Developer's Handbook

Search Server and AgentXpert

for PublishingXpert

Version 2.01

The Team:
Engineering: Jerry Ko, Tim Walters, Jason Lim, Dong Zhang, Jenhwa Tan, Nick Huang, James Wang, Ryh-Wei Yeh, Rick Henderson, Ravi Devesetti, Chi Nguyen, Juan Jose Mata, Roscoe Shih, Trang Le Tuyet, Jason Li, Aln Reddy, Prasad Yendluri, Ravi Kumar, Eric Chen, Xin Chen, Kwang-Chi Liang
Marketing: Mike Micucci, Kyung Kim
Publications: Meera Holla, Laura Zupko, Meredith Tanner, Lynn Gold, Gary McCue, Bill Branca, Mike Zampiceni
Quality Assurance: Jerome Fong, Gayatri Rimola, Beverly Chan, Carl Le, Wenge Song, June Ann Martin, Chao-Sheng Yang
Technical Support: Heather Trumbower, Harry Harper, Jeff Jones
Search Server and AgentXpert Mascot: Fuzzball

Version 2.01
©Netscape Communications Corporation 1997
All Rights Reserved
Printed in USA
97 10 9 8 7 6 5 4 3 2 1
Netscape Communications Corporation 501 East Middlefield Road, Mountain View, CA 94043

Contents

About This Book

This *Handbook* describes the operations of the Search and AgentXpert Servers, Version 2.01 from Netscape Communications Corporation.

This preface discusses the intended audience, the organization, and provides a listing of typographic conventions used in this document. If you spend a few minutes looking through this preface before reading the rest of the *Handbook*, you will be able to utilize the *Handbook* more effectively.

Before You Begin

This handbook is written with the assumption that you understand the basics of a relational database system and that you understand the operating system on which you are running this software.

You do not need to be an expert on the Internet, the World Wide Web, or HTML, but you will find it helps to know the basics of these technologies.

A knowledge of CORBA and IIOP is helpful in understanding the underlying infrastructure, but it is not required for developing agents. Knowledge of C++ is sufficient for writing your own agents.

Audience

This handbook is written for the site administrator maintaining a Web site system with a range of online services. It also addresses developers who would like to customize the software.

Organization

This *Handbook* has two parts. It is divided into six chapters and two appendixes:

- Chapter 1, “Introducing Search Server and AgentXpert,” describes how the Search and AgentXpert server modules fit into the PublishingXpert system.

Part 1: The Search Server

- Chapter 2, “Introducing the Search Server,” provides a brief introduction to the architecture and components of the search server.
- Chapter 3, “The Search Server API,” describes the API function calls for the search server Software Development Kit (SDK).
- Chapter 4, “Document Loader,” describes the functionality of the document loader command-line utility `psloader`.

Part 2: The AgentXpert Framework

- Chapter 5, “Introducing AgentXpert,” provides a brief introduction to the architecture, operating requirements, and components of AgentXpert. It also explains how to modify the dispatcher and command server configuration files.
- Chapter 6, “Building an Agent,” explains how to build an agent, describing the four enclosed agents and how they work. It also explains how to start and run a dispatcher and command server.

Part 3: Appendixes

- Appendix A, “Search Server Example,” contains a sample search server application.
- Appendix B, “PublishingXpert SDK Makefile,” describes the file `Makefile.basic` and how to configure it and build your software.

Conventions

Typographic conventions are used throughout this handbook to help you recognize special terms and instructions. These conventions are summarized in the following table.

Convention	Meaning	Example or Context
boldface	items on the screen	Click the **Submit** button to save your changes.
	names of keys	Press **Enter** to clear the message.
boldface numbered steps	higher level descriptions of tasks you perform (more detailed instructions follow)	3. **Enter the group information.** Enter the name in the **Group Name** field, and a short description in the **Description** field.
italics	key words, such as terms that are defined in the text	The notices posted on an electronic BBS are called *articles*.
	names of books	For more information, refer to the *Getting Started with Netscape Navigator* manual.
`courier font`	command line input or output	Enter the following command: `ls *.mle`
	text file content, such as HTML templates and configuration files	`<TITLE>Password Check</TITLE>` `<IMG SRC="/ui/icons/hd_svcs.gif">`
	code samples	`const char* getName() const`

Chapter

1

Introducing Search Server and AgentXpert

This chapter presents an overview of the Search Server and AgentXpert Server foundation layer in the following sections:

- Architectural Overview
- Search Server and AgentXpert Framework
- Command Line Utilities and API
- Configuring Your Environment

Architectural Overview

The problem of skimming through large amounts of data and working with them takes enormous amounts of time. Consider the problem of sending daily news reports to users who meet a particular criteria when the user database contains 10,000,000 IDs. If accessing each row and determining whether that user meets the set criteria takes 10 milliseconds, the time it takes to process 10 million rows is 27.7 hours. In this case some users would get their reports one day late. Furthermore, this delay would accumulate over time. One solution for this could be to divide the user space and span multiple concurrent process to do the job. Although this is a well-known technique, it requires in-depth knowledge and experience in developing a solution.

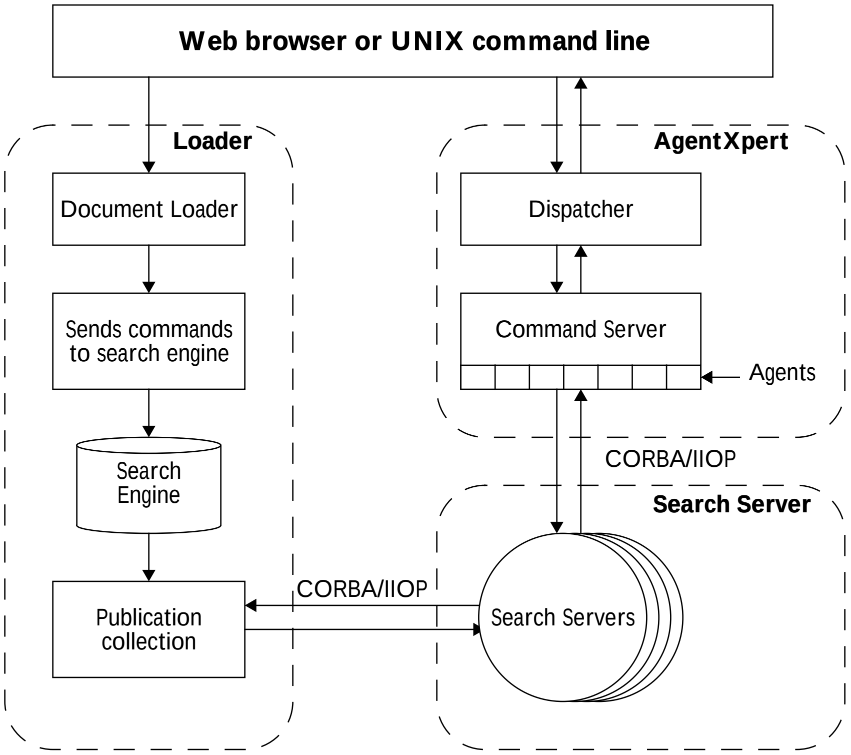
To address this problem, the Search Server and AgentXpert framework has been developed whereby any given computational task can be divided into a specified number of concurrent processes to exploit the inherent parallelism. This framework provides a dispatcher and server mechanism that lets users control the granularity of the size of the commands each server (thread) works on. Optimal performance is achieved by distributing these servers on different machines.

The AgentXpert framework and search server constitute the Information Retrieval component of the foundation layer underlying PublishingXpert.

The search server framework is based on the CORBA architecture and also has a distributed environment. Using the search server, you can distribute a search amongst multiple search servers running simultaneously on the same host or on different hosts.

The following figure illustrates the Search Server and AgentXpert server architecture.

Figure 1.1 AgentXpert architecture schematic representation



The following steps describe the interaction between the document loader, the search server, and the AgentXpert framework:

1. The user runs the document loader (`psloader`) to load a publication collection from a search engine into the search server and index it.
2. The administrator must start one or more command servers (`acpsacmdsrv`).
3. The user runs the dispatcher (`acpsadisp`) to instruct the command server to create an agent.
4. In the example in Figure 1.1, the command server starts an agent that accesses the search servers. The search servers return the results of the query to the agent, which in turn sends an Acknowledgment to the dispatcher and exits.

Search Server and AgentXpert Framework

This section briefly describes the Search Server and AgentXpert framework. For additional information on these entities, refer to Chapter 2, “Introducing the Search Server,” and Chapter 5, “Introducing AgentXpert,” in this handbook.

The Search Server

The *search server* is a feature that provides text search capabilities in PublishingXpert. Its architecture allows multiple persistent search servers, where each server can perform search requests for multiple clients. The command server receives requests from a PublishingXpert Server-side JavaScript (SSJS) client or AgentXpert’s search agent (`GenerateResults`).

The *document loader* is also part of the search server framework. The document loader uses a third-party search engine, such as Verity or PLS, and indexes a publication collection for the search servers.

AgentXpert

AgentXpert is a comprehensive CORBA-based agent server that provides a scalable framework for creation, scheduling, recovery, and administration of customized agents. Agents bundled with the AgentXpert framework include a personalized search agent (`GenerateResults`), an HTML formatter (`FormatResult`), a sample agent to insert an advertisement into a formatted result (`Advertiser`), and one that e-mails results to a list of user IDs (`Mailer`).

The AgentXpert framework consists of a dispatcher server, a command server, and several agents. The dispatcher is responsible for sending events and messages to distributed command servers. Each command server dispatches the event and its messages to one of the agents within the command server. Using a search profile for each user and an agent scheduler on the administration end, the agents shipped with AgentXpert provide a complete personalized search application. You can customize or extend this to integrate with other systems. Using the AgentXpert framework, you can also develop new agent-based applications.

Command Line Utilities and API

The Search Server and AgentXpert framework uses a combination of command line utilities and an API to gather and distribute data.

Command Line Utilities

The Search Server and AgentXpert command line utilities let you bypass the administrative forms and perform repetitive tasks.

With AgentXpert command line utilities you can:

- start a command server
- dispatch commands to various command servers
- retrieve documents from search servers
- format data and send e-mail through mail servers

Search Server API

The search server has its own API that interfaces with the underlying search engine. Using the API in your code makes program development faster and easier. In addition, your code can be independent of the actual search engine. The search server allows client programs to initiate searches, configure the search context in the server, manage document sets, and retrieve search results.

Note Netscape recommends you always use the API when writing code that accesses the database. This way your code is not affected if the database schema changes.

Configuring Your Environment

The following files and products are necessary for using the Search Server and AgentXpert framework.

File Locations

The default installation directories for files associated with the APIs are listed in Table 1.1. The `PS_HOME` environment variable is the Netscape Applications home directory (`/.../ns-apps`).

Table 1.1 Files associated with Search Server and AgentXpert

Files For	Default Location
Command Line Utilities	`$PS_HOME/admin/bin`
C++ Include Files	`$PS_HOME/sdk/$CMODULE/include`
C++ Lib Files	`$PS_HOME/lib`
C++ Makefiles	`$PS_HOME/sdk/$CMODULE/src/Makefile`

Software Versions

The following versions of the compiler were used in developing this release of the Search Server and AgentXpert Software Development Kit (SDK). If you are using earlier versions, you should upgrade your software before working with the APIs.

Table 1.2 Software requirements for Search Server and AgentXpert

Tools and Libraries	Version
Sun C++ compiler	4.1
g++ library `libgcc`.a (included in `$PS_HOME/lib`)	2.6.3

Table 1.2 Software requirements for Search Server and AgentXpert (Continued)

Tools and Libraries	Version
perl	5.000
gmake (only needed with Netscape make files)	3.74

Note RogueWave version S-220 `dbtools.h++` and `tools.h++` header files are used with the source code for Search Server and AgentXpert. If you want to use RogueWave `dbtools.h++` and `tools.h++` classes in your development, you must obtain a development license from RogueWave.

1 The Search Server

- **Introducing the Search Server**
- **The Search Server API**
- **Document Loader**

Chapter

2

Introducing the Search Server

This chapter introduces you to the search server, which is a part of the Information Retrieval foundation layer.

