configure Shibboleth as a WAYF, you need to edit the shibboleth.xml file. To the section beginning with these lines:

```xml
<RequestMap applicationId="default"/>
```
<Host name="SP-SERVER" scheme="http" port="80">

After this line:
<Path name="shib" applicationId="default" requireSession="true">

Add a separate mapping line for each institute to be used:
<Path name="Institute" applicationId="Institute" requireSession="true" exportAssertion="true"/>

or
<Path name="INSTITUTE" requireSession="true" requireSessionWith="INSTITUTE" exportAssertion="true"/>

Add a separate <Application> for each institute, each with a unique handlerURLshareURL and each with a Identity Provider specific wayfURL
<Application id="INSTITUTE">
<Sessions lifetime="7200" timeout="3600" checkAddress="true"
handlerURLshareURL="/shib/INSTITUTE/Shibboleth.sso" handlerSSLshare="false"
wayfURL=https://IDP-SERVER/shibboleth-idp/SSO/HS/>
</Application>

Note
For security reasons, you should run PDS on https and set handlerSSL to true.

5 PDS Customization

5.1.1 PDS HTML Directory: html_form
The html_form directory stores the HTML files and GIFs used by the PDS. There are by default three subdirectories:
/global
/calling_system-<product>
/icon

The global directory includes default HTML files used by PDS. The calling_system-metalib contains customized screens for MetaLib, the calling_system-digitool contains the DigiTool screens and the ALEPH screens are in calling_system-aleph. In addition every institution may have its own directory named /institute-institutecode with its own customized HTML files.

PDS first checks if there is a special institution login file page. If there is no such file, PDS searches for the calling system login file, failing which, it seeks and display the global login screen.

The order of the search for the login HTML page is:
1. /institute-<institute name>/login
2. /calling_system-calling-application/login
3. /global/login

The icon directory includes GIF files used by PDS.

To check which HTML file PDS is displaying and where it was found, use the “view source” option from the browser. At the top of the HTML source is a comment with the full path to the file.

<!-- The file is : $TMPDIR/utf_files/pds/html_form/calling_system-aleph/login -->

The HTML files are converted to UTF using the pds_utf_prog definition in PDSDefinitions:

```
our ($pds_utf_prog) = "/exlibris/calling application/xn_n/aleph/exe/pds_utf_file_name";
```

The output is stored in the ./tmp/utf_files/pds directory.

For example, if the file is html_form/calling_system-aleph/login and the $TMPDIR is /exlibris/aleph/m3_1/tmp, then the UTF file is located at /exlibris/aleph/m3_1/tmp/utf_files/pds/html_form/calling_system-aleph/login.

It is important to note that if the UTF file exists, it is used, therefore when making changes in the HTML files or when installing new code that changes values in the HTML, the UTF files should be deleted.

To remove the PDS UTF files:

```
> rm –r $TMPDIR/utf_files/pds/html_form/
```

5.1.2 The login page

The global login page resides in ./pds/html_form/global/login. It is displayed when the user accesses the PDS without a calling system parameter in the URL:

For example when accessing PDS without the calling system, use http://hostname:port/pds.

![Ex Libris Login Page](image)

*Figure 6: Ex Libris global PDS login page*

If the calling_system=metalib was passed in the URL, then the calling system - MetaLib HTML page is displayed:
Figure 7: MetaLib PDS login page

If the calling_system=digitool was passed in the URL, then the calling system - DigiTool HTML page is displayed:

Figure 8: DigiTool PDS login page

If the calling_system=aleph was passed in the URL, then the calling system - ALEPH HTML page is displayed:
Figure 9: ALEPH PDS login page

The Institution pull-down menu in the login page takes the names of institutions from names defined in the files in the conf_table directory. All files in the tab_service.xxxxxx format is used whereas files of the tab_service.xxx.xxx format is ignored.

5.1.3 Customized institution login screen

To create a customized institution login screen for the ARTS institution, first create a new directory for this institution as follows:

> pdsroot
> cd html_form
> mkdir institute-arts
> cp calling_system-metalib or calling_system-digitool or calling_system-aleph/* institute-arts

The file to be customized is “login” under the newly created directory:

./pds/html_form/institute-arts/login

Once this is done and the ARTS institution is selected, the view source of the login screen shows that the login HTML file was taken from the institute-arts directory.

The file is: $TMPDIR/utf_files/pds/html_form/institute-arts/login

5.1.4 PDS Direct access

There is an interface to PDS which does not entail a calling system.
http://host:port/pds . Through this interface one can check the “AUTHENTICATE” and “BOR_INFO” functionality without interactions from the calling application. This is mainly a debugging tool. The following sections explain the /pds interface.
5.1.5 PDS Direct access: The remote-and-local-list page instead of the login page

When the institute parameter is not sent to PDS, and one of the institutions is configured for REMOTE_LOGIN, then PDS displays a special page called remote-and-local-list instead of the login page for example:

![Figure 10: PDS Remote and Local List](image)

5.1.6 PDS Direct access: main-menu-logged-out

The main-menu-logged-out page resides in ./pds/html_form/global. It is displayed when the user accesses the PDS directly via a http://hostname:port/pds URL.

![Figure 11: PDS Main Menu](image)

The main-menu-logged-out file needs to be edited to reflect the choices presented to users when logging into the PDS. That includes a list of the applications sharing the PDS, the "href" links to the appropriate application/s and the display names. Following is an example of two applications defined in main-menu-logged-out.
The lines in bold below need to be edited to configure appropriate local settings/values:

```html
PDS - Patron Directory Services $0100
</TD>
</TR>
<TR STYLE="PADDING:2PX 0PX 6PX 0PX">
<TD CLASS="LableBoldDark">
<img src="&icon_path/f-separator.gif" border="0" alt="">
<a class= LableBold href="http://hostname:port/V" title = "Metalib - 3">
Metalib 3 on Mldev03 (m3_9) </a>
</TD>
</TR>
<TR STYLE="PADDING:2PX 0PX 6PX 0PX">
<TD CLASS="LableBoldDark">
<img src="&icon_path/f-separator.gif" border="0" alt="">
<a class= LableBold href="http://hostname:port/R" title = "DigiTool 3.0">
DigiTool 3 on Ram7 (d3_1) </a>
</TD>
</TR>
```

**Note**

If a site uses a single Ex Libris application, You can modify this screen so that it links back to the Ex Libris application instead of displaying the above menu. To activate the auto logoff option, add the following line:

```html
<body onload="top.location = 'http://www.ml-univ.com:8332/V'">
```

in the following file

PDS/html_form/calling_system-product/main-menu-logged-out

Below is a sample file defined to link back to ALEPH:

```html
<!-- main-menu-logged-out -->
<html>
<head>
  <title> Main Menu - Logoff </title>
  <include>meta-tags</include>
  <link rel="stylesheet" href="&server_httpd/pds.css" TYPE="text/css">
</head>
<body onload="top.location = 'http://www.ml-univ.com:8332/F'">

</body>
</html>
```

### 5.1.7 PDS Direct access: main-menu-logged-on

The main-menu-logged-on page resides in ./pds/html_form/global. It is displayed when the user accesses the PDS directly via a http://hostname:port/pds URL after a valid authentication.
5.1.8 Error messages from the PDS

The ./pds/conf_table/ directory holds the heading_error.eng file that defines error messages used by the PDS. You can customize PDS error messages by editing this file.

```
our ($default_con_lng) = "ENG";
```

The following error messages are currently used:

- 0001 L Missing Password
- 0002 L Missing ID
- 0003 L Service not defined in tab_service.
- 0004 L Invalid UserID and/or Password. Please re-enter.
- 0005 L Invalid institution.

