R E F E R E N C E G U I D E

Integrated Services™
DIRECTORY

Version 4.0

Version 1.1 of Manual
Published October 16,1998

Software.com™
THE INTERNET INFRASTRUCTURE COMPANY™

Document number: R-981016

Table of Contents

Preface

This manual provides information on the Integrated Services Directory and its role in the InterMail system. This document assumes you have a technical background as well as a working knowledge of InterMail.

This manual is supplemented by the InterMail manual set, including the *InterMail Operations Guide* and *InterMail Reference Guide*.

Overview of the Manual

The content of this manual is organized as follows:

- Chapter 1 provides an introduction to the Integrated Services Directory, including the structure of this directory, and the methods of moving data in and out of the Integrated Services Directory database.
- Chapter 2 discusses the specific data objects that are managed by the Integrated Services Directory, including domains, e-mail accounts, classes of service, and InterManager objects.
- Chapter 3 covers the administrative commands that can be used to create, modify, and delete objects in the Integrated Services Directory.
- Chapter 4 describes the relationship, tables, and schema of the Integrated Services Directory.
- Chapter 5 contains information on the InterMail C API, which can be used to create applications that access data in the Integrated Services Directory.
- Chapter 6 contains information on the InterMail Perl API, which can be used to create Perl scripts that access data in the Integrated Services Directory.
- Chapter 7 contains information on the InterManager Perl API, which can be used to access and manipulate LDAP data used in the InterManager product.
- Chapter 8 provides instructions for administering the Integrated Services Directory database, including monitoring the database and expanding the system tablespace.
- Chapter 9 discusses the tasks required to back up the Integrated Services Directory, and subsequently restore from backup.

Style and Conventions

To assist you in understanding the material presented in this manual, the following conventions have been observed:

- Commands and configuration options are referenced by their proper names.
- Environment variables (whose value is set at time of installation) are referenced with a preceding "`$`" (e.g., `$spoolDir`).
- Commands (and other entries you might type) appear in `monospaced` type.
- Variable names for elements within a command either appear between <angle brackets>.

- Optional entries within a command appear in [square brackets].
- Optional entries separated by a vertical bar (pipe) as in [option 1 | option 2] are exclusive—you can choose only one item from such a list.
- Optional entries followed by an ellipsis (…) can be repeated. When an ellipsis follows a bracketed set, the expression within the brackets can be repeated.
- {Curly braces} surround a list of options, one of, which is required as an argument.
- **Boldface** indicates literal input used in an example

Questions and Comments

To suggest improvements or provide feedback on the content of this manual, send E-mail to `ISD.Manual@Software.com`.

Legal Notices

Trademarks

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this documentation, and Software.com was aware of a trademark claim, the designations have been printed in initial caps or all caps.

InterMail and Software.com are trademarks of Software.com, Inc.

Licensing Agreement

The RSA Data Security, Inc. MD5 Message-Digest Algorithm

License is also granted to make and use derivative works provided that such works are identified as “derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm” in all material mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either the merchantability of this software or the suitability of this software for any particular purpose. It is provided “as is” without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this documentation and/or software.

RSA Data Security, BSAFE

InterMail incorporates a derivative work of the BSAFE cryptographic toolkit, copyright 1992-1996, RSA Data Security, Inc. All rights reserved.

BSAFE is a trademark of RSA Data Security, Inc.

The RSA Public Key Cryptosystem is protected by U.S. Patent #4,405,829.

SSL Plus: SSL 3.0 Integration Suite Toolkit

InterMail incorporates a derivative work of the SSL Plus: SSL 3.0 Integration Suite Toolkit, copyright 1996, 1997 Consensus Development Corporation. SSL Plus: SSL 3.0 Integration Suite is a trademark of Consensus Development Corporation, which reserves all rights thereto.

Portions of the SSL Plus: SSL 3.0 Integration Suite Toolkit software are based on SSLRef(tm) 3.0, which is copyright (c)1996 by Netscape Communications Corporation. SSLRef(tm) was developed by Netscape Communications Corporation and Consensus Development Corporation.

The Regular Expression Routines

The Regular Expression Routines used in InterMail are © 1992-94 Henry Spencer. All rights reserved. This software is not subject to any license of the American Telephone and Telegraph Company or of the Regents of the University of California.

Permission is granted to anyone to use this software for any purpose on any computer system, and to alter it and redistribute it, subject to the following restrictions:

1. The author is not responsible for the consequences of use of this software, no matter how awful, even if they arise from flaws in it.
2. The origin of this software must not be misrepresented, either by explicit claim or by omission. Since few users ever read sources, credits must appear in the documentation.
3. Altered versions must be plainly marked as such, and must not be misrepresented as being the original software. Since few users ever read sources, credits must appear in the documentation.
4. This notice may not be removed or altered.

The Regents of the University of California Copyright

InterMail includes software that is © 1990, 1993, 1994. The Regents of the University of California. All rights reserved.

This code is derived from software contributed to Berkeley by Mike Olson.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Re-distributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Re-distributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgment: This product includes software developed by the University of California, Berkeley and its contributors.
4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

GNU General Public License

Version 1, February 1989

Copyright (C) 1989 Free Software Foundation, Inc.

675 Mass Ave., Cambridge, MA 02139, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The license agreements of most software companies try to keep users at the mercy of those companies. By contrast, our General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. The General Public License applies to the Free Software Foundation's software and to any other program whose authors commit to using it. You can use it for your programs, too.

When we speak of free software, we are referring to freedom, not price. Specifically, the General Public License is designed to make sure that you have the freedom to give away or sell copies of free software, that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of a such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING,

DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The “Program”, below, refers to any such program or work, and a “work based on the Program” means either the Program or any work containing the Program or a portion of it, either verbatim or with modifications. Each licensee is addressed as “you”.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this General Public License and to the absence of any warranty; and give any other recipients of the Program a copy of this General Public License along with the Program. You may charge a fee for the physical act of transferring a copy.

2. You may modify your copy or copies of the Program or any portion of it, and copy and distribute such modifications under the terms of Paragraph 1 above, provided that you also do the following:

a) cause the modified files to carry prominent notices stating that you changed the files and the date of any change; and

b) cause the whole of any work that you distribute or publish, that in whole or in part contains the Program or any part thereof, either with or without modifications, to be licensed at no charge to all third parties under the terms of this General Public License (except that you may choose to grant warranty protection to some or all third parties, at your option).

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the simplest and most usual way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this General Public License.

d) You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

Mere aggregation of another independent work with the Program (or its derivative) on a volume of a storage or distribution medium does not bring the other work under the scope of these terms.

3. You may copy and distribute the Program (or a portion or derivative of it, under Paragraph 2) in object code or executable form under the terms of Paragraphs 1 and 2 above provided that you also do one of the following:

a) accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Paragraphs 1 and 2 above; or,

b) accompany it with a written offer, valid for at least three years, to give any third party free (except for a nominal charge for the cost of distribution) a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Paragraphs 1 and 2 above; or,

c) accompany it with the information you received as to where the corresponding source code may be obtained. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form alone.)

Source code for a work means the preferred form of the work for making modifications to it. For an executable file, complete source code means all the source code for all modules it contains; but, as a special exception, it need not include source code for modules which are standard libraries that accompany the operating system on which the executable file runs, or for standard header files or definitions files that accompany that operating system.

4. You may not copy, modify, sublicense, distribute or transfer the Program except as expressly provided under this General Public License. Any attempt otherwise to copy, modify, sublicense, distribute or transfer the Program is void, and will automatically terminate your rights to use the Program under this License. However, parties who have received copies, or rights to use copies, from you under this General Public License will not have their licenses terminated so long as such parties remain in full compliance.

5. By copying, distributing or modifying the Program (or any work based on the Program) you indicate your acceptance of this license to do so, and all its terms and conditions.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein.

7. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of the license which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the license, you may choose any version ever published by the Free Software Foundation.

8. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

9. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

10. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to humanity, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>

Copyright (C) 19yy <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 1, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave., Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) 19xx name of author

Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.

This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program `Gnomovision' (a program to direct compilers to make passes at assemblers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989

Ty Coon, President of Vice

Oracle

Oracle Programs are the proprietary products of Oracle and are protected by copyright and other intellectual property laws. Customer acquires only the right to use Oracle Programs and does not acquire any rights, express or implied, in Oracle Programs or media containing Oracle Programs other than those specified by License. Oracle, or its licensor, shall at all times retain all rights, title, interest, including intellectual property rights, in Oracle Programs and media.

SmartHeap

EMANATE

InterMail incorporates the EMANATE server as part of its monitoring functionality. Software.com licenses EMANATE pursuant to a license agreement with SNMP Research International, Incorporated. The copying and distribution of EMANATE is with the permission of SNMP Research International, Incorporated.

Apache Server License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgment:

"This product includes software developed by the Apache Group for use in the Apache HTTP server project (http://www.apache.org/)."

4. The names "Apache Server" and "Apache Group" must not be used to endorse or promote products derived from this software without prior written permission.
5. Redistributions of any form whatsoever must retain the following acknowledgment:

"This product includes software developed by the Apache Group for use in the Apache HTTP server project (http://www.apache.org/)."

THIS SOFTWARE IS PROVIDED BY THE APACHE GROUP ''AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE GROUP OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT

NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)

ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the Apache Group and was originally based on public domain software written at the National Center for Supercomputing Applications, University of Illinois, Urbana-Champaign. For more information on the Apache Group and the Apache HTTP server project, please see <http://www.apache.org/>.

1

Introduction

This chapter provides an introduction to the Integrated Services Directory, and includes the following information:

- The structure of the Integrated Services Directory.
- The role of the Integrated Services Directory in the InterMail system.
- The distribution of Integrated Services Directory data to the various InterMail, LDAP, and InterManager components.
- The available methods for creating and modifying data in the Integrated Services Directory.

1.1 Directory Structure

The Integrated Services Directory is a group of data objects controlled by products in the InterMail system. These objects include e-mail accounts, users, and organizations, which are used by LDAP and the InterMail and InterManager, products to provide end-user services. As a centralized directory, the Integrated Services Directory is at the heart of the InterMail system.

The following diagram illustrates the structure of the Integrated Services Directory:

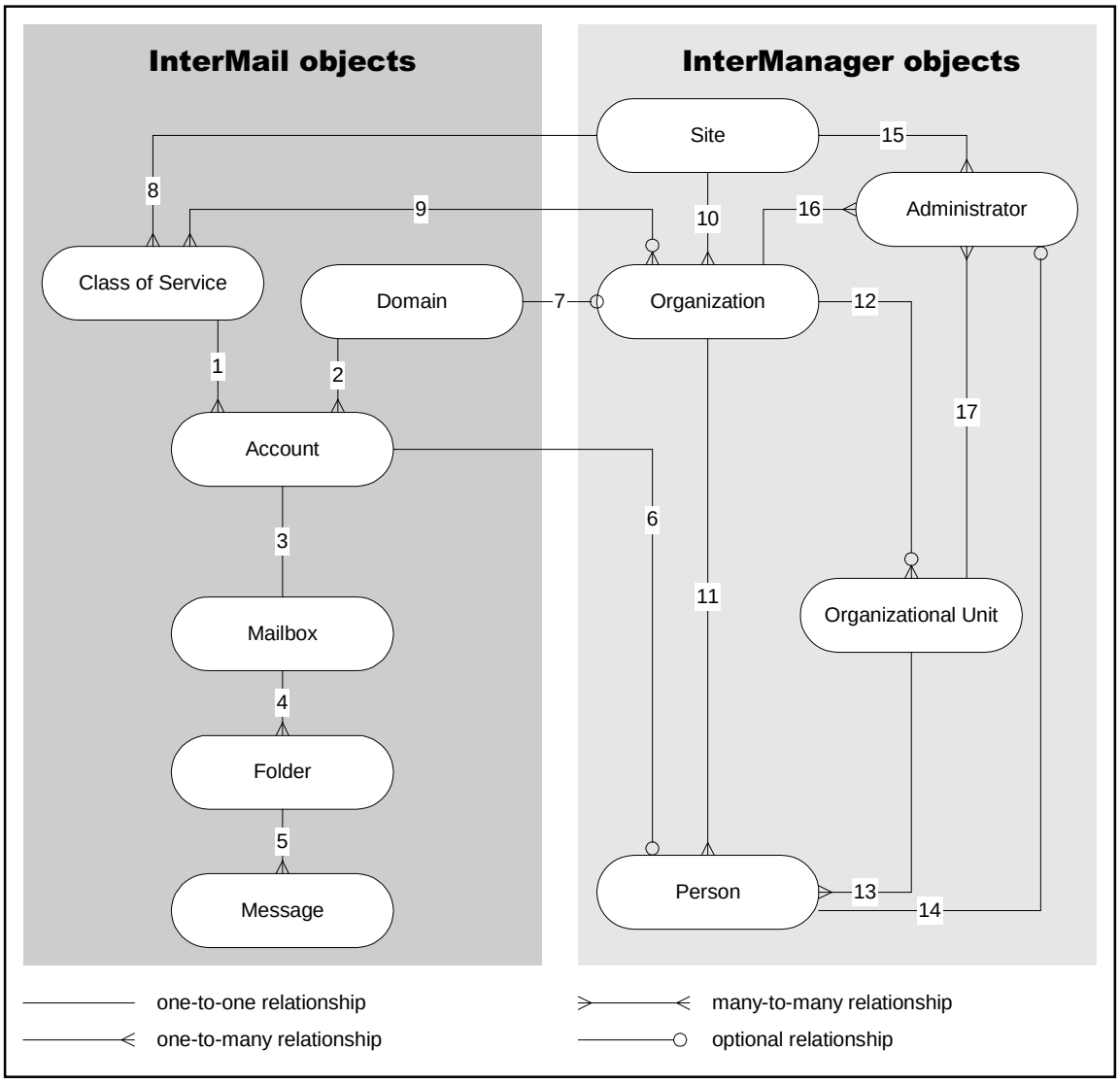


Figure 1. The structure of the Integrated Services Directory.

The relationships between these objects are as follows:

1. Each InterMail *account* is associated with a particular *class of service*. This class of service defines certain attributes of the e-mail account, such as the account's mailbox quotas and the InterMail services (IMAP access, authenticated SMTP, etc.) that are used by the account. Typically, only a few classes of service are defined, with many accounts associated with each class of service.
2. Each account is also associated with a specific *domain*. The name of this domain is included in the account's primary e-mail address. An account can optionally be associated with additional domains through the use of alias and forwarding address.
3. Each account typically has an associated *mailbox*, which is a container for the folders in which messages received for the account are stored.

4. Each mailbox contains a series of *folders*, which contain the messages received for the account. All mailboxes include an `INBOX` folder, which contains all new messages sent to the account, as well the folders `SentMail` and `Trash`, which can be used by IMAP clients to manage sent and deleted messages, respectively. IMAP clients can create new folders within a mailbox, or within other folders.
5. Each folder can contain many *messages*, which are the actual e-mail messages that are sent to an account. A single message can be stored in multiple folders within a mailbox.
6. An account may be optionally associated with an InterManager *person* object. A person object is the InterManager equivalent to an InterMail account, and corresponds to an individual e-mail user. Although an account is not required to be associated with a person object, all person objects must have a corresponding account.
7. A domain may be optionally associated with an InterManager *organization*. An organization is analogous to a top-level domain, and typically corresponds to a company that purchases e-mail from the service provider. Although a domain is not required to be associated with an organization, all organizations must have a corresponding domain.
8. Within InterManager, classes of service are associated with the *site* object, which is the highest-level InterManager object. This means that classes of service are not contained within organizations or other lower-level objects.
9. Each InterManager organization is associated with one or more classes of service. These classes of service define the e-mail account types that may be defined within the organization. This allows the service provider to allocate blocks of e-mail accounts according to the services provided to those accounts.
10. All organizations are contained within the site object, the highest-level InterManager object. Organizations typically correspond to companies or other top-level organizations that purchase e-mail access from the service provider. A site may include many organizations, but there is one—and only one—site object.
11. Organizations contain person objects, which define the users whose accounts are managed via InterManager. Each person object is a member or a specific organization, which can contain many person objects.
12. Organizations may optionally contain multiple *organizational units*. These organizational units define sub-levels within an organization, such as a department within a company. All organizational units are associated with a specific organization.
13. Organizational units may contain person objects. This association typically denotes that an individual e-mail user works within a specific department of an organization.
14. A person may optionally be an *administrator* for an organization, an organizational unit, or the site. Each of these objects may have multiple administrators, who each have authority over the objects within their organization, organizational unit, or site.
15. Site administrators have authority over top-level objects, such as domains, classes of service, and organizations.
16. Organization administrators have authority over the organizational units and e-mail accounts within their organization.
17. Organizational unit administrators have authority over the e-mail accounts within their organizational unit.

Chapter 2 of this manual provides more information on the data types in the Integrated Services Directory.

1.2 Role of the Integrated Services Directory

The Integrated Services Directory is the definitive source of all directory information for the InterMail and InterManager products. These components continuously refer to the Integrated Services Directory to determine such things as mail routing instructions, delivery options for a specific account, LDAP white pages information, and access levels for InterManager administrators.

Note: *Configuration information for InterMail and InterManager components is stored in the Configuration Database, and not in the Integrated Services Directory.*

Server components do not access the Integrated Services Directory directly. Instead, information in the Integrated Services Directory is distributed by database replication servers, which provide high-speed access to Integrated Services Directory data.

1.3 The Integrated Services Directory Database

The Integrated Services Directory is stored in an Oracle RDB. This database provides journaling, redo logs, and other important backup and recovery mechanisms.

Within its Oracle RDB, the Integrated Services Directory maintains specific data in a Lightweight Directory Access Protocol (LDAP) database format. This format provides a hierarchical data structure that is used to store objects such as organizations, organizational units, and mail policies, which have hierarchical relationships.

1.4 Distributing Directory Information

Because the Integrated Services Directory provides a centralized directory database, all requests for user information and authentication must go through this database. The centralized nature if the Integrated Services Directory provides simplified administration. However, providing only one source of this information would create a bottleneck on the system.

For this reason, the Integrated Services Directory uses database replication servers to distribute directory data. Database replication servers maintain their own copies of information that is stored in the directory database, and each responds to requests from InterMail components to validate or retrieve information. By using multiple replication servers, the InterMail system provides scalability and eliminates bottlenecks on requests for information from the Integrated Services Directory.

Note: *Database replication servers are the only components that communicate directly with the Integrated Services Directory. All other components get Integrated Services Directory information via the database replication servers.*

1.4.1 Directory Cache Server

The main database replication server is the Directory Cache Server. The Directory Cache Server is used by the InterMail and InterManager products to retrieve and validate Integrated Services Directory information. The InterMail system can include multiple Directory Cache Servers, and typically includes at least two.

Whenever an InterMail or InterManager component needs to access information that is stored in the Integrated Services Directory, it first contacts one of these Directory Cache Servers for the information. If the Directory Cache Server can provide the requested information, then it does so. The Directory Cache Server makes a request directly to the Integrated Services Directory only if it does not include contain the requested information or cannot verify authentication data.

For example, if the POP Server requests authentication for a user, and the Directory Cache Server cannot authenticate the user because of an incorrect login name or password, a second authentication request is made by the Directory Cache Server directly to the Integrated Services Directory to verify the login information. This prevents a user request from being denied because the information maintained by the Directory Cache Server is out-of-date.

The information maintained by the Directory Cache Server is updated from the Integrated Services Directory at a configurable interval (by default, 60 seconds). Updates are also made if a read-through to the Integrated Services Directory reveals that the information in the Directory Cache Server is out-of-date with the directory database.

An InterMail system can contain as many Directory Cache Servers as necessary to support the lookup throughput needed by other system components. The Configuration database stored on each host in the system contains a list of available Directory Cache Servers. A server that attempts to access a Directory Cache Server initially tries the first entry on its list (the primary Directory Cache Server). If communication with this cache server fails for some reason, it will go down the list until it finds one that is active. Clients periodically attempt to reconnect with their primary Directory Cache Server. Using this mechanism, the load on the Directory Cache Servers can be spread across all available servers, while still supporting redundancy.

1.5 Accessing Integrated Services Directory Data

There are three basic methods available for creating, viewing, modifying, and deleting data in the Integrated Services Directory:

- The SelfCare and InterManager web interfaces
- Scripts that invoke the database-related administrative commands `imdbcontrol` and `imldapcontrol`.
- Programs that use the InterMail or InterManager API libraries.

The following sections describe these three methods. Additional methods are available for batch loading information, which is typically done only when setting up accounts during initial system setup. The tools available for these setup operations are discussed in the respective product manuals.

1.5.1 SelfCare Web Interface

The SelfCare component of InterMail includes an HTML-based user interface that can be executed within a web browser. This web interface can be accessed by end users to manage Integrated Services Directory data related to their e-mail accounts.

1.5.2 InterManager Web Interface

Like SelfCare, the InterManager product includes a web interface to Integrated Services Directory data. This interface can be accessed by a variety of administrators to manage e-mail accounts and other data. InterManager provides delegated administration, which allows certain end users to manage e-mail accounts within a particular organization or organizational unit.