This chapter contains the following sections:

- Search Server Overview
- Using the Search Server
- Search Server API Call Sequence
- Search Server Configuration

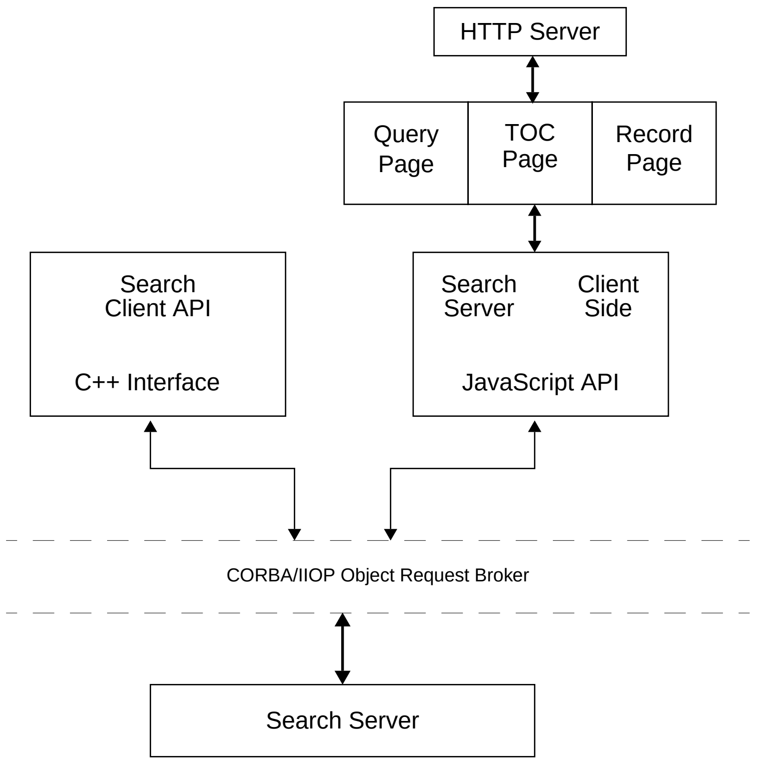
Search Server Overview

The Search Server is based on the CORBA architecture. It allows multiple search servers to run on the same host or on different hosts.

Search clients can communicate with search server through the Search Server Software Development Kit (SDK), which is a set of C++ APIs, as well as through Netscape Server Side JavaScript (SSJS).

The following figure gives an overview of the search server interface.

Figure 2.1 Search server interface



The search client API and the SSJS API communicate with the search server through the CORBA/IIOP object request broker.

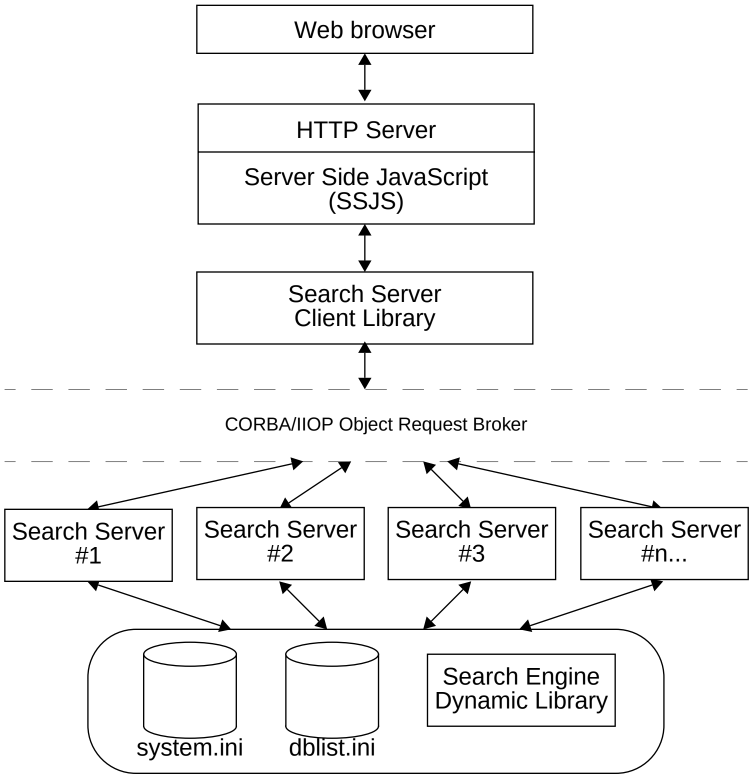
For details on the query page, the Table of Contents (TOC) page, and the record page, see the *PublishingXpert Administrator Handbook*.

Using the Search Server

Users can run multiple search servers across multiple machines. The CORBA search server listens to the CORBA agent at port `OSAGENT_PORT` and responds to all requests from CORBA clients. A CORBA agents dispatches the incoming requests in sequential order.

Figure 2.2 shows the flow of a search initiated by a user using a web browser, such as Netscape Communicator.

Figure 2.2 User-initiated search


Web browser
HTTP Server
Server Side JavaScript (SSJS)
Search Server Client Library
CORBA/IIOP Object Request Broker
Search Server #1
Search Server #2
Search Server #3
Search Server #n...
system.ini
dblist.ini
Search Engine Dynamic Library

The following steps are executed when a user calls the search server:

1. Using a web browser, the user initiates a search for text, such as ‘Netscape’.

2. The search request is passed by the browser to the HTTP server, which in turn uses a Server Side JavaScript (SSJS) interface to translate the search string and search parameters to structures understood by the search server.
3. The client-side CORBA interface passes the search parameters to a server-side CORBA interface that distributes the search parameters to various search servers.
4. The various search servers all read the same `system.ini` and `dblist.ini` files to interface with the search engine dynamic library.
5. The search engine dynamic library sends a search result set containing the results of the search to the search servers.
6. The CORBA interface sends the search result set from the search engines to the SSJS interface.
7. The SSJS interface processes the information in the search result set and passes it to the HTTP server.
8. The HTTP server sends the search result set to the browser window, where the results are presented. For the available options, see Chapter 3, "The Search Server API."

Search Server API Call Sequence

The following steps describe a sample call sequence for submitting a query to a CORBA-based search server and retrieving the result.

1. Create a `PSACSearch` object to initialize an ORB handle.
2. Call the `PSACSearch` object method `GetCollection()` or `GetLangCollection()` to obtain all name and path pairs from the search server. The name and path pairs are used to form the `SearchArg` structure. For more information on the name and path pairs, see "PSACCollSet" on page 22 in Chapter 3, "The Search Server API."
3. Create a `PSACSearchArg` object to package an argument to be sent to the search server. For details about the `PSACSearchArg` class, see Chapter 3, "The Search Server API."

4. Call the `PSACSearch` member function `Search()` to pass the formed search argument to the search server and place a search result into a `PSACSearchRes` object.

5. Create a `PSACSearchRes` object to process the results of the search. For a detailed description of `PSACSearchRes`, see Chapter 3, "The Search Server API."

Search Server Configuration

In a search session, the client establishes a connection with one or more servers using data from the `system.ini` file. The entries in the `system.ini` file provide information about the search engine library, the location of the `dblist.ini` file, and tracing information.

The following parameters are in the `system.ini` file. You, as the System Administrator, have to provide the values for these parameters:

Parameter	Section	Description
`NS-dblist-ini`	`NS-search`	location of `dblist.ini` file
`NS-search-eng`	`NS-search`	location of search engine dynamic library
`NS-dbgroup-ini`	`NS-search`	location of `dbgroup.ini` file
`trace-work-dir`	`search_trace`	location of the search server trace file
`trace-file-name`	`search_trace`	name of search server trace file
`trace-level`	`search_trace`	search trace level
`trace-components`	`search_trace`	tracing component (should be `searchserv`)

The section `NS-search-engine-specific` in the `system.ini` file contains search engine-specific information and is passed to the underlying search engine during search server startup. For more information about the `system.ini` file, see the *PublishingXpert Administrator Handbook*.

Netscape supports either Verity or PLS as the default search engine. When configuring your `system.ini` file to use Verity as the default search engine, you must include the line:

```
Vrty-lib-path = $PSHOME/lib/search/verity
```

If your default search engine is PLS, you must include the following line in the `system.ini` file:

```
pls-install-dir = $PSHOME/lib/search/pls
```

Chapter 3

The Search Server API

This chapter documents the API classes, structures, and function calls for the search server.

PSACSearch

The PSACSearch class is a base class that establishes a CORBA connection to a CORBA search server. The PSACSearch class initializes an ORB handle.

To query a search server, you must first create a PSACSearch object. You can retrieve publication collection information from a remote search server using the PSACSearch object's GetCollection() and GetLangCollection() methods of this class. You can also use this class to query or restart the remote search server.

Methods The following table shows the methods you can use to manipulate PSACSearch objects:

Constructors and Destructors

`PSACSearch()`	Creates a PSACSearch object.
`~PSACSearch()`	Deletes a PSACSearch object.

Manipulating search object parameters

`GetCollection()`	Obtains a set of opened publications from the search server.
`GetLangCollection()`	Obtains a language-specific set of publications from the search server.
`Search()`	Searches the opened data for specified criteria and obtains the results of the search.
`Restart()`	Restarts the search server.
`Bad()`	Obtains an error number.
`Clear()`	Clears the error number buffer.

Class Definition

Interface	`searchsdk.h`
Superclasses	None
Subclasses	None
Friend Classes	None
Syntax	`class PSACSearch`

Data Types and Constants

PSACCollSet

Defines a search collection item.

Syntax

```
struct PSACCollSet
{
  int   count;
  char  **collname;
  char  **collpath;
}
```

`count`	The number of open data collections.
`collname`	The collection name array.
`collpath`	The collection path array.

Constructors and Destructors

PSACSearch( )

Creates a `PSACSearch` object.

Syntax `PSACsearch();`

~PSACSearch( )

Deletes a `PSACSearch` object.

Syntax `~PSACsearch();`

PSASearch Methods

GetCollection( )

Retrieves a list of opened publications from the search server.

Syntax `PSACSearch& GetCollection(PSACCollSet *collset);`

Parameters The function has the following parameters:

`PSACCollSet`	The collection of data being searched.

Returns A list of opened collections from the search server.

Discussion If successful, the function returns a list of opened collections from the search server; if unsuccessful, the function sets the error code to -1 to indicate the operation failed or 0 to indicate the operation was successful.

Example

```
// Get server collection list
if (acsearch->GetCollection(collset).Bad() == -1) {
        cout << "GetCollection failed" << endl;
        return(-1);
}

    for (i=0; i<collset.count; i++) {
      cout << "GetCollection" << i << " " << collset.collname[i] << " "
                             << collset.collpath[i] << endl;
}
```

See Also The `PSACCollSet` structure on page 22.

GetLangCollection( )

Retrieves a language-specific set of publications from the search server.

Syntax

```
PSACSearch& GetLangCollection(const char *langparm,
                              PSACCollSet *collset);
```

Parameters The function has the following parameters:

`langparm`	The language in which to return the search data.
`PSACCollSet`	The collection of data being searched.

Returns A list of opened collections from the search server in the specified language.

Discussion If successful, the `GetLangCollection()` function returns a collection of data in a language specified by `langparm`, a string of *`language_name;characterset`* such as `english;8859`. If unsuccessful, the function sets the error code to -1 to indicate the operation failed or 0 to indicate the operation was successful.

Example

```
// Get English-only collection list}
if (acsearch->GetLangCollection(“english;8859”,collset).Bad() == -1) {
        cout << "GetLangCollection failed" << endl;
        return(-1);
}
```

See Also The `GetCollection( )` method on page 23.

Search( )

Searches the opened data for a specified parameter and assigns the results of the search to a `PSASearchRes` object.

Syntax `PSACSearch& Search(PSACSearchArg &srcharg, PSACSearchRes *res)`

Returns The results of a search.

Discussion The `Search()` function passes the `PSACSearch` argument to a remote search server and passes the result back into a `PSACSearchRes` object. If unsuccessful, the function sets the error code to -1 to indicate the operation failed or 0 to indicate the operation was successful.

Example

```
if ((acsearch->Search(acsearcharg, acsearchres)).Bad() == -1) {
        cout << "Search failed." << endl;
        return(-1);
}
```

See Also The `PSACSearchArg` class on page 27. The `PSACSearchRes` class on page 40.

Restart( )

Restarts the search server.

Syntax `PSACSearch& SClient_Restart(ACSearch_var ACSearch_object);`

Returns An integer: 0 if restart is successful, 1 if restart is unsuccessful.

Discussion The `Restart()` function rereads all configuration files and restarts all connections made by the search server.

Example

```
if ((acsearch->Restart()).Bad() == -1) {
        cout << "Restart failed." << endl;
        return(-1);
}
```

Bad( )

Returns an error number from the search server.

Syntax `PSACSearch& SClient_Restart(ACSearch_var ACSearch_object);`

Returns An integer: 0 if restart is successful, 1 if restart is unsuccessful.

Example

```
if ((acsearch->Restart()).Bad() == -1) {
        cout << "Restart failed." << endl;
        return(-1);
}
```

Clear( )

Clears the search server error number buffer.

Syntax `PSACSearch& SClient_Restart(ACSearch_var ACSearch_object);`

Returns The integer 0.

Example

```
acsearch->Clear();
```

PSACSearchArg

The `PSACSearchArg` class passes a search argument to a CORBA search server.

Use this class to prepare all search criteria for the `PSACSearch` object method `Search()`, which passes the search argument structure to the search server and fills a `PSACSearchRes` object with a structure containing the results of the search.

Methods The following table shows the methods you can use to manipulate `PSACSearchArg` objects:

Constructor and destructor

`PSACSearchArg()`	Creates a `PSACSearchArg` object.
`~PSACSearchArg()`	Destroys a `PSACSearchArg` object.