The file has three columns:

- The first column defines the error number used by PDS programs and must NOT be changed.
- The second column is a language indicator.
- The third column defines the text to be displayed to the end user. Sites may customize this text.

5.1.9 PDS debug mode.

In the pds/program/PDSDefinitions file there is a debug flag. The default mode of the debug is off

```
our ($debug) = "N";
```

If the mode is turned to on

```
our ($debug) = "Y";
```
The $LOGDIR/pds_server.log tracks which pds programs were run with which parameters which can sometimes be a debugging aid. See 8 PDS Logs on page 58 for more information on pds_server.log.

From PDS 1.3 the PDS runs under mod_perl this means all the PDS programs are loaded once. Therefore in order to change the debug mode in addition to changing the PDSDefinitions touch PDSUtil.pm and PDStoucfile in the pdsroot program directory. This causes the Apache to reload the changed PDSDefintions.

There are pds_debug_on/pds_debug_off routines in the program directory under pdsroot to turn debugging on and off.

## 6 PDS Programs

As explained above, the tab_service tables contain definitions for the services that should be run when a user attempts to log in to an institution.

The program line lists programs for authentication, user information or remote login routines. They reside in the ./pds/service_proc directory. The following programs are available:

<table>
<thead>
<tr>
<th>Program Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aleph_141.pl</td>
<td>Uses the ALEPH X-Server – sends host and port and other parameters to the X–Server. For example: <a href="http://www.cityuniv,8991,BOR_AUTH,USM50,N">www.cityuniv,8991,BOR_AUTH,USM50,N</a></td>
</tr>
<tr>
<td>aleph_142.pl</td>
<td>Which is sent to the X-Server as <a href="http://www.cityuniv:8991/X?op=BOR_AUTHENTICATE&amp;BOR_ID=">http://www.cityuniv:8991/X?op=BOR_AUTHENTICATE&amp;BOR_ID=</a>&lt;xxx&gt;&amp;VERIFICATION=&lt;xxx&gt;&amp;BOR_LIBRARY=USM50. Depending on the N the call is httpd or httpds and the port is 443 and sets &amp;TRANSLATE=N</td>
</tr>
<tr>
<td>aleph.pl</td>
<td>Interfaces with an LDAP server for remote authentication. It requires a configuration file for each institute.</td>
</tr>
<tr>
<td>ldap.pl</td>
<td>Interfaces with an LDAP server for remote authentication. It requires a configuration file for each institute.</td>
</tr>
</tbody>
</table>

Important Note!

The ALEPH interfaces depend on the ALEPH version. Aleph.pl works with both ALEPH 16 and ALEPH 17 and on on, but Aleph_152.pl does not work with ALEPH 17.

In addition, verify the ALEPH X-Server is working correctly (license and so on) and that the WWW-X user name / and WWW-X password is defined in your management interface.

When requesting a secure HTTPS connection (Y) in the interface, the port must be the secure port for example:

**PROGRAM-NAME** 

<table>
<thead>
<tr>
<th>PROGRAM-NAME</th>
<th>aleph.pl</th>
</tr>
</thead>
<tbody>
<tr>
<td>160.45.152.196,</td>
<td></td>
</tr>
<tr>
<td>443,BOR_AUTHENTICATE,FUB00,Y,WWW-X,WWW-X</td>
<td></td>
</tr>
<tr>
<td>Program Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ldap_prn.conf</td>
<td></td>
</tr>
</tbody>
</table>
| metalib_x_server.pl          | Uses the MetaLib X-Server – sends host, port and other parameters to X-Server
    
    ram47,832,BOR-AUTH,N,pds,xxx  |
| remote_cgi_hook.pl           | Remote authentication hook
    
    GET,130.182.125.3:2080,cgi-bin/rpa_cla.pl  |
| remote_cgi_hook_ssl.pl       | Remote authentication hook with SSL (Secure Sockets Layer) support for secure communication between the PDS hook and the local CGI program. For example
    
    POST,https://vulture.library.colostate.edu:443,cgi-bin/metauth.cgi,BOR_ID  |
| ichain.pl                    | Ichain PDS component - checks for the existence of an environment variable HTTP_AUTHORIZATION from which it extracts the user ID and password.                                                                                                                                                                                             |
| cas.pl                       | CAS PDS component - checks for the existence of an environment variable HTTP_PDS_ID which indicates an authenticated CAS user.                                                                                                                                                                                                             |
| shib.pl                      | Shibboleth PDS component - checks for the existence of an environment variable HTTP_SHIB_APPLICATION_ID which indicates an authenticated Shibboleth user.                                                                                                                                                                                          |

These programs are explained in the next sections.

### 6.1 Authentication against X-servers

PDS contains standard programs to authenticate users against X-Servers for Ex Libris applications (aleph.pl, metalib_x_server.pl, digitool.pl).

The URL of a request to an X-server in PDS consists of two parts:

Hostname and port: http://www.exlibrisgroup.com:8331

Query string: op=bor_authenticate&bor_library=USM50

BOR_AUTH and BOR_AUTHENTICATE are both accepted formats of the same service

For ALEPH X-servers it is recommended to use BOR_AUTH service instead of the BOR_INFO for performance reasons as the records are shorter.

Refer to X-Services documentation for the list of values the X-servers can return.

### Examples of X-server PDS Configurations:

For ALEPH

```
[AUTHENTICATE]
program = aleph.pl
params  = ram11,8993,BOR_AUTHENTICATE,USM50,N,WWW-X,WWW-X
[END]

and

[BOR_INFO]
program = aleph.pl
params  = ram11,8993,BOR_AUTHENTICATE,USM50,N,WWW-X,WWW-X
```
For ALEPH 15.2:

[AUTHENTICATE]
program = aleph_152.pl
params  = ram9,8082,BOR_AUTHENTICATE,USM50,N
[END]

[BOR_INFO]
program = aleph_152.pl
params  = ram9,8082,BOR_INFO,USM50,N
[END]

For ALEPH 14.1:

[AUTHENTICATE]
program = aleph_141.pl
params  = ram9,8081,BOR_AUTHENTICATE,USM50,N
[END]

[BOR_INFO]
program = aleph_141.pl
params  = ram9,8081,BOR_INFO,USM50,N
[END]

For ALEPH 14.2:

[AUTHENTICATE]
program = aleph_142.pl
params  = ram37,8082,BOR_AUTHENTICATE,USM50,N
[END]

[BOR_INFO]
program = aleph_142.pl
params  = ram37,8082,BOR_INFO,USM50,N
[END]

For DigiTool 3.0

[AUTHENTICATE]
program     = digitool.pl
params      = ram7:8881,op=BOR_AUTH,DAT01,PDS,PDS,N
[END]

[BOR_INFO]
program     = digitool.pl
params      = ram7:8881,op=BOR_INFO,DAT01,PDS,PDS,N
[END]

For MetaLib 3.1x

[AUTHENTICATE]
program     = metalib_x_server.pl
params      = 10.1.235.60,8339,BOR-AUTH,N,PDS,PDS
[END]

[BOR_INFO]
program     = metalib_x_server.pl

6.2 LDAP Services

PDS has a standard program ldap.pl to interface with an LDAP server for remote authentication and the fetching of user attributes.