For more information on the InterManager web interface, refer to the *InterManager Administration Guide*.

1.5.3 Administrative Commands

Two administrative commands provide general access to Integrated Services Directory data: `imdbcontrol` and `imldapcontrol`. These commands are installed with the Integrated Services Directory, and can perform a variety of actions (create, modify, delete, etc.) on all data stored in the directory database.

As with other InterMail administrative commands, `imdbcontrol` and `imldapcontrol` can be executed manually at the UNIX command-prompt. However, these commands are typically executed by scripts that are invoked by a site's existing provisioning system. For example, the program used by a customer service representative to create new e-mail accounts might invoke a script that executes an `imdbcontrol CreateAccount` operation.

The `imdbcontrol` command operates on mail objects in the Integrated Services Directory. These objects include e-mail accounts and domains. These are the objects used by InterMail.

Meanwhile, the `imldapcontrol` command operates on the LDAP objects in the Integrated Services Directory. These LDAP objects are described in Chapter 2, and include organizations, organizational units, and persons.

Note: *Both* `imdbcontrol` *and* `imldapcontrol` *are described in detail in Chapter 3.*

1.5.4 API Libraries

The Integrated Services Directory package includes a set of API libraries that allow you to implement your own interface to the directory database. This allows an ISP to integrate InterMail with its existing billing system and customer database, or create its own customized interface.

The InterMail API is available as both C and Perl libraries. The functions in these libraries allow the programs that call them to directly create, modify, or delete objects in the Integrated Services Directory. Refer to Chapters 5 and 6 of this manual for more information on these APIs.

There is also an InterManager Perl API that is used specifically to operate on InterManager data. This API is documented in the Chapter 7 of this manual.

2

Data Types

This chapter describes the various objects that are stored in the Integrated Services Directory, and provides a list of object attributes for each. This includes the following data types:

- domains
- e-mail accounts
- classes of service
- InterManager LDAP objects

2.1 Domains

The first InterMail objects that must be defined in the Integrated Services Directory are domains. The domains defined in the Integrated Services Directory—along with the `MX` records in your DNS—determine the domains for which your site receives mail. All other InterMail objects, such as accounts and forwarding addresses, are associated with a domain.

2.1.1 Types

There are four types of domains in InterMail: local, rewrite, non-authoritative, and deleted. InterMail accounts and addresses may be associated only with local and non-authoritative domains; no accounts or addresses can be associated with rewrite or deleted domains.

Local Mail Domains

A *local domain* is a domain for which InterMail claims exclusive control. If a domain (or subdomain) is defined as a local domain for your site and mail is received for an address within that domain, InterMail considers itself the ultimate destination for that mail. Messages that are addressed to non-existent recipients within a local domain are considered undeliverable, and will be returned to sender.

Most InterMail e-mail accounts and SMTP aliases are associated with a local domain.

Non-authoritative (Semi-local) Domains

A *non-authoritative domain* (also known as a *semi-local domain*) is similar to a local domain, but does *not* define the InterMail system as the definitive destination for all mail addressed to that domain. Non-authoritative domains allow InterMail to accept mail for a domain, but relay it to another mail host if necessary. Non-authoritative domains can be established to define domains for which your site is an MX backup, or used when InterMail is run in parallel with an existing mail system (i.e., during migration of e-mail accounts from a legacy mail system to InterMail).

All non-authoritative domains are associated with the name of a mail host. When mail is received for addresses in a non-authoritative domain, and the recipient is not found in the directory database, the mail is relayed to the host associated with this domain.[1]

For example, say `softwarenow.com` is created as a non-authoritative domain, and is associated with the host name `jupiter.software.com`. When mail arrives for an address within this domain (e.g., `jane.doe@softwarenow.com`), InterMail first checks for the existence of this address in the Integrated Services Directory. If a match is not found, the message is relayed to the mail server on `jupiter.software.com`.

Note: *When mail addressed to a non-authoritative domain is relayed, the operation affects mail routing only. No changes are made to the original address information of the message because of this routing.*

Rewrite Domains

Rewrite domains are very different from local and non-authoritative domains, as they do *not* define domains for which InterMail receives mail. A rewrite domain simply defines a rule for rewriting the recipient address of incoming mail, and must be associated with a local or non-authoritative domain. E-mail accounts or SMTP aliases *cannot* be associated with a rewrite domain.

When an incoming message is received, and the domain of the recipient address is defined as a rewrite domain, the address is rewritten to include the local or non-authoritative domain associated with the rewrite domain. This allows InterMail accounts to receive messages addressed to the same user in multiple domains without requiring SMTP aliases for each account.

For example, say `accordance.com` is created as a rewrite domain, and is associated with the local mail domain `software.com`. When mail arrives for `joe.schmoe@accordance.com`, InterMail rewrites this address to `joe.schmoe@software.com`, and then attempts to deliver the message to the rewritten address.

Note: *By default, domain rewriting affects only the envelope address of an incoming message. However, rewrite domains may also be used for header rewriting, which is described in Chapter 5 of the InterMail Operations Guide.*

Deleted Domains

A *deleted domain* is a domain that was previously a local, non-authoritative, or rewrite domain, but which was marked for deletion. A deleted domain cannot have accounts or SMTP aliases associated with it, but continues to exist in the Integrated Services Directory until it is explicitly removed.

Note: *Whenever possible, domains should be marked for deletion instead of permanently removed from the Integrated Services Directory. This allows the MTA and other InterMail components to more easily accommodate the change to the list of mail domains.*

---

[1] The relay host associated with a non-authoritative domain presumably hosts the e-mail server that is the definitive destination for mail addressed to that domain.

2.1.2 Wildcard Delivery

All local domains can have an optional *wildcard account* associated with them. A wildcard account is simply a normal e-mail account within the domain that receives all mail that is sent to non-existent addresses within the domain. This feature allows you to collect all mail sent to a particular domain in a single account.

For example, if the account `john.doe@software.com` is defined as a wildcard account for the local mail domain `software.com`, then any message sent to a non-existent address within `software.com` will be delivered to `john.doe@software.com`.

Note: *Delivery to a wildcard account is, from InterMail's perspective, a last resort; it occurs only when the destination address of a message does not exist as an account primary address or as an SMTP alias.*

Wildcard delivery can be enabled or disabled for an existing local domain at any time. However, a domain can only have one wildcard account associated with it at any time. To change the wildcard account for a domain, you must disable wildcard delivery before defining the new account.

2.1.3 Default Domains

There are two different default domains in InterMail. The first is the default domain in the Integrated Services Directory, while the second is the default MTA domain defined in the configuration database.

Note: *The two default domains are completely unrelated. There is no requirement that these values match, or even that both domains exist in the Integrated Services Directory.*

Integrated Services Directory

The Integrated Services Directory default domain is a convenience for creating e-mail addresses and accounts. When a default domain is specified, you can omit a domain name when creating new addresses or accounts; the default domain will be assumed in this case.

For example, the `imdbcontrol` utility (described in Chapter 3) requires a domain name as a parameter for several of its operations. If you define a default domain in the Integrated Services Directory, you can omit domain name parameters when executing certain `imdbcontrol` operations.

Configuration Database

A second default domain is defined in the configuration database with the MTA configuration key `defaultDomain`. The domain defined by this configuration key is used by the MTA as the default domain for several operations, including address completion.

Refer to Chapter 5 of the *InterMail Operations Guide* for more information on the MTA default domain, defined in the configuration database.

2.2 Accounts

Information about the users who receive e-mail at your site is organized by InterMail accounts. The information included in an e-mail account identifies the recipient, determines how messages are accepted and delivered for that account, and whether or not the account makes use of any special features (such as auto-reply).

The following sections describe the attributes associated with each InterMail account.

2.2.1 General Account Data

The basic information associated with each InterMail account includes the name of the user, the domain with which the account is associated, the user's password, the type of the account, and the status of the account. All accounts include information for these account attributes.

Username

The *username* of an account typically reflects the name of an individual user who will use the account. This name is the part of the account's e-mail address that precedes the "@" symbol. When the username and domain of an account are combined, they form the primary e-mail address of the account.

For example, an account whose primary address is `susie.queue@software.com` has the username `susie.queue`.

Usernames can be up to 64 characters in length. They can include all alphanumeric characters, as well as the underscore (_), period (.), and hyphen (-) characters. Usernames are not case-sensitive, so the usernames `John.Doe`, `john.doe`, and `JOHN.DOE` are identical. Because two accounts cannot share the same e-mail address, usernames must be unique within a domain.

Password

The account password is used by e-mail clients when connecting to the POP or IMAP server to retrieve mail from the account's mailbox. The password is also used to allow end user access to the InterManager web interface.

Passwords can be up to 64 characters in length, can include all printable characters, and are case-sensitive.

Domain

All InterMail accounts are associated with a specific domain. The domain of an account is specified when the account is created, and must be an existing local or non-authoritative domain. The username and the domain name together define the primary e-mail address of an InterMail account.

Class of Service

Each e-mail account is associated with a particular class of service. Except for the general account data described in this section, most account attributes can be configured at the class of service level. This allows account-related features—such as message retrieval, mailbox quotas, and end user access to the InterMail web interfaces—to be defined once for large numbers of accounts. These attributes can then be set for all accounts in the class of service by simply changing the value of the attribute for the class of service.

For more information on the interaction of accounts and classes of service, refer to Section 2.3, which also includes a list of account attributes that can be set on the class of service level.

Type

There are two types of InterMail accounts: *standard* and *administrative*. The type of an account has no effect on InterMail behavior, so an administrative account has no more (or less) access to InterMail services than a standard account. However, this information can be used in conjunction with a billing or provisioning system to indicate certain information. For example, you may create a rule in your billing system to ignore all administrative accounts when generating usage and billing information.

Status

Each account has an associated status, which defines the current state of the account. There are six possible account status values:

- **Active**. This is the most common account status, and indicates that the account is in a normal state. An active account is permitted to send and receive messages normally.
- **Maintenance**. The maintenance status is used when an account's mailbox is temporarily unavailable. For example, if an MSS database is shut down (to expand the tablespace, reorganize indexes, etc.), the mailboxes in that database cannot accept new messages, and cannot answer client requests for mail. Messages that arrive for accounts in maintenance mode are queued internally, and are delivered normally when the status of the account is reset to active. When requests are made to download mail from the POP or IMAP server, the client is rejected with a message indicating that the account is in maintenance mode and is temporarily unavailable.
- **Suspended**. The suspended status is used to prevent access to the account's mailbox, and is typically used when the account's user has not paid his or her bill for e-mail access. When an account is suspended, mail sent to that account is returned to sender, and user requests to download mail from the account via the POP or IMAP server are rejected with an unknown username/password error. Setting the status to active can easily restore normal delivery and client access to a suspended account.
- **Locked**. The locked status is identical to the suspended status, preventing user access to the mailbox and returning all mail sent to the account. This status type is supported for backward compatibility only.

- **Deleted**. Setting an account's status to deleted completely halts mail activity for that account. Mail addressed to the account is treated as undeliverable, and client requests to access the account's mailbox via the POP or IMAP servers are rejected with an unknown username/password error. Unlike permanently deleting an account from the Integrated Services Directory, setting an account's status to deleted allows you to later restore the account by resetting its status.
- **Proxy**. The proxy status indicates that mail activity for the account is being handled by another mail system. This status is used when migrating accounts from an existing mail system to InterMail. Incoming mail that is received for an account in proxy mode is redirected to the proxy MTA defined for the account. Similarly, client requests to retrieve message via the POP or IMAP server are redirected to account's proxy POP/IMAP server. This redirection is transparent, so end users can specify InterMail hosts as their SMTP and POP/IMAP server while their accounts remain on a legacy mail system.

Note: *When the status of an account is proxy, the purpose of two account attributes—the MSS host and auto-reply host—is redefined. Instead of their typical use, these attributes are used to define the proxy SMTP and POP server, respectively, for the proxy account.*

2.2.2 E-mail Addresses

All accounts have at least one e-mail address, known as the *primary address*. Accounts may also have additional addresses, or *SMTP aliases*.

Primary Address

The primary address is the "official" Internet e-mail address of an account. Although SMTP alias addresses are equally valid for sending mail to an account, the primary address is the only address used to identify the account in operations related to the Message Store database and the Integrated Services Directory.

The primary e-mail address of an InterMail account is defined implicitly by combining the username of the account with its domain. For example, if an account has the username `jane.doe`, and was created with the domain `software.com`, then the primary address of this account is `jane.doe@software.com`.

SMTP Aliases

SMTP aliases are additional e-mail addresses for an account. Mail addressed to an SMTP alias is delivered to the account exactly as if it had been addressed to the account's primary address. As with primary addresses, SMTP aliases must be unique throughout the system.

Aliases allow individual accounts to receive mail at multiple domains, and to use multiple addressing formats. For example, an account with the primary address `john.doe@software.com` might have the following SMTP aliases:

```
john.doe@softwarenow.com
jdoe@software.com
```


Alias Limit

Each account also includes an optional limit on the number of SMTP aliases that the account can include. Because end users cannot define SMTP alias addresses themselves, this limit is intended to prevent InterManager administrators from creating an unlimited number of alias addresses.

2.2.3 Delivery Information

Each account includes a series of attributes that determine the account's method of mail delivery. The available types of mail delivery are *local* and *forwarding*. Accounts may use one or both of these delivery methods. Any account that uses local delivery also has an associated *mailbox*.

Mailbox

InterMail accounts are typically associated with a *mailbox*, a storage area for mail sent to the account. The information stored in the Integrated Services Directory for each account includes the name of a specific MSS host—this is the host on which the account's mailbox is stored. It is to this mailbox that messages sent to the account are delivered, and from which mail is retrieved by the POP and IMAP servers.

Note: *A mailbox is needed by an account only if it uses the local delivery method. Accounts that use only forwarding delivery do not require or make use of a mailbox.*

Mailboxes are defined in the Message Store database, and not the Integrated Services Directory. A mailbox need not be created for an account when the account itself is created; mailbox are typically created automatically when first needed (that is, when the first message arrives for the account, or during the first user request to retrieve mail).

See Chapter 6 of the *InterMail Operations Guide* for more information on managing mailboxes.

Local Delivery

The most common method of mail handling is local delivery, which stores messages in the account's InterMail mailbox. From this mailbox, messages can be retrieved via the POP Server, IMAP Server, and WebMail. Local delivery can be enabled or disabled for an account at any time.

Note: *Accounts that use local delivery are subject to mailbox quotas, which are described in Section 2.2.5.*

Login Name

All accounts that use local delivery have an associated *login name* (also referred to as the *POP login name*). When retrieving messages via the POP or IMAP server, a user must supply the login name and password for his or her account. Mail in the account's mailbox can be accessed only if the user supplies the correct login information.

There is no requirement that the login name and username of an account be identical, so you may choose any format for specifying login names.

Forwarding

The forwarding method of delivery is similar to the forwarding of postal mail. When mail arrives for an account that uses forwarding delivery, InterMail modifies the message's destination address, and then sends it to the new location.

Forwarding can be enabled or disabled for an account at any time. Accounts that use forwarding delivery must also be associated with one or more forwarding addresses. There is no limit to the number of forwarding addresses for an account.

Mail Filtering

The *mail filtering* account option can be used to prevent the delivery of unsolicited commercial e-mail (also known as "spam") to an account. When this option is enabled for an account, and a message which violates one of InterMail's mail blocking is received for that account, the message will not be delivered.

Note: *Mail filtering simply allows accounts to accept or ignore the entire set of administrator-defined blocking policies. New blocking criteria—such as additional domains and addresses to block—cannot be defined on an account-by-account basis.*

The mail filtering option is useful only if InterMail's mail blocking policies have been configured to operate on a per-account basis. Refer to Chapter 4 of the *InterMail Operations Guide* for information on setting mail blocking policies.

2.2.4 Auto-Reply

InterMail accounts include an optional auto-reply feature. When mail is received for an account that uses the auto-reply feature, InterMail immediately sends the account's auto-reply message to the sender of the original message.

Mode

The auto-reply functionality of an account can operate in three modes:

- **Reply**. The Reply mode sends the account's auto-reply message *every* time mail is sent to the account. This mode is commonly used to automatically distribute information on sales, system policies, directions, etc.
- **Echo**. The Echo mode is the same as Reply, but also includes the sender's original message as a MIME attachment to the auto-reply message.
- **Vacation**. The Vacation mode is similar to Reply, but sends only one copy of the auto-reply message to each sender during the defined auto-reply expire period (by default, one week). This means that if a user sends ten messages within a week to a particular account, and the account uses the Vacation mode of auto-reply, that user will receive only one copy of the account's auto-reply message.[2] This mode is typically used when the account's user is on vacation or otherwise temporarily unavailable.

---

[2] If the user sends another message to the account after the auto-reply expiration period, he or she receives a second copy of the auto-reply message.

Note: *The auto-reply expire period affects only the Vacation mode of auto-reply, and is defined in the configuration key* `autoReplyExpireDays`.

Message

If an account uses the auto-reply feature, an associated auto-reply message must be defined for the account. This auto-reply message defines the contents (that is, the message body) of the automatic response to incoming mail. There is no limit on the size of an auto-reply message.

Note: *An auto-reply message cannot contain a MIME attachment, such as a document or graphics file.*

The auto-reply message associated with an account is *not* stored in the Integrated Services Directory. The information stored for an account in the Integrated Services Directory includes an *auto-reply host*, which specifies the name of am MSS host with which a particular account's auto-reply message is associated. The Message Store Database on that host stores the name and location of the individual auto-reply message file. The file itself is stored in the associated Message File System.

The location of an account's auto-reply message is configured by using the `imreplyctrl` administrative command. This command is described in Chapter 12 of the *InterMail Reference Guide*.

2.2.5 Mailbox Quotas

All InterMail accounts that use the local delivery method have an associated set of mailbox quotas. These quotas are limits on the number and size of messages that can be delivered to the account's mailbox.

Note: *Mailbox quotas are typically set on the class of service level. However, class of service mailbox quota values can be overridden on the account level.*

There are three different quotas for each account:

- **Maximum message size**. This quota limits the size of the largest message that can be delivered to the account.
- **Maximum mailbox size**. This quota limits the total storage size of the account's mailbox.
- **Maximum number of messages**. This quota limits the total number of messages that can be in the account's mailbox at any time.

If delivery of mail to an account's mailbox would violate any of these quotas, the mail is bounced or deferred (depending on the value of the configuration key `bounceOnQuotaFull`). If the over-quota notification service option is enabled, the user also receives a message that describes the problem.

Quota Warning Threshold

To prevent users from exceeding the maximum mailbox size quota for their accounts, InterMail sends a notification to users when their mailboxes reach a certain percentage of that limit. This allows you to alert a user that his or her mailbox is, say, 90% full, and that it may reach its limit if action is not taken. The percentage of the maximum mailbox size quota at which this warning message is sent is set individually for each account.

Over-Quota Notifications

By default, an end user receives a notification message whenever mail cannot be delivered to his or her account because it would cause one of the account's mailbox quotas to be exceeded. However, the sending of these over-quota notifications is configurable. If over-quota notifications are disabled for an account, and delivery of a message is prevented because of an over-quota condition, the end user receives no indication that mail delivery has failed.

2.2.6 Mail System Access

A series of mail access options control the methods available to the user for accessing the various InterMail services. For example, the local delivery attribute of an account controls the delivery of messages to the account's mailbox, while the POP3 service option controls whether or not the user can log in to the POP server to retrieve mail from the mailbox.

The following options define user access to InterMail services:

- **POP3**. This option controls access to standard POP3 service. When this option is enabled, the user is allowed to access the POP server through the standard (non-secure) POP3 port. Because POP3 is the most common method of accessing mail, this option is typically enabled for all accounts that use local delivery.
- **POP3 with SSL**. This option controls POP access using SSL (secure socket layer). When this option is enabled, the user is allowed to access the POP server through the SSL (secure) POP port. A user must have an SSL-compliant POP3 client to use this feature.
- **IMAP4**. This option controls access to standard IMAP service. Because IMAP access usually requires longer connection times and more storage capacity than POP, this option is typically enabled for an account only as a premium service.
- **Authenticated SMTP**. This option specifies that the user *must* provide authentication information when sending mail via SMTP. When this option is enabled, and the user submits mail to an MTA, the message is accepted only if the user provides the appropriate username and password information. If authentication information is not given (or is incorrect), the message is rejected.
- **SMTP**. This option is used only when the Authenticated SMTP option is enabled, and controls user access to sending e-mail via the standard (non-secure) SMTP port.
- **SMTP with SSL**. This option is used only when the Authenticated SMTP option is enabled, and controls user access to sending e-mail via the SSL (secure) SMTP port.

2.2.7 Web Interface Options

Each account includes a series of attributes that define end user options related to the available web interfaces to InterMail. These interfaces are:

- SelfCare, which provides end user control over a limited set of account information.
- WebMail, which provides web-based access to viewing and sending e-mail.
- InterManager, which provides partitioned administration of groups of e-mail accounts.