Accessor functions

`AddCollection()`	Adds a collection of publications to the specified search.
`AddTableFields()`	Sets the attribute list which can retrieve the attribute result from the search result set later.
`CollectionCount()`	Gets the number of collection sets
`FieldCount()`	Gets the attribute count.
`FieldTableAttr()`	Gets a field table.
`HLon()`	Toggles highlighting of the search string.
`HLbegin()`	Gets or sets the value of the string marking the beginning of highlighted data.
`HLend()`	Gets or sets the value of the string marking the end of highlighted data.
`LargestSet()`	Gets or sets the maximum number of documents the search engine can return.
`MaxAttributeSize()`	Sets the maximum attribute size.

`MaxRecords()`	Gets or sets the maximum result size that can be passed back.
`Query()`	Gets or sets the query string.
`QueryType()`	Gets or sets the query parser.
`SearchBase()`	Gets or sets the search result base.
`SortBy()`	Gets or sets a field containing a keyword and whether to sort in ascending or descending order.
`SummaryType()`	Gets or sets the summary type.

Class Definition

Interface	`searchsdkarg.h`
Superclasses	`PSACSearch`
Subclasses	None
Friend Classes	None
Syntax	`class PSACSearchArg`

Constructors and Destructors

PSACSearchArg( )

Creates a `PSACSearchArg` object.

Syntax `PSACSearchArg();`

~PSACSearchArg( )

Destroys a `PSACSearchArg` object.

Syntax

```
~PSACsearchArg();
```

Discussion The `~PSACSearchArg` function frees up all resources.

PSASearchArg Methods

AddCollection( )

Adds a publication collection to the specified search.

Syntax

```
PSACSearchArg& AddCollection(const char *collname,
                                   const char *collpath);
```

Parameters The function has the following parameters:

`collname`	A pointer to a string that specifies the name of the collection.
`collpath`	A pointer to a string that specifies the search collection path.

Returns A reference to this object.

Discussion The referenced collection of publications must already be opened by the search server. Use the `GetCollection()` method to open the collection set.

Example

```
PSACSearchArg acsearcharg;
PSACSearch::GetCollection(PSACCollSet *collset);
for (i=0; i<collset.count; i++) {
if ((acsearcharg.AddCollection(collset.collname[i],
      collset.collpath[i])).Bad() == -1) {
         cout << "AddCollection Failed: collname  "
              << collset.collname[i] << endl;
         }
}
```

See Also The `GetCollection( )` method on page 23.

AddTableFields( )

Sets the attribute list to be retrieved with the attribute result from the search result set.

Syntax `PSACSearchArg& AddTableFields(int cnt, const char *attlist);`

Parameters The function has the following parameters:

`cnt`	An integer that specifies an attribute number.
`attlist`	A constant string that specifies an attribute list.

Returns A reference to this object.

Discussion The attributes list (`attlist`) is a set of attributes separated by semicolons. For example:

```
Title;Subject;Date
```

Example

```
PSACSearchArg acsearcharg;
acsearcharg.AddTableFields(3,”Title;Subject;Date”);
```

CollectionCount( )

Obtains the number of collection sets.

Syntax `int CollectionCount();`

Returns A reference to this object.

Discussion This function obtains the number of collection sets for this `PSACSearchArg` object.

Example

```
PSACSearchArg acsearcharg;
int coll_count = acsearcharg.CollectionCount();
```

FieldCount( )

Obtains the number of field attributes.

Syntax `int FieldCount() const;`

Returns An integer value containing the number of field attributes.

Discussion This returns the number of fields the user has set in the `PSACSearchArg` object.

Example

```
PSACSearchArg acsearcharg;
int field_count = acsearcharg.FieldCount();
```

FieldTableAttr( )

Gets a field table.

Syntax `const char * FieldTableAttr() const;`

Returns A string containing a semicolon-separated attribute string.

Discussion A field table attribute is a semicolon-separated string the user has set in the `PSACSearchArg` object.

Example

```
PSACSearchArg acsearcharg;
char *field = acsearcharg.FieldTableAttr();
```

See Also The `AddTableFields()` method on page 30.

HLon( )

Turns highlighting of the search string on or off.

Syntax `PSACSearchArg& HLon(int highlighton);`

Parameters The function has the following parameters:

`highlighton`	An integer that specifies whether highlighting of a search string is on or off.

Returns A reference to this object.

Discussion The `HLon()` function sets one of the following values:

`1`	Turns on the highlight function.
`0`	Turns off the highlight function.

When highlighting is on, the `HLon()` method has the search server highlight a search string when it is found in a document. For example, with highlighting or search result string might appear:

This is a **book**.

where 'book' is the search string. With highlighting off, your search result string would appear:

```
This is a book.
```

The kind of highlighting used depends on the values of `HLbegin` and `HLend`.

Example

```
PSACSearchArg acsearcharg;
acsearcharg.HLon(1);
```

See Also The `HLbegin()` method on page 32. The `HLend()` method on page 33.

HLbegin( )

Gets or sets the contents of a string that specifies the HTML command to use when beginning highlighting.

Syntax

```
const char * HLbegin() const;
PSACSearchArg &HLbegin(const char * highlightbegin);
```

Parameters The function has the following parameters:

`highlightbegin`	The character string to indicate the beginning of the highlighted data.

Returns A pointer to a character string that contains the HTML command or a reference to this object.

Discussion Use the first syntax form to obtain the highlight begin string. Use the second form to set the value of this string. When highlighting is turned on, the search engine inserts the characters in this string before the text being highlighted.

Example The following example shows the highlight begin string being set to `<B>`, which turns on bold text in HTML:

```
PSACSearchArg acsearcharg;
acsearcharg.HLbegin(“<B>”);
```

The following example obtains the highlight begin string:

```
char *highlight_begin = acsearcharg.HLbegin();
```

See Also The `HLon()` method on page 31. The `HLend()` method on page 33.

HLend( )

Gets or sets the contents of a string that specifies the HTML command to use when ending highlighting.

Syntax

```
const char * HLend() const;
PSACSearchArg &HLend(const char * highlightend)
```

Parameters The function has the following parameters:

`highlightend`	The end of the highlighted data.

Returns A pointer to a character string that contains the HTML command or a reference to this object.

Discussion Use the first syntax form to obtain the highlight end string. Use the second form to set the value of this string. When highlighting is turned on, the search engine appends the characters in this string to the text being highlighted.

Example The following example shows the highlight begin string being set to `</B>`, which turns off bold text in HTML:

```
PSACSearchArg acsearcharg;
acsearcharg.HLend(“</B>”);
```

The following example obtains the highlight end string:

```
char *highlight_end = acsearcharg.HLend();
```

See Also The `HLon()` method on page 31. The `HLbegin()` method on page 32.

LargestSet( )

Gets or sets the maximum number of documents the search engine can return.

Syntax

```
int LargestSet() const;
PSACSearchArg& MaxAttributeSize(int LargestSet);
```

Parameters The function has the following parameters:

`LargestSet`	An integer that specifies the largest set size.

Returns An integer containing the maximum number of documents or a reference to this object.

Discussion Use the first form of the `LargestSet()` method to obtain the maximum number of documents the search engine can return. Use the second form to set the maximum number of documents. If you do not set the size, the default value of `LargestSet` is 200.

Example The following example sets the maximum number of documents the search engine can return:

```
PSACSearchArg acsearcharg;
acsearcharg.LargestSet(500);
```

The following example obtains the maximum number of documents the search engine can return:

```
int LargestSet = acsearcharg.LargestSet();
```

MaxAttributeSize( )

Sets the maximum attribute size.

Syntax `PSACSearchArg& MaxAttributeSize(int MaxAttributeSize);`

Parameters The function has the following parameters:

`MaxAttributeSize`	An integer that specifies the maximum attribute size.

Returns A reference to this object.

Discussion If you do not set the size, the default value is 255 characters.

Example

```
PSACSearchArg acsearcharg;
acsearcharg.MaxAttributeSize(511);
```

MaxRecords( )

Gets or sets the maximum number of records that can be passed back from the search server.

Syntax

```
int MaxRecords() const;
PSACSearchArg& MaxRecords(int MaxRecords);
```

Parameters The function has the following parameters:

`MaxRecords`	An integer that specifies the maximum number of records that can be passed back.

Returns An integer containing the maximum number of records or a reference to this object.

Discussion Use the first form of the `MaxRecords()` method to obtain the maximum number of records that can be passed back from the search server. Use the second form to set the maximum number of records. If you do not set the size, the default value is 20 records.

Example The following example sets the maximum number of records the search engine can return:

```
PSACSearchArg acsearcharg;
acsearcharg.MaxRecords(100);
```

The following example obtains the maximum number of records the search engine can return:

```
int MaxRecords = acsearcharg.MaxRecords();
```

Query( )

Gets or sets the query string.

Syntax

```
const char * Query() const;
PSACSearchArg& Query(const char * query);
```

Parameters The function has the following parameters:

`query`	A character string containing the value of the query string.

Returns A pointer to a character string that contains the value of the query string, or a reference to this object.

Discussion Use the first form of the `Query()` method to obtain the query string. Use the second form to set the value of the query string. Netscape supports the Verity and PLS search engines. For more information on what can be in a query string, see the documentation for your search engine.

Example The following example sets the query string:

```
PSACSearchArg acsearcharg;
char query[8192];
cout << "BEGIN SEARCH, enter your query:" << endl;
cin.getline(query, sizeof(query));
acsearcharg.Query(query);
```

The following example obtains the query string:

```
char *query_string = acsearcharg.Query();
```

QueryType( )

Gets or sets the query parser.

Syntax

```
int QueryType() const;
PSACSearchArg& QueryType(int queryParser);
```

Parameters The function has the following parameters:

queryParser	An integer containing a value indicating the query type

Returns An integer containing the value of the query parser, or a reference to this object.

Discussion Use the first form of the `QueryType()` method to obtain the query parser. Use the second form to set the value of the query parser. The following values of `queryParser` have the following internally-set meanings:

QP_BOOLEAN	Boolean (value is 1).
QP_FREETEXT	Free text (value is 3).
QP_PASSTHRU	Pass through (value is 4).

If any other value for `queryParser` is passed to `QueryType()`, a value of -1 is returned to indicate failure.

Example The following example sets the query parser:

```
PSACSearchArg acsearcharg;
cout << "ENTER QUERY PARSER (F)REETEXT, (B)OOLEAN" << endl;
char parserType[512];
cin.getline(parserType, sizeof(parserType));
    switch (*parserType) {
      case 'f':
      case 'F' : acsearcharg.QueryType(QP_FREETEXT);
      case 'b':
      case 'B' :  acsearcharg.QueryType(QP_BOOLEAN);
      default:    acsearcharg.QueryType(QP_BOOLEAN);
};
```

The following example obtains the query parser:

```
int query_type = acsearcharg.QueryType();
```

SearchBase( )

Gets or sets the base value from which results of the search are returned.

Syntax

```
int SearchBase() const;
PSACSearchArg& SearchBase(int SearchBase);
```

Parameters The function has the following parameters:

`SearchBase`	An integer containing the value of the search result base.

Returns An integer containing the base value from which results of the search are returned, or a reference to this object.

Discussion Use the first form of the `SearchBase()` method to obtain the base value from which results of the search are returned. Use the second form to set the base value. When the results of a search are found, rather than having them all returned at once, the search server can feed results to the browser in increments. As the information is delivered to the browser, the search base is increased by the increment in which the information is being delivered.

Example The following example sets the base value:

```
PSACSearchArg acsearcharg;
acsearcharg.SearchBase(20);
```

The following example obtains the base value:

```
int search_base = acsearcharg.SearchBase();
```

SortBy( )

Gets or sets a field containing a keyword and whether to sort in ascending or descending order.

Syntax

```
const char * SortBy() const;
PSACSearchArg& SortBy(const char * sortby);
```

Parameters The function has the following parameters:

`sortby`	A constant string that consists of an attribute and a keyword.

Returns A pointer to a character string that contains the value of a field containing a keyword and whether to sort in ascending or descending order, or a reference to this object.

Discussion Use the first form of the `SortBy()` method to set the name of the field on which the sort is based, such as `Author`, `Date`, or `Subject`, and whether to sort in ascending or descending order. Use the second form to obtain the value of the sortby field. The *sortby* string consists of an attribute, such as `Title`, `Date`, or `Subject`, a space, and either the keyword `asc` to indicate ascending order or `desc` to indicate descending order, as in ‘attribute order’.

Example The following example sets the sortby string:

```
PSACSearchArg acsearcharg;
acsearcharg.SortBy(“Title asc”);
```

The following example obtains the sortby string:

```
char *sort_by = acsearcharg.SortBy();
```

SummaryType( )

Gets or sets the summary type.

Syntax

```
int SummaryType() const;
PSACSearchArg& SummaryType(int summary);
```

Parameters The function has the following parameters:

`summary`	An integer that specifies the summary type

Returns An integer containing a value representing the summary type, or a reference to this object.