The ldap.pl program needs to interface with many different LDAP servers and many different configurations. It has a configuration file that defines how it interacts with the server. See section 6.2.1 LDAP Configuration file on page 50 below for a full explanation of all the parameters.

The flow of the ldap.pl script is

- The LDAP script (ldap.pl) reads the conf file.
- PDS connects to the given LDAP hostname and establishes a secure or non-secure communication (depending on "secure_ldap" flag).
- If initial bind parameters ("init_bind_dn", and "init_bind_password" ) are defined in the configuration file, then an initial bind using the configuration file parameters is performed.
- PDS searches the LDAP tree to find the user’s record according to the given search_base and search_filter. The hardcoded USERNAME token is replaced with the name from the login page and the token PASSWORD is replaced with the verification from the login page.
- If the results are not unique (or zero size result), repeat the search step for the next given base/filter.
- If there is no “bor_info_only = Y” parameter a bind is attempted for the search object using the password from the login page.
- If the bind succeeds <auth>Y</auth> is returned in the XML and the “AUTHENTICATE” or “BOR_INFO” step succeed.
- Otherwise the XML contains <auth>N</auth> the return code is “11” and an error message is displayed in the logon page. A more specific error message is printed to the $LOGDIR/pds_server.log. See 8 PDS Logs on page 58 for more information about the pds_server.log.

6.2.1 LDAP Configuration file

The LDAP configuration file has the following sections:

[general]

This section defines attributes of the LDAP server interaction:

The **hostname and port** of the remote LDAP server.

**Initial Username and password.** Many LDAP servers enable anonymous login, in which case no username/password is needed. If it is required, for initial binding, enter the username/password assigned to the calling application for accessing the LDAP server in the init_bind_dn and init_bind_password parameters.
secure_ldap Y/N flag which defaults to N. If the flag is set to yes SSL protocol is used to communicate with the LDAP server.

init_bind_dn enter the full “dn” (distinguished name) for the initial bind.

init_bind_password enter the dn's password for the initial bind.

search_base enter the full path search in the ldap directory tree to the user.

search_filter enter the parameter to filter the results to return only 1 object.

The search_base and search_filter parameters can be repeated to search in more then one tree.

ldap_version is not a mandatory field. If it is not present or if it is empty, the ldap.pl attempts a bind with its default version which is 2. If the LDAP server is a version 3 server, the ldap_version in the conf. table must be set to 3 for the bind to be successful.

bor_info_only is an optional line. If it is present the ldap.pl skips the bind step and return bor_info without attempting to bind.

auth_only is an optional line. If it is present the ldap.pl only validates the password and does not return bor_info.

If auth_only = Y

dn is a mandatory line and contains the full distinguished name for the bind the search base and search filter are not used in this scenario.

auth_only = Y and bor_info_only = Y are mutually exclusive.

start_tls = Y switch to secure

Calling this method converts the existing connection to using Transport Layer Security (TLS), which provides an encrypted connection. This is only possible if the connection uses LDAPv3.

utf_to_charset = ISO-8859-1

### 6.2.2 LDAP Configuration file examples

Below is an example of an initial bind setup using an LDAP version 3 server.

```plaintext
[general]
host_name = cityuniv.library.edu
port = 389
init_bind_dn = cn=Ldap,o=Citytown University,c=US
init_bind_password = t670P11
ldap_version = 3
```

Below is an example of a search base and search filter for querying the LDAP Server. The search_base defines the LDAP base to be searched and the search_filter, the LDAP user; the USERNAME token is a placeholder and is replaced with the name value from the login page.
search_base = o= City University, st=New York ,c=US
search_filter = uid=USERNAME

In the above example, if a user "johndoe" is trying to authenticate against the LDAP server, PDS performs LDAP searches on the base City University, and filters the results where uid="johndoe".

Multiple search_base and search_filter pairs can be defined. For example, to search in both student and faculty trees:

- search_base = ou=fac,o=sunynp
- search_filter = cn=USERNAME
- search_base = ou=stu,o=sunynp
- search_filter = cn=USERNAME

PDS supports references from LDAP servers. The system tries to bind against the defined LDAP server, and follows a reference to another LDAP server, when given.

Example of auth_only: no attributes are returned only an <auth>Y</auth> based on the dn parameter

[general]
host_name = cityuniv.library.edu
port = 389
ldap_version = 3
auth_only=Y
dn= cn=USERNAME,o=Citytown University,c=US

Example of bor_info_only: no bind is performed on the object found by the LDAP search.

[general]
host_name = cityuniv.library.edu
port = 389
search_base = ou=fac,o=sunynp
search_filter = cn=USERNAME
bor_info_only = Y

When working with “SECURE” mode (LDAP over SSL), enter a flag in the LDAP configuration file, as follows:

[general]
host_name = cityuniv.library.edu
port = 636
search_base = o=City University, st=Ohio, c=US
search_filter = uid=USERNAME
secure_ldap=Y

[xml setting]
xml_root_node = bor_authentication

[attributes mapping]
cn = user_name
mailLocalAddress = email_address
PortalName = portal_name
portalName,EBSCO = z303-birth-date,19910101
This section is required by PDS and must not be modified or deleted:

```xml
[xml setting]
  xml_root_node = bor_authentication
```

### Attributes mapping

This section is used to map the LDAP user record fields to the calling application’s user record fields in the user table.

Example:

```xml
[attributes mapping]
  cn = name
  mail = email_address
```

### Defaults

This section sets defaults for user attribute mapping. You can set a default value for any of the fields that can be mapped to the user record, particularly if the site wishes to force a default value for all users. For example, to set all users expiry-date to 20201231, use the following:

```xml
[defaults]
  expiry-date,20201231
```

#### 6.2.3 Attributes mapping and defaults

**Note**

Underscores and hyphens are interchangeable in the calling application update but not in the mapping tables.

The tag names in the LDAP XML need to be mapped to names which the calling system recognizes. (For a full list of the current recognized tags see the section on Attribute Mapping on page 56).