End user access to each of these interfaces in configurable, and can be granted on an account-by-account basis. An additional series of WebMail-specific options control end user behavior in this web interface.

Login Access

The following account options control end user login access to the available web interfaces:

- **SelfCare access**. Controls the ability of the user to log in to the SelfCare web interface. If this option is not enabled for an account, the end user cannot use SelfCare.
- **InterManager administrative access**. Controls the ability of the user to log into the InterManager web interface as the administrator of a site, organization, or organizational unit. This option should be available only to users who have been designated as the administrator of a site, organization, or organizational unit.
- **WebMail access**. Controls the ability of the user to log into WebMail to access e-mail in his or her mailbox. If this option is not enabled for an account, the end user cannot use WebMail.
- **Bypass authentication**. This option allows the end user to bypass login authentication when accessing SelfCare or WebMail. If this option is enabled for an account, user authentication data is saved in a “cookie” on the client system, allowing the end to access SelfCare and/or WebMail directly without providing a login name and password.

WebMail Options

The following account options and limits control end user behavior within the WebMail client interface:

- **Attachment size limit**. This option sets a limit on the maximum size of file attachments that can be sent by the end user via the WebMail client. If the end user attempts to attach to a message a file that exceeds this size limit, the operation is denied.
- **Message attachment number limit**. This option sets a limit on the number of file attachments that can be sent with a single message.
- **Session attachment number limit**. This option sets a limit on the cumulative number of file attachments that can be sent with messages in a single WebMail client session.
- **Address book entry limit**. This option sets a limit on the number of address book entries that an individual end user may define. When this number of address book entries have been created, no new addresses may be entered until existing addresses are deleted.

- **Address book list entry limit**. This option sets a limit on the number of e-mail addresses that can be included in a single address book entry. The use of multiple addresses in a single address book entry allows end users to create distribution lists. This option limits the size of these distribution lists.
- **Signature flag**. This option enables/disables the use of a message signature. When this option is enabled, all messages sent by an end user with the WebMail client will have that user's signature text appended to the body of the message.
- **Confirm before deleting messages**. This option causes the end user to be prompted for confirmation before a message is deleted via WebMail.
- **Width of the message display area**. This option controls the formatting of messages in the WebMail interface, and is designed to accommodate end users with low screen resolutions. The width of message lines in the WebMail interface can be either 80 characters (for normal resolutions) or 64 characters (for low resolutions).

2.3 Classes of Service

A *class of service* is an Integrated Services Directory object that defines a common set of InterMail account attributes that are available to end users. Each e-mail account in InterMail is associated with a particular class of service, which acts as a template for those accounts. Most account attributes can be defined at both the account and class of service levels.

Classes of service are used to bundle features in distinct packages for consumers. For example, a service provider may charge customers one monthly rate for e-mail accounts with a limited range of services (simple POP3 access, low mailbox quotas, etc.), and a second rate for e-mail accounts with a larger set of allowed services (IMAP access, higher mailbox quotas, security features, etc.). Classes of service allow you to easily manage these sets of features for groups of end users.

2.3.1 Sample Classes of Service

The following diagram illustrates three sample classes of service that a service provider might use to differentiate specific service options. Each of these classes of service has a name, a series of associated service options, and a price to end users:

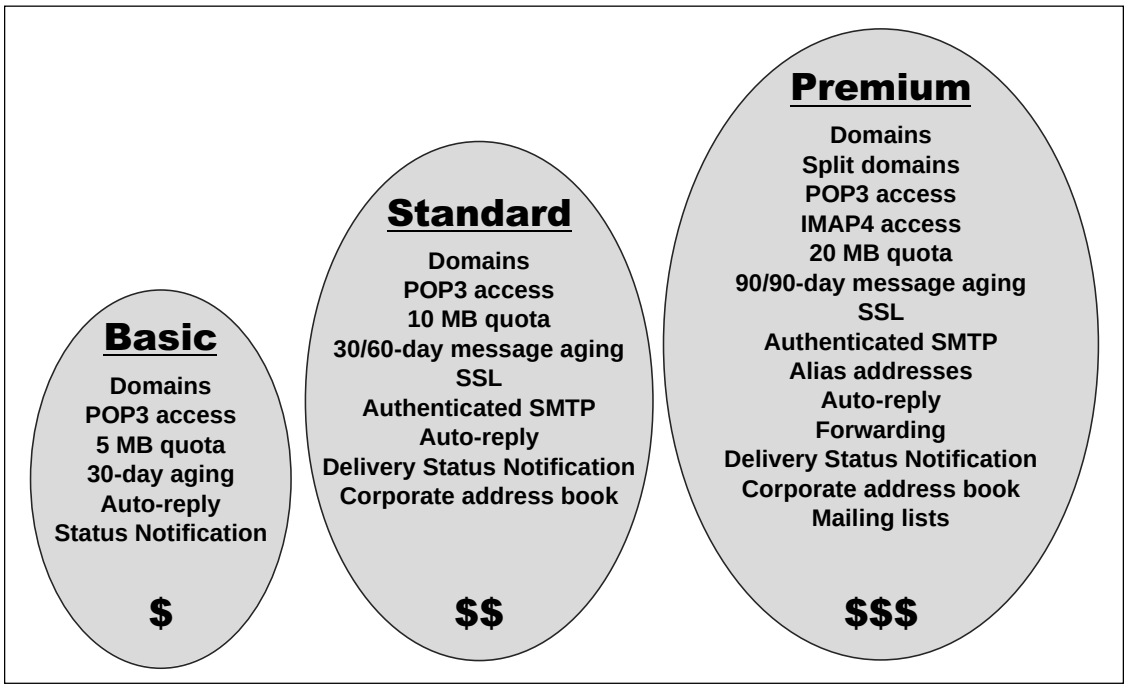


Figure 2: Simple set of classes of service that allows a service provider to provide multiple pricing policies.

In this example, the service provider's customer has a choice of three account levels when purchasing e-mail service:

- The **Basic** class of service defines e-mail accounts that are allowed only a basic set of service options, such as POP3 access, auto-reply, and delivery status notification. The mailbox of a Basic account is allotted only 5 MB of disk space.
- The **Standard** class of service defines e-mail accounts that include the features of Basic accounts, but with added service options that provide access to security features (SSL and Authenticated SMTP), as well as corporate address book data. The message aging service option of the Standard class of service allows messages to remain on the server for a longer period of time, and the mailbox quota of 10 MB is twice that of the Basic mailbox quota.
- The **Premium** class of service defines e-mail accounts that include all of the service options of Standard accounts, but with additional service options: IMAP4 mailbox access, alias addresses, forwarding delivery, and access to mailing lists. The message aging policy and mailbox quota of a Premium account are also increased over the values used by the Standard class of service.

2.3.2 Attributes

Classes of service are made up of a series of *class of service attributes*. These attributes specify the individual InterMail features that are used by end users, and also control end user access to setting these features themselves. A class of service may be associated with as many or as few of these attributes as you wish

There are two types of class of service attributes:

- *preferences*, which define a value for an account option. Preferences are used to define whether a feature is in use (as in the case of mail forwarding) and to set account-related limits (such as mailbox quotas).
- *permissions*, which define end users' access to setting preferences for their accounts. For example, the option to enable or disable end user access to mail delivery options (such as forwarding delivery) in the SelfCare web interface is a permission.

Although each permission attribute has an associated preference attribute, the opposite is not true—several preferences have no explicitly-defined permission. This is the case for preferences such as mailbox quotas and web interface access, which cannot be set by an end user. Because end user access is denied by default, no permission needs to be explicitly defined.

Note: *Additional preference and permission attributes can be added to classes of service, allowing you to create new end user services not currently provided by InterMail.*

Account Settings vs. Class of Service Settings

As described in Section 2.2.1, class of service attributes can be defined at both the account level and the class of service level. The administrator always sets attributes for a class of service, while individual account attributes are typically set by the user through the SelfCare web interface. Although it is possible for an administrator to set attributes at the account level, this type of operation typically occurs only if a particular end user requires special access or privileges (for example, a mailbox quota larger than quota defined by the account's class of service).

To provide the greatest flexibility in the use of class of service attributes, InterMail uses a set of precedence rules to determine whether the value of an attribute that is applied to an account should be taken from the class of service or account level. For example, if the mailbox quota of an account is 3 Mb, while the quota set for the account's class of service is 1 Mb, a precedence rule associated with the mailbox quota attribute determines whether the quota value enforced on the account is 3 Mb or 1 Mb.

The four precedent rules are:

1. **Class of service value is used in favor of the account value**. This rule is typically used for attributes that cannot be modified by the end user.
2. **Account value is used in favor of the class of service value**. This rule is typically used for preferences that are at the discretion of the end user.
3. **Lesser of the account and class of service values is used**. This rule is typically used for preferences which allow access to services, such as `pref_imap`.
4. **Greater of the account and class of service values is used**. This rule is used for preferences that the user can set to a higher value.

These rules are influenced by whether a value exists for the attribute at the class of service level. The following table defines the interaction between attribute values and the resultant values that are applied to accounts.

Precedence Rule	Attribute Values Defined	Value Applied to Account
Class of service value used	For neither	Default value
	For class of service only	Class of service value
	For account only	Default value
	For class of service and account	Class of service value
Account value used	For neither	Default value
	For class of service only	Class of service value
	For account only	Account value
	For class of service and account	Account value
Lesser of account and class of service values	For neither	Default value
	For class of service only	Class of service value
	For account only	Default value
	For class of service and account	Lesser of the two values
Greater of account and class of service values	For neither	Default value
	For class of service only	Class of service value
	For account only	Default value
	For class of service and account	Greater of the two values

For boolean attributes, the value `true` is considered greater than `false`.

The default values associated with class of service attributes are as follows:

- For boolean attributes: `false`
- For number attributes: `0`
- For character attributes: `null`

In the following example, the mailbox quota attribute has been set to 3 Mb for an account, but is set at 1 Mb for this user's class of service. The quota value that is enforced on this account's mailbox for each precedence rule is as follows:

Precedence Rule	Value Applied to Account
Class of service value used (C)	1 Mb
Account value used (A)	3 Mb
Lesser value used (L)	1 Mb
Greater value used (G)	3 Mb

Note: *By default, the precedence rule for the mailbox quota preference specifies that the account value is used.*

The tables in the following sections provide information on the available class of service attributes, and include the precedent rule of each attribute.

Preferences

The following table provides a list of the available class of service preference attributes. Custom attributes can also be added to this list if you need to extend the available class of service options. Note that the Rule column uses the following for precedence rules:

- C - Class of service value used
- A - Account value used
- L - Lesser value used
- G - Greater value used

Attribute	Rule	Type	Description
pref_bypassauthentication	L	Boolean	Allow authentication bypass for both SelfCare and WebMail. If a user's session token is valid, the user can access SelfCare or WebMail without re-authenticating.
pref_forwarding	G	Boolean	Mail forwarding is enabled.
pref_imap	A	Boolean	IMAP access is allowed.
pref_intermanager	A	Boolean	Administrator web access via InterManager is allowed.
pref_localdelivery	G	Boolean	When forwarding is enabled, user may disable local delivery.
pref_mtafilter	A	Boolean	Mail is rejected if sent from system-wide list of addresses.

Attribute	Rule	Type	Description
pref_numaliases	A	Numeric	Maximum number of SMTP aliases.
pref_pop	A	Boolean	POP access is allowed.
pref_popssl	A	Boolean	POP access via SSL is allowed.
pref_quotabouncenotify	L	Boolean	Notify recipient when a message is bounced because it would cause one of the recipient's mailbox quotas to be exceeded.
pref_quotamsgkb	A	Numeric	Maximum message size quota (in kb).
pref_quotathreshold	G	Numeric	Quota warning threshold (set as a number indicating a percentage).
pref_quotatotkb	A	Numeric	Total mailbox size quota (in kb).
pref_quotatotmsgs	A	Numeric	Total number of messages quota.
pref_replymode	A	Numeric	Auto-reply mode (vacation, reply, or echo).
pref_selfcare	A	Boolean	Web access to SelfCare interface.
pref_selfcaressl	A	Boolean	Web access to SelfCare interface through SSL.
pref_smtp	A	Boolean	SMTP access is allowed.
pref_smtpauth	A	Boolean	SMTP connections must use SMTP authentication.
pref_smtpssl	A	Boolean	SMTP access via SSL is allowed.
pref_webdisplay	A	Numeric	Width of the web interface column display.
pref_webmail	A	Boolean	Access to WebMail.
pref_webmailaddressbooklimit	A	Numeric	The maximum number of address book entries allowed in WebMail.
pref_webmailaddressbooklistlimit	A	Numeric	The maximum number of mailing lists allowed in an address book in WebMail.

Attribute	Rule	Type	Description
pref_webmailattachlimit	A	Numeric	WebMail per-session attachment count limit.
pref_webmailattachsizelimit	A	Numeric	WebMail per-attachment size limit (in kb).
pref_webmailconfirmdelete	L	Boolean	Confirm message deletion in WebMail. This controls whether a “confirm deletion” prompt will be displayed.
pref_webmailmsgattachLimit	A	Numeric	WebMail per-message attachment count limit.
pref_webmailusesignature	L	Boolean	Insert signature in WebMail messages.

Permissions

The following table defines the account permissions associated with a class of service. In all cases, permissions are numeric data types, with a value of 1 indicating true (access allowed), and a value of 0 indicating false (access denied).

Attribute	Rule	Type	Description
perm_autoreply	L	Numeric	Controls end user's ability to select the reply mode of auto-reply.
perm_bypassauthentication	L	Numeric	Controls end user's ability to set pref_bypassauthentication.
perm_echo	L	Numeric	Controls end user's ability to select the echo mode of auto-reply.
perm_forwarding	L	Numeric	Controls end user's ability to set pref_forwarding.
perm_localdelivery	L	Numeric	Controls end user's ability to set pref_localdelivery.
perm_mtafilter	L	Numeric	Controls end user's ability to set pref_mtafilter.
perm_quotabouncenotify	L	Numeric	Controls end user's ability to set pref_quotabouncenotify.
perm_quotathreshold	L	Numeric	Controls end user's ability to set pref_quotathreshold.
perm_vacation	L	Numeric	Controls end user's ability to select the vacation mode of auto-reply.
perm_webdisplay	L	Numeric	Controls end user's ability to set pref_webdisplay.

Attribute	Rule	Type	Description
`perm_webmailconfirmdelete`	L	Numeric	Controls end user's ability to set `pref_webmailconfirmdelete`.
`perm_webmailusesignature`	L	Numeric	Controls end user's ability to set `pref_webmailusesignature`.

Note: *For permission attributes, the value* `1` *means that the permission is granted, while a value of* `0` *means that the permission is denied.*

Notice that the names of permission attributes are taken from the preference attribute to which the permission allows access. For example, the attribute `perm_forwarding` controls end user access to setting the `pref_forwarding` preference attribute. Although there is no requirement that you follow this convention when defining new attributes, it is recommended.

2.3.3 Default Class of Service

A default class of service is defined in the Integrated Services Directory upon installation. This class of service—named "default"—contains only the attributes required to allow end user access to simple SMTP and POP3 access:

Attribute	Value
`pref_smtp`	1 (enabled)
`pref_pop`	1 (enabled)

The default class of service is used for backward compatibility. When InterMail 3.x is upgraded to version 4.0, all existing accounts on that system are associated with the default class of service. As with other classes of service, you can modify the default class of service to provide additional services to all accounts within it.

Note: When additional classes of service are created (i.e other than the "default" class of service), no class of service attributes are included; all attributes (including *`pref_smtp`* and *`pref_pop`*) need to be manually specified by the administrator. See Chapter 3 for more information on how to create classes of service and to set attributes.

2.4 Additional InterManager Objects

The InterManager product allows you to manage InterMail domains, classes of service, and e-mail accounts. However, the Integrated Services Directory also contains objects that are specific to InterManager. These objects are:

- organization
- organizational unit
- person

Note: *Refer to the InterManager Administration Guide for complete information on the objects and values that can be created and modified via InterManager.*

2.4.1 Organization

An *organization* corresponds to an entire company or other “top-level” institution. Organizations are typically created when a company purchases an initial set of e-mail accounts and services. Organizations are containers for organizational units and e-mail accounts.

2.4.2 Organizational Unit

An *organizational unit* corresponds to a group within an organization, such as a department within a company. Organizational units are simply a convenience for subdividing an organization, and are optional. Each organizational unit is a member of a specific organization, and can contain any number of e-mail accounts or sub-organizational units.

2.4.3 Person

A *person* object corresponds to an individual user within an organization. Each person object is associated with an InterMail e-mail account. Person objects also contain “white pages” information; this is the data that is available to LDAP clients, and includes attributes such as the person’s name, e-mail address, and telephone number.

3

Utilities for Updating the Database

This chapter describes the command-line utilities that can be used to update InterMail data in the Integrated Services Directory database. These utilities are:

- `imdbcontrol`
- `imldapcontrol`

Although it is recommended that all sites create their own database provisioning tools using the API libraries described in Chapters 5, 6 and 7, the utilities discussed in this chapter can also be used to create database objects and are provided as a convenience.

3.1 imdbcontrol

The utility used to access mail objects in the Integrated Services Directory is `imdbcontrol`. To execute `imdbcontrol`, you must be logged in as the InterMail user on any InterMail host. Because the location of `imdbcontrol` was added to this user's path at installation time, the utility can be executed from any directory.

Note: *The imdbcontrol utility operates similar to the UNIX superuser and allows an administrator to override class of service settings and options set through the InterManager interface. For example, if a class of service is created with a maximum of three aliases for any user, this limit will be strictly enforced in InterManager; however,by using imdbcontrol, the administrator can overwrite this limit. This allows flexibility when individual accounts contained in a class of service require expanded privileges.*

3.1.1 Syntax

The syntax for using `imdbcontrol` is as follows:

```
imdbcontrol [-help] [-d] [-e] [-q] [-v] [-u <user/password@db>]
      [-p <#>] [-] [-h] <option> ...
```

Where:

`-help`	Provides complete usage information.
`-d`	Prints debugging information during execution.
`-e`	Causes the program to exit when errors are encountered during batch operations.
`-q`	Suppresses progress reports when the program is run in batch mode.
`-v`	Reports the success or failure of each operation when run in batch mode.

`-u`	Allows specification of database administrator information, in the form:
	`username/password@db-service`
	For example:
	`imail/imail@IMD1`

`-p`	Prints progress reports every # lines in batch mode. The default number of lines if 100.
`-`	Specifies that multiple lines should be processed from STDIN and executed one-by-one, just as if each line was invoked with a separate invocation of `imdbcontrol`. (This increases the performance of `imdbcontrol` by a factor of about 5 in doing batched operations.)
`-h`	Displays usage information for the specified option. For example, entering `imdbcontrol -h CreateAccount` returns usage information (number and type of parameters, etc.) for the `CreateAccount` option.
`option`	Specifies the operation to be performed (see table below).

The particular operation performed by a given `imdbcontrol` instruction is specified by a command-line option. The available `imdbcontrol` options are divided between operations involving domains, e-mail accounts, forwarding addresses, SMTP Alias addresses, objects, and system logging.

The `imdbcontrol` command line options are case-insensitive. For example, you can create an account by entering any of the following:

```
imdbcontrol CreateAccount ...
imdbcontrol createaccount ...
imdbcontrol CrEaTeAcCoUnT ...
```

In addition, `imdbcontrol` command line options can be abbreviated, such that:

```
imdbcontrol CreateAccount
```

can be expressed as:

```
imdbcontrol ca
```


3.1.2 Execution Options

The following tables provide information on the execution options available with `imdbcontrol`.

Domain Operations

`CreateDomain`	Creates a mail domain.
`DeleteDomain`	Deletes an existing mail domain.
`GetDefaultDomain`	Gets the default mail domain.
`ListDomains`	Shows a list of domains for which InterMail accepts mail.
`SetDefaultDomain`	Sets the default mail domain.
`SetWildcardAccount`	Sets the wildcard account for a local domain.
`UnsetWildcardAccount`	Disables wildcard delivery for a domain.
`UpdateDomain`	Modifies an existing domain.

Account Operations

`CreateAccount`	Creates an account.
`DeleteAccount`	Deletes an existing account.
`DeleteAccountCos`	Deletes a per-account class of service value for an account.
`GetAccount`	Gets data for an existing account.
`GetAccountCos`	Displays the per-account class of service attributes and values associated with an account.
`GetPassword`	Gets the password for an existing account.
`ListAccounts`	Shows a list of all accounts.
`ModifyAccount`	Modifies an existing account.
`ModifyAccountPop`	Modifies the login name of an existing account.
`ModifyAccountSmtp`	Modifies the username of an existing account.
`SetAccountCos`	Defines a per-account class of service value for an account.
`SetAccountQuota`	Sets mailbox quotas for an account.
`SetAccountStatus`	Sets the status of an account (active, locked, etc.)
`SetAutoReply`	Sets the auto reply mode for an account.
`SetAutoReplyHost`	Sets the location of an account's auto-reply message.
`SetPassword`	Sets the password for an existing account.