Discussion Use the first form of the `SummaryType()` method to obtain an integer representing the summary type. Use the second form to set the summary type. This method specifies which summary type to use to display search results. The available integer values for `SummaryType()`, along with their definitions, are in the `Summarytype-`*`N`* entries in the `system.ini` file. For example:

```
Summarytype-0 = 2
Summarytype-1 = KeyWORDS
SummaryType-2 = 1
SummaryType-3 = Indexed
SummaryType-4 = KeywordsDYNAMIC
SummaryType-5 = -10
SummaryType-6 = -20
SummaryType-7 = -50
```

Example The following example sets the summary type:

```
PSACSearchArg acsearcharg;
acsearcharg.SummaryType(1);
```

The following example obtains the summary type:

```
int summary_type = acsearcharg.SummaryType();
```

PSACSearchRes

The `PSACSearchRes` class returns the results of a search placed on a CORBA search server.

Methods The following table shows the methods you can use to manipulate `PSACSearchRes` objects::

Constructors and Destructors

`PSACSearchRes()`	Creates a `PSACSearchRes` object.
`~PSACSearchRes()`	Deletes a `PSACSearchRes` object.

Manipulating search result object parameters

`AltKey()`	Obtains the alternate key from the offset of the search result set.

`CollName()`	Obtains the name of the publication collection at the offset of the search result set.
`DocPath()`	Obtains the document path of the offset of the search result set.
`DocsFound()`	Obtains the number of documents found in this search.
`DocsSearched()`	Obtains the number of documents searched in this search.
`DocsReturned()`	Obtains the names of the documents returned in this search.
`DocScore()`	Obtains the document score for the offset of the returned documents.
`DocType()`	Obtains the document type for the returned documents.
`ErrorCode()`	Gets an error number.
`Field()`	Obtains the field value of a specified field for the offset of the returned documents.
`Bad()`	Returns an error number.
`Clear()`	Clears the error number buffer.

Class Definition

Interface	`searchsdkres.h`
Superclasses	`PSACSearch`
Subclasses	`None`
Friend Classes	`None`
Syntax	`class PSACSearchRes`

Constructors and Destructors

PSACSearchRes( )

Creates a PSACSearchRes object.

Syntax `PSACSearchRes()`

~PSACSearchRes( )

Destroys a PSACSearchRes object.

Syntax `~PSACSearchRes()`

PSASearchRes( ) Methods

AltKey( )

Gets the alternate key from the offset of the search result set.

Syntax `char * AltKey(int offset);`

Parameters The function has the following parameters:

`offset`	An integer containing the value of the offset in the publication set.

Returns The alternate key from the offset of the search result set.

Discussion The alternate key is search engine-specific information a user can pass down to a search engine.

Example

```
PSACSearchRes acsearchres;
for (j=0; j< acsearchres.DocsReturned(); j++) {
    cout << acsearchres.AltKey(j) << endl;
}
```

Bad( )

Returns an error number from the search server.

Syntax `int Bad();`

Returns An integer: 0 if restart is successful, 1 if restart is unsuccessful.

Example

```
PSACSearchRes acsearchres;

if ((acsearchres->DocPath()).Bad() == -1) {
        cout << "Could not find document." << endl;
        return(-1);
}
```

Clear( )

Clears the search server error number buffer.

Syntax `PSACSearchRes& SClient_Restart(ACSearch_var ACSearch_object);`

Returns The integer 0.

Example

```
acsearchres->Clear();
```

CollName( )

Gets the collection path of the offset of the search result set.

Syntax `char * CollName(int offset);`

Parameters The function has the following parameters:

`offset`	An integer containing the value of the offset in the publication set.

Returns The name of the publication collection at the offset of the search result set.

Discussion A search can return multiple results back from multiple collections. The `GetCollectionPath()` member function returns the collection paths for a specific search result.

Example

```
PSACSearchRes acsearchres;
for (j=0; j< acsearchres.DocsReturned(); j++) {
    cout << acsearchres.CollName(j) << endl;
}
```

DocPath( )

Gets the document path of the document located at the offset of the search result set.

Syntax

```
char * DocPath(int offset);
```

Parameters The function has the following parameters:

`offset`	An integer containing the value of the offset in the publication set.

Returns The document path of the document located at the offset of the search result set.

Example

```
PSACSearchRes acsearchres;
for (j=0; j< acsearchres.DocsReturned(); j++) {
cout << acsearchres.DocPath(j) << endl;
}
```

DocsFound( )

Gets the number of documents found from this search.

Syntax

```
int DocsFound() const;
```

Returns The number of documents found from this search.

Example

```
PSACSearchRes acsearchres;
int num_of_docs = acsearchres.DocsFound();
```

DocsSearched( )

Gets the number of documents searched in this search.

Syntax `int DocsSearched() const;`

Returns The number of documents searched in this search.

Example

```
PSACSearchRes acsearchres;
int num_of_docs = acsearchres.DocsSearched();
```

DocsReturned( )

Gets the number of documents returned from the set of documents found by this search.

Syntax `int DocsReturned() const;`

Returns The number of documents returned from the set of documents found by this search.

Example

```
PSACSearchRes acsearchres;
for (j=0; j< acsearchres.DocsReturned(); j++) {
cout << acsearchres.DocPath(j) << endl;
}
```

DocScore( )

Gets the document score for the offset of the returned documents.

Syntax `int DocScore(int offset);`

Parameters The function has the following parameters:

`offset`	An integer containing the value of the offset in the publication set.

Returns The document score for the offset of the returned documents.

Discussion The document score is a number that determines how relevant a document is to your search query.

Example

```
PSACSearchRes acsearchres;
for (j=0; j< acsearchres.DocsReturned(); j++) {
cout << acsearchres.DocScore(j) << endl;
}
```

DocType( )

Gets the document type for the document located at the offset of the returned documents.

Syntax `const char * DocType(int offset);`

Parameters The function has the following parameters:

`offset`	An integer containing the value of the offset in the publication set.

Returns The document type for the document located at the offset of the returned documents.

Example

```
PSACSearchRes acsearchres;
for (j=0; j< acsearchres.DocsReturned(); j++) {
cout << acsearchres.DocType(j) << endl;
}
```

ErrorCode( )

Gets an error number.

Syntax `int ErrorCode() const;`

Returns The error number.

Discussion The result -1 means failure; 0 means successful.

Example

```
PSACSearchRes acsearchres;
cout << “Error number: ” << acsearchres.ErrorCode() << endl;
```

Field( )

Gets the value of a specified field of the returned documents located at the offset.

Syntax `char * Field(int offset, const char *fieldname);`

Parameters The function has the following parameters:

`offset`	An integer containing the value of the offset in the publication set.
`fieldname`	A pointer to a character string that contains the name of the specified field.

Returns A pointer to a character string that contains the value of the specified field.

Example

```
PSACSearchRes acsearchres;
for (j=0; j< acsearchres.DocsReturned(); j++) {
char *fieldname = acsearchres.Field(j,”Author”) << endl;
}
```

See Also The `AddTableFields()` method on page 30.

Chapter

4

Document Loader

This chapter explains how to use the `psloader` command line utility to manipulate the document *loader*.

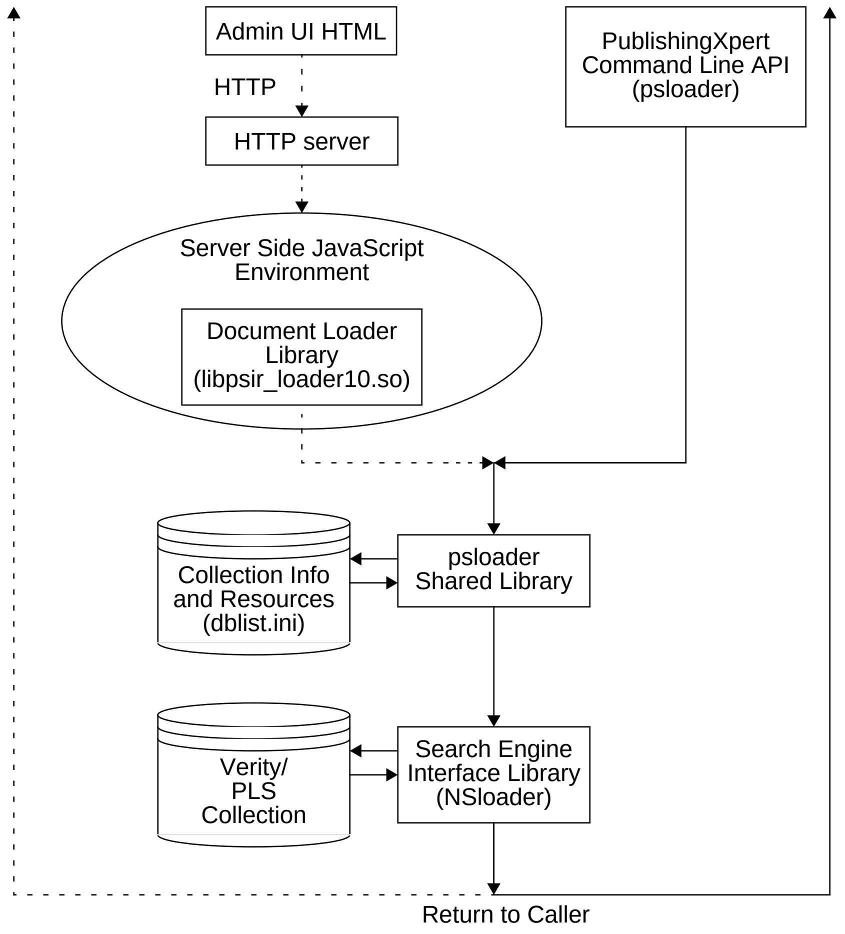
This chapter contains the following sections:

- Overview
- Creating a Publication
- Deleting a Document
- Listing Documents in a Publication
- Editing Publication Information
- Working with Groups of Publications
- Listing Information About a Publication
- Obtaining the Names of Publications
- Optimizing the Publication Index
- Updating a Publication

Overview

The document loader takes documents and indexes the text in them for future searches. The loader structure is shown in Figure 4.1.

Figure 4.1 Loader control structure


Admin UI HTML
HTTP
HTTP server
PublishingXpert Command Line API (psloader)
Server Side JavaScript Environment
Document Loader Library (libpsir_loader10.so)
Collection Info and Resources (dblist.ini)
psloader Shared Library
Verity/ PLS Collection
Search Engine Interface Library (NSloader)
Return to Caller

The document loader can be accessed either through a browser using HTML and the Server Side JavaScript (SSJS) environment, or through the UNIX command line utility `psloader`. Both interfaces access the `psloader` shared library, which first reads the `dblist.ini` file to obtain information about available collection lists and search engines.

The loader then accesses a Verity or PLS search engine using the `NSloader` search engine interface library. When the document collection is loaded, command returns to the caller (either the HTML interface or the UNIX command line.

Document Loader Command Line Utility

The `psloader` utility accepts commands for the document loader. Using the `psloader` utility you can determine which documents gets indexed, as well as the granularity of the index. The loaded document can then be accessed by the search server.

In most cases, the `psloader` utility computes directory and collection information defaults, which in turn makes most commands much shorter. For more information on publication collections, see the *PublishingXpert Administrator Handbook*.

For example, to update the collection `mycoll`, you can use the command:

```
psloader update mycoll
```

The syntax of the `psloader` command line utility is:

Syntax **psloader** -*switches arguments*

Table 4.1 Switch list for the `psloader` utility

Switch	Arguments	Description
-trace_level	0, 1, 2, or 3	selects the diagnostic output level, overriding the `trace_level` set in the `system.ini` file; 0 is the least verbose, and 3 is most verbose
-f	*filename*	reads attributes from file *filename*; if *filename* is set to “**-**”, uses stdin
-op	*operation_name*	the name of the operation; see Table 4.2, “Operation command arguments for the psloader utility,” on page 53 for a complete list

Table 4.1 Switch list for the `psloader` utility

Switch	Arguments	Description
-arg	*argument*	an argument being passed to an operation
-sysinipath	*pathname*	location of `system.ini` file; this overrides the path `$PS_HOME/config/pubsys/system.ini`

Command arguments to `psloader` can be specified either on the command line or in an argument file. On the command line, arguments names and values can be specified either as `-arg val` or `arg=val`. In an external file, arguments must be specified as `arg=val`, using one line per argument. The first two unnamed arguments are assumed to have names of `op` and `pub`, referring to the operation and publication respectively. Additional unnamed arguments are called `arg`*`N`* where n is the sequential number of the argument, such as `arg1`, `arg2`, and `arg3`.

Thus, the following commands are equivalent:

```
psloader update mycoll /dir/file.html
psloader -op update -pub mycoll arg1=/dir/file.html
psloader update mycoll -f inpfile.txt
```

where `inpfile.txt` contains the line:

```
/dir/file.html
```

Note The `psloader` command can only handle one command argument per invocation.

Table 4.2 shows the command arguments for the `psloader` command line utility.