In the above code, the LDAP returns

```
sn: Orange
cn: Becky Orange
mailLocalAddress: becky.orange@exlibris.co.il
PortalName: EBSCO
```

The attribute mapping section normalizes the names and can also be used to map values and add defaults. For example the LDAP “cn” is mapped to the field “user_name”. This is name normalization for the calling application’s update program. An example of mapping values is the last line of the attributes mapping below, the z303-birth-date,19910101 is appended to the XML when the input contains a portalName whose value is EBSCO. The defaults section example is add an expiry-date containing the date “20050101” if there is no expiry-date field in the incoming XML.

```xml
[attributes mapping]
  cn = user_name
  mailLocalAddress = email_address
  PortalName = portal_name
  portalName,EBSCO = z303-birth-date,19910101
```
6.3 Remote CGI Hook

PDS provides a custom hook to enable an interface with user defined external authentication servers (for authentication) and user defined sources for user attributes. The hook is enabled through remote_cgi_hook.pl or remote_cgi_hook_ssl.pl (both programs are included in the PDS package).

An institution wishing to use this method should write a CGI program that interacts with the PDS hook, as specified below, and handles regular or secure transactions/communication between PDS and the relevant authentication server.

6.3.1 Remote CGI Hook: Setup

To set up PDS to work with a remote hook for authentication, configure the AUTHENTICATE and BOR_INFO services in the tab_service.institute table. The syntax is GET or POST and http address to a user defined CGI script.

For example:

```
[AUTHENTICATE]
program     =  remote_cgi_hook.pl
params      =  GET,10.1.235.39:8997,aleph-cgi/remote_hasharon.pl
[END]

[BOR_INFO]
program     =  remote_cgi_hook.pl
params      =  GET,10.1.235.39:8997,aleph-cgi/remote_hasharon.pl
[END]
```

6.3.2 Remote CGI Hook interface

The remote_cgi_hook.pl receives the address of the user CGI script from the tab_service.institute (the params of AUTHENTICATE or BOR_INFO service) and adds to it the following parameters from the login page.

- `id` User ID
- `verification` User Password
- `institute` User Institution

An example of an authentication request:

```
```

6.3.3 Remote CGI program

The CGI program communicates with the PDS hook in the following way:

**Input:**

Accepts a user’s ID, user’s password (optional when the hook is used to retrieve attributes) and user’s institution.

**XML Output:**
For Authentication: Sends an authentication flag (Y/N back to PDS). Optionally, it may also provide user attributes.

For Attribute Retrieval: Sends the user’s attributes back to PDS.

The communication between the CGI program and PDS is based on the HTTP protocol and XML.

**Note**

It is possible for each institution to have and use its own CGI program.

Use the remote_cgi_hook_ssl.pl program if your Apache server supports SSL for secure communication between PDS and your CGI program.

The CGI program should return an XML document containing all the mandatory fields marked with an asterisk below.

This is a partial list, the full list of fields can be found in Appendix B – The bor_info.tags attribute on page 67.

<table>
<thead>
<tr>
<th>Field</th>
<th>XML tag</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>*Authenticated ?</td>
<td>&lt;auth&gt;</td>
<td>Y – authenticated</td>
</tr>
<tr>
<td></td>
<td>&lt;/auth&gt;</td>
<td>N – Not authenticated</td>
</tr>
<tr>
<td>*User ID</td>
<td>&lt;id&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;/id&gt;</td>
<td></td>
</tr>
</tbody>
</table>

Examples of CGI program replies:

When used to authenticate a user:

```xml
<?xml version="1.0" ?>
<bor_authentication>
  <auth>Y</auth>
</bor_authentication>
Or:

<?xml version="1.0" ?>
<bor_authentication>
  <auth>N</auth>
</bor_authentication>
```

When used to retrieve user attributes:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bor_info>
  <id>9036860</id>
  <institute>Harvard</institute>
  <group>GRAD</group>
  <name>Smith, John</name>
  <open-date>00000000</open-date>
  <name>Smith, John</name>
  <title>Mr.</title>
  <academic_status></academic_status>
  <address_0>10 Tree street</address_0>
  <address_1></address_1>
  <address_2></address_2>
  <address_3></address_3>
```

© Ex Libris Ltd., 2007  Page 55 of 76
Patron Directory Services version 1.3
Last Update: February 13, 2007
Note

A staff user testing the results of the program can view the resulting user record via the User Administration module (/M in the MetaLib Management interface). The first time a user logs in, a user record is created in the local calling application’s user file with the information provided via the CGI program. Thereafter, every time a user logs in, the user’s details are updated.

7 Attribute Mapping

Information is delivered to PDS in many formats and its PDS’s responsibility to normalize all the different types of information to a format the calling application can handle.

The attributes coming from the remote authentication pass through several filters:

- When using LDAP, the ldap_inst.conf file processes the raw output of the LDAP server. For more information on LDAP configuration refer to section 6.2.3 Attributes mapping and defaults on page 53.
- All XML’s pass through the bor_info.tags file located under pds/program/conf/. This is a system file and it should not be modified as changes made to it is overridden by updates and fixes made to PDS. This file defines the normalizations needed between all the calling applications. It has tables for normalizing input from ALEPH, DigiTool and MetaLib.
- Optionally the user can define custom mapping files. These files are created in the ./pds/conf_table directory and named INSTITUTE.tags or INSTITUTE_calling_application.tags. If an INSTITUTE.tags file is found the INSTITUTE_calling_application file is ignored. It is recommended to use the naming convention <Institute code>.tags (for example HUJI.tags).

7.1 User defined attribute and value mapping and defaults

When there is an institute-specific configuration table, it can modify the XML in three ways.

- Field to Field mapping
- Value to Value mapping
- Adding new fields and values

7.1.1 Field to Field mapping

These mapping lines are in the [attributes+values mapping] section of the INSTITUTE.tags file and they consist of two columns divided by an "=" sign.
This option lets you map values to be transferred from field A to field B.

When the input is one of the X-Server programs the Field to Field mapping is less relevant since the default bor-info.tags file knows the original field names and how they should be normalized.

An example of Field to Field mapping:

```
[attributes+values mapping]
z312-name = name
```

If a field exists in the original input, <z312-name> with a value. The action is to change the name of the field to <name> in the output XML.

### 7.1.2 Value to Value mapping

These mapping lines are also in the [attributes+values mapping] section of the INSTITUTE.tags file.

The structure of the table consists of two columns (name,value pairs) divided by an "=" sign. The left hand side has the input field name and value as defined in the input XML and the right hand side has the field and value to be added to the output XML.

This is conditional mapping and it means that based on an existing field and value an additional field and value is added to the output XML.

For example:

```
[attributes+values mapping]
z305-bor-status,01 = group,STAFF
```

If a field exists in the original input, <z305-bor-status>, with the value, “01”. The action is to create an additional field, <group>, with the value, “STAFF”. The string matching process is not case-sensitive.

### 7.1.3 Adding new fields and values

These mapping lines are in the [defaults] section of the INSTITUTE.tags file.

The defaults portion of the table consists of a name and a value separated by a comma.

For example:

```
[defaults]
portal-name,SCIENCE
```

If there is no field called <portal-name> in the original input a field portal-name with the value, “SCIENCE” is added to the output XML.

For example:

```
[defaults]
expiry_date,today+1y
```
For date fields there is a dynamic default mechanism. The input is: today +ny +nm +nd, where n is any number and y is years, m is months and d is days. Any or all of months days and years can be used in any order and spaces do not matter.

If this is the input XML:

```xml
<bor-info>
  <z305-bor-status>01</z305-bor-status>
</bor-info>
```

It is converted to the following Output XML. Note the two additional entries modified according to the attribute mapping rules defined: Note the z305-bor-status was changed to bor-status by the bor_info.tags file.