Mail Delivery Operations

`CreateRemoteForward`	Creates a remote forwarding address for an existing InterMail account.
`DeleteRemoteForward`	Deletes an existing remote forwarding address.
`DisableForwarding`	Disables forwarding for an account.
`DisablePOPDelivery`	Disables local delivery (POP and IMAP) for an account.
`EnableForwarding`	Enables forwarding for an account.
`EnablePOPDelivery`	Enables local delivery (POP and IMAP) for an InterMail account.
`ListAccountForwards`	Shows a list of forwarding addresses associated with an InterMail account.

SMTP Alias Operations

`CreateAlias`	Creates an SMTP alias for an existing InterMail account.
`DeleteAlias`	Deletes an existing SMTP alias.
`ListAliases`	Shows a list of SMTP aliases.

Class of Service Operations

`CreateCos`	Creates a new class of service.
`CreateCosAttribute`	Creates a new class of service attribute, which can then be associated with one or more classes of service.
`DeleteCos`	Deletes an existing class of service.
`DeleteCosAttribute`	Deletes an existing class of service attribute.
`ListCosAttribute`	Shows a list of the existing class of service attributes.
`ListCosNames`	Shows a list of existing classes of service.
`ModifyCosAttribute`	Modifies and existing class of service attribute.
`SetCosAttribute`	Sets the value of a particular attribute for a class of service.
`ShowCos`	Lists the attributes associated with a class of service.
`UnsetCosAttribute`	Removes an attribute and its value from a class of service.

System Operations

`ExpireLogs`	Expires log entries that are older than a specified number of hours.
`GetFirstModified`	Gets the ID of the earliest log entry in the database.
`GetLastModified`	Gets the ID of the latest log entry in the database.

3.1.3 Domain Operations

This section pertains specifically to using `imdbcontrol` for domain-related operations, such as creating and deleting local mail domains.

CreateDomain

Mail domains can be created with `imdbcontrol` by using the `CreateDomain` option. Only one parameter is required with this option: the name of the new domain being created. By default, new domains are created as local mail domains, but optional parameters allow you to define the new domain as a non-authoritative (semi-local) or rewrite domain.

Usage:

```
imdbcontrol CreateDomain <domain>
      [ nonauth <relayHost> | rewrite <rewriteValue> ]
```

Where:

`domain`	The name of the new domain.
`relayHost`	The host to which mail received for the non-authoritative domain is routed. This parameter is required only if the new domain is a non-authoritative domain.
`rewriteValue`	The domain name that replaces the `domain` value when header addresses are rewritten. This parameter is required only if the new domain is a rewrite domain.

Example:

```
imdbcontrol CreateDomain software.com
```

This command creates the local mail domain software.com.

```
imdbcontrol CreateDomain accordance.com rewrite software.com
```

This command creates the rewrite domain `accordance.com`. When mail arrives for users in this domain, the domain name will be rewritten as `software.com`.

```
imdbcontrol CreateDomain venus.software.com nonauth
pluto.software.com
```

This command creates the non-authoritative domain `sales.software.com`. When messages are received for accounts in this domain, they will be routed to the host `pluto.software.com`.

Notes:

Accounts and alias addresses cannot be created in a domain until the domain has been created. Also, the `CreateDomain` command will fail if the domain being created already exists in the database.

UpdateDomain

Each domain has an associated type: local, non-authoritative, or rewrite. The type of an existing domain can be modified with `imdbcontrol` by using the `UpdateDomain` option. Two parameters are required with this option: the name of the domain, and the new type.

A local or non-authoritative domain cannot be changed to a rewrite domain directly. To make this change, set the type of a local or non-authoritative domain to deleted, and then set the type to rewrite.

Usage:

```
imdbcontrol UpdateDomain <domain> { local | nonauth <relayHost> |
      rewrite <rewriteValue> }
```

Where:

`domain`	The name of the existing domain.
`relayHost`	The host to which mail received for the non-authoritative domain is routed. This parameter is required only if the type of the domain is being changed to non-authoritative.
`rewriteValue`	The domain name that replaces the `domain` value when header addresses are rewritten. This parameter is required only if the type of the domain is being changed to rewrite.

Example:

```
imdbcontrol UpdateDomain accordance.com nonauth mail.accordance.com
```

This command changes the domain `accordance.com` to a non-authoritative domain, and sets its relay host value to `mail.accordance.com`.

```
imdbcontrol UpdateDomain accordance.com local
```

This command changes the `accordance.com` domain to a local mail domain.

Notes:

Accounts and alias addresses cannot be created in a rewrite domain. If you attempt to change the type of a local or non-authoritative domain to rewrite, and accounts are associated with the domain, the operation will fail.

DeleteDomain

Domains can be deleted with `imdbcontrol` by using the `DeleteDomain` option. Only one parameter is required with this option: the name of the domain being deleted.

By default, deleting a domain with this option does *not* cause the domain to be removed from the Integrated Services Directory. Instead, the domain is marked as deleted. This allows the domain to be subsequently restored with the `UpdateDomain` option.

Usage:

```
imdbcontrol DeleteDomain <domain> [force]
```

Where:

`domain`	The name of the domain being deleted.
`force`	Option to permanently remove the domain from the Integrated Services Directory.

Example:

```
imdbcontrol DeleteDomain software.com
```

This example deletes the domain `software.com`.

Notes:

Domains cannot be deleted if there are any accounts or alias addresses associated with the domain. Before attempting to delete a domain, use the `ListAccounts` and `ListAliases` options to check for the existence of accounts, aliases, and forwarding addresses within this domain.

ListDomains

To get a list of mail domains for which InterMail receives e-mail, use `imdbcontrol` with the `ListDomains` option. This option takes no parameters, and simply generates a list of domains.

Usage:

```
imdbcontrol ListDomains
```

Example:

This command generates output like the following:

```
vvv
Domain:         software.com
Type:           L
RelayHost:
RewriteDomain:
WildcardAcct:   unknown@software.com
---
Domain:         venus.software.com
Type:           R
RelayHost:
RewriteDomain:  software.com
WildcardAcct:
---
^^^ # Domains = 2
```


The above data includes the following information for a domain:

- Domain name (`Domain`).
- Type of the domain (`Type`). The value of this field can be `L` (local), `N` (non-authoritative), `R` (rewrite), or `D` (deleted).
- The relay host of the non-authoritative domain (`RelayHost`). This attribute is relevant only for non-authoritative domains.
- The rewrite value of the rewrite domain (`RewriteDomain`). The name of the rewrite domain will be replaced by this value in recipient addresses. This attribute is relevant only for rewrite domains.
- The address of the wildcard account of the local (`WildcardAcct`). This attribute is relevant only for local domains.

SetDefaultDomain

Several account-related `imdbcontrol` operations allow you to omit a domain parameter; in this case, the default domain is assumed. This default domain is set with `imdbcontrol` by using the `SetDefaultDomain` option. Only one parameter is required with this option: the name of the domain to be set as the default.

Usage:

```
imdbcontrol SetDefaultDomain <domain>
```

Where:

`domain`	The name of the domain that is being set as the default domain.

Example:

```
imdbcontrol SetDefaultDomain software.com
```

This example sets the default domain to software.com. When creating accounts, deleting aliases, or performing numerous other `imdbcontrol` operations, omitting the domain parameter specifies that this default domain be used in the operation.

Notes:

This default domain is different from the MTA default domain, which is set in the configuration key `mta/defaultDomain`. The MTA default domain is used for mail routing tasks, such as address completion, and has no affect on the Integrated Services Directory.

GetDefaultDomain

The current default domain can be retrieved with `imdbcontrol` by using the `GetDefaultDomain` option. This command takes no parameters, and simply outputs the name of the default domain.

Usage:

```
imdbcontrol GetDefaultDomain
```


SetWildcardAccount

All mail domains in InterMail can have an optional "wildcard" account associated with them. A wildcard account is simply a normal e-mail account that receives all e-mail that is sent to unknown addresses within the domain.

For example, if the account `john.doe@software.com` is defined as a wildcard account for the local mail domain `software.com`, then any message sent to a non-existent address within `software.com` will be delivered to `john.doe@software.com`.

Note: *Delivery to a wildcard account is, from InterMail's perspective, a last resort; it occurs only when the destination address of a message does not exist as an account primary address or as an SMTP alias. A message that is successfully delivered to a normal InterMail account is never delivered to a wildcard account as well.*

To set a wildcard account for a domain, use `imdbcontrol` with the `SetWildcardAccount` option. This option takes two parameters: the domain name, and the primary SMTP address of the existing account that will be used as the wildcard account.

Usage:

```
imdbcontrol SetWildcardAccount <domain> <username>
```

Where:

`domain`	The name of the domain.
`username`	The address of the e-mail account that will be used as a wildcard account for this domain.

Example:

```
imdbcontrol SetWildcardAccount mars.software.com unknown
```

This command sets the e-mail account `unknown@mars.software.com` as the wildcard account for the domain `mars.software.com`.

UnsetWildcardAccount

If a wildcard account has been defined for a domain, you can disable wildcard delivery by using `imdbcontrol` with the `UnsetWildcardAccount` option. This option takes only parameter: the domain name.

Usage:

```
imdbcontrol UnsetWildcardAccount <domain>
```

Where:

`domain`	The name of the domain for which wildcard delivery is being removed.

Example:

```
imdbcontrol UnsetWildcardAccount mars.software.com
```

This command removes the wildcard account delivery defined for the domain `mars.software.com` in the `SetWildcardAccount` example.

3.1.4 Account Operations

The `imdbcontrol` operations described in this section are specific to InterMail account-related mail objects.

CreateAccount

Accounts are created with `imdbcontrol` by using the `CreateAccount` option and specifying several additional pieces of account information. Some of these additional flags are required, while others are optional. If any of the optional parameters are omitted, then all subsequent optional parameters must be omitted.

Usage:

```
imdbcontrol CreateAccount <username> <mssHost>
      <internalId> [<popLogin>] [<password> [-convert]
      clear|md5|UNIX] [<domain>] [<status>] [<type>] [cosName]
```

Where:

`username`	The local part of the SMTP address for the account (the part which precedes the "@" symbol in the e-mail address).
`mssHost`	The name of the MSS host on which the mailbox for this account will be located.
`internalId`	A unique ID shared between the Directory database and the MSS.
`popLogin`	POP/IMAP login name.

`password`
`clear|md5|unix`
`-convert`
	Password for POP/IMAP access. If a password is specified, the hashing scheme (or lack thereof) must be indicated. Note that if the `-convert` option is omitted, `imdbcontrol` does not hash passwords.

`domain`	Domain with which the account is associated. If none is specified, the default domain is assumed.
`status`	Status of the account. The possible values for this parameter are: `A` (active), `L` (locked), `D` (deleted), `M` (maintenance), `S` (suspended), `P` (proxy). The default is `A` (active).
`type`	Account type of the account. The possible values for this parameter are `A` (administrative) and `S` (standard). The default is `S` (standard).
`cosName`	Class of service with which the account is associated. If none is specified, the default class of service is assumed.

Example:

```
imdbcontrol CreateAccount john.doe pluto.software.com 12345 jdoe
rosebud clear software.com A S Basic
```


This command creates an account which has the following attributes:

- The SMTP address `john.doe` in the domain `software.com` (i.e., this account has the e-mail address `john.doe@software.com`).
- The class of service `Basic`.
- A mailbox on the host `pluto.software.com`.
- The POP/IMAP login name `jdoe`.
- The password `rosebud`.
- The internal ID number `12345`.
- The account is active (`A`) and is a standard user account (`S`).

GetAccount

In addition to setting account attributes, `imdbcontrol` allows you to simply view a definition of all of the attributes of a particular account. This is done using the `GetAccount` option, which requires two additional parameters: the username portion of the account's primary SMTP address, and the domain with which this address is associated.

Usage:

```
imdbcontrol GetAccount <username> <domain>
```

Where:

`username`	The local part of the SMTP address for the account (the part which precedes the "@" symbol in the e-mail address).
`domain`	Domain with which the account is associated.

Example:

```
imdbcontrol GetAccount john.doe software.com
```

This example retrieves all of the mail account information for the account `john.doe@software.com`. The output from this command looks like the following:

```
Account Info:
---
          Type: S
        Status: A
   Internal-ID: 10671
 Delivery Host: pluto.software.com
   Domain Name: software.com
 PSMTP Address: john.doe
   POP Address: jdoe
      Password: rosebud
 Password Hash: C
       Forward: N
 LocalDelivery: P
AutoReply Host: pluto.software.com
AutoReply Mode: N
      COS Name: default
---
```


The preceding data includes the following information for an account:

- Account type (`Type`). The value of this field can be `S` (standard), or `A` (administrative).
- Account status (`Status`). The value of this field can be `A` (active), `L` (locked), `D` (deleted), `M` (maintenance), `S` (suspended), or `P` (proxy).
- Unique number used to identify the account (`Internal-ID`).
- MSS host where the account's mailbox is stored (`Delivery Host`).
- Domain of the account's primary address (`Domain Name`).
- Login password (`Password`)
- Password hashing scheme (`Password Hash`). The value of this field can be `C` (clear), `U` (Unix), or `M` (MD5).
- Flag that indicates whether mail forwarding is enabled (`Forward`). The value of this field can be `F` (enabled), or `N` (disabled).
- Flag that indicates whether local delivery is enabled (`LocalDelivery`). The value of this field can be `P` (enabled), or `N` (disabled).
- MSS host where the account's auto-reply message is stored (`AutoReply Host`).
- Flag that indicates the auto-reply mode (`AutoReply Mode`). The possible values for this field are `N` (none),`V` (vacation), `R` (reply), or `E` (echo).
- Class of service associated with the account (`COS Name`).

ModifyAccount

Accounts can be modified with `imdbcontrol` by using the `ModifyAccount` option and specifying several additional pieces of account information. Unlike `CreateAccount`, the `ModifyAccount` option requires that all account parameters be specified.

Usage:

```
imdbcontrol ModifyAccount <username> <mssHost>
      <internalId> <password> [-convert] <clear|md5|unix>
      <domain> <status> <type> <cosName>
```

Where:

`username`	The local part of the SMTP address for the account (the part which precedes the "@" symbol in the e-mail address).
`mssHost`	The name of the MSS host where the mailbox for this account is located.
`internalId`	A unique ID for the account shared between the Integrated Services Directory database and the MSS.

`password`
`clear|md5|unix`
`-convert`
	Password for POP/IMAP access. If a password is specified, the hashing scheme (or lack thereof) must be indicated. Note that if the `-convert` option is omitted, `imdbcontrol` does not hash passwords.

`domain`	Domain with which the account is associated. If none is specified, the default domain will be used.
`status`	Status of the account. The possible values for this parameter are: `A` (active), `L` (locked), `D` (deleted), `M` (maintenance), `S` (suspended), `P` (proxy).
`type`	Account type of the account. The possible values for this parameter are `A` (administrative) and `S` (standard).
`cosName`	Class of service with which the account is associated.

Example:

```
imdbcontrol ModifyAccount john.doe pluto.software.com 12345 rosebud
unix -convert software.com A S Basic
```

This command modifies the account created in the `CreateAccount` example. The only change that has been made is that the hashing scheme of the user's password has been changed from clear to UNIX.

Notes:

When using `ModifyAccount`, it is recommended that you first execute a `GetAccount` operation to obtain current account information, and then use `ModifyAccount` to make changes to the account.

ModifyAccountSmtp

The primary e-mail address of an account can be changed with `imdbcontrol` by using the `ModifyAccountSmtp` option. Four parameters are required for this option: the current username of the account, the domain with which the account is associated, the new username value, and the new domain.

Usage:

```
imdbcontrol ModifyAccountPop <username> <domain> <newUsername>
      <newDomain>
```

Where:

`username`	The local part of the SMTP address for the account (the part which precedes the "@" symbol in the e-mail address).
`domain`	Domain with which the account is associated.
`newUsername`	The new username value for the account.
`newDomain`	The new domain for the account.

Example:

```
imdbcontrol ModifyAccountSmtp john.doe software.com jdoe
sales.software.com
```

This example changes the primary e-mail address of an account from `john.doe@software.com` to `jdoe@sales.software.com`.

ModifyAccountPop

The login name of an account can be set with `imdbcontrol` by using the `ModifyAccountPop` option. Three parameters are required for this option: the username portion of the account's primary SMTP address, the domain with which the account is associated, and the new login name value.

Usage:

```
imdbcontrol ModifyAccountPop <username> <domain> <popLogin>
```

Where:

`username`	The local part of the SMTP address for the account (the part which precedes the "@" symbol in the e-mail address).
`domain`	Domain with which the account is associated.
`popLogin`	The new POP/IMAP login name value.

Example:

```
imdbcontrol ModifyAccountPop john.doe software.com johndoe
```

This example changes the POP/IMAP login name of the account `john.doe@software.com` to `johndoe`.

DeleteAccount

Accounts are deleted with `imdbcontrol` by using the `DeleteAccount` option. Two additional parameters are required for this option: the username and domain of the account's e-mail address.

Usage:

```
imdbcontrol DeleteAccount <username> <domain>
```

Where:

`username`	The local part of the SMTP address for the account (the part which precedes the "@" symbol in the e-mail address).
`domain`	Domain with which the account is associated.

Example:

```
imdbcontrol DeleteAccount john.doe software.com
```

This example deletes the account that has the e-mail address `john.doe@software.com`.

ListAccounts

While the `GetAccount` option provides information on a particular account, an additional `imdbcontrol` option allows you to view this information for all accounts in InterMail, or in a particular domain. This is done using the `ListAccounts` option, which takes only one optional parameter: the domain for which account information is being queried.

Usage:

```
imdbcontrol ListAccounts [<domain>]
```

Where:

`domain`	Domain that is being queried for account information. If no domain is specified, information is retrieved for all domains.

Example:

```
imdbcontrol ListAccounts software.com
```

This example retrieves mail account information for all accounts in the domain `software.com`.

```
imdbcontrol ListAccounts
```

This example retrieves mail account information for all accounts in all domains.

The information displayed by `ListAccounts` for each account is formatted as follows:

```
Addr:           john.doe@software.com
Pass:           rosebud
Pass-Type:      C
Type:           S
Status:         A
Host:           venus
IntID:          6
Local Delivery: P
ForwardFlag:    N
AutoReplyMode:  N
AutoReplyHost:  pluto
```

The above data includes the following information for an account:

- Primary e-mail address (`Addr`).
- Login password (`Pass`)
- Password hashing scheme (`Pass-Type`). The value of this field can be `C` (clear), `U` (Unix), or `M` (MD5).
- Account type (`Type`). The value of this field can be `S` (standard), or `A` (administrative).
- Account status (`Status`). The value of this field can be `A` (active), `L` (locked), `D` (deleted), `M` (maintenance), `S` (suspended), or `P` (proxy).
- MSS host where the account's mailbox is stored (`Host`).
- Unique number used to identify the account (`IntID`).
- Flag that indicates whether local delivery is enabled (`Local Delivery`). The value of this field can be `P` (enabled), or `N` (disabled).
- Flag that indicates whether mail forwarding is enabled (`ForwardFlag`). The value of this field can be `F` (enabled), or `N` (disabled).
- Flag that indicates the auto-reply mode (`AutoReplyMode`). The possible values for this field are `N` (none),`V` (vacation), `R` (reply), or `E` (echo).
- MSS host where the account's auto-reply message is stored (`AutoReplyHost`).

Notes:

A domain cannot be deleted if there are mail accounts in the Integrated Services Directory that is associated with the domain. Before attempting to delete a domain, you should first use the `ListAccounts` option to check for the existence of mail accounts in this domain, and delete those accounts if appropriate.

SetPassword

Account password values can be set with `imdbcontrol` by using the `SetPassword` option. Four parameters are required for this option: the username portion of the account's primary SMTP address, the domain with which the account is associated, the new password, and the hashing scheme for the password.

Usage:

```
imdbcontrol SetPassword <username> <domain>
      <password> clear|md5|UNIX [-convert]
```

Where:

`username`	The local part of the SMTP address for the account (the part which precedes the "@" symbol in the e-mail address).
`domain`	Domain with which the account is associated.
`password`	The new password value for the account.
`clear|md5|UNIX`	The hashing algorithm used to store the password in the database.
`-convert`	Optional flag that specifies that `imdbcontrol` should hash the password according to the given hashing scheme.

Example:

```
imdbcontrol SetPassword john.doe software.com $ecret clear
```

This example changes the password for the account `john.doe@software.com` to `$ecret`, stored in clear text format.

GetPassword

Account password values can be retrieved with `imdbcontrol` by using the `GetPassword` option. Two additional parameters are required for this option: the username portion of the account's primary SMTP address, and the domain with which the account is associated.

Usage:

```
imdbcontrol GetPassword <username> <domain>
```

Where:

`username`	The local part of the SMTP address for the account (the part which precedes the "@" symbol in the e-mail address).
`domain`	Domain with which the account is associated.