Table 4.2 Operation command arguments for the `psloader` utility

Command Argument	Description
create *pubName pubDesc docroot* [*attributes*]	creates a publication named *pubName* with description *pubDesc* in the directory *docroot*; see page 54
delete *pubName* { *pathName* | **publication** }	deletes the specified document (*pubName*) from a publication index (*pathName*) or from an entire **publication**; see page 57
edit *pubName attributes*	changes information specified by *attributes* in publication *pubName*; see *page 56* For more information about attributes, see the discussion following Table 4.3, "Parameters for the psloader create command," on page 54.
info *pubName*	gets information about publication *pubName*; see page 62
names [*parameter*]	gets the names of all publications on the system, or if an information parameter is supplied, returns the value of the specified publication information *parameter* for each publication; see page 64
optimize *pubName*	optimizes the index files of publication *pubName*; see page 64
update [*pubName* [*attributes*]]	updates documents in a collection; see page 65

Parameters in Initialization Files

The `psloader create` command copies the contents of the [`psir/loader/defaults`] section of the `system.ini` file to the `dblist.ini` file when a publication is created.

The `psloader grpcreate` command copies the contents of the [`psir/loader/grpdefaults`] section of the `system.ini` file to the `dbgroup.ini` file when a publication group is created.

You must also set the `PS_HOME` environment variable to point to the directory containing your PublishingXpert files before you invoke the `psloader` command line utility.

Creating a Publication

The `psloader create` operation creates a publication.

Syntax **psloader create** *pubName* **doc-root=***docrootpath* [**description=***pubDesc*]
[**convert-docs=**yes **|** no] [**extract-metatags=**yes **|** no]
[**file-format=**html **|** ascii **|** mail **|** pdf **|** news]
[**filename-pattern=***pattern* [**index-recurse=**yes **|** no]
[**language=***langopt*] [**template-dir=***pattern_dir*] [*attributes*]

Table 4.3 Parameters for the `psloader create` command

Parameter	Usage
pubName	The publication name.
doc-root	The path to the directory where the document resides.
description	An ASCII character string that describes the publication. (Optional)
convert-docs	Specify whether to convert documents into HTML; must be set to ‘yes’ or ‘no’. The default value is ‘no’. (Optional)
extract-metatags	specify whether to extract HTML metatags; must be set to yes or no. The default value is ‘no’. (Optional)

Table 4.3 Parameters for the `psloader create` command (Continued)

Parameter	Usage
file-format	Format of the document file or files; must be set to 'html', 'ascii', 'mail', 'pdf', 'meta', or 'news'. The default value is 'html'. (Optional)
filename-pattern	A wildcard pattern, such as `*.html`, that specifies a group of document files. (Optional) **Note:** There is no default value, but Netscape recommends you use the following patterns for certain types of files: • HTML files: *.html • news or mail files: * • ASCII files: *.txt • PDF (Acrobat) files: *.pdf • meta files: *.meta
index-recurse	Indicates whether documents in subdirectories of the publication's main directory are included in the publication; must be set to 'yes' or 'no'. The default value is 'yes'. (Optional)
language	The language in which the publication is presented. The languages supported by Netscape are 'dutch;8859', 'english;8859', 'french;8859', 'german;8859', italian;8859', norwegian;8859', portuguese;8859', spanish;8859', and 'swedish;8859'; the default value is 'english;8859'. (Optional) **Note:** Your search engine might support a different set of languages. See the documentation with your search engine for details.

Table 4.3 Parameters for the `psloader create` command (Continued)

Parameter	Usage
template-dir	A directory containing templates used by PublishingXpert user search applications such as `psquery` to find patterns in documents. (Optional) For information on the psquery application, see the *PublishingXpert Administrator Handbook.*
attributes	These define attribute information; see the Discussion following this table for more information. (Optional)

Discussion The attribute arguments define attribute information in the following format:

```
description=Publication_Description
attr-nameN=attribute_name_N
attr-aliasN=attribute_alias_N
attr-typeN=attribute_type_N
```

where:

`attr-nameN`	is the *N*th attribute name, such as `attr-name1`, `attr-name2`, and `attr-name3`, and `attribute_name_N` is a string defining the name of the attribute, such as `Title`
`attr-aliasN`	is the *N*th attribute alias, such as `attr-alias1`, `attr-alias2`, and `attr-alias3`, and `attribute_alias_N` is an alternate search alias for the tag. For example, an alias of `Subject` could be added to `Title` to enable searching under both attribute names
`attr-typeN`	is the *N*th attribute type, such as `attr-type1`, `attr-type2`, and `attr-type3`, and `attribute_type_N` is the type of attribute. Acceptable values are `text`, `date`, `numeric`, and `zone`; the default value is `text`

Example

```
psloader create mypublication doc-root=$ACHOME/mydocuments
```

Deleting a Document

The `psloader delete` command deletes a document from the publication index or a publication from the system. If an entire publication is deleted, converted documents are also deleted.

Syntax **psloader delete** *pubName* **{** *pathname* **|** **publication** **|** **missing** **}**

Table 4.4 Parameters for the `psloader delete` command

Parameter	Usage
pubName	The publication name.
pathname	The path name of the document to delete from the index.
publication	Remove the entire publication; use instead of *pathName*.
missing	Delete all documents in the index file for which the source file no longer exists.

Discussion The `psloader delete` command requires either a *pathname*, the keyword **publication**, or the keyword **missing**. If the single keyword **publication** is given as an argument, the entire publication is removed. If the keyword **missing** is given as an argument, all documents for which there is no source file are deleted. Otherwise, only individual filenames are removed from the index. Multiple documents can be removed with the same `psloader delete` command.

Example

```
psloader delete mypub publication
```

Listing Documents in a Publication

The `psloader docnames` command lists the pathnames of all documents in a publication.

Syntax **psloader docnames** *pubName*

Table 4.5 Parameters for the `psloader docnames` command

Parameter	Usage
pubName	The publication name.

Example

```
psloader docnames mypub
```

Editing Publication Information

The `psloader edit` command edits information about a publication.

Syntax **psloader edit** *pubName* [*properties*]

Table 4.6 Parameters for the `psloader edit` command

Parameter	Usage
pubName	The publication name.
properties	These define property information; see the Discussion following Table 4.3, “Parameters for the psloader create command,” on page 54 for more information. (Optional) Note: File format and language properties cannot be changed with the `psloader edit` command.

Discussion If the single keyword `publication` is given as an argument, the entire publication is removed; otherwise, only individual filenames are removed from the index.

Example

```
psloader edit mypub convert-docs=yes description=”My New Title” \
filename-pattern=\*.html
```

Working with Groups of Publications

The following `psloader` commands enable you to manipulate groups of documents. The following operations are available:

- Creating a New Publication Group
- Deleting a Group of Publications
- Changing Information in a Group of Publications
- Obtaining Information About a Publication Group
- Obtaining a List of Publication Group Names

Creating a New Publication Group

The `psloader grpcreate` command creates a new publication group.

Syntax **psloader grpcreate** *groupName* [*pubName1 pubName2...*] [*properties*]

Table 4.7 Parameters for the `psloader grpcreate` command

Parameter	Usage
groupName	The publication group name.
pubName1, pubname2...	The publication name.
properties	These define property information; see the Discussion following Table 4.3, “Parameters for the psloader create command,” on page 54 for more information. (Optional)

Example

```
psloader grpcreate mygroup mypub1 mypub2 \
description=”My Publication Group”
```

To create a group named `All` containing every publication name, enter:

```
psloader names | psloader grpcreate All \
description=”Every Publication” -f -
```

See also “Obtaining the Names of Publications” on page 64.

Deleting a Group of Publications

The `psloader grpdelete` command removes a group of publications.

Syntax **psloader grpdelete** *groupName* **{** *pubName* **|** **group** **}**

Table 4.8 Parameters for the `psloader grpdelete` command

Parameter	Usage
groupName	The publication group name.
pubName	The publication name.
group	Keyword; a list of publications.

Discussion The **group** keyword deletes an entire section from the `dbgroup.ini` file; a publication name argument causes the entry to be deleted from a section in the `dbgroup.ini` file.

Example

```
psloader grpdelete mygroup mypub
```

Changing Information in a Group of Publications

The `psloader grpedit` command changes the information in a group of publications.

Syntax **psloader grpedit** *groupName* [*pubName1 pubName2...*] [*properties*]

Table 4.9 Parameters for the `psloader grpedit` command

Parameter	Usage
groupName	The publication group name.

Table 4.9 Parameters for the `psloader grpedit` command (Continued)

Parameter	Usage
pubName1 pubname2...	The publication name.
properties	These define property information; see the Discussion following Table 4.3, “Parameters for the psloader create command,” on page 54 for more information. (Optional)

Discussion If you do not enter a publication name, the `psloader grpedit` command changes your publication group to contain zero publications.

Example

```
psloader grpedit mygroup mypub1 mypub2 \
description=”My Publication Group”
```

Obtaining Information About a Publication Group

The `psloader grpinfo` command returns information about a publication group.

Syntax **psloader grpinfo** *groupName*

Table 4.10 Parameters for the `psloader grpinfo` command

Parameter	Usage
groupName	The publication group name.

Example

```
psloader grpinfo mygroup
```

Obtaining a List of Publication Group Names

The `psloader grpnames` command returns a list of publication group names.

Syntax **psloader grpnames**

Discussion The `psloader grpnames` command returns the list in the format of one publication group per line to the standard output (`stdout`).

Example

```
psloader grpnames
```

Listing Information About a Publication

The `psloader info` command returns information about a publication.

Syntax **psloader info** *pubName*

Table 4.11 Parameters for the `psloader info` command

Parameter	Usage
pubName	The publication name.

Discussion Information is returned for the named publication. The output is sent to the standard output and is similar to the data in the `dblist.ini` file. For example:

```
attr-name1=author
attr-type1=text
attr-name2=Title
attr-type2=text
attr-name3=SourceType
attr-type3=text
convert-docs=no
description=Publication a1 - Documents in /tmp
doc-root=/tmp
extract-metatags=no
file-format=html
filename-pattern=*.html
index-recurse=no
```

```
language=german;8859
template-dir=/h/aura/d2/ns-apps-20/lib/locale/en/psir/templates
```

The following table explains each line in the example:

Line	Description
attr-name1=author	first attribute name; here the value is set to `author`
attr-type1=text	first attribute type; here the value is set to `text`
attr-name2=Title	second attribute name; here the value is set to `Title`
attr-type2=text	second attribute title; here the value is set to `text`
attr-name3=SourceType	`\SourceType`
attr-type3=text	third attribute type; here the value is set to `text`
convert-docs=no	do not convert the documents to HTML
description= Publication a1 - Documents in /tmp	an ASCII character string describing the publication; here the value is set to `Publication a1 - Documents in /tmp`
doc-root=/tmp	the path to the directory where the document resides; here the value is set to `/tmp`
extract-metatags=no	metatags are not being extracted
file-format=html	format of the document file or files; here the value is set to `html`
filename-pattern=*.html	a wildcard pattern that specifies a group of document files; here the pattern is `*.html`
index-recurse=no	documents in subdirectories of the publication's main directory are not included in the publication
language=german;8859	The language in which the publication is presented; here the value of `language` is `german;8859`.
template-dir=/h/aura/d2/ns-apps-20/lib/locale/en/psir/templates	Utilities such as psquery are to use templates located in the directory `/h/aura/d2/ns-apps-20/lib/locale/en/psir/templates` to find patterns in documents.

Obtaining the Names of Publications

The `psloader names` command returns the names of all publications on the system, or returns the value of a single publication and information item for each publication.

Syntax **psloader names** [*information_item*]

Table 4.12 Parameters for the `psloader names` command

Parameter	Usage
information_item	An information item, such as an attribute or a keyword. (Optional)

Discussion See “Creating a Publication” on page 54 for a list of possible attributes and parameters.

Example

```
psloader names doc-root
```

Optimizing the Publication Index

The `psloader optimize` command optimizes the collecting of publications.

Syntax **psloader optimize** *pubName*

Table 4.13 Parameters for the `psloader optimize` command

Parameter	Usage
pubName	The publication name.

Discussion The publication collection’s index can contain many entries spread across many files. The `psloader optimize` command uses the search engine’s optimization algorithms to make the searching the index faster, such as consolidating information into fewer files. See your search engine’s documentation for more details on how optimization works.

Updating a Publication

The `psloader update` command updates the document index in a publication.

Syntax **psloader update** *pubName* [*filepath*] [**modify-time=***newer_than*]

Table 4.14 Parameters for the `psloader update` command

Parameter	Usage
pubName	The publication name.
filepath	The pathname of a specific file to be reindexed or deleted. (Optional)
modify-time	The document modified time in ISO 8601 format: *yyyy-mm-dd hh:mm:ss*Z where Z indicates Universal Time (UTC); the time from which documents are to be considered for updating. If *newer_than* is an empty string, all files are indexed. (Optional)

Discussion If no pathname is specified, `psloader update` looks for all documents under the `doc-root`, compares their index times to the time last indexed, and updates any documents that have changed.

Example

```
psloader update my_publication /docs/mypub/*.html
```

2 *The AgentXpert Framework*

Chapter

5

Introducing AgentXpert

This chapter provides a description of AgentXpert, a framework that lets you develop custom agents for various tasks. It introduces you to AgentXpert concepts, components, and processes.