```xml
<group>STAFF</group>
<portal_name>SCIENCE</portal_name>
```

- `<bor-info>
  <z305-bor-status>01</z305-bor-status>
  <bor-status>01</bor-status>
  <group>STAFF</group>
  <portal-name>SCIENCE</portal-name>
</bor-info>

7.1.4 Tips for successful attribute mapping

The fields defined in the ldap.conf file is filtered by the bor_info.tags so the name may change between the ldap.conf file and the institute.tags file. Check the bor_info.tags file to avoid name space collisions.

Value to Value mapping works on the input before the bor_info.tags field to field mapping. The mapping has to be z305-bor-status,01 = group,STAFF and not bor-status,01= group,STAFF.

Only fields with values and fields that match a column on the left side of the bor_info.tags file appears in the final output XML.

8 PDS Logs

The pds log file named pds_server.log can be found in the $LOGDIR directory. PDS logs are restarted with the start_w command. PDS logs older than 30 days are automatically removed.

9 PDS setup validity testing

PDS has a validity testing utility designed to check the setup configuration of the PDS system to ensure that definitions are valid and that destinations are reachable.

The validity of the configuration files per institution can be checked by entering the user, password and a template file for that institution or by checking all institutions based on predefined users and templates that are stored in a table.
Activation: cd to calling application proc directory

`pds_check`

The following menu is displayed:

```
PDS Check Utility - Main Menu
================================
0. Exit
1. Edit pds_tab_users
2. Check file PDSDefinitions
3. Check tab_service tables for institutes
4. View log files
```

Select [0]:

Each option is explained below.

**Option 1:** Edit pds_tab_users: Allows changing information in pds_tab_users.

Submenu Options:
- Change default template – select template general to all institutes that have no private template
- Edit institute information – Prompt user per institute for user_id, verification and template. The passwords are encrypted.

**Option 2:** Check file PDSDefinitions

- Check if directories defined in ../pds/program/PDSDefinitions exist.
- Check if IP addresses defined are reachable.

**Option 3:** Check tab_service tables for institutes

User is prompted as follows:

Select institute or ALL
- If a single institute is selected, the user is prompted to enter:
  - user ID
  - verification
  - template extension
  - confirmation
- If all institutes (ALL) is selected, user is prompted to enter:
  - Default template extension
  - confirmation

Check that all services that appear in template exist in tab_service.institute.

Check that all parameter lines services that appear in template exist in tab_service.institute.

Check correctness as follows:

Presence of the program file in the service_proc directory.
  - Are the parameters valid?
Does “function” parameter match the service?
- Is the port reachable?
- Are the code and language supplied?
- Is authentication obtained with the supplied parameters?

**Option 4: View log files**
Enable view of summarize errors in the log files.

**10 PDS encrypt password utility**

From PDS 1.3 there is a utility `./pds/program/pds_encrypt.pl` which encrypts the service password to the X-server session. This is an optional step as the x-server.pl programs work in both encrypted and decrypted mode.

The `pds_encrypt.pl` program takes one parameter, institute, the suffix of the `tab_service.institute` file.

For example to update `tab_service.digitool`

```
>>pdsroot
>>cd program
>>perl pds_encrypt.pl digitool
```

**Originally:**

```
[AUTHENTICATE]
program = digitool.pl
params = ram7:8881,op=BOR_AUTH,DAT01,PDS,PDS,N
[END]

[BOR_INFO]
program = digitool.pl
params = ram7:8881,op=BOR_INFO,DAT01,PDS,PDS,N
[END]
```

**After pds_encrypt.pl**

```
[AUTHENTICATE]
program = digitool.pl
params = ram7:8881,op=BOR_AUTH,DAT01,PDS,!encrypt@GldX3qXt0cS7nZMuMqOpqzAIBF8BFsPcjMghu4TGbm0=,N
[END]

[BOR_INFO]
program = digitool.pl
params = ram7:8881,op=BOR_INFO,DAT01,PDS,!encrypt@G1RT2KXt0cS7nZMuMqOpqzAIBF8BFsPcjMghu4rFZW8=,N
[END]
```
11 X-Server Security patch

A security flaw exists that can expose details exposed. Manual changes are needed for blocking access to restricted content. The flaw causes attributes of an authenticated user to be available in XML using the following URL syntax:


11.1 Manual changes

- Map allowed IP addresses. Add to the $apache_home/conf/hosts-allow file calling system’s server IP address such as in the following example:

  10.1.235.47  FOUND

- Add a restriction in Apache using mod_rewrite so that only the calling system machine can access this link. In the $apache_home/conf/httpd.conf section after the declaration “RewriteEngine On”, add the following rewrite rules. These are based on the hosts-allow file configurations.

```perl
RewriteMap  hosts-allow  txt:/exlibris/pds/p1_2/apache/conf/hosts-allow
RewriteCond  %{QUERY_STRING} ^func=get-attribute&attribute=BOR_INFO [NC]
RewriteCond  %{hosts-allow:%{REMOTE_ADDR}|NOT-FOUND} NOT-FOUND
RewriteRule  ^.*$  /PDSAccessNotAllowed.html
```

These rules determine that when the $query_string contains a func=get-attribute and the $remote_addr is not found in the hosts-allow lookup table, the browser is redirected to a NotAllowed.html.

- Create the file, $apache_home/htdocs/PDSAccessNotAllowed.html. The contents of which should include the following code.

```html
<html><head><title>PDS Error</title></head><body bgcolor="#ffffff" LEFTMARGIN="0" TOPMARGIN="0" MARGINWIDTH="0" MARGINHEIGHT="0">
<TABLE CELLPADDING="0" CELLSPACING="0" BORDER="0" width="100%">
  <TR bgcolor="#C0DDEA">
    <TD height="27" style="COLOR: #43596B; FONT-SIZE: 70%;FONT-WEIGHT:BOLD; FONT-FAMILY: TAHOMA,ARIAL,VERDANA; PADDING-LEFT: 12PX;PADDING-RIGHT: 12PX;" width="100%">Access Denied Error:</TD>
  </TR>
</TABLE>
<div style="height:150px;"></div>
<table bgcolor="#C4E0ED" CELLPADDING="1" CELLSPACING="1" align="center">
  <tr>
    <td bgcolor="#EEF4F7">
      <table CELLPADDING="4" CELLSPACING="4">
        <tr>
          <td style="color: #FF0022; font-weight: bold; FONT-FAMILY: VERDANA,TAHOMA,ARIAL; FONT-SIZE: 70%;">Cannot process request: The service you are attempting to access is denied</td>
        </tr>
      </table>
    </td>
  </tr>
</table>
</body></html>
```
Appendix A – How to set up PDS

There is an example table of all the tab_service sections in the tab_service-example.txt file in the ./pds/conf_table directory.

Adding an institution

Open a new tab_service.institute file under the ./pds/conf_table directory.

- pdsroot
- cd conf_table
- vi tab_service.institute

Note

In MetaLib a new tab_service.institute file is created automatically when creating a new MetaLib institute.

ALEPH Standard Configuration

The following example shows the tab_service.science entries for a new institution named “SCIENCE”. This institution has decided to use the ALEPH authentication service and the ALEPH user database to obtain user attributes.