Example:

```
imdbcontrol GetPassword john.doe software.com
```

This command gets the password for the account john.doe@software.com.

SetAccountStatus

To set the status of an account, use `imdbcontrol` with the `SetAccountStatus` option. This command takes three parameters: the username portion of the account's primary SMTP address, the domain with which this address is associated, and a flag that indicates the account's new status.

Usage:

```
imdbcontrol SetAccountStatus <username> <domain>
      <status>
```

Where:

`username`	The local part of the SMTP address for the account (the part which precedes the "@" symbol in the e-mail address).
`domain`	Domain with which the account is associated.
`status`	New status of the account. The possible values for this parameter are: `A` (active), `L` (locked), `D` (deleted), `M` (maintenance), `S` (suspended), `P` (proxy).

Example:

```
imdbcontrol SetAccountStatus john.doe software.com S
```

This example changes the status of the account `john.doe@software.com` to suspended (`S`).

SetAccountQuota

Quota values can be set at the account level with `imdbcontrol` by using the `SetAccountQuota` option. This command takes up to six parameters: the username portion of the account's primary SMTP address, the domain with which this address is associated, and the four available quota options. When setting values for the maximum number of messages, maximum message size, and quota warning threshold, you must specify a keyword (`maxMsgs`, `maxMsgBytes`, and `quotaThreshold`, respectively) with the value.

Usage:

```
imdbcontrol SetAccountQuota <username> <domain>
      <mboxMaxBytes> [maxMsgs=<mboxMaxMsgs>]
      [maxMsgBytes=<mboxMaxMsgBytes>]
      [quotaThreshold=<percentage>]
```


Where:

`username`	The local part of the SMTP address for the account (the part which precedes the "@" symbol in the e-mail address).
`domain`	Domain with which the account is associated.
`mboxMaxBytes`	The maximum size of the account's mailbox; expressed in number of bytes.
`mboxMaxMsgs`	The maximum number of messages allowed in the account's mailbox.
`mboxMaxMsgBytes`	The largest message that can be received by the account; expressed in number of bytes.
`percentage`	The quota warning threshold; expressed as a percentage of the `mboxMaxBytes` limit.

Example:

```
imdbcontrol SetAccountQuota john.doe software.com 10000000
maxMsgs=2000 maxMsgBytes=3000000 quotaThreshold=90
```

This command sets values for all of the available quota options for the account `john.doe@software.com`. These values set the following behaviors for the account:

- The account's mailbox can contain no more than 10 Mb (10,000,000 bytes) of mail.
- The account's mailbox can contain no more than 2,000 messages.
- Only messages of less than 3 Mb (3,000,000 bytes) will be accepted for the account.
- When the account's mailbox reaches 90% of capacity—that is, when the mailbox contains 9 MB of mail—the user is sent a warning message.

Notes:

The default value for all account quotas is 0, which specifies that there are no limits on the account.

EnablePOPDelivery

The most common method of mail delivery for InterMail accounts is local delivery. This method of delivery causes all messages sent to the account to be stored in a mailbox, from which they can be accessed by a POP3 or IMAP4 mail client.

Local delivery can be enabled for an account with `imdbcontrol` by using the `EnablePOPDelivery` option. Two parameters are required with the this option: the username portion of the account's primary SMTP address, and the domain with which this address is associated.

Usage:

```
imdbcontrol EnablePOPDelivery <username> <domain>
```

Where:

`username`	The local part of the SMTP address for the account (the part which precedes the "@" symbol in the e-mail address).
`domain`	Domain with which the account is associated.

Example:

```
imdbcontrol EnablePOPDelivery john.doe software.com
```

This command enables local delivery for the account `john.doe@software.com`. Once this delivery method is enabled, all messages sent to `john.doe@software.com` will be stored in the account's mailbox, from which they can be retrieved via the POP3 Server or IMAP Server.

DisablePOPDelivery

Local delivery can be disabled for an account with `imdbcontrol` by using the `DisablePOPDelivery` option. Two parameters are required with the this option: the username portion of the account's primary SMTP address, and the domain with which this address is associated.

Usage:

```
imdbcontrol DisablePOPDelivery <username> <domain>
```

Where:

`username`	The local part of the SMTP address for the account (the part which precedes the "@" symbol in the e-mail address).
`domain`	Domain with which the account is associated.

Example:

```
imdbcontrol DisablePOPDelivery john.doe software.com
```

This command disables local delivery for the account `john.doe@software.com`. When local delivery is disabled for an account, messages sent to that account will no longer be stored in the account's mailbox.

Notes:

An account which does not use local delivery must use the mail forwarding.

CreateAlias

Alias addresses can be created for an account with `imdbcontrol` by using the `CreateAlias` option. Two parameters are required with this option: the username portion of the account's primary SMTP address, and the username portion of the alias address that is being added. If a default domain has not been set, the domains associated with these addresses must also be specified.

Usage:

```
imdbcontrol CreateAlias <username> <aliasUsername>
      [<domain>] [<aliasDomain>]
```

Where:

`username`	The local part of the SMTP address for the account (the part which precedes the "@" symbol in the e-mail address).
`aliasUsername`	The local portion of the new alias address.
`domain`	Domain with which the account's primary SMTP address is associated. This parameter is optional only if a default domain has already been set.
`aliasDomain`	Domain with which the new alias address is associated. If omitted, the default domain is assumed.

Example:

```
imdbcontrol CreateAlias john.doe sales software.com software.com
```

This command adds the alias address `sales@software.com` to the account whose primary SMTP address is `john.doe@software.com`. Aliases are commonly created in this manner to allow users to receive mail sent to multiple addresses.

DeleteAlias

Alias addresses can be deleted with `imdbcontrol` by using the `DeleteAlias` option. Two parameters are required with the `DeleteAlias` option: the username portion of the alias address, and the domain with which this alias is associated.

Usage:

```
imdbcontrol DeleteAlias <username> <domain>
```

Where:

`username`	The local portion of the alias address to be deleted.
`domain`	Domain with which the alias address is associated.

Example:

```
imdbcontrol DeleteAlias sales software.com
```

This command deletes the alias address `sales@software.com` that was created in the `CreateAlias` example.

ListAliases

In addition to information about a particular account, you can use `imdbcontrol` to query the Integrated Services Directory for a list of all alias addresses in the system, or the aliases associated with a particular account. This is done using the `ListAliases` option, which takes two optional parameters: the username portion of the account's primary SMTP address, and the domain with which this address is associated (note that you must specify either both or neither of these arguments).

Usage:

```
imdbcontrol ListAliases [<username> <domain>]
```

Where:

`username`	The local part of the SMTP address for the account (the part which precedes the "@" symbol in the e-mail address).
`domain`	Domain with which the account's primary SMTP address is associated.

Example:

```
imdbcontrol ListAliases john.doe software.com
```

This command returns a list of SMTP aliases associated with the account `john.doe@software.com`.

```
imdbcontrol ListAliases
```

This example retrieves information for all alias addresses in all domains.

Notes:

A domain cannot be deleted if there are one or more alias addresses in the Integrated Services Directory that is associated with the domain. Before attempting to delete a domain, you should first use the `ListAliases` option to check for the existence of alias addresses in this domain, and delete those aliases if appropriate.

EnableForwarding

To enable mail forwarding for an account, use `imdbcontrol` with the `EnableForwarding` option. Two parameters are required with the this option: the username portion of the account's primary SMTP address, and the domain with which this address is associated.

Usage:

```
imdbcontrol EnableForwarding <username> <domain>
```

Where:

`username`	The local part of the SMTP address for the account (the part which precedes the "@" symbol in the e-mail address).
`domain`	Domain with which the account's primary SMTP address is associated

Example:

```
imdbcontrol EnableForwarding john.doe software.com
```

This command enables forwarding delivery for the account `john.doe@software.com`. Once this delivery method is enabled, all messages sent to `john.doe@software.com` will be forwarded to whatever forwarding addresses have been defined for this account.

DisableForwarding

Forwarding delivery can be disabled for an account with `imdbcontrol` by using the `DisableForwarding` option. Disabling forwarding delivery this way allows you to stop mail forwarding from an account without permanently deleting all forwarding addresses associated with the account. Two parameters are required with the this option: the username portion of the account's primary SMTP address, and the domain with which this address is associated.

Note: *Executing this option automatically enables local (POP) delivery if it was previously disabled.*

Usage:

```
imdbcontrol DisableForwarding <username> <domain>
```

Where:

`username`	The local part of the SMTP address for the account (the part which precedes the "@" symbol in the e-mail address).
`domain`	Domain with which the account's primary SMTP address is associated

Example:

```
imdbcontrol DisableForwarding john.doe software.com
```

This command disables forwarding delivery for the account `john.doe@software.com`. Once this delivery method is disabled, no messages addressed to this account will be forwarded, regardless of whether any local or remote forwarding addresses have been defined for this account.

CreateRemoteForward

To forward mail from an InterMail account to an account in a remote mail domain, create a forwarding address with `imdbcontrol` by using the `CreateRemoteForward` option. Three parameters are required with the `CreateRemoteForward` option: the username portion of the account's primary SMTP address, the domain with which this address is associated, and the full address (username + "@" + domain) to which mail will be forwarded.

Usage:

```
imdbcontrol CreateRemoteForward <username> <domain>
      <forwardTo>
```


Where:

`username`	The local part of the SMTP address for the account (the part which precedes the "@" symbol in the e-mail address).
`domain`	Domain with which the account's primary SMTP address is associated. This parameter is optional only if a default domain has already been set. If omitted, the default domain is assumed.
`forwardTo`	The complete remote address which is being defined as a forwarding address.

Example:

```
imdbcontrol CreateRemoteForward john.doe software.com
jdoe@Wossamotta-U.edu
```

This example creates the forwarding address `jdoe@Wossamotta-U.edu` for the account `john.doe@software.com`.

Notes:

Even if you have created a forwarding address for an account, mail will not be forwarded from this account unless forwarding delivery is specifically enabled for the account.

DeleteRemoteForward

Forwarding addresses can be deleted from an account with `imdbcontrol` by using the `DeleteRemoteForward` option. Three parameters are required with the this option: the username portion of the account's primary SMTP address, the domain with which this address is associated, and the full forwarding address (username + "@" + domain) that is being deleted.

Usage:

```
imdbcontrol DeleteRemoteForward <username> <domain> <forwardTo>
```

Where:

`username`	The local part of the SMTP address for the account (the part which precedes the "@" symbol in the e-mail address).
`domain`	Domain with which the account's primary SMTP address is associated
`forwardTo`	The complete remote forwarding address which is being deleted.

Example:

```
imdbcontrol DeleteRemoteForward john.doe software.com
jdoe@Wossamotta-U.edu
```

This example deletes the remote forwarding address `jdoe@Wossamotta-U.edu` that was created for the local account `john.doe@software.com` in the `CreateRemoteForward` example.

Notes:

Deleting an account's last forwarding address automatically enables local (POP) delivery for the account if it was previously disabled.

ListAccountForwards

To get a list of the forwarding addresses that are associated with an e-mail account, use `imdbcontrol` with the `ListAccountForwards` option. Two parameters are required with the this option: the username portion of the account's primary SMTP address, and the domain with which this address is associated.

Usage:

```
imdbcontrol ListAccountForwards <username> <domain>
```

Where:

`username`	The local part of the SMTP address for the account (the part which precedes the "@" symbol in the e-mail address).
`domain`	Domain with which the account's primary SMTP address is associated

Example:

```
imdbcontrol ListAccountForwards john.doe software.com
```

This command lists the local and remote forwarding addresses associated with the account `john.doe@software.com`. Output from this command includes only the relevant forwarding addresses, one per line. For example:

```
joe.schmoe@accordance.com
jdoe@Wossamotta-U.edu
```

SetAutoReply

Auto-reply can be enabled for an account with `imdbcontrol` by using the `SetAutoReply` option. Three parameters are required with the this option: the username portion of the account's primary SMTP address, the domain with which this address is associated, and the mode of auto-reply.

Usage:

```
imdbcontrol SetAutoReply <username> <domain> <autoReplyMode>
```

Where:

`username`	The local part of the SMTP address for the account (the part which precedes the "@" symbol in the e-mail address).
`domain`	Domain with which the account is associated.
`autoReplyMode`	The mode of auto-reply used by the account. The possible values for this parameter are: `N` (none),`V` (vacation), `R` (reply), `E` (echo).

Example:

```
imdbcontrol SetAutoReply john.doe software.com V
```

This example enables an auto-reply for the account `john.doe@software.com`, using the vacation mode.

Notes:

The auto-reply message associated with an account is not stored in the Integrated Services Directory, and therefore cannot be set using `imdbcontrol`. Auto-reply messages are stored as text files on the hosts that contain account mailboxes; that is, the hosts which run the InterMail Message Store Server (MSS). Use the `SetAutoReplyHost` option with `imdbcontrol` to set the location of an account's auto-reply message.

SetAutoReplyHost

All InterMail accounts include an optional auto-reply feature. The status of this feature (enabled or disabled), and its mode of operation (vacation, reply, echo) is stored in the Integrated Services Directory. However, the actual text of an account's auto-reply message is stored on the file system of a host that runs the InterMail Message Store Server (MSS). Setting up the auto-reply feature for an account therefore includes specifying the MSS host on which the account's auto-reply message is stored.

The location of an auto-reply message can be set for an account with `imdbcontrol` by using the `SetAutoReplyHost` option. Three parameters are required with the this option: the username portion of the account's primary SMTP address, the domain with which this address is associated, and the hostname of the system on which the auto-reply message is stored.

Usage:

```
imdbcontrol SetAutoReplyHost <username> <domain> <autoReplyHost>
```

Where:

`username`	The local part of the SMTP address for the account (the part which precedes the "@" symbol in the e-mail address).
`domain`	Domain with which the account is associated.
`autoReplyHost`	The host name of the system on which the account's auto-reply message is stored.

Example:

```
imdbcontrol SetAutoReplyHost john.doe software.com
venus.software.com
```

This command sets the host `venus.software.com` as the location of the auto-reply message for the account `john.doe@software.com`.

Notes:

The name of the file that contains an account's auto-reply message is defined in the MSS database. To specify the auto-reply file for an account, you must use the InterMail utility `imreplyctrl`. This command is described in Chapter 12 of the *InterMail Reference Guide*.

SetAccountCos

The value of a class of service attribute can be defined for an account with `imdbcontrol` by using the `SetAccountCos` option. Four parameters are required with this option: the username portion of the account's primary SMTP address, the domain with which the account is associated, the name of the class of service attribute, and the value of the attribute. The attribute and value parameters may be repeated to allow for setting multiple attributes in a single operation.

Usage:

```
imdbcontrol SetAccountCos <username> <domain> <attribute> <value>
...
```

Where:

`username`	The local part of the SMTP address for the account (the part which precedes the "@" symbol in the e-mail address).
`domain`	Domain with which the account is associated.
`attribute`	The name of the attribute. More than one attribute can appear in a single execution, each with its own value.
`value`	The value of the attribute.

Example:

```
imdbcontrol SetAccountCos jdoe software.com pref_imap 1
```

This example sets values for the class of service attribute `pref_imap` for the account `jdoe@software.com`. The value `1` indicates that this boolean attribute is being set to on.

GetAccountCos

The class of service attribute values for an account can be retrieved with `imdbcontrol` by using the `GetAccountCos` option. This command displays the names of all InterMail class of service attributes, their values (if any) at both the class of service and account levels, and the attribute value that is currently applied to the account. Two parameters are required with this option: the username of the account's primary SMTP address and the domain with which the account is associated.

Usage:

```
imdbcontrol GetAccountCos <username> <domain>
```

Where:

`username`	The local part of the SMTP address for the account (the part which precedes the "@" symbol in the e-mail address).
`domain`	Domain with which the account is associated.

Example:

```
imdbcontrol GetAccountCos jdoe software.com
```

This example retrieves class of service attribute values for the account `jdoe@software.com`. This command generates output like the following:

```
    Attribute Name              Syntax Rule COS Value Account Value Result Value
    ==============              ====== ==== ========= ============= ============

perm_autoreply                    N     L      0        <undef>           0
perm_bypassauthentication         N     L      0        <undef>           0
perm_echo                         N     L      0        <undef>           0
perm_forwarding                   N     L      0        <undef>           0
perm_localdelivery                N     L      0        <undef>           0
perm_mtafilter                    N     L      0        <undef>           0
perm_quotabouncenotify            N     L      0        <undef>           0
perm_quotathreshold               N     L      0        <undef>           0
perm_vacation                     N     L      0        <undef>           0
perm_webdisplay                   N     L      0        <undef>           0
perm_webmailconfirmdelete         N     L      0        <undef>           0
perm_webmailusesignature          N     L      0        <undef>           0
pref_bypassauthentication         B     L      0        <undef>           0
pref_forwarding                   B     G      0        <undef>           0
pref_imap                         B     A      1*       <undef>           1
pref_intermanager                 B     A      0        <undef>           0
pref_intermanagerssl              B     A      0        <undef>           0
pref_localdelivery                B     G      0        <undef>           0
pref_mtafilter                    B     A      0        <undef>           0
pref_numaliases                   N     A      0        <undef>           0
pref_pop                          B     A      1*       <undef>           1
pref_popssl                       B     A      0        <undef>           0
pref_quotabouncenotify            B     L      0        <undef>           0
pref_quotamsgkb                   N     A      0        <undef>           0
pref_quotathreshold               N     G      0        <undef>           0
pref_quotatotkb                   N     A      0        <undef>           0
pref_quotatotmsgs                 N     A      0        <undef>           0
pref_replymode                    N     A      0        <undef>           0
pref_selfcare                     B     A      0        <undef>           0
pref_selfcaressl                  B     A      0        <undef>           0
pref_smtp                         B     A      1*       <undef>           1
pref_smtpauth                     B     A      0        <undef>           0
pref_smtpssl                      B     A      0        <undef>           0
pref_webdisplay                   N     A      0        <undef>           0
pref_webmail                      B     A      0        <undef>           0
pref_webmailaddressbooklimit      N     A      0        <undef>           0
pref_webmailaddressbooklistlimit  N     A      0        <undef>           0
pref_webmailattachlimit           N     A      0        <undef>           0
pref_webmailattachsizelimit       N     A      0        <undef>           0
pref_webmailconfirmdelete         B     L      0        <undef>           0
pref_webmailmsgattachlimit        N     A      0        <undef>           0
pref_webmailusesignature          B     L      0        <undef>           0
```

Notes:

The Result column of the table displayed by the `GetAccountCos` option contains the value of the attribute that currently applies to the queried account. This value is determined by the precedence rule of the attribute. For information on attribute precedence rules, see Chapter 2.

The attributes containing an asterisk (*) indicate that the value shown is not the default value.

DeleteAccountCos

A class of service attribute can be deleted from an account with `imdbcontrol` by using the `DeleteAccountCos` option. Three parameters are required with this option: the username portion of the account's primary SMTP address, the domain with which the account is associated, and the name of the class of service attribute being deleted from the account.

Usage:

```
imdbcontrol DeleteAccountCos <username> <domain> <attribute>
```

Where:

`username`	The local part of the SMTP address for the account (the part which precedes the "@" symbol in the e-mail address).
`domain`	Domain with which the account is associated.
`attribute`	The name of the attribute being deleted from the account.

Example:

```
imdbcontrol DeleteAccountCos jdoe software.com pref_imap
```

This example deletes the class of service attribute `pref_imap` from the account `jdoe@software.com`.

3.1.5 Class of Service Operations

The `imdbcontrol` operations described in this section are related to InterMail classes of service.

CreateCos

New classes of service can be created with `imdbcontrol` by using the `CreateCos` option. Only one parameter is required with this option: the name of the new class of service being created.

Usage:

```
imdbcontrol CreateCos <classOfService>
```

Where:

`classOfService`	The name of the new class of service.

Example:

```
imdbcontrol CreateCos Premium
```

This example creates a new class of service named "Premium".

DeleteCos

Classes of service can be deleted with `imdbcontrol` by using the `DeleteCos` option. Only one parameter is required with this option: the name of the class of service being deleted.

Usage:

```
imdbcontrol DeleteCos <classOfService>
```

Where:

`classOfService` The name of the class of service to be deleted.

Example:

```
imdbcontrol DeleteCos Premium++
```

This command deletes the class of service named “Premium++”.

ListCosNames

The list of existing classes of service can be displayed with `imdbcontrol` by using the `ListCosNames` option. This option takes no parameters, and simply produces a list of classes of service.

Usage:

```
imdbcontrol ListCosNames
```

ShowCos

The attributes associated with a class of service, as well as their values, can be displayed with `imdbcontrol` by using the `ShowCos` option. Only one parameter is required with this option: the name of the class of service being queried.

Usage:

```
imdbcontrol ShowCos <classOfService>
```

Where:

`classOfService` The name of the class of service to be queried.