This chapter contains the following sections:

- AgentXpert Overview
- Components of AgentXpert
- Configuring and Invoking an Agent

AgentXpert Overview

The AgentXpert framework consists of a dispatcher and one or more command servers. A command server waits for commands from the dispatcher and processes those commands. A command specifies which agent is to be invoked.

The dispatcher sends commands to one or more command servers. The dispatcher takes the input file and the information about the agent to be invoked and sends it to the command servers.

The dispatcher can be invoked in two modes:

- blocking

  In this mode, the dispatcher waits, blocking further input, until it receives all acknowledgments.

- timeout (default)

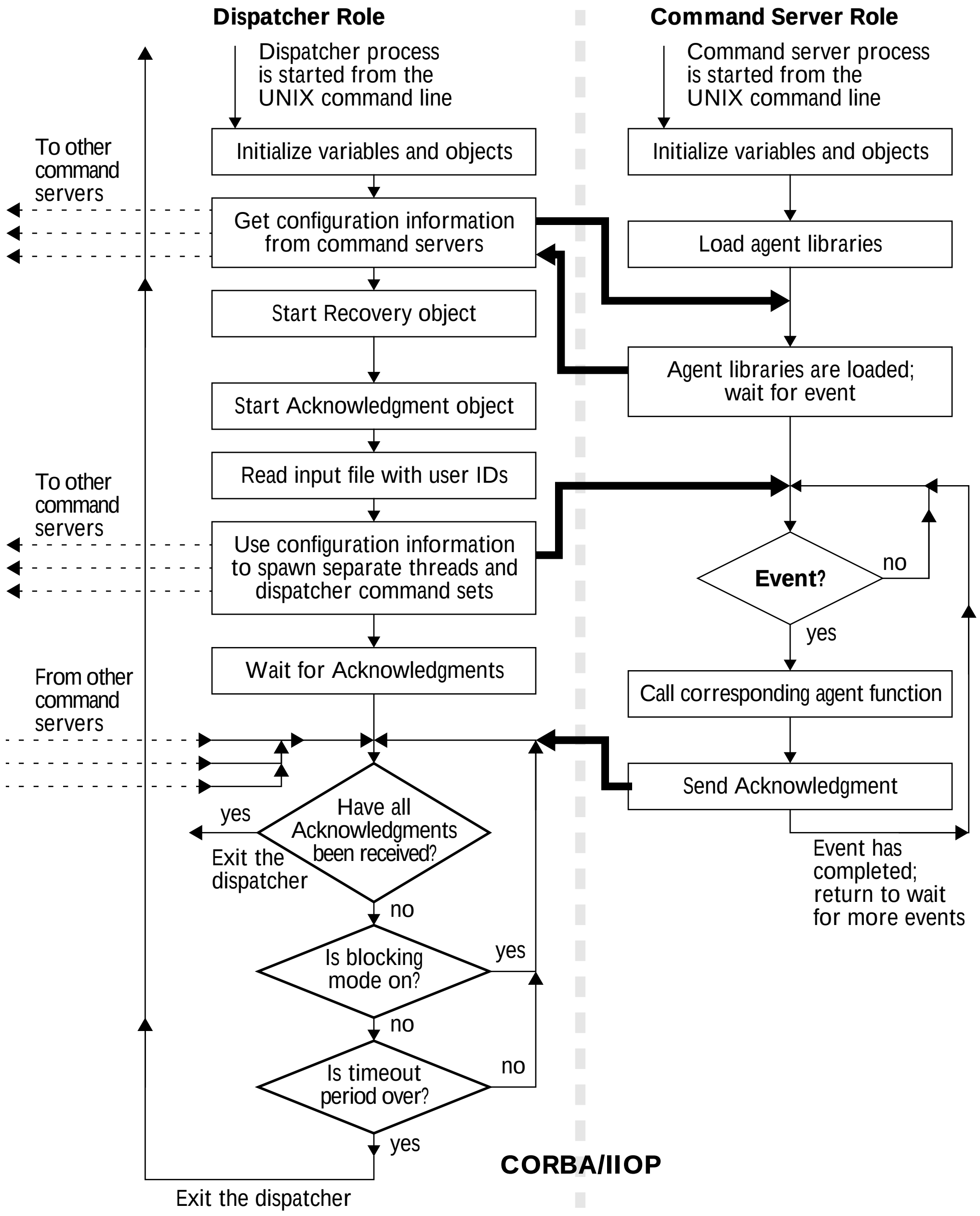
  In this mode, the dispatcher times out after a period of time determined by the command servers. Each command server specifies its own timeout period.

There is no limit to the number of command servers you can have. If there is more than one command server available, the dispatcher balances the load equally among the available command servers.

Using the AgentXpert framework, you can develop agents that can respond to a variety of events and take specified actions. For example, you can develop agents that search through collections of documents based on profile information stored in the database and distribute those search results to subscribers.

Figure 5.1 contains a diagram of an AgentXpert framework scenario.

Figure 5.1 The AgentXpert framework


Dispatcher Role
Command Server Role
Dispatcher process is started from the UNIX command line
Initialize variables and objects
To other command servers
Get configuration information from command servers
Start Recovery object
Start Acknowledgment object
Read input file with user IDs
To other command servers
Use configuration information to spawn separate threads and dispatcher command sets
Wait for Acknowledgments
From other command servers
Have all Acknowledgments been received?
yes
Exit the dispatcher
no
Is blocking mode on?
yes
no
Is timeout period over?
no
yes
Exit the dispatcher
CORBA/IIOP
Command server process is started from the UNIX command line
Initialize variables and objects
Load agent libraries
Agent libraries are loaded; wait for event
Event?
no
yes
Call corresponding agent function
Send Acknowledgment
Event has completed; return to wait for more events

Components of AgentXpert

The two main components of the AgentXpert framework are:

- the Dispatcher
- the Command Server

This section gives an overview of how each of them work.

Dispatcher

The dispatcher provides a means to distribute events to command servers. It takes an event name, command name, and a list of name-value pairs and distributes them to one or more command servers. Each command server, in turn, sends an event specified by the dispatcher to a corresponding agent.

As part of every event, the dispatcher includes an IIOP reference of an Acknowledgment object. This object provides a callback mechanism for the agent to notify the dispatcher of the success or failure of an event.

When activated, the dispatcher performs the following sequence of steps:

1. All variables and object request brokers (ORBs) are initialized.
2. The dispatcher creates an event to query the command servers for their configuration information (timeout period and the number of threads the command servers can support).
3. The Recovery object is started.
4. The Acknowledgment object is started.
5. The dispatcher reads an input file containing user IDs or e-mail addresses.
6. The dispatcher uses configuration information to spawn separate threads and dispatcher command sets.
7. If all Acknowledgment messages have been received, exit the dispatcher.
8. If *blocking mode* is on (the dispatcher is blocked from further input), continue to wait for Acknowledgment messages (step 7).

9. If blocking mode is not set, and if the timeout period is over, the dispatcher exits; otherwise, the dispatcher continues to wait for Acknowledgment messages (step 7).

Command Server

The command server functions as a gateway for various events, deciding which event goes to which agent. When the command server decides an event goes to a particular agent, it builds an event structure and passes it to the agent function. Each agent must be able to interpret the contents of an event to correctly act upon the information contained in the event.

The command server is also a full-fledged CORBA object. Agents sit on the command server side and wait for events. When activated by an event from the dispatcher, the command server performs the following sequence of steps (see Figure 5.1 on page 71):

1. The object request broker (ORB) and all variables are initialized.
2. The command server loads agents from the configuration file (`psasrvevent.conf`).
3. The command server waits for events. When an event arrives, the command server invokes the agent to which the event has been sent.
4. If there is a valid Acknowledgment reference, the command server sends an Acknowledgment message.
5. When the event has completed, the command server repeats step 3 to wait for more events.

Configuring and Invoking an Agent

This section describes the steps you must follow to configure and invoke the agents distributed with Search Server and AgentXpert. The topics discussed are:

- Creating the Command Server Initialization File
- Creating the Command Server Configuration File
- Starting the Command Server
- Starting the Dispatcher

The agents included with Search Server and AgentXpert are described in the section "Using Existing Agents" on page 90 in Chapter 6, "Building an Agent."

Creating the Command Server Initialization File

You must create the command server initialization file `acpsacmdsrv.ini` for the command server to run. The following is a sample `acpsacmdsrv.ini` file with a description of its components.

Note The command server reads the `acpsacmdsrv.ini` file only once: when the command server is first started. If you change any of the command server configuration files, you must restart the command server for your changes to take effect.

```
# -----------------
# Trace file setup
# -----------------

[acpsacmdsrv_trace]
trace-work-dir = .
trace-file-name = trace_acpsacmdsrv.log
trace-level   = 3
trace-components = acpsacmdsrv

# -------------------------
# acpsacmdsrv configuration
# -------------------------
# This is the Command Server configuration section of the init file
[acpsacmdsrv_conf]
ACPSA-srv-priority = 0
ACPSA-srv-instance-num = 6
ACPSA-srv-cmdack-timeout = 60
```

The following table explains each line in the `acpsacmdsrv.ini` file. The pound-sign (#) character at the beginning of a line denotes a comment.

Line	Description
[acpsacmdsrv_trace]	identifies is the trace section of the initialization file. Use the trace file when you want to trace the execution for troubleshooting
trace-work-dir = .	the directory where you want to put the trace log; in this example it is the current working directory
trace-file-name = trace_acpsacmdsrv.log	the log file for the trace; here it's set to `trace_acpsacmdserv.log`
trace-level = 3	the level of trace information to be supplied; can be 0, 1, 2, or 3, with 0 being least verbose and 3 being most verbose. In this example it is set to the most verbose mode.
trace-components = acpsacmdsrv	the trace component name; defaults to `acpsacmdsrv`. Do not change this name
[acpsacmdsrv_conf]	identifies the command server configuration section of the initialization file
ACPSA-srv-priority = 0	reserved for future use
ACPSA-srv-instance-num = 6	specifies the number of threads the command server can spawn; this example specifies 6 threads
ACPSA-srv-cmdack-timeout = 60	sets the duration in seconds the Event Dispatcher waits for Acknowledgment; in this example it is set to 60 seconds

Creating the Command Server Configuration File

The command server looks for a configuration file called `psasrvevent.conf` that contains event specifications. The entries to specify an agent are:

- **Event name**: a unique name given to a particular event or agent. This name is used by the AgentXpert scheduler user interface to identify an agent. It is represented by surrounding square brackets; for example:

  ```
  [MyAgent]
  ```

  An indent name may be any alphanumeric string but may not contain spaces.

- **Command name**: the name given to the function that corresponds to the agent. It is specified using the syntax:

  ```
  command_name = event_template
  ```

  where event_template is the template agent function name.

- **Execute command:** the command line syntax for running the dispatcher with different events. This keyword is also used by the PSAdmin AgentXpert scheduler to schedule batch jobs using the UNIX `at` system command. (See *Getting Started with the PublishingXpert System* for more information on the PSAdmin commands.)

- **Library name**: the name of the library that contains the agent function:

  ```
  library_name = libpsagent_eventtemplate10.so
  ```

  If the `LD_LIBRARY_PATH` environment variable is not set, you must specify the absolute path. For example:

  ```
  library_name = /usr/ps2.01/libs/libpsagent_eventtemplate10.so
  ```

- **Initialization function name**: the initialization function that must be called when the library is loaded for the first time. This can be an empty function, but it must be present in the library:

  ```
  init_function_name = event_template_init
  ```

For example, if the name of your agent is 'MyAgent' and the function names are `myagent_init()` and `myagent()`, your configuration file entry will look similar to the following:

```
[MyAgent]
command_name = myagent                  # agent function name
execute_command = acpsadisp dispatcher dispatcherRole -e MYAGENT -c
myagent -f input.users
library_name = libmyagent10.so          # agent dynamic library
init_function_name = myagent_init       # Initialization function
```

The following table explains each line in the `psasrvevent.conf` configuration file:

Line	Description
[MyAgent]	The event name of this section in the `psasrvevent.conf` file; in this example, the event name is 'MyAgent'.
command_name = myagent	The agent function name; in this case, the name `myagent` has been given to the function that corresponds to the agent 'MyAgent'.
execute_command = acpsadisp dispatcher dispatcherRole -e MYAGENT -c myagent -f input.users	Executes a command from the UNIX shell; in this case, the command is 'acpsadisp dispatcher dispatcherRole -e MYAGENT -c myagent -f input.users'.
library_name = libmyagent10.so	The agent's dynamic library; `libmyagent10.so` contains the agent function.
init_function_name = myagent_init	The name of the initialization function; instructs the command server to call the initialization function `myagent_init` when the library `libmygent10.so` is first loaded.

Note Each event name must be unique. If there are multiple events with the same name, the command server only recognizes the first one in the file.