```
[AUTHENTICATE]
program = aleph.pl
params  = <server_name>,<port>, BOR_AUTH,USM50,N,ALEPH,ALEPH

[END]

[BOR_INFO]
program = aleph.pl
params  = <server_name>,<port>, BOR_INFO,USM50,N,ALEPH,ALEPH

[END]

[BOR_VERIFICATION]
program = get_pds_verification.pl

[END]

[INSTITUTE_DISPLAY]
code    = SCIENCE
lang    = ENG
desc    = Science Institute

[END]
```

MetaLib Standard Configuration

The following example shows the tab_service.science entries for a new institution named “SCIENCE”. This institution has decided to use the MetaLib authentication service and the MetaLib user database to obtain user attributes.

```
[AUTHENTICATE]
program = metalib_x_server.pl
```

© Ex Libris Ltd., 2007  Page 63 of 76
DigiTool Standard Configuration

The following example shows the tab_service.science entries for a new institution named “SCIENCE”. This institution has decided to use the LDAP followed by the DigiTool for both authentication and user attributes. If the username/password is authenticated by the LDAP, the authentication succeeds; if the LDAP authentication is unsuccessful, PDS attempts to authenticate against the local DigiTool database.

```
[AUTHENTICATE]
program = digitool.pl
params = <server_name>:<port>, op=BOR_AUTH,DAT01,PDS,PDS,N
[END]

[BOR_INFO]
program = digitool.pl
params = <server_name>:<port>, op=BOR_INFO,DAT01,PDS,PDS,N
[END]

[INSTITUTE_DISPLAY]
code = SCIENCE
lang = ENG
desc = Science Institute
[END]
```

LDAP Configuration

The following example shows the tab_service.law entries for a new institution named “LAW”. This institution has decided to use an LDAP server as its authentication method and it also uses the LDAP user database to obtain user attributes.

```
[AUTHENTICATE]
program = ldap.pl
params = ldap_law.conf
[END]

[BOR_INFO]
program = ldap.pl
params = ldap_law.conf
[END]

[INSTITUTE_DISPLAY]
code = LAW
```
The PDS LDAP configuration file must be configured for this institution. This is done in a file called ldap_law.conf in the ./pds/conf_table directory.

**ALEPH X-server Configuration**

The following example shows institution “CITYUNIV” using an X-Server configuration (for ALEPH 17), to authenticate users and retrieve user attributes.

```
[AUTHENTICATE]
program = aleph.pl
params  = ram19,8081,BOR_AUTHENTICATE,USM50,N,WWW-X,WWW-X
[END]

[BOR_INFO]
program = aleph.pl
params  = ram19,8081,BOR_AUTHENTICATE,USM50,N,WWW-X,WWW-X
[END]

[INSTITUTE_DISPLAY]
code    = CITYUNIV
LANG    = ENG
Desc    = City University
[END]
```

Explanation of the ALEPH X-server service’s third line, column three data string:

```
ram19,8081,BOR_AUTHENTICATE,USM50,N,WWW-X,WWW-X
```

- **ram19** – Server name
- **8081** – Port
- **BOR_AUTHENTICATE** – X-Server function / service
- **USM50** – Active library
- **N** – Use non secure HTTPD (“Y” – Use secure HTTPS)
- **WWW-X** – X-Server username
- **WWW-X** – X-Server password

**Remote CGI Configuration**

The following example shows a new institution named “ARTS”.

This institution uses a remote CGI hook program for authentication and retrieval of user attributes.

```
[AUTHENTICATE]
program = remote_cgi_hook.pl
params  = GET,www.university.edu:8992,cgi-bin/remote_cgi
[END]