Example:

```
imdbcontrol ShowCos Premium
```

This command lists the attributes associated with the class of service named “Premium++”. This command generates output like the following:

```
pop:1
smtp:1
```


CreateCosAttribute

New class of service attributes can be created with `imdbcontrol` by using the `CreateCosAttribute` option. Three parameters are required with this option: the name of the new attribute being created, its precedence rule, and its type.

Usage:

```
imdbcontrol CreateCosAttribute <attribute> <rule> <type>
```

Where:

`attribute`	The name of the new class of service attribute being created.
`rule`	The precedence rule that defines whether the value of the attribute for an account is defined in the account or its class of service. The possible values for this parameter are: `C` (class of service value is used) `A` (account value is used) `L` (the lesser of the class of service and account values is used) `G` (the greater of the class of service and account values is used) In all cases, if the attribute is defined for either the class of service or the account, but not both, the value of the one instance is used.
`type`	The type of the new attribute. The possible values for this parameter are: `S` (string), `N` (numeric), and `B` (boolean).

Example:

```
imdbcontrol CreateCosAttribute pref_pager L B
```

This example creates a new class of service attribute named `pref_pager`. This attribute is defined as boolean (`B`), and the precedence rule specifies that the lower (`L`) of the class of service and per-account values should be used as the value for an account.

ModifyCosAttribute

The precedence rule associated with a class of service attribute can be modified with `imdbcontrol` by using the `ModifyCosAttribute` option. Two parameters are required with this option: the name of the new attribute being modified, and the new precedence rule value.

Usage:

```
imdbcontrol ModifyCosAttribute <attribute> <rule>
```

Where:

`attribute` The name of the class of service attribute being modified.

`rule` The precedence rule that defines whether the value of the attribute for an account is defined in the account or its class of service. The possible values for this parameter are:

`C` (class of service value is used)
`A` (account value is used)
`L` (the lesser of the class of service and account values is used)
`G` (the greater of the class of service and account values is used)

In all cases, if the attribute is defined for either the class of service or the account, but not both, the value of the one instance is used.

Example:

```
imdbcontrol ModifyCosAttribute pref_pager A
```

This example changes the precedence rule for the attribute `pref_pager` to use the per-account (`A`) value of the attribute.

ListCosAttributes

Information on existing class of service attributes can be listed by `imdbcontrol` by using the `ListCosAttributes` option. This option takes no parameters, and simply outputs the following information for each class of service attribute:

- ID number
- name
- precedence rule, which can be one of the following values:

  `C` (class of service value overrides per-account values)
  `A` (per-account values override class of service value)
  `L` (the lesser of the class of service and per-account values is used)
  `G` (the greater of the class of service and per-account values is used).

- data type, which can be one of the following values:

  `B` (boolean)
  `N` (numeric)
  `S` (string)

Usage:

```
imdbcontrol ListCosAttributes
```

Example:

This option generates output like the following:

```
ID: 14; Known ID: 14; Name: pref_autoreply; Rule: C; Syntax: B
ID: 15; Known ID: 15; Name: permm_echo; Rule: C; Syntax: B
ID: 12; Known ID: 12; Name: pref_forwarding; Rule: C; Syntax: B
ID: 3; Known ID: 3; Name: pref_imap; Rule: C; Syntax: B
...
```

DeleteCosAttribute

Existing class of service attributes can be deleted with `imdbcontrol` by using the `DeleteCosAttribute` option. Only one parameter is required with this option: the name of the attribute being deleted.

Usage:

```
imdbcontrol DeleteCosAttribute <attribute>
```

Where:

`attribute`	The name of the class of service attribute being deleted.

Example:

```
imdbcontrol DeleteCosAttribute pref_pager
```

This example deletes the class of service attribute named `pref_pager`.

SetCosAttribute

The value of a class of service attribute can be defined for a class of service with `imdbcontrol` by using the `SetCosAttribute` option. Three parameters are required with this option: the name of the class of service, the name of the attribute, and the value of the attribute.

Usage:

```
imdbcontrol SetCosAttribute <classOfService> <attribute> <value>
```

Where:

`classOfService`	The name of the class of service for which the attribute is being defined.
`attribute`	The name of the attribute.
`value`	The value of the attribute.

Example:

```
imdbcontrol SetCosAttribute Basic pref_popSSL 1
```

This example sets the attribute `pref_popSSL` for a class of service named “Basic”. The given value of `1` indicates that this boolean attribute is being set to on.

UnsetCosAttribute

An attribute can be deleted from a class of service with `imdbcontrol` by using the `SetCosAttribute` option. When this option is executed, the existing value of the attribute is deleted for the given class of service, which is no longer associated with that attribute. Two parameters are required with this option: the name of the class of service and the name of the attribute.

Usage:

```
imdbcontrol UnsetCosAttribute <classOfService> <attribute>
```

Where:

`classOfService`	The name of the class of service for which the attribute is being removed.
`attribute`	The name of the attribute.

Example:

```
imdbcontrol UnsetCosAttribute Basic pref_popSSL
```

This example disassociates the attribute `pref_popSSL` with a class of service named “Basic”. The value for this attribute in the Basic class of service is deleted.

3.1.6 System Operations

The `imdbcontrol` operations described in this section are specific to managing the Integrated Services Directory logs.

Warning! These `imdbcontrol` operations are not meant to be executed manually, and can cause serious problems if used incorrectly. These options should be used only in conjunction with the assistance of Software.com technical support or Professional Services.

ExpireLogs

The Integrated Services Directory logs that are older than a certain number of hours can be expired with `imdbcontrol` by using the `ExpireLogs` option. This option takes one parameter: the minimum age, in hours, of logs which should be expired.

Usage:

```
imdbcontrol ExpireLogs <hours>
```

Where:

`hours`	The minimum age, in hours, of logs which should be expired. For example, if the specified number of hours is 6, any logs that are more than six hours old will be expired.

Example:

```
imdbcontrol ExpireLogs 12
```

This example expires all logs which are more than 12 hours old.

GetFirstModified

The ID of the earliest log entry in the Integrated Services Directory can be retrieved with `imdbcontrol` by using the `GetFirstModified` option. This option takes no parameters, and returns the log entry ID and the time of the first entry in the current Integrated Services Directory log.

Usage:

```
imdbcontrol GetFirstModified
```

Output:

The `imdbcontrol GetFirstModified` command generates output of the following form:

```
Earliest Log Entry: 87, at 10/08/97 11:24:33
```

GetLastModified

The ID of the latest Integrated Services Directory log entry can be retrieved with `imdbcontrol` by using the `GetLastModified` option. This option takes no parameters, and simply returns the log ID.

Usage:

```
imdbcontrol GetLastModified
```

Output:

The `imdbcontrol GetLastModified` command generates output of the following form:

```
Latest Log ID: 88
```

3.2 imldapcontrol

The `imldapcontrol` command provides low-level inspection and editing of information stored in the LDAP data within the Integrated Services Directory. InterManager makes extensive use of this LDAP data. Since the LDAP data model does not explicitly enforce consistency constraints or access control in its data model, many of these mechanisms are enforced within InterManager itself above the Integrated Services Directory database representation. `imldapcontrol` bypasses this InterManager layer and directly accesses the LDAP data, and can therefore inadvertently cause data integrity or security problems when used for editing data.

Warning! `imldapcontrol` is not recommended for regular administrative use, and should not be used for editing data except with a complete understanding of the security and data integrity issues within the ISD, and even then only in conjunction with Software.com professional services or support.

3.2.1 Syntax

The syntax for using `imldapcontrol` is as follows:

```
imdbcontrol [-d] [-u <user>/<password>@<db>] [-help] [-] <options>
```

Where:

`-debug`	Prints progress information for the operation.

`-user`	Specifies database user information, in the form of:
	`username/password@db-service`
	For example:
	`imail/imail@IMD1`

`-help`	Displays usage information for imldapcontrol.
`-`	Specifies that multiple lines should be processed from STDIN and executed one-by-one, just as if they were specified with separate invocations of `imldapcontrol`.
`-echo`	Echoes lines from STDIN as they are read.
`-errcountinue`	Continue on error while reading from STDIN.
`-errexit`	Abort on error while reading from STDIN.
`-remark`	Ignore the remainder of the command-line.

3.2.2 Execution Options

The operation performed by `imldapcontrol` is specified by a command-line option. The available options are.

`-addentry`	Adds a new LDAP object.
`-checkschema`	Enables/disables checking of LDAP schema rules upon the creation or modification of all entries.
`-delentry`	Deletes an LDAP object.
`-listattribute`	Displays the description of an LDAP attribute.
`-listclass`	Displays the attributes of an LDAP class.
`-listentry`	Displays the attributes for a specific entry.
`-modentry`	Modifies an existing LDAP object.
`-modrdn`	Modifies the relative distinguished name (RDN) of an existing LDAP object.
`-rename`	Renames an existing entry's distinguished name (DN).
`-replicate`	Enables/disables replication of data.

Note: *Unlike the* `imdbcontrol` *options described in Section 3.1.2, the* `imldapcontrol` *option flags are case-sensitive.*

3.2.3 LDAP Operations

The `imldapcontrol` options described in this section relate specifically to LDAP objects stored in the Integrated Services Directory. These are the objects used by InterManager.

addentry

New LDAP objects can be added with `imldapcontrol` by using the `-addentry` option. This command takes as parameters the attributes (or binary attributes) of this new entry. Any number of attributes may be specified on the command-line, which must also include the DN of the new entry.

Usage:

```
imldapcontrol -addentry <dn>
      -attribute <attribute> <value> ...
      -attributebinary <attribute> <file> ...
```

Where:

`dn`	The distinguished name of the new entry.
`attribute`	The name of each attribute (or binary attribute) that is defined for the new LDAP object.
`value`	The ASCII text value of the specified attribute.
`file`	The binary file which contains the value for the specified binary attribute.

Example:

```
imldapcontrol -addentry c=us -attr objectclass country
```

This command creates a new object of the country class, which has the distinguished name `c=us`.

```
imldapcontrol -addentry o=Software.com,c=us -attr objectclass
organization
```

This command creates a new object of the organization class, which has the distinguished name `o=Software.com,c=us`.

```
imldapcontrol -addentry "cn=Jeff Doe,o=Software.com,c=us" -attr
objectclass person -attr cn "Jeff Doe" -attr cn Jeff -attr sn Doe -
attr c us -attr l Bellevue -attribute telephoneNumber "(425) 867-
5309" -attr o Software.com -attr givenname Jeff
```

This command creates a new object of the person class, which has the distinguished name `cn=Jeff Doe,o=Software.com,c=us`, and the following attributes:

- The common names (`cn`) Jeff Doe, Jeff.
- The surname (`sn`) Doe.
- The country (`c`) of the United States (`us`).
- The locality (`l`) Bellevue.
- The phone number (`telephonenumber`) 425-867-5309.
- The organization (`o`) Software.com.

modentry

Existing LDAP objects can be modified with `imldapcontrol` by using the `-modentry` option. This command itself has a variety of command-line modifiers to specify that attributes should be added, deleted, or modified for the given DN. The parameters required to execute this command are dependent on the operation being performed.

Usage:

```
imldapcontrol -modentry <dn>
      -attribute <attribute> <value>
      -delattribute <attribute> [<value>]
      -modattribute <attribute> <value>
      -attributebinary <attribute> <file>
      -delattributebinary <attribute>
      -modattributebinary <attribute> <file>
```

Modifiers:

The following command-line modifiers control the behavior of the `-modentry` option:

`-attribute`	Adds a new attribute tot he specified entry.
`-delattribute`	Deletes an attribute from the specified entry. If the attribute is multi-valued, you must also specify the value that should be deleted from this attribute.
`-modattribute`	Modifies the value of an existing attribute.
`-attributebinary`	Adds a new binary attribute to an existing entry.
`-delattributebinary`	Deletes a binary attribute from the specified entry.
`-modattributebinary`	Modifies the value of an existing binary attribute. This option requires the name of an existing binary file that contains the new data for the binary attribute.

Parameters:

`dn`	The distinguished name of the entry to be modified.
`attribute`	The name of the attribute that is being added, deleted, or modified.
`value`	The value of the added or modified attribute. If a deleted attribute is multi-valued, this parameter specifies the value to be deleted.
`file`	A binary file that contains the data for an attribute (for example, a GIF image file).

Example:

```
imldapcontrol -modentry "cn=Joe Smith,o=Software.com,c=us" -
attribute telephoneNumber"(425) 867-5309"
```

This command adds the telephone number attribute to the LDAP entry `cn=Joe Smith,o=Software.com,c=us`.

```
imldapcontrol -modentry "cn=Joe Smith,o=Software.com,c=us"
-delattribute l
```

This command removes the locality (`l`) attribute from the same LDAP entry.

delentry

Existing LDAP objects can be deleted from the ISD with `imldapcontrol` by using the `-delentry` option. This command takes one parameter: the distinguished name of the entry to be deleted.

Usage:

```
imldapcontrol -delentry <dn>
```

Where:

`dn`	The distinguished name of the entry being deleted.

Example:

```
imldapcontrol -delentry "cn=Joe Smith,o=Software.com,c=us"
```

This command deletes the LDAP entry for Joe Smith.

listentry

Information for an existing DN can be retrieved from the ISD with `imldapcontrol` by using the `-listentry` option. This command requires one parameter: the DN of the entry in question. By default, this option outputs all attributes associated with the DN, but you can narrow the output to one or more attributes

Usage:

```
imldapcontrol -listentry <dn> [-attribute <attribute>]
```

Where:

`dn`	The distinguished name of the entry being queried.
`attribute`	The attribute being queried.

Example:

```
imldapcontrol -listentry "cn=Jeff Doe,o=Software.com,c=us" -
attribute telephoneNumber
```

This command requests the value of the telephone number attribute associated with the entry `cn=Jeff Doe,o=Software.com,c=us` (created in one of the `-addentry` examples above).

```
imldapcontrol -listentry "cn=Jeff Doe,o=Software.com,c=us"
```

This command requests information for all attributes associated with the entry `cn=Jeff Doe,o=Software.com,c=us`, and creates output like the following:

```
cn=Jeff Doe,o=Software.com,c=us
      objectclass = person
      cn = Jeff Doe
      cn = Jeff
      sn = Doe
      c = us
      l = Bellevue
      o = Software.com
      telephonenumber = (425) 867-5309
      givenname = Jeff
```


modrdn

The final RDN component of an existing DN can be modified with `imldapcontrol` by using the `-modrdn` option. This command takes three parameters: the DN of the entry in question, the new RDN data that is being added, and a flag for specifying that the replaced information should be saved as an attribute of the DN.

Usage:

```
imldapcontrol -modrdn <dn> <new-rdn> <save-old>
```

Where:

`dn`	The distinguished name of the entry being modified.
`new-rdn`	The RDN of the portion of the selected entry that is being modified.
`save-old`	Specifies that the replaced RDN should be saved as an attribute of the DN. The possible values for this parameter are `Y` (yes) and `N` (no).

Example:

```
imldapcontrol -modrdn "cn=Rick Brown,o=Software.com,c=us" "cn=Dr.
Richard J. Brown III" N
```

This command changes common name (`cn`) RDN of the entry `cn=Rick Brown,o=Software.com,c=us` from "Rick Brown" to "Dr. Richard J. Brown III." This changes the DN of this entry to:

```
cn=Dr. Richard J. Brown III, o=Software.com,c=us
```

rename

The distinguished name (DN) of an existing entry can be modified with `imldapcontrol` by using the `-rename` option. This command requires two parameters: the DN of the entry in question, and the new relative distinguished name (RDN) for the entry. You may also specify the DN of the entry's new parent object. If the DN of the parent object is present, the specified RDN is combined with the parent's DN to create the new DN for the entry. If the parent DN is omitted, only the RDN of the entry is modified.

Usage:

```
imldapcontrol -rename <dn> <new-rdn> [<parent-dn>] [-deloldattrval]
```

Where:

`dn`	The distinguished name of the entry being modified.
`new-rdn`	The new RDN for the entry.
`parent-dn`	The DN of the entry's new parent. If omitted, the current parent of the entry is assumed.
`-deleteoldval`	Specifies that the original RDN value should be removed from the appropriate (multi-valued) attribute.

Example:

```
imldapcontrol -rename "cn=Barney Brown,ou=engineering,dc=software,
      dc=com" "cn=Barney Smith"
```


This example changes the RDN of this entry from "cn=Barney Brown" to "cn=Barney Smith", and also adds the new RDN value to the cn attribute of the entry. This means that the cn attribute of this entry now contains both the values "Barney Brown" and "Barney Smith".

```
imldapcontrol -rename "cn=Barney Brown,ou=engineering,dc=software,
      dc=com" "cn=Barney Smith" "ou=marketing,dc=software,dc=com"
```

This operation is similar to the previous example, and changes the RDN of this entry from "cn=Barney Brown" to "cn=Barney Smith". However, because a parent DN — in this case, "ou=marketing,dc=software,dc=com" — is also included on the command-line, this operation also changes the parent of the entry. The result is that the DN of the entry becomes "cn=Barney Smith,ou=marketing,dc=software,dc=com". The cn attribute of this entry now contains both the values "Barney Brown" and "Barney Smith", while the ou attribute now contains both the values "engineering" and "marketing".

```
imldapcontrol -rename "cn=Barney Smith,ou=marketing,dc=software,
      dc=com" "cn=Barney Brown" -deleteoldval
```

This example changes the RDN of the entry shown in the previous examples from "cn=Barney Smith" back to "cn=Barney Brown". However, because the -deleteoldval flag is used, the previous RDN value is removed from the cn attribute of the entry. This means that the cn of this entry now contains only the value "Barney Brown".

listattribute

Information for a particular LDAP attribute can be viewed with imldapcontrol by using the -listattribute option. This command takes one parameter: the name of the attribute being described.

Usage:

```
imldapcontrol -listattribute <attribute>
```

Where:

attribute	The name of the attribute for which a description is being requested.

Example:

```
imldapcontrol -listattribute telephonenumber
```

This command requests information on the telephonenumber attribute, and gives output like the following:

```
Attr ID:    21
Attr Name:  telephonenumber
Attr Type:  String
```

listclass

LDAP objects are created from classes, which are containers for attributes. Objects created from a class can have only the attributes defined in that class. The list of the attributes of a specific LDAP class can be viewed with `imldapcontrol` by using the `-listclass` option. This command takes one parameter: the name of the class being described.

Usage:

```
imldapcontrol -listclass <class-name>
```

Where:

`class-name`	The name of the class for which attribute information is being requested.

Example:

```
imldapcontrol -listclass person
```

This command requests attribute information for the `person` class, and gives output like the following:

```
Class Name:  person
Class Desc:  Describes attributes appropriate for white pages
listings of people
Attr Count:  9

Attr ID:    3
Attr Name:  cn
Attr Type:  String

Attr ID:    4
Attr Name:  sn
Attr Type:  String

Attr ID:    35
Attr Name:  userpassword
Attr Type:  String
Attribute:  Optional

Attr ID:    34
Attr Name:  seealso
Attr Type:  String
Attribute:  Optional

Attr ID:    21
Attr Name:  telephonenumber
Attr Type:  String
Attribute:  Optional
```



```
Attr ID:    14
Attr Name:  description
Attr Type:  String
Attribute:  Optional

Attr ID:    42
Attr Name:  givenname
Attr Type:  String
Attribute:  Optional

Attr ID:    57
Attr Name:  mail
Attr Type:  String
Attribute:  Optional

Attr ID:    1
Attr Name:  objectclass
Attr Type:  String
```

checkschema

As an error-checking mechanism, you can request that `imldapcontrol` check all transactions against the LDAP schema rules of the Integrated Services Directory. This option can be globally enabled or disabled by executing `imldapcontrol` with the `-checkschema` option, which takes no parameters. If schema checking is enabled, and an attempted `imldapcontrol` operation does not adhere to the appropriate schema rules, the operation fails.

Usage:

```
imldapcontrol -checkschema
```

replicate

Changes to LDAP data in the Integrated Services Directory are typically propagated to the LDAP server, which makes this information available to LDAP clients. However, the propagation of changes made via `imldapcontrol` can be globally enabled or disabled with the `-replicate` option, which takes no parameters.

Usage:

```
imldapcontrol -replicate
```

4

Integrated Services Directory Schema

The Integrated Services Directory (ISD) contains database tables for administration, mail, class of service, and LDAP information. This chapter contains descriptions of the database tables for each of these three types. In addition, the following information is included:

- a diagram indicating the relationship of keys and constraints between tables
- the name of each table
- the name of each column in these tables
- data type, possible values, and other information associated with each database column

Warning! The database schema is subject to change without notice by Software.com. When making changes to values in the database, never make those changes directly; instead use the utilities and/or APIs described in later chapters of this manual.

Existing database schema should not be modified. Attempts to modify the schema may result in invalidation of your support contract.

4.1 ISD Database Tables

The Integrated Services Directory contains database tables for the following types of information:

- Mail—tables that specifically contain information related to mail operations (e.g., account, domains)
- COS—tables that describe the class of service (COS) implementation; these tables are used by InterMail and InterManager.
- LDAP—tables that describe the LDAP implementation; these tables are used by InterManager.
- Administrative—static tables that are used in all Software.com Inter* products.