The following `psasrvevent.conf` file defines the four agents included with Search Server and AgentXpert, as well as several related functions:

```
[GenerateUserList]
command_name = dummyfunc
execute_cmd = acpsadisp userList writeUserQueryList -f input.users
library_name = libpsagent_psaevents10.so
init_function_name = dummyfunc

[GenerateResult]
command_name = psaGenRes
execute_cmd = acpsadisp dispatcher dispatcherRole -e GenerateResult -c
psaGenRes -f input.users
library_name = libpsagent_psaevents10.so
init_function_name = psaGenRes_init

[FormatResult]
command_name = psaFrmRes
execute_cmd = acpsadisp dispatcher dispatcherRole -e FormatResult -c
psaFrmRes -f input.users
library_name = libpsagent_psaevents10.so
init_function_name = psaFrmRes_init

[Advertiser]
command_name = advtsr
execute_cmd = acpsadisp dispatcher dispatcherRole -e Advertiser -c
advtsr -f input.users
library_name = libpsagent_psaevents10.so
init_function_name = advtsr_init

[Mailer]
command_name = mail_event
execute_cmd = acpsadisp dispatcher dispatcherRole -e Mailer -c
mail_event -f input.users
library_name = libpsagent_psaevents10.so
init_function_name = mail_event_init

[Message_Mailer]
command_name = message_mail
execute_cmd = acpsadisp dispatcher dispatcherRole -e Mailer -c
mail_event -f input.emails -M
library_name = libpsagent_psaevents10.so
init_function_name = mail_event_init
```

```
[DummyEvent]
command_name = event_template
execute_cmd = acpsadisp dispatcher dispatcherRole -e event_template -c
event_template
library_name = libpsagent_eventtemplate10.so
init_function_name = event_template_init
```

This file tells the command server to load the `GenerateUserList`, `GenerateResult`, `FormatResult`, `Advertiser`, `Mailer`, `Message_Mailer`, and `DummyEvent` agents. The following table explains the first ten lines in the configuration file:

Line	Description
[GenerateUserList]	The event name of this section in the `psasrvevent.conf` file; here it is `GenerateUserList`.
command_name = dummyfunc	the agent function name; the name dummyfunc has been used so AgentXpert can schedule this function. This is actually a special dispatcher that queries the membership database to generate a list of users.
execute_cmd = acpsadisp userList writeUserQueryList -f input.users	Executes a command from the UNIX shell; in this case, the command is 'acpsadisp userList writeUserQueryList -f input.users'.
library_name = libpsagent_psaevents10.so	The agent dynamic library; in this example, the library `libpsagent_psaevents10.so` contains the agent function
init_function_name = dummyfunc	The name of the initialization function; tells the command server to call the function `dummyfunc` when the library `libpsagent_psaevents10.so` is first loaded
[GenerateResult]	The event name of this section in the `psasrvevent.conf` file; here it is `GenerateResult`.

Line	Description
command_name = psaGenRes	The agent function name; the name `psaGenRes` has been given to the function the agent `GenerateResult` serves.
execute_command = acpsadisp dispatcher dispatcherRole -e GenerateResult -c psaGenRes -f input.users	Executes a command from the UNIX shell; in this case, the command is 'acpsadisp dispatcher dispatcherRole -e GenerateResult -c psaGenRes -f input.users'.
library_name = libpsagent_psaevents10.so	The agent dynamic library; in this example, the library `libpsagent_psaevents10.so` contains the agent function.
init_function_name = psaGenRes_init	The name of the initialization function; tells the command server to call the function `psaGenRes_init` when the library `libpsagent_psaevents10.so` is first loaded.

See "Using Existing Agents" on page 90 in Chapter 6, "Building an Agent," for more information.

Starting the Command Server

Use the `acpsacmdsrv` command line utility in a terminal window to start a command server:

Syntax **acpsacmdsrv** *command_server*

Table 5.1 Parameter list for the `acpsacmdsrv` utility

Switch	Arguments	Description
	command_server	name of the Command Server

Discussion To start more than one command server on the same command line, use semicolons to separate the commands. For example:

```
acpsacmdsrv cmdsrv1 & ; acpsacmdsrv cmdsrv2 & ; acpsacmdsrv cmdsrv3 &
```

Example

```
acpsacmdsrv cmdsrv &
```

Starting the Dispatcher

Start the dispatcher using the acpsadisp command line utility:

Syntax **acpsadisp** *dispatcher* **dispatcherRole -e** *event_name* **[***event_name2...***]**
-c *command_name* **-f** *user_IDs_file* **[-b]**

Table 5.2 Parameter list for the acpsacmdsrv utility

Switch	Arguments	Description
	dispatcher	the name of the library from which to draw commands
	dispatcherRole	constant; tells the program to assume a Dispatcher role. Use the **dispatcherRole** constant to generate a list of user IDs
-e	*event_name*	the unique name given to a particular event or agent
-c	*command_name*	the name given to the function the agent serves
-f	*user_IDs_file*	the name of the file containing a list of user IDs
-b		specifies blocking mode (Optional)

Discussion You can also send multiple commands to the dispatcher using the following syntax:

```
acpsadisp dispatcher dispatcherRole -e event1,event2,event3
    -c command1,command2,command3 -f sample.users
```

Example

```
acpsadisp dispatcher dispatcherRole -e GenerateResult, FormatResult,\
    Mailer -c psaGenRes, psaFrmRes, mail_event -f sample.users
```

Chapter

6

Building an Agent

This chapter provides a road map for developing an agent. An agent template is provided as part of the explanation on how to build one. This template file is located in the `$PS_HOME/SDK/agentXpert` directory in your installation. Netscape recommends compiling and testing this agent before you build one of your own.

This chapter contains the following sections:

- Defining an Agent
- Developing a New Agent
- Using Existing Agents

Defining an Agent

Before embarking on any development, you should familiarize yourself with the architecture and the composition of the various components of AgentXpert.

Agents are shared library functions. They are loaded by the command server at startup time; once loaded, they wait for events. The command server is the gateway for various events, deciding which event goes to which agent. After the command server decides to dispatch an event to a particular agent, the command server builds an event structure and passes it to the agent function as a `void` pointer. An agent must be aware of the contents of the structure to make use of the information it contains.

Developing a New Agent

All agents must contain the following two functions:

- an Initialization Function
- an Agent Function

Initialization Function

The signature of the initialization function must be `int` *`myagent`*`_Init(void *)`. The function must return 1 on success and 0 (zero) on failure. Use this function to initialize your agent. This function is called by the command server only at the time the dynamic library (`*.so`) is loaded.

Agent Function

The signature of the agent function must be `int` *`myagent`*`(void *)`. The function must return 1 on success and 0 (zero) on failure. When a command server receives a request for your agent function, the agent function is invoked with an appropriate event structure.

Note Netscape recommends that you not write an agent that invokes or creates a separate process using `fork`, `vfork`, or similar methods, as this is not part of the AgentXpert framework.

Writing Functions

The following example shows the two functions that implement an agent. The `event_template_init` function initializes the agent; the `event_template` function interprets the event structure and performs the desired action.

```
//------------include necessary files--------------
#include <iostream.h>
#include <fstream.h>
#include "eventtemplate.h"
#include "axeventdispatcher.h"

//--------------------------------
#define TRUE           1
#define FALSE          0

//-------------------------------------
//System variables
//While building your event application you put your own
//variables and definitions here

#define tempFileName "dummyEvent.log"
ofstream fop;

//------------------------------------------
//init function template:
//         This function does all the initialization of the event
//         processing. If properly done, it helps you avoid lots
//         of costly mistakes
//-------------------------------

// This must be defined as a C function to prevent Name Mangling
extern "C"
int event_template_init (void *msgStruct)
{
        //STEP 1
        //Initialize
        cout <<"Event template initialization done..." << endl;

        //dummy operations
        fop.open(tempFileName,ios::out);
        fop << "Initialization done....."<< endl;
        //STEP 2
        // depending on the success or failure of your processing
        // return TRUE or FALSE

        // This is to show you must return
        // TRUE or FALSE in appropriate cases
        if(!(1))
        {
                return FALSE;
        }
        else {
                return TRUE;
        }
}

//-------------------------------------
//This is the actual agent function event_template.
//Follow the steps described in the code
//--------------------------------------

extern "C"
int event_template(void *msgStruct)
{
   axEventDispatcher *test=0;
  //STEP 1
  // Pass the msgStruct to the structure
  // The class axEventDispatcher is obtaining an event
  // with structure msgStruct
   test = axEventDispatcher::getAxEvent(msgStruct);

   if(!test)
      {
        cout << "Event conversion failed" << endl;
        return FALSE;
      }
  //STEP 2
        RWCString nameStr;
        RWCString valStr;
  // do your preprocessing
  // The following sample code prints out all the name/value pairs
while((test->getNameValuePair(nameStr,valStr)).Good())
```

```
    {
        cout << "Name= " << nameStr.data() << "  ;Value = " <<
valStr.data() << endl;
    }
  fop << "Done Processing...."<< endl;
  fop.flush();

  // get rid of memory we don't need
  delete test;
  //STEP 3 return TRUE/FALSE
  if(!(1))
    {
      return FALSE;
    }
  else {
    return TRUE;
  }

}
```

The next example uses methods from the `axEventDispatcher` class. The `getNameValuePair()` method obtains the next parameter from the event structure. The following is the definition file for the `agentEvent` class (`axeventdispatcher.h`):

```
#ifndef  _AXEVENTDISPATCHER_H_
#define  _AXEVENTDISPATCHER_H_
#include <rw/slistcol.h>
#include <rw/gslist.h>
#include "acpsafrmwrk_c.hh"

//DEFINES
#define AX_NOERROR                0
#define AX_SUCCESS                0
#define AX_FAILURE                1
#define AX_NVLISTEMPTY            3
#define AX_ORBINITFAILURE         4
#define AX_CMDSERVER_BINDFAILURE  5
#define AX_UNKNOWN_ERROR          6

//typedef struct namevalue pair
```

```
class nvPair{

  public:
  nvPair();
  ~nvPair();
void setName(RWCString name);
RWCString getName();
void setValue(RWCString value);
RWCString getValue();
private:
  RWCString  name_;
  RWCString  value_;
};

typedef class nvPair NVPAIRRECORD;

//RANGERECORD;
declare(RWGSlist,NVPAIRRECORD);
typedef RWGSlist(NVPAIRRECORD)    NVPAIR_LIST;

class axEventDispatcher
{
public:
 axEventDispatcher();
 ~axEventDispatcher();
 axEventDispatcher& addNameValuePair(RWCString,RWCString);
 axEventDispatcher& getNameValuePair(RWCString&,RWCString&);
 axEventDispatcher& sendEvent(RWCString& serverName,RWCString&
hostName);
 static axEventDispatcher* getAxEvent(void *evtPtr);
 NVPAIR_LIST* axQueryServer(const char *serverName,const char
*hostName,NVPAIR_LIST& queryList);
 NVPAIR_LIST* axQueryServerConfig(const char *serverName,const char
*hostName);


 //Reflection methods
 void setEventId(long);
 long getEventId();
 void setName(RWCString&);
 void setName(const char *);
 RWCString getName();
 void setAckObject(RWCString&);
 RWCString getAckObject();
 void setAttributeList(NVPAIR_LIST& attrList);
 NVPAIR_LIST& getAttributeList();


 //Error handling methods
 long Error();
 void Error(long);
 int Good();// returns 1 on good, 0 on Bad
 int Bad();
 void Clear();

private:
 long eventId_;               // To maintain the UUID
 RWCString name_;             // Name of the event
 RWCString command_;          // Name of the command to be invoked
 RWCString token_;            // Token to be exchanged,
                              // Can be used to keep the transaction
                              // semantics
 RWCString ackObject_;        // IIOP object reference, if need to
                              // be acknowledged
 NVPAIR_LIST attributeList_;  // Name value pair list to be passed
                              // to the agent
 long err_;
};

#endif /* _AXEVENTDISPATCHER_H_ */
```

Edit the Event Configuration File

Update the event configuration file to contain the functions and the library name as described in the `psasrvevent` configuration file. For more information about the event configuration file format, see “Creating the Command Server Configuration File” on page 76 in Chapter 5, “Introducing AgentXpert.”

Using Existing Agents

This section contains descriptions of each of the agents included with the Search Server and AgentXpert product.

This section explains how to use the agents packaged with the AgentXpert framework. It includes a description of each agent, a description of the configuration file needed by the command server to invoke the agents, and the commands needed to get the dispatcher and command server to run the agents. These agents are part of the `libpsagent_psaevents10.so` shared library and are loaded at runtime. For information on the `libpsagent_psaevents10.so` shared library, see *Getting Started with the PublishingXpert System*.

For a description of the configuration file needed by the command server to invoke the agents, see "Creating the Command Server Configuration File" on page 76 in Chapter 5, "Introducing AgentXpert." For a description of the command needed to get the dispatcher and command server to run the agents, see "Starting the Command Server" on page 80 in Chapter 5, and for a description of the command server initialization file, see "Creating the Command Server Initialization File" on page 74, also in Chapter 5.