[BOR_INFO]
program = remote_cgi_hook.pl
params  = GET,www.university.edu:8992,cgi-bin/remote_cgi
```

© Ex Libris Ltd., 2007  
Patron Directory Services version 1.3  
Last Update: February 13, 2007
Setup Help for the System Librarian/Administrator

In order to customize tab_service:

1. Copy the existing file and save it under a new file name.
2. Replace ARTS with your local institute.
3. Replace GET, www.university.edu:8992, cgi-bin/remote_cgi with the appropriate:
   <Method>, <Base-URL>, <remote-cgi path>.

Login Configuration

To set up PDS to SSO (Single Sign-On) across all Ex Libris products:

> pdsroot
> cd conf_table
> vi sso_conf

Change sso_conf to:

[LOGON]
TYPE1 = metalib, digitool, aleph (names of the sharing applications)
[END]
(Users is logged on to all Applications)

Logout Configuration

To set up PDS to SSO (Single Sign-Off) across All Ex Libris products:

> pdsroot
> cd conf_table
> vi sso_conf

Change this file to:

[LOGOUT]
TYPE1 = metalib, digitool, aleph (names of the sharing applications)
[END]
(Users are logged off all Applications)

To set up PDS to selectively sign-off: (only relevant for Ex Libris products)

> pdsroot
> cd conf_table
> vi sso_conf

Change this file to:

[LOGOUT]
TYPE2 = metalib, digitool, aleph (prompts the user with the application s/he wishes to log off from)
[END]
Appendix B – The bor_info.tags attribute mapping table

The table (pds/program/conf/bor_info.tags) is divided into sections, each section lists the possible user attributes originating from a defined application, such as ALEPH, MetaLib or DigiTool.

Even input that doesn’t come from one of these sources is translated through this standard table.

The table is constructed of two columns divided by a "=" sign. The left column defines the values that the PDS receives from the source of the user attributes (ALEPH, LDAP, CGI hook, and so on) and the right column defines the normalized attribute names PDS sends to the calling application (ALEPH, MetaLib or DigiTool).

This file handles all user attributes passed through PDS and should **not be changed**.

**ALEPH attributes**

<table>
<thead>
<tr>
<th>User Attribute</th>
<th>Normalized Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>z303-alpha</td>
<td>alpha</td>
</tr>
<tr>
<td>z303-birth-date</td>
<td>birth-date</td>
</tr>
<tr>
<td>z303-budget</td>
<td>budget</td>
</tr>
<tr>
<td>z303-con-lng</td>
<td>con-lng</td>
</tr>
<tr>
<td>z303-delinq-1</td>
<td>delinq-1</td>
</tr>
<tr>
<td>z303-delinq-1-cat-name</td>
<td>delinq-1-cat-name</td>
</tr>
<tr>
<td>z303-delinq-1-update-date</td>
<td>delinq-1-update-date</td>
</tr>
<tr>
<td>z303-delinq-2</td>
<td>delinq-2</td>
</tr>
<tr>
<td>z303-delinq-2-cat-name</td>
<td>delinq-2-cat-name</td>
</tr>
<tr>
<td>z303-delinq-2-update-date</td>
<td>delinq-2-update-date</td>
</tr>
<tr>
<td>z303-delinq-3</td>
<td>delinq-3</td>
</tr>
<tr>
<td>z303-delinq-3-cat-name</td>
<td>delinq-3-cat-name</td>
</tr>
<tr>
<td>z303-delinq-3-update-date</td>
<td>delinq-3-update-date</td>
</tr>
<tr>
<td>z303-delinq-n-1</td>
<td>delinq-n-1</td>
</tr>
<tr>
<td>z303-delinq-n-2</td>
<td>delinq-n-2</td>
</tr>
<tr>
<td>z303-delinq-n-3</td>
<td>delinq-n-3</td>
</tr>
<tr>
<td>z303-field-1</td>
<td>field-1</td>
</tr>
<tr>
<td>z303-field-2</td>
<td>field-2</td>
</tr>
<tr>
<td>z303-field-3</td>
<td>field-3</td>
</tr>
<tr>
<td>z303-home-library</td>
<td>home-library</td>
</tr>
<tr>
<td>z303-id</td>
<td>id</td>
</tr>
<tr>
<td>z303-ill-library</td>
<td>ill-library</td>
</tr>
<tr>
<td>z303-name</td>
<td>name</td>
</tr>
<tr>
<td>z303-name-key</td>
<td>name-key</td>
</tr>
<tr>
<td>z303-open-date</td>
<td>open-date</td>
</tr>
<tr>
<td>z303-profile-id</td>
<td>profile-id</td>
</tr>
<tr>
<td>z303-proxy-for-id</td>
<td>proxy-for-id</td>
</tr>
<tr>
<td>z303-title</td>
<td>title</td>
</tr>
<tr>
<td>z303-update-date</td>
<td>update-date</td>
</tr>
<tr>
<td>z304-address-0</td>
<td>address-0</td>
</tr>
<tr>
<td>z304-address-1</td>
<td>address-1</td>
</tr>
<tr>
<td>z304-address-2</td>
<td>address-2</td>
</tr>
<tr>
<td>z304-address-3</td>
<td>address-3</td>
</tr>
<tr>
<td>z304-email-address</td>
<td>email-address</td>
</tr>
<tr>
<td>z304-telephone</td>
<td>telephone</td>
</tr>
<tr>
<td>z304-zip</td>
<td>zip</td>
</tr>
<tr>
<td>z305-bor-status</td>
<td>bor-status</td>
</tr>
<tr>
<td>z305-expiry-date</td>
<td>expiry-date</td>
</tr>
</tbody>
</table>
DigiTool attributes

z312-birth-date = birth_date
z312-bor-id = bor_id
z312-id = id
z312-open-date = open_date
z312-source-id = id
z312-title = title
z312-update-date = update_date
z312_birth_date = birth_date
z312_bor_id = bor_id
z312_id = id
z312_open_date = open_date
z312_source_id = id
z312_title = title
z312_update_date = update_date
z312m_bor_dept_m = bor_dept_m
z312m_bor_group_m = bor_group_m
z312m_bor_tuples_m = bor_tuples_m
z312m_course_enrollment_m = course_enrollment_m

MetaLib attributes

Predefined attribute mapping of PDS to MetaLib applications.

Tags on the right side are PDS normalized tags.

Note - all attr with "-" can be written with "_" as well.

The user internal name (secondary key of the institute)
Field type > 20 characters

bor_id = id
source_id = id
id = id
z312-source-id = id
z312_source_id = id

User institution

institute = institute
z312-institute = institute
z312_institute = institute

User name to be displayed in the \V & \M interface
Field type > 200 characters

user_name = name
name = name
z312-name = name
z312_name = name

User title (name prefix)
Field type > 10 characters

user_title = title
Groups defined in the MetaLib application
Field type > 30 characters

user_group = group
group = group
z312-group = group

Portal Name
- *Portal Name is a mandatory field in the Z32 record.
- *User's portal.
- *The Portal should be in uppercase.

Note
The Portal field value is taken from the first portal of the relevant institution in ./vir00/tab_institute if no parameter is found in the XML file.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>z312_portal_name</td>
<td>portal_name</td>
</tr>
<tr>
<td>portal_name</td>
<td>portal_name</td>
</tr>
</tbody>
</table>

Language to be used in the \V application.
Refer to ./dat01/www_m_eng/user-language-include
Field type > 3 characters
If no lang is send the default is English = "eng".

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>con_lng</td>
<td>con-lng</td>
</tr>
<tr>
<td>z312-con-lng</td>
<td>con-lng</td>
</tr>
</tbody>
</table>

Resource status flag
Field type -> 1 character
Valid Values:
- A = active
- T = active + testing (IRDS authentication)

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>resource-status</td>
<td>resource-status</td>
</tr>
<tr>
<td>z312-resource-status</td>
<td>resource-status</td>
</tr>
</tbody>
</table>

User's academic status.
Refer to: ./dat01/www_m_eng/academic-status-include
Field type > 30 characters
Valid Values:
Undergraduate
Graduate

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_academic_status</td>
<td>academic-status</td>
</tr>
<tr>
<td>academic_status</td>
<td>academic-status</td>
</tr>
<tr>
<td>z312-academic-status</td>
<td>academic-status</td>
</tr>
</tbody>
</table>

Field type > 8 characters in YYYYMMDD format

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>expiry_date</td>
<td>expiry-date</td>
</tr>
</tbody>
</table>
User address details
Field type > 50 characters for each address
user_address_1 = address_0
z312_address_0 = address_0