Figure 3 illustrates the relationship between these types of tables.

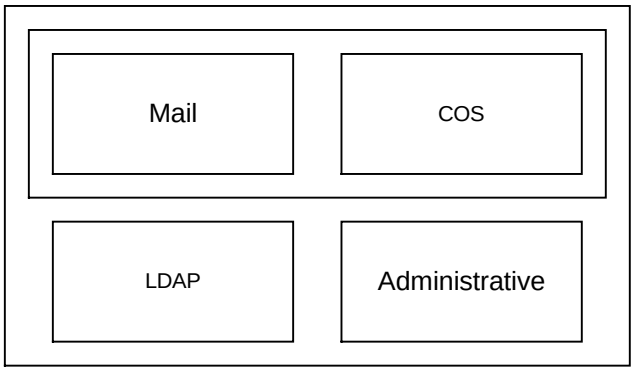


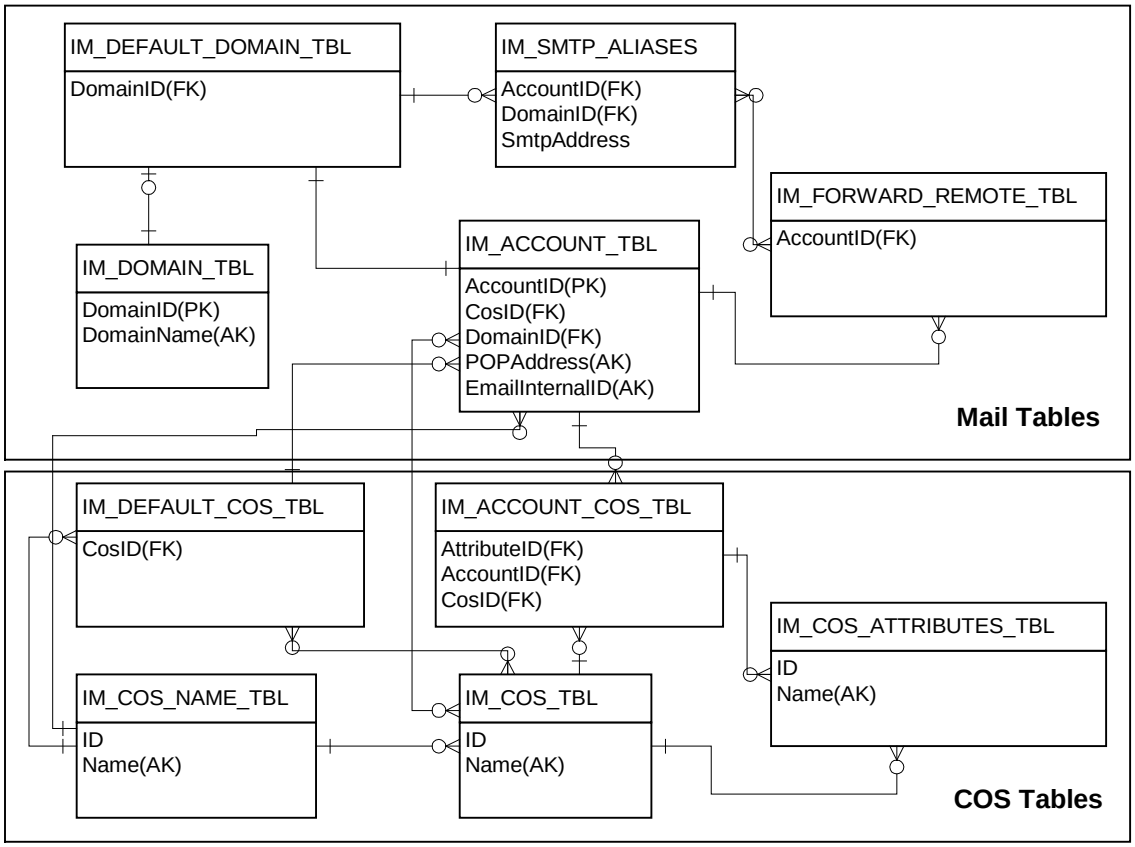
Figure 3. Relationship of Objects in the ISD.

Mail and class of service tables work together and have Entity/Relationships with each other (see Figure 4). InterMail, SelfCare, and WebMail all use the Mail, class of service, and Administrative tables. The LDAP tables are used by InterManager only. The LDAP tables exist only if the InterManager product has been installed. However, updates that occur in Mail, class of service, or LDAP tables can have an effect on Administrative tables (e.g. adding an account will affect the IM_LAST_MODIFIED_TBL table).

Note: *The ISD implementation contains both tables and views. Tables contain a '_TBL' extension (i.e. IM_LOG_TBL) and views have no extension (i.e. IM_BINARY_VALUE).*

4.2 Relationship of Mail and COS Tables

The following diagram describes the relationships of schema elements in all mail and class of service (COS) tables. Keys are listed in each table so that the user can determine the specific relationship of constraints.



Legend

- One-to-One Relationship
- One-to-Many (Optional) Relationship
- One-to-One (Optional) Relationship

AK	Unique and Indexed
IE	Indexed
FK	Foreign Key
PK	Primary Key

Figure 4. Mail and COS E/R.

Figure 4 depicts the database tables associated with mail as well as the relationship these tables have with one another. Tables are grouped by logical primary function (e.g., Mail, Administration, COS). In addition, Administrative tables (e.g., IM_LOG_TBL) perform administrative or maintenance functions and do not affect data in other tables in the database.

***Note**: The IM_ACCOUNT_TYPES_TBL, IM_STATUS_TYPES_TBL, and IM_DOMAIN_TYPES_TBL tables are described in the following section; however, these tables are static and do not have an entity/relationship. Therefore, they are not depicted in Figure 4.*

4.3 Mail Object Tables

The tables described in this section are related specifically to the mail objects (accounts, domains, etc.) stored in the Integrated Services Directory.

4.3.1 IM_ACCOUNT_TBL

This table contains all the information about InterMail accounts; these are the e-mail accounts for which your installation of InterMail accepts messages.

Attribute	Description	Type	Length	Notes
`AccountID`	Unique identifier of the account.	NUMBER	< 10^38	Primary key; Unique. The value of this field is generated automatically by the database.
`AccountIDType`	The type of account.	CHAR	1	Possible values: `S` (standard), `A` (administrative) This value is cached.
`POPAddress`	User's POP/IMAP login name.	VARCHAR2	129	Unique. This value is cached.
`PrimarySMTPAddress`	The local portion of the account's primary SMTP address.	VARCHAR2	64	
`DomainID`	The ID of the local mail domain with which the account is associated.	NUMBER		Foreign key to `DomainID` in `IM_DOMAIN_TBL`.
`DeliveryHost`	Hostname of the MSS which includes the account's mailbox.	VARCHAR2	32	This value is cached.
`LocalDelivery`	Indicates delivery of mail to the account's message store.	CHAR	3	Possible values: `P` (enabled), `N` (disabled) This value is cached.
`HandlerDeliver`	(This field is reserved for future use.)	CHAR	3	

Attribute	Description	Type	Length	Notes
`Status`	The status of the account.	CHAR	1	Possible values: `A` (active), `D` (deleted), `S` (suspended), `L` (locked), `M` (maintenance), `P` (proxy)
`Password`	POP/IMAP password	VARCHAR2	64	Possible values: `C` (clear), `M` (MD5-PO), `U` (Unix)
`PasswordType`	Hash Algorithm	CHAR	1	
`EmailInternalID`	Unique ID kept in customer database and MSS database.	NUMBER	< 10^38	Unique. This value is cached.
`AutoReplyMode`	Indicates that the auto-reply feature is enabled/disabled.	CHAR	1	Possible values: `N` (none), `V` (vacation), `R` (reply), `E` (echo). This value is cached.
`AutoReplyHost`	The name of the host on which the account's auto-reply message is stored.	VARCHAR2	32	This value is cached.
`ForwardFlag`	Flag to enable/disable mail forwarding.	CHAR	1	Possible values: `F` (enabled), `N` (disabled) This value is cached.
`ForwardCount`	The number of forwarding addresses associated with the account.	NUMBER	< 10^38	
`COSID`	The class of service associated with an account.	NUMBER		Foreign key to `IM_COS_TBL`

4.3.2 IM_ACCOUNT_TYPES_TBL

This static table lists and describes the available account types.

Attribute	Description	Type	Length	Notes
`AccountIDType`	The type of account.	CHAR	1	Possible values: `S` (user), `A` (administrator),
`Description`	The textual description of the account.	VARCHAR2	50	Possible values: user, admin

4.3.3 IM_DEFAULT_DOMAIN_TBL

This table contains a single entry: the default local mail domain. Specifying a default domain is a convenience to simplify database operations. For example, when creating e-mail accounts with `imdbcontrol`, if domain is specified, the default domain specified here is assumed.

Attribute	Description	Type	Length	Notes
`DomainID`	Unique identifier of the default domain.	NUMBER	< 10^38	Primary key; Foreign key to `DomainID` in `IM_DOMAIN_TBL`.
`DomainName`	The name of the default domain (for example, `mydomain.com`)	VARCHAR2	64	

4.3.4 IM_DOMAIN_TBL

This contains a list of all of the domains in the Integrated Services Directory.

Attribute	Description	Type	Length	Notes
`DomainID`	Unique identifier of the domain	NUMBER	< 10^38	Primary key; Indexed The value of this field is generated automatically by the database.
`DomainName`	The domain name text string (e.g., `mydomain.com`)	VARCHAR2	64	Indexed; Unique
`ReversedDomainName`	The domain name text string in reverse order (e.g., `moc.niamodym`); used to efficiently locate sub-domains of a given domain.	VARCHAR2	65	

Attribute	Description	Type	Length	Notes
Type	The type of the domain.	CHAR	1	Possible values: L (local), N (non-authoritative), R (rewrite), D (deleted).
RelayHost	Used for non-authoritative domain relay. This value is the name of the host to which mail should be relayed.	VARCHAR2	97	
RewriteDomain	Used with domain rewriting. This value is the domain name that should be substituted for the original domain name on rewritten messages.	VARCHAR	64	Foreign key to DomainName in IM_DOMAIN_TBL.
WildcardAccount	The primary SMTP address of the e-mail account which receives messages sent to unknown addresses in the domain.	VARCHAR2	64	
LastModified	(For internal use only)	DATE		

4.3.5 IM _DOMAIN_TYPES_TBL

This static table lists and describes the available domain types.

Attribute	Description	Type	Length	Notes
DomainType	Unique identifier of the domain type.	CHAR	1	Primary key; Unique
Description	Description of the domain type.	VARCHAR2	50	

4.3.6 IM_FORWARD_REMOTE_TBL

This table defines mail forwarding from an InterMail account to an e-mail account in a remote domain. Note that this table does not define forwarding delivery to other InterMail accounts in local mail domains.

Note: *The IM_FORWARD_REMOTE_TBL stores all forwards. In previous implementations of the database schemas, the IM_FORWARD_LOCAL table stored local forwards and IM_FORWARD_REMOTE table stored remote forwards.*

Attribute	Description	Type	Length	Notes
`AccountID`	Unique identifier of the InterMail account from which e-mail is forwarded.	NUMBER	< 10^38	Primary key
`ForwardTo`	Address to which mail is forwarded (for example, `jdoe@software.com`)	Text	128	Primary key

4.3.7 IM_SMTP_ALIASES_TBL

This table contains all of the SMTP alias addresses used by InterMail accounts.

Attribute	Description	Type	Length	Notes
`AccountIDType`	Type of the account ID.	Text	1	
`AccountID`	Unique identifier of the InterMail account for which the alias is defined.	NUMBER	< 10^38	
`DomainID`	Unique identifier of the domain in which the alias is defined.	NUMBER	< 10^38	Primary key; Foreign key to `DomainID` in `IM_DOMAIN_TBL`
`SMTPaddress`	SMTP address for the alias.	Text	64	Primary key

4.3.8 IM_STATUS_TYPES_TBL

This static table defines the available account status types. Status flags are used to define e-mail accounts as active, deleted, suspended, locked, or in maintenance mode.

Attribute	Description	Type	Length	Notes
`StatusType`	The single character that represents the account status type.	CHAR	1	Possible values: `A` (active), `D` (deleted), `S` (suspended), `L` (locked), `M` (maintenance), `P` (proxy)
`Description`	A description of the status type.	VARCHAR2	50	

4.4 COS Tables

The following tables are related to the class of service (COS) Strategy and specify the relationships between accounts and classes of service.

4.4.1 IM_ACCOUNT_COS_TBL

This table contains the class of service attribute values for all accounts.

Attribute	Description	Type	Length	Notes
`AccountID`	Unique identifier of the InterMail account.	NUMBER	< 10^38	Unique; Indexed
`AttributeID`	Unique identifier for each class of service attribute.	NUMBER		
`Value`	Value of the class of service attribute.	VARCHAR2	255	This value is both used for numeric and boolean values (0 represents “off” and 1 represents “on” when the attribute is a boolean).

4.4.2 IM_COS_ATTRIBUTE_TBL

This table describes all class of service attributes in terms of the rule and syntax.

Attribute	Description	Type	Length	Notes
AttributeID	Unique numeric value for each class of service attribute.	NUMBER		
Name	Class of service attribute.	VARCHAR2	255	
Rule	Precedent rule that determines what value will be used for an attribute in the event that a conflict between class of service and account level exists.	CHAR	1	Possible values: C (class of service value), A (Account value), L (Lesser value), G (Greater value)
Syntax	Determines the syntax for the value of the precedent rule.	CHAR	1	Possible values: N (Numeric), B (Boolean)

4.4.3 IM_COS_NAME_TBL

This table contains all the global class of service names.

Attribute	Description	Type	Length	Notes
COSID	The ID for the global class of service.	NUMBER		
Name	The name of the global class of service.	VARCHAR2	255	

4.4.4 IM_COS_TBL

This table contains the class of service attribute values for all global class of service.

Attribute	Description	Type	Length	Notes
COSID	The ID for each class of service.	NUMBER		
AttributeID	Unique numeric value for each class of service attribute.	NUMBER		
Value	Value of the class of service attribute.	VARCHAR2	255	This value is both used for numeric and boolean values (0 represents “off” and 1 represents “on” when the attribute is a boolean).

4.4.5 IM_DEFAULT_COS_TBL

This table contains the name of the default class of service.

Attribute	Description	Type	Length	Notes
COSID	The ID for the default class of service.	NUMBER		

4.5 LDAP Object Tables

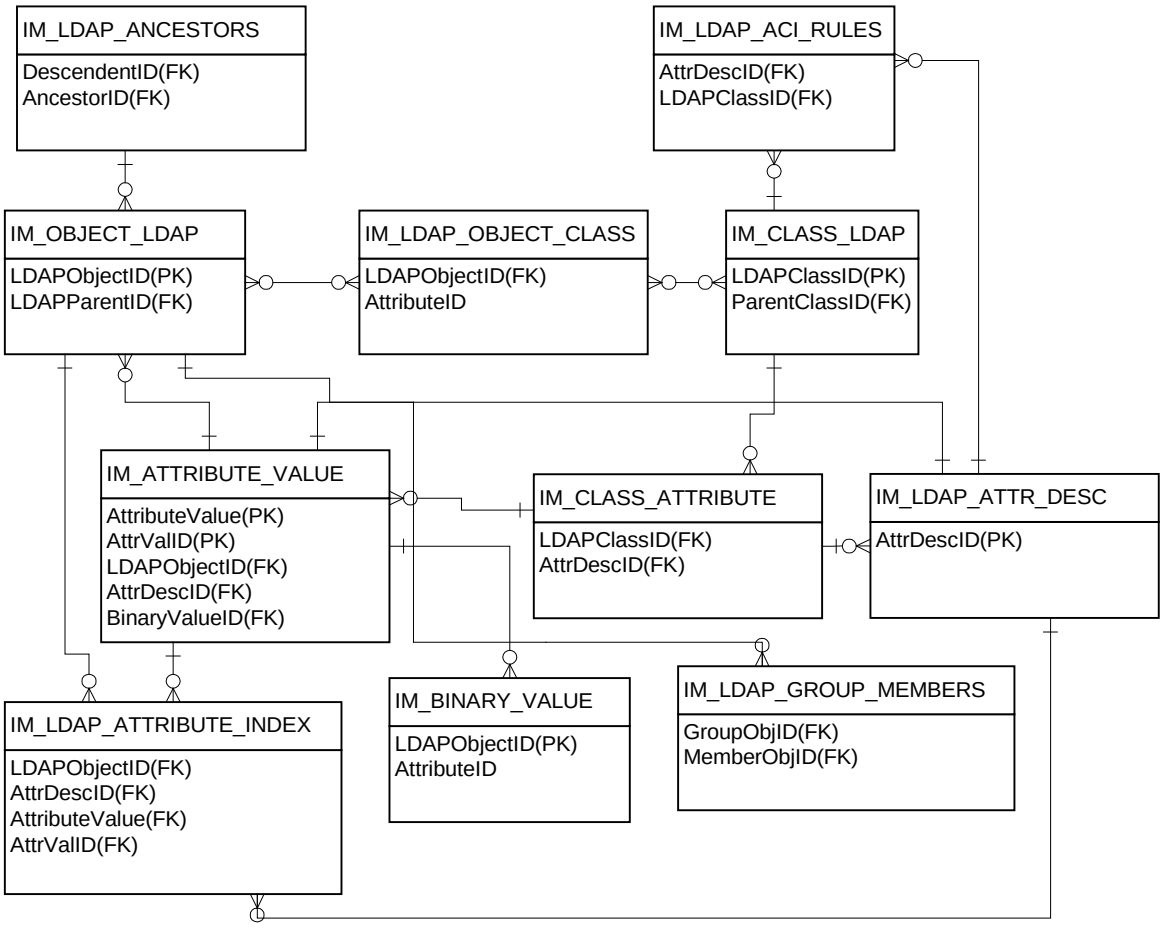
The tables described in this section are related to LDAP, used by InterManager.

The information stored in the LDAP tables relates to the following data types:

- A *class* is a container for a list of attribute descriptors. Each class is identified by a unique name. Classes are also referred to as object classes.
- An *object* is a container for a list of attributes. Each object is an instance of a particular class, and is uniquely identified by a Distinguished Name (DN). Objects are also referred to as LDAP entries.
- An *attribute descriptor* contains a description of an attribute. Within a class, it is identified by a unique name.
- An *attribute* is an instance of an attribute descriptor, and contains the value of an attribute. An attribute may be multi-valued, and the value may be a string, a number, or binary data.

4.6 Relationship of LDAP Tables

The following diagram illustrates the relationship of LDAP tables.



Legend

	One-to-One Relationship
	One-to-Many (Optional) Relationship
	One-to-One (Optional) Relationship
AK	Unique and Indexed
IE	Indexed
FK	Foreign Key
PK	Primary Key

Figure 5. LDAP E/R.

Note: *The IM_LDAP_ACI_RULES, IM_LDAP_CONFIGURATION, IM_LDAP_FILTER_PATTERN,IM_LDAP_LAST_FILTER, IM_LDAP_SCHEMA_VERSION, and IM_LOG_LDAP tables are described in the following section; however, these tables are static and have no relationship with another table. Therefore, they are not shown in Figure 5.*

4.6.1 IM_ATTRIBUTE_VALUE

This table contains the values of all attributes that are associated with LDAP objects. These values are typically short data types, such as strings of less than one kilobyte, or numeric data stored in ASCII format. Attributes that have a binary value include an entry in this table, which references the entry in the `IM_ATTRIBUTE_BINARY_VALUE` table that holds the binary value.

Attribute	Description	Type	Length	Notes
`AttrValID`	Forms a referential integrity constraint relationship between an attribute value and its indexed value (if indexed). Used to automatically purge the indexed value if the value is deleted and for consistency.	NUMBER		Foreign key to `AttrValID` in `IM_LDAP_ATTRIBUTE_INDEX`
`LDAPObjectID`	Unique ID of the LDAP object with which the attribute is associated.	NUMBER		Foreign key to `LDAPObjectID` in `IM_OBJECT_LDAP`
`AttrDescID`	Unique ID of the type of the attribute.	NUMBER		Foreign key to `AttrDescID` in `IM_LDAP_ATTR_DESC`
`BinaryValueID`	Unique ID of the entry in `IM_BINARY_VALUE` which contains the value for binary attributes.	NUMBER		Foreign key to `BinaryID` in `IM_BINARY_VALUE`
`AttributeValue`	The ASCII value of the attribute.	VARCHAR2	2000	Primary key

4.6.2 IM_BINARY_VALUE

This table contains all LDAP attribute values that are binary data, such as graphics. Normally, an attribute value is stored in either `IM_ATTRIBUTE_VALUE` or `IM_ATTRIBUTE_BINARY_VALUE`, but never in both tables for a single attribute.