The AgentXpert package comes with the following preconstructed agents:

- Personalized Search
- HTML Formatter
- Advertiser
- Mailer

The code for these agents is located in the `$PS_HOME/SDK/agentXpert` directory. A description of each agent follows.

Personalized Search

The `GenerateResult` agent takes a list of user IDs, searches the database for a user-defined query, and generates the results into a file. Each file has a name of the format *`userid`*`.res` where *`userid`* is the user ID number. For example, if the user ID number is 2000, the output filename is `2000.res`. The generated output is written to the directory specified in the `cmaserver.conf` configuration file.

The command server checks for the validity of the directory name given to it before making any attempt to write to it. If the command server is given a list of directories, it writes to the first one in the list with write access enabled. The code for the `GenerateResult` agent is in the header file `psagenres.h` and the file `psagenres.cpp`.

HTML Formatter

The `FormatResult` agent takes a list of user IDs for which the result files (*`userid`*`.res`) have been generated and formats them. Like the `GenerateResult` agent, the `FormatResult` agent looks for the generated result files in the given list of directories. The formatted output files have the filename extension *`userid`*`.final` where *`userid`* is the user ID number. For example, if the user ID number is 2000, the output filename is `2000.final`. The code for the `FormatResult` agent is in the file `psafrmres.cpp`.

Advertiser

The `Advertiser` agent is a demonstration of what you can do with the formatted results of a query. This agent takes the list of user IDs and inserts quasi-randomly selected advertisement images into the formatted files. The code for the `Advertiser` agent is in the file `advtsr.cpp`.

Mailer

The `Mailer` agent takes a list of user IDs, looks for the files with a `.final` extension, and mails the files to the user whose e-mail address is in the first line of the file. The `Mailer` agent changes the name of the file to *`userid`*`.don` where *`userid`* is the user ID number. For example, if the given user ID is 2000, the `Mailer` agent looks for `2000.final`, mails the file to the user, and changes the filename to `2000.don`. The code for the `Mailer` agent is in the file `mailer_event.cpp`.

3 *Appendixes*

- **Search Server Example**
- **PublishingXpert SDK Makefile**

Appendix

A

Search Server Example

This chapter contains the file searchsdk_ex.cpp, an example that uses the search server software development kit (SDK). For information on the methods used in this example, see Chapter 3, “The Search Server API.”

Note Make sure you have installed the appropriate compilers on your machine before building your application. Refer to the *Getting Started* document for details on the recommended compilers.

```
#include <iostream.h>
#include <search_c.hh>
#include <searchsdk.h>

int main(int argc, char *const *argv)
{
    int              i,j;
    PSACSearch       *acsearch;
    PSACSearchArg    acsearcharg;
    PSACSearchRes    acsearchres;
    PSACCollSet      collset;

    // Initiate the CORBA client
    acsearch = new PSACSearch;

    // Get server collection list
    if (acsearch->GetCollection(collset).Bad() == -1) {
        cout << "GetCollection Failed" << endl;
        return(-1);
    }

    // Get collection name and path
    for (i=0; i<collset.count; i++) {
    cout << "GetCollection" << i << " " << collset.collname[i] << " "
        << collset.collpath[i] << endl;
    }

    // Set the maximum number of records and turn off highlighting
    acsearcharg.MaxRecords(100);
    acsearcharg.HLon(0);

    // Set the Query type
    cout << "ENTER QUERY PARSER (F)REETEXT, (B)OOLEAN" << endl;
    char parserType[512];
    cin.getline(parserType, sizeof(parserType));
    switch (*parserType) {
      case 'f':
      case 'F' : acsearcharg.QueryType(QP_FREETEXT);
      case 'b':
      case 'B' :  acsearcharg.QueryType(QP_BOOLEAN);
      default:    acsearcharg.QueryType(QP_BOOLEAN);
    };
    // Add collection to searcharg
    for (i=0; i<collset.count; i++) {
        if ((acsearcharg.AddCollection(collset.collname[i],
            collset.collpath[i])).Bad() == -1) {
            cout << "AddCollection Failed: collname  "
                 << collset.collname[i] << endl;
         }
    }

    // Set the query string
    char query[8192];
    cout << "BEGIN SEARCH, enter your query:" << endl;
    cin.getline(query, sizeof(query));
    acsearcharg.Query(query);

    // Do the search
    if ((acsearch->Search(acsearcharg, acsearchres)).Bad() == -1) {
        cout << "Search failed." << endl;
        return(-1);
    }

    // Print out the number of documents found.
    for (j=0; j< acsearchres.DocsReturned(); j++) {
    cout << acsearchres.DocPath(j) << endl;
    }

    return(1);

}
```

Appendix

B

PublishingXpert SDK Makefile

This appendix explains how to configure the PublishingXpert Makefile for your system and compile the files for your component of the PublishingXpert system.

Overview

This appendix presents the makefile `Makefile.basic` that contains all the required flags to build all components of the PublishingXpert Software Development Kit (SDK). The file is located in the directory `$PS_HOME/sdk`.

Each of the following components of the PublishingXpert SDK contains a Makefile that includes `$PS_HOME/sdk/Makefile.basic`:

- AgentXpert
- Search Server
- Membership:
    - Member
    - Assets
    - Access Control Services
- Billing

Configuring the Makefile

Each component of the PublishingXpert system contains a Makefile and example files which briefly explain how to use the SDK classes. Each product-specific Makefile reads the file `Makefile.basic` for global flags.

You must execute the following steps before using the `Makefile.basic` file:

1. Make sure `PS_HOME` is set to your PublishingXpert installation directory.
2. Set the following flags in `Makefile.basic` to point to the correct directory paths:
    - `LDAP_LIBS_PATH` to the *`$suitespot/`*`lib` directory where *`suitespot`* is the directory where the Netscape Enterprise Server is installed
    - `USER_LIB` to the directory that contains `libresolv.so` (usually `/usr/lib`)

3. Edit the `Makefile.basic` file to set the following flags to point to the proper locations:

   - `CPP` to the correct C++ compiler, for example:

     ```
     CPP        = /usr/local/tools/sparcworks/SUNWspro/bin/CC
     ```

   - `CPUTYPE`, to the type of CPU, for example:

     ```
     CPUTYPE    = sparc
     ```

   - `OSTYPE`, to the operating system, for example:

     ```
     OSTYPE     = solaris
     ```

   - `OSVERTYPE`, to the version of the operating system, for example:

     ```
     OSVERTYPE   = solaris2_5
     ```

4. Change to the subdirectory containing the PublishingXpert component you wish to compile:

For...	Enter...
AgentXpert	`cd $PS_HOME/sdk/agentxpert`
Search Server	`cd $PS_HOME/sdk/search/src`
Membership	`cd $PS_HOME/sdk/membership/member`
Assets	`cd $PS_HOME/sdk/membership/assets`
Access Control Services	`cd $PS_HOME/sdk/membership/aclsvc`
Billing	`cd $PS_HOME/sdk/billing`

To build the files in your component of the PublishingXpert SDK, enter:

```
make -f Makefile
```

Listing of Makefile.basic

```
#Generic definitions

CPUTYPE          = sparc
OSTYPE           = solaris
OSVERTYPE        = solaris2_5
CP               = /usr/local/tools/sparcworks/SUNWspro/bin/CC
LDAP_LIBS_PATH   = /disk01/suitespot/lib
USER_LIB = /usr/lib

CPPFLAGS       = -mt -DRW_MULTI_THREAD -DCPU=CPU_$(CPUTYPE) -DOS=OS_$(OSTYPE) -D_RCSID
-DOSVER=OSVER_$(OSVERTYPE) -g -DDEBUG=1 -DDB_ORACLE_SEQ=0 -DSECURITY=acsec_domestic
-DEXPORT_PRODUCT=0 -DBUILD_DLL=1 -pic

CPPSUFFIX      = cpp

# PS_HOME should be setup as a environment already
SDK_HOME       = $(PS_HOME)/sdk

PS_LIBS_PATH   = -L$(PS_HOME)/lib
NS_LIBS_PATH   = $(PS_HOME)/lib
GCC_LIBS_PATH  = $(PS_HOME)/lib

PS_INCLUDE        = -I$(SDK_HOME)/common/private -I$(SDK_HOME)/common/public
ROGUEWAVE_INCLUDE = -I$(SDK_HOME)/common/private/roguewave/s-220
VB_INCLUDE        = -I$(SDK_HOME)/common/private/Orbeline2.0/s-156/include \
           -I$(SDK_HOME)/common/private/eventservice/include

PS_LIBS    =  $(PS_LIBS_PATH) -lpsagent_axsdk10 -lpsagent_client10 -lpsagent_server10
-lmember_client10 -lmemcommon_client10 -lassets_client10 -laecore_client10
-lacsrch_client10 -lpsagent_dbaccess10 -lsysmgmt_server10 -lsysmgmt_util10
-lmemcommon_dbaccess10 -lsearch_client10 -laecore_server10 -laecore_client10
-lcorba_client10 -laecore10 -laccore10 -lpsagent_client10 -lcrypto10

BILL_LIBS   =  $(PS_LIBS_PATH) -lbilling_client10 -lcorba_client10 -lacsrch_client10
-laecore_client10 -lmember_client10 -lmemcommon_client10  -laecore10 -laccore10 -lcrypto10

PS_MEM_LIBS = $(PS_LIBS_PATH) -lsdk_membership10 -lsdk_assets10 -lpscmd_cutils10
-laclsvc_client10 -lcorba_client10 -lmember_client10 -lassets_client10 -laclsvc_client10
-lbilling_client10 -lmemcommon_client10 -lacsrch_client10 -laecore_client10
-lmemcommon_dbaccess10 -laecore10 -lcrypto10 -laccore10

ORACLE_LIBS =

RW_LIBS     =  -L$(PS_HOME)/lib -ldbt7d -ltls7d -lmny7d -ldl

SYS_LIBS     =  -lnsl -lsocket

VB_LIBS      =  -L$(PS_HOME)/lib/ -lorb_r -lposix4 -levent
```

```
EXT_LIBS        = \
-L$(NS_LIBS_PATH) -lsec-us -lxp $(NS_LIBS_PATH)/libdbm.a \
$(NS_LIBS_PATH)/libnspr.a $(NS_LIBS_PATH)/libares.a $(NS_LIBS_PATH)/libsslio.a \
-lposix4 -L$(GCC_LIBS_PATH) -lgcc -ldl $(USER_LIB)/libresolv.so \
-L$(LDAP_LIBS_PATH) -ladmin -lframe -laccess -lbase -lsi18n -llldapu \
$(LDAP_LIBS_PATH)/libldap10.so $(PS_LIBS_PATH)/libssldap10.a \
$(LDAP_LIBS_PATH)/liblcache10.so

BILL_EXT_LIBS = $(EXT_LIBS)

##################################################
# DO NOT EDIT BELOW THIS PART
##################################################

COMMON_EXTENSION   = $(SDK_HOME)/common/public
psagent_server     = $(COMMON_EXTENSION)/pubsys/psagent/idlpsagent/server/public
aecore_server      = $(COMMON_EXTENSION)/acaecore/idlacaecore/server/public
psir_common        = $(COMMON_EXTENSION)/pubsys/psir/common/public
search_client      = $(COMMON_EXTENSION)/pubsys/psir/search/idlsearch/client/public
member_client      = $(COMMON_EXTENSION)/membership/member/idlmember/client/public
assets_client      = $(COMMON_EXTENSION)/membership/assets/idlassets/client/public
aclsvc_client      = $(COMMON_EXTENSION)/membership/aclsvc/idlaclsvc/client/public
memcommon_client   = $(COMMON_EXTENSION)/membership/common/idlcommon/client/public
aecore_client      = $(COMMON_EXTENSION)/acaecore/idlacaecore/client/public
acsrch_client      = $(COMMON_EXTENSION)/accommon/acsrch/idlsrch/client/public
corba_client       = $(COMMON_EXTENSION)/accommon/corba/client/public
psagent_dbaccess   = $(COMMON_EXTENSION)/pubsys/psagent/dbaccess/public
memcommon_dbaccess = $(COMMON_EXTENSION)/membership/common/dbaccess/public
pubsys_errmsg      = $(COMMON_EXTENSION)/pubsys/common/errmsg/public
search_common      = $(COMMON_EXTENSION)/pubsys/psir/search/common/public
aecore             = $(COMMON_EXTENSION)/acaecore/dbaccess/public
accore             = $(COMMON_EXTENSION)/accore/public
crypto             = $(COMMON_EXTENSION)/crypto/public
psagent_client     = $(COMMON_EXTENSION)/pubsys/psagent/idlpsagent/client/public
aecore_client      = $(COMMON_EXTENSION)/acaecore/idlacaecore/client/public
bill_include       = $(COMMON_EXTENSION)/billing/public
licence            = $(COMMON_EXTENSION)/pubsys/licence/public
sdk_member         = $(SDK_HOME)/membership/member/public
sdk_assets         = $(SDK_HOME)/membership/assets/public
sdk_aclsvc         = $(SDK_HOME)/membership/aclsvc/public

misc         = $(COMMON_EXTENSION)/misc/
```

Index