z312-address-0 = address_0

When address-0 is the incoming attribute, the followed addresses are shifted accordingly
user_address_2 = address_1
z312_address_1 = address_1
z312-address-1 = address_1
user_address_3 = address_2
z312_address_2 = address_2
z312-address-2 = address_2
user_address_4 = address_3
z312_address_3 = address_3
z312-address-3 = address_3
z312_address_4 = address_4
z312-address-4 = address_4
z312_address_5 = address_5
z312-address-5 = address_5

User address zip code
Field type > 9 characters
user_zip = zip
zip = zip
z312-zip = zip
z312_zip = zip

User’s e-mail address
Field type > 60 characters
user_email_address = email_address
email_address = email_address
z312_email_address = email_address

User’s two telephone numbers
Field type -> 30 characters
user_telephone_1 = telephone-1
z312_telephone_1 = telephone-1
user_telephone_2 = telephone-2
z312_telephone-2 = telephone-2
z312_telephone_2 = telephone-2

END OF METALIB ATTRIBUTES
Appendix C - apachectl.shib

#!/bin/sh
#
# Apache control script designed to allow an easy command line
# interface to controlling Apache. Written by Marc Slemko, 1997/08/23
#
# The exit codes returned are:
# 0 - operation completed successfully
# 1 -
# 2 - usage error
# 3 - httpd could not be started
# 4 - httpd could not be stopped
# 5 - httpd could not be started during a restart
# 6 - httpd could not be restarted during a restart
# 7 - httpd could not be restarted during a graceful restart
# 8 - configuration syntax error
#
# When multiple arguments are given, only the error from the _last_
# one is reported. Run "apachectl help" for usage info
#
# START CONFIGURATION SECTION

PIDFILE=/exlibris/metalib/m3_1/apache/logs/httpd.pid
SHIBDPIDFILE=SHARPIDFILE=/exlibris/metalib/m3_1/apache/logs/shibdshar.pid

HTTPD="/exlibris/metalib/m3_1/product/bin/httpd -d /exlibris/metalib/m3_1/apache -DSSL"

LYNX="lynx -dump"

STATUSURL="http://localhost/server-status"

END CONFIGURATION SECTION

ERROR=0
ARGV="$@
if [ "$ARGV" = "x" ] ; then
  ARGS="help"
fi

for ARG in $@ $ARGS
do

© Ex Libris Ltd., 2007  Page 71 of 76
Patron Directory Services version 1.3
Last Update: February 13, 2007
# check for pidfile
if [ -f $PIDFILE ] ; then
  PID=`cat $PIDFILE`
  if [ "x$PID" != "x" ] && kill -0 $PID 2>/dev/null ; then
    STATUS="httpd (pid $PID) running"
    RUNNING=1
  else
    STATUS="httpd (pid $PID?) not running"
    RUNNING=0
  fi
else
  STATUS="httpd (no pid file) not running"
  RUNNING=0
fi

if [ -f $SHIBDPIDFILESHARPIDFILE ] ; then
  SHIBDPIDSHARPID=`cat $SHIBDPIDFILESHARPIDFILE`
fi

case $ARG in
  start)
    if [ $RUNNING -eq 1 ]; then
      echo "$0 $ARG: httpd (pid $PID) already running"
      continue
    fi
    if $HTTPD ; then
      echo "$0 $ARG: httpd started"
    else
      echo "$0 $ARG: httpd could not be started"
      ERROR=3
    fi
    ;;
  stop)
    if [ $RUNNING -eq 0 ]; then
      echo "$0 $ARG: $STATUS"
      continue
    fi
    if kill $PID ; then
      echo "$0 $ARG: httpd stopped"
    else
      echo "$0 $ARG: httpd could not be stopped"
      ERROR=4
    fi
    if [ "x$SHIBDPIDSHARPID" != "x" ] ; then
      if kill $SHARPID ; then
        echo "$0 $ARG: shibdshar stopped"
      else
        echo "$0 $ARG: shibdshar could not be stopped"
      fi
      rm $SHARPIDFILE
    fi
    ;;
  restart)
    if [ $RUNNING -eq 0 ]; then
      echo "$0 $ARG: httpd not running, trying to start"
      if $HTTPD ; then
        echo "$0 $ARG: httpd started"
      else
        echo "$0 $ARG: httpd could not be started"
        ERROR=5
      fi
else
  if $HTTPD -t >/dev/null 2>&1; then
    if kill -HUP $PID ; then
      echo "$0 $ARG: httpd restarted"
    else
      echo "$0 $ARG: httpd could not be restarted"
      ERROR=6
    fi
  else
    echo "$0 $ARG: configuration broken, ignoring restart"
    echo "$0 $ARG: (run 'apachectl configtest' for details)"
    ERROR=6
  fi
fi

graceful)
  if [ $RUNNING -eq 0 ]; then
    echo "$0 $ARG: httpd not running, trying to start"
    if $HTTPD ; then
      echo "$0 $ARG: httpd started"
    else
      echo "$0 $ARG: httpd could not be started"
      ERROR=5
    fi
  else
    if $HTTPD -t >/dev/null 2>&1; then
      if kill -USR1 $PID ; then
        echo "$0 $ARG: httpd gracefully restarted"
      else
        echo "$0 $ARG: httpd could not be restarted"
        ERROR=7
      fi
    else
      echo "$0 $ARG: configuration broken, ignoring restart"
      echo "$0 $ARG: (run 'apachectl configtest' for details)"
      ERROR=7
    fi
  fi

status)
$LYNX $STATUSURL | awk '/process$/ { print; exit } { print }' 

fullstatus)
$LYNX $STATUSURL

configtest)
  if $HTTPD -t; then
    :;
  else
    ERROR=8
  fi

")
  echo "usage: $0 (start|stop|restart|fullstatus|status|graceful|configtest|help)"
cat <<EOF
start    - start httpd
stop     - stop httpd
restart  - restart httpd if running by sending a SIGHUP or start if not running
fullstatus - dump a full status screen; requires lynx and mod_status
enabled
status     - dump a short status screen; requires lynx and mod_status
enabled
graceful   - do a graceful restart by sending a SIGUSR1 or start if not
running
configtest - do a configuration syntax test
help       - this screen

EOF
    ERROR=2
  ;;
esac
done
exit $ERROR

## ====================================================================
## The Apache Software License, Version 1.1
##
## Copyright (c) 2000-2002 The Apache Software Foundation. All rights
## reserved.
##
## Redistribution and use in source and binary forms, with or without
## modification, are permitted provided that the following conditions
## are met:
##
## 1. Redistributions of source code must retain the above copyright
## notice, this list of conditions and the following disclaimer.
## 2. Redistributions in binary form must reproduce the above copyright
## notice, this list of conditions and the following disclaimer in
## the documentation and/or other materials provided with the
## distribution.
## 3. The end-user documentation included with the redistribution,
## if any, must include the following acknowledgment:
##   "This product includes software developed by the
##   Apache Software Foundation (http://www.apache.org/)."
## Alternately, this acknowledgment may appear in the software
## itself,
## if and wherever such third-party acknowledgments normally appear.
## 4. The names "Apache" and "Apache Software Foundation" must
## not be used to endorse or promote products derived from this
## software without prior written permission. For written
## permission, please contact apache@apache.org.
## 5. Products derived from this software may not be called "Apache",
## nor may "Apache" appear in their name, without prior written
## permission of the Apache Software Foundation.
##
## THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
## WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
## OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
## DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
## ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
## SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
## LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
Appendix D - apachetcl_auto.shib

#!/usr/local/bin/tcsh
source ${metalib_dev}/metalib_conf/metalib_start
/usr/local/shibboleth/bin/shidbshar -f >& /dev/null &
echo $! > ${httpd_root}/logs/shibdshar.pid
${httpd_bin}/httpd -d $httpd_root -DSSL