Attribute	Description	Type	Length	Notes
`BinaryID`	Unique ID of the binary value. This is referenced by `BinaryValueID` in `IM_ATTRIBUTE_VALUE`.	NUMBER		Primary key
`BinaryValue`	The binary value of the attribute.	LONG RAW		

4.6.3 IM_CLASS_ATTRIBUTE

This table contains the attributes associated with LDAP classes.

Attribute	Description	Type	Length	Notes
`LDAPClassID`	Unique ID of the class with which the attribute is associated.	NUMBER		Foreign key to `LDAPClassID` in `IM_CLASS_LDAP`
`AttrDescID`	Unique ID of the description of the attribute, which is defined in `IM_LDAP_ATTR_DESC`.	NUMBER		Foreign key to `AttrDescID` in `IM_LDAP_ATTR_DESC`
`AttrRequired`	Flag that specifies whether the option is required.	CHAR	1	Possible values: `R` (required), `O` (optional), `S` (suggested)

4.6.4 IM_CLASS_LDAP

This table contains the definition of LDAP classes. The information included in this table includes the unique ID and name of the class.

Attribute	Description	Type	Length	Notes
LDAPClassID	Unique ID of the class.	NUMBER		Primary key; Unique
ParentClassID	Unique ID of the class' parent class.	NUMBER		Foreign key to LDAPClassID in IM_CLASS_LDAP
SuppressFlag	Enacts policy to write (or not write) to LDAP logs to record operations made on this class. This policy decides if class data gets replicated to slapd.	CHAR	1	
ClassName	Name of the class.	VARCHAR2	256	
Description	Description of the class.	VARCHAR2	256	

4.6.5 IM_LDAP_ACI_RULES

This tables stores ACI rules pertaining to LDAP entries. These rules specify the bind DN operating upon a target DN. This table lists rules stating "Who" has permission to do "What". Both "Who" and "What" are LDAP entries, former being the bind entry, latter being the target entry. The bind entry (or "Who") represents the current user logged in (or bound) to the LDAP directory.

Attribute	Description	Type	Length	Notes
ACIID	ID for the ACI Record.	NUMBER		
AttrDescID		NUMBER		Foreign key to AttrDescID in IM_LDAP_ATTR_DESC
BindID	ID of the Bind entry, which defines who has access to which LDAP information.	NUMBER		
BindType	Indicates if the rule applies directly to the bind entry as the logged in user or a group of bind entries. If group, BindID is the group entry.	CHAR	1	
LDAPClassID	Unique ID of the class.	NUMBER		Foreign key to LDAPClassID in IM_CLASS_LDAP

Attribute	Description	Type	Length	Notes
`Permissions`	The permissions granted or denied to the bind entry(ies).	VARCHAR2	4	
`Realm`	Indicates if the rule applies directly to the target entry or to th target entry's children. This is a method of applying the ACI to a subtree.	CHAR	1	
`TargetID`	The entry id specifying the target(s) being operating upon.	NUMBER		

4.6.6 IM_LDAP_ANCESTORS

This table stores pointers for each LDAP entry back to their ancestral entries. This storage excludes the immediate parent, which is already stored in the IM_LDAP_OBJECT table.

Attribute	Description	Type	Length	Notes
`AncestorID`	Refers to an ancestoral entry of another entry.	NUMBER		Foreign key to `LDAPObjectID` in `IM_OBJECT_LDAP`
`DescendentID`	Refers to the entry, whose ancestors are being mapped.	NUMBER		Foreign key to `LDAPObjectID` in `IM_OBJECT_LDAP`

4.6.7 IM_LDAP_ATTR_DESC

This table contains data related to attribute descriptors. Included in this table are the name of the attribute, the type of data its values can contain (string, binary, or number), and a flag to specify if the attribute can have multiple values.

Note that an attribute type is defined independently of the object classes that may contain it. Therefore, an attribute is always the same underlying type, and is always either multi-valued or single-valued, across all object classes.

Attribute	Description	Type	Length	Notes
AttrDescID	Unique ID of the attribute descriptor.	NUMBER		Primary key; Unique
AttrName	Name of the attribute.	VARCHAR2	128	Unique
AttrType	Type of data contained in the value of the attribute.	CHAR	1	Possible values: B (binary), S (ASCII text string)
AttrMultival	Flag to specify that multiple values are allowed for the attribute.	CHAR	1	Possible values: M (multi-value), S (single value)
AttrHidden	Flag to specify that the attribute is hidden from the LDAP server component of LDAP.	CHAR	1	Possible values: Y (hidden), N (not hidden)

4.6.8 IM_LDAP_ATTRIBUTE_INDEX

This table stores all indexed attribute values from the IM_ATTRIBUTE_VALUE table. This table has a few indexes so that the retrieval of attribute values is optimized for certain attributes.

Attribute	Description	Type	Length	Notes
LDAPObjectID	Unique ID of the LDAP object.	NUMBER		Foreign key to LDAPObjectID in IM_OBJECT_LDAP
AttrDescID	Unique ID of the attribute descriptor.	NUMBER		Foreign key to AttrDescID in IM_LDAP_ATTR_DESC
AttributeValue	The ASCII value of the attribute.	VARCHAR2	2000	Foreign key to AttributeValue in IM_ATTIBUTE_VALUE
AttrValID	Forms a referential integrity constraint relationship between an attribute value and its indexed value (if indexed). Used to automatically purge the indexed value if the value is deleted and for consistency.	NUMBER		Foreign key to AttrValID in IM_ATTRIBUTE_VALUE

4.6.9 IM_LDAP_CONFIGURATION

This table contains information on LDAP configuration options.

Attribute	Description	Type	Length	Notes
CheckSchemaFlag	Indicates whether to enforce LDAP schema rules upon LDAP data.	CHAR	1	Possible values: Y (yes), N (no)
ReplicateDataFlag	Indicates whether to record LDAP operations in the LDAP log.	CHAR	1	

4.6.10 IM_LDAP_FILTER_PATTERN

This table stores the filter patterns available for performing LDAP searches (using the ldap_search_s function). Patterns express general filter format and ultimately map to a function conducting the actual query(-ies) that implement the filter.

Attribute	Description	Type	Length	Notes
FilterChgID		NUMBER		
FilterClass		CHAR	1	
Pattern		VARCHAR2	2000	
Query		VARCHAR2	2000	

4.6.11 IM_LDAP_GROUP_MEMBERS

This table is used for internal purposes only.

Attribute	Description	Type	Length	Notes
GroupObjID	(For internal use only.)	NUMBER		Foreign key to LDAPObjectID in IM_OBJECT_LDAP
MemberObjID	(For internal use only.)	NUMBER		Foreign key to LDAPObjectID in IM_OBJECT_LDAP

4.6.12 IM_LDAP_LAST_FILTER

This table records the last change to filter patterns. It indicates that filters need to be re-cached.

Attribute	Description	Type	Length	Notes
`LastFilterChg`		NUMBER		

4.6.13 IM_LDAP_OBJECT_CLASS

This table defines the class to which an LDAP object belongs. This information is used to validate the set of attributes that can be associated with an object.

Attribute	Description	Type	Length	Notes
`LDAPObjectID`	Unique ID of the LDAP object.	NUMBER		Foreign key to `LDAPObjectID` in `IM_OBJECT_LDAP`
`LDAPClassID`	Unique ID of the class with which the object is associated.	NUMBER		Foreign key to `LDAPObjectID` in `IM_OBJECT_LDAP`

4.6.14 IM_LDAP_SCHEMA_VERSION

This static table contains a single entry: the version number of the current LDAP database table schema.

Attribute	Description	Type	Length	Notes
`Version`	Version number of the LDAP schema	NUMBER	< 10^38	

4.6.15 IM_LOG_LDAP

This table stores all records by recording all creates/changes/deletions made to LDAP entries. It is used by a replication LDAP server to keep entries up-to-date.

Attribute	Description	Type	Length	Notes
ID	Log id	NUMBER	< 10^38	Primary key; Unique
TransactionID		NUMBER		
LogDate	Date/Time of log entry	DATE	7	Indexed
UserName	Name of Oracle user who made this change	VARCHAR2	30	
Operation	Name of the logged operation.	VARCHAR2	32	
NumFields	Number of fields updated in operation.	NUMBER	1	
LogContinue		CHAR	1	
s1	Short field 1	VARCHAR2	256	
s2	Short field 2	VARCHAR2	256	
s3	Short field 3	VARCHAR2	256	
s4	Short field 4	VARCHAR2	256	
s5	Short field 5	VARCHAR2	256	
s6	Short field 6	VARCHAR2	256	
s7	Short field 7	VARCHAR2	256	
s8	Short field 8	VARCHAR2	256	
s9	Short field 9	VARCHAR2	256	

4.6.16 IM_OBJECT_LDAP

This table defines individual LDAP objects, and includes the distinguished name (DN) of the object, as well as the unique ID of its parent.

Attribute	Description	Type	Length	Notes
LDAPObjectID	Unique ID of the LDAP object.	NUMBER		Primary key
LDAPDN	Distinguished name (DN) of the LDAP object.	VARCHAR2	2,000	
LDAPParentID	Unique ID of the parent object for this LDAP object.	NUMBER		Foreign key to LDAPObjectID in IM_OBJECT_LDAP

4.7 Administration Tables

The Integrated Services Directory database tables described in this section are used for administrative purposes. Some of these tables are used by multiple products (i.e. IM_LOG_TBL, used by all Inter* products). Other tables are used by a specific product, but are static (i.e. IM_STATUS_TYPES_TBL, used by InterMail).

4.7.1 IM_LOG_TBL

This table contains log entries for all database operations. This information is used by various product components to determine the changes that must performed to locally-cached database information. For example, the Directory Cache Server component of InterMail checks the IM_LOG_TBL for cached information.

The nine "short" fields in this table are used to store various attributes of the operation being logged. For example, when an account is created, the short fields for this operation's log entry contain attributes of the new account — `PopAddress`, `host`, `internalID`, `password`, etc.

Attribute	Description	Type	Length	Notes
`ID`	Log id	NUMBER	< 10^38	Primary key; Unique
`LogDate`	Date/Time of log entry	DATE	7	Indexed
`UserName`	Name of Oracle user who made this change	VARCHAR2	30	
`Operation`	Name of the logged operation (`CREATE_ACCOUNT`, `CREATE_ALIAS`, etc.)	VARCHAR2	32	
`NumFields`	Number of fields updated in operation	NUMBER	1	
`s1`	Short field 1	VARCHAR2	256	
`s2`	Short field 2	VARCHAR2	256	
`s3`	Short field 3	VARCHAR2	256	
`s4`	Short field 4	VARCHAR2	256	
`s5`	Short field 5	VARCHAR2	256	
`s6`	Short field 6	VARCHAR2	256	
`s7`	Short field 7	VARCHAR2	256	
`s8`	Short field 8	VARCHAR2	256	
`s9`	Short field 9	VARCHAR2	256	

4.7.2 IM_LAST_EXPIRED_TBL

Attribute	Description	Type	Length	Notes
ID		NUMBER		

4.7.3 IM_LAST_MODIFIED_TBL

This table contains the ID of the most recent entry in the IM_LOG_TBL table. There is only one entry in this table.

Attribute	Description	Type	Length	Notes
ID	Largest ID in the IM_LOG table (i.e., the ID of the most recent entry).	NUMBER	< 10^38	
LogType	The type of operation that has been logged.	CHAR	1	

4.7.4 IM_SCHEMA_VERSION_TBL

This static table contains a single entry: the version number of the InterMail database tables.

Attribute	Description	Type	Length	Notes
Version	Version number of the InterMail schema	NUMBER	< 10^38	Primary key

5

InterMail C API

This chapter describes the C API library available for creating programs that access data in the Integrated Services Directory and InterMail MSS databases, and includes the following information:

- An overview of the semantics of the API library.
- Descriptions of the data types used in the API.
- Listing of individual API functions.
- Description of compiling and linking applications with the API library.

5.1 Introduction

Data maintained by the Integrated Services Directory database, MSS database, and Configuration database can be accessed using an object-oriented C API. InterMail objects can be created, destroyed, read, and modified using this API. Using this library, it is possible to develop customized system administration commands, and to extend the features of InterMail while remaining compatible as InterMail is upgraded.

The InterMail Perl API described in Chapter 6 is based on this C library.

Note: *You must have one of the recommended C ++ compilers for your specific Unix platform to use the InterMail C API. If a suitable compiler is not available, consider using the InterMail Perl API instead*

5.2 Naming Conventions

Types, functions, and constants in the InterMail C API are prefixed with `IM_` to avoid conflicts, and to make these types easy to identify. Header filenames use only lower case, and are prefixed with `im_` instead. The library is named `libim4.so.4.`*x*.

Note: *Due to the large set of software libraries (both InterMail and third party) accessed by the API, programs that link with the API may encounter other naming conflicts that are impossible to avoid.*

5.3 Function Semantics

Most of the functions follow a consistent naming pattern:

- IM_InitX() initializes an object of type "X", and must be called before that object is used in other functions.
- IM_FreeX() de-allocates the resources that are in use by an object of type "X".
- IM_ReadX() reads information relevant to that type. For example, IM_ReadAccount() reads information about the account identified by the fields of an IM_Account object.
- IM_UpdateX() updates fields for an object of type "X". If a char* field of that object contains a NULL pointer, this particular field is not updated. Numeric fields can also be set to a value that implies no change. All other fields containing data, except those that uniquely identify the object, are incorporated into the update operation.
- IM_CreateX() creates an object of type "X" in the Integrated Services Directory or MSS database using the data from the object's fields.
- IM_DeleteX() deletes an object of type "X" from the Integrated Services Directory or MSS database.

5.4 Calling Conventions

With few exceptions, the InterMail API calls operate on objects. These objects are represented using standard C structures that can be located on the stack, in the heap, or in global memory, depending on the caller's preference and the requirements of the application.

All function return integer values, with zero indicating success, and non-zero indicating failure. More detailed information about errors is returned in an `IM_Error` struct. Any function that can return an error takes an `IM_Error*` as its last argument.

5.4.1 Library Initialization

```
int IM_InitApplication(char* appName, IM_Error*);
```

This function must be called at least once before any other function in the library (except `IM_InitX()` functions or `IM_InitLibrary()`). If it returns a non-zero value, then the library was not properly initialized and no further calls to the library should be attempted.

The `appName` argument will be used to control access to the InterMail configuration database, and to name the InterMail log file that may be created if errors occur in the application.

Calling this function more than once has no effect.

```
int IM_InitLibrary();
```

This is a simple wrapper around `IM_InitApplication()` that uses a generic `appName` value ("IM-C-API"). It is here for convenience and backwards compatibility.

5.4.2 Object Data Types

Every InterMail object's struct begins with a header that allows the API to do run-time type verification and to detect improperly initialized objects.

The caller of the API need not worry about the contents of these headers, they are initialized using a function of the form `IM_Init`*`Type`*`()`. Calling an API function with an object that has not been properly initialized will result in an error. When the caller is finished with an object, it is necessary to free the resources of that an object may have allocated using a function of the form `IM_Free`*`Type`*`()`. As a convenience, it is also possible to pass any properly initialized object to the function `IM_Free()`. Failure to pass objects to their `IM_Free`*`Type`*`()` function will result in memory leaks and poor performance. The `Init` and `Free` functions correspond roughly to constructors and destructors in C++.

All objects are passed by pointer, not by value, e.g. `IM_InitMbox(IM_Mbox*)`.

Some objects have a void* field called "internalID" following the header. Extra state related to the object is kept here to improve performance when accessing an object multiple times. This state is also deallocated, if necessary, by calling `IM_Free`*`Type`*`()`.

Except for the header and internalID, most fields are `int`'s and `char*`'s, and `enum`'s.

The following conventions must be observed by the caller when manipulating `char*`'s in these structs. The InterMail API guarantees that the same conventions are followed internally.

- All `char*` values must be allocated on the heap with malloc(), or a derivative like strdup().
- Any non-NULL char* value may be passed to free() or realloc() inside an API function.
- All non-NULL char* values will be passed to `free()` in the `IM_Free` function.

Callers are free to use `char*` values from an object, provided that they set that field to NULL before passing that object to another API function (particularly `IM_Free`).

Some functions require that one or more `char*` fields be set by the caller, and these conventions must be followed in such instances.

5.5 Overview of Types

The types of objects supported by the InterMail API are:

`IM_Error`	An error returned by an API function.
`IM_StringArray`	A collection of NULL-terminated strings.

The `IM_Error` and `IM_StringArray` types are general purpose utility types.

`IM_Domain`	An e-mail domain, e.g. “my.company.com”.
`IM_Account`	An account with associated attributes and a mailbox.

The `IM_Domain` and `IM_Account` types are associated with the InterMail Directory. They control account provisioning, e-mail address aliases, and the routing of e-mail to forwarding addresses and the InterMail Message Store Server (MSS).

`IM_Mbox`	A mailbox belonging to an account, containing folders.
`IM_Folder`	A folder in a mailbox, organizing e-mail messages hierarchically.
`IM_Msg`	An e-mail message from a folder.
`IM_MimeInfo`	Information about the MIME (RFC 1521) structure of a message.

The `IM_Mbox`, `IM_Folder`, `IM_Msg` and `IM_MimeInfo` types are associated with the InterMail MSS. These objects control the delivery, removal, and organization of e-mail messages.

`IM_Reply`	An auto-reply message for an account.

The `IM_Reply` type is associated with both the InterMail Directory and the InterMail MSS. This object controls the content and manner of automated replies to incoming mail for an account or a group of accounts.

`IM_Cos`	Class of service definition
`IM_CosAttrDef`	Attribute referred to by one or more IM_Cos objects
`IM_CosAttrDefArray`	Collection of `IM_CosAttrDef` objects

The `IM_Cos` types are associated with the InterMail Directory. These objects control the features of InterMail that are enabled for an account.

`IM_Config`	Structure to hold configuration data from the configuration database.

The `IM_Config` type stores information managed by the InterMail Configuration Server .

`IM_LogContext`	Context used when generating log messages
`IM_LogMsg`	A log file entry

5.6 Errors

Interface: im_error.h

```
typedef struct IM_Error
{
      IM_Header _header;
      int       number;
      char*     string;
} IM_Error;
```

When an error is encountered in an API function, the `IM_Error` object that was passed into the function is used to report the details of the error. The number field represents the type of error, and the string field contains details that may help in determining the source of the error. In some situations, and for all instances of some errors, the string field may be NULL.

Error numbers greater than `0xFFFF` are two-part error codes. The high-order part refers to a category of errors and the low-order part determines the specific error within the category. These error codes can be converted to a short mnemonic string using `IM_GetErrorMnemonic()`. Other error numbers are related to improper use of the library and other miscellaneous problems. These errors are documented below.

```
enum IM_ErrorNumber {
    IM_ERROR_INVALID_ARG       = -100,
    IM_ERROR_MALLOC,
    IM_ERROR_MISSING_ARG,
    IM_ERROR_MISSING_ATTR,
    IM_ERROR_DEAD_DATA,
    IM_ERROR_NOT_FOUND,
    IM_ERROR_ACCOUNT_COS,        /* invalid account COS */
    IM_ERROR_NUMBER_NOT_IN_USE = 0,
    IM_DIR_GENERIC_ERROR       = 2000,
    IM_DIR_INTERNAL_ERROR,       /* unknown internal error */
    IM_DIR_BAD_DOMAIN,
    IM_DIR_BAD_SMTP,
    IM_DIR_BAD_ALIAS,
    IM_DIR_BAD_POP,
    IM_DIR_BAD_PASSWORD,
    IM_DIR_BAD_ALIAS_DOMAIN,
    IM_DIR_DUP_POP,             /* duplicate pop address*/
    IM_DIR_DUP_DOMAIN,          /* duplicate domain */
    IM_DIR_DUP_SMTP_ALIAS,      /* duplicate smtp alias */
    IM_DIR_DUP_SMTP_OR_INT_ID,  /* duplicate pSmtp or Int Id */
    IM_DIR_CANT_DELETE_PSMTP,   /* can't delete primary smtp */
    IM_DIR_BAD_SCHEMA,          /* tables & package are
                                   incompatible */
    IM_DIR_NO_FORWARDS,         /* can't set forwarding if
                                   no forwards */
    IM_EXISTS,                  /* object to be created
                                   already exists */
    IM_NOT_ALLOWED              /* operation is not allowed */
};
```

The functions below manipulate the `IM_Error` structure.

```
void IM_InitError(IM_Error* error);
void IM_FreeError(IM_Error* error);
int  IM_SetError(IM_Error* error, int number, const char* string);
int  IM_GetErrorMnemonic(const IM_Error* error, char* buf,
     int bufsize);
int  IM_Errno2Mnemonic(int errnum, char* buf, int bufsize)
```

Before passing an `IM_Error` structure to an API function, it must be initialized with `IM_InitError()`.

After handling an error reported by an API function, the object resources of `IM_Error` should be released using `IM_FreeError()`.

The `IM_SetError()` function fills the IM_Error object's fields, copying the string into a newly allocated buffer. Most users will never need to call IM_SetError() directly.

The IM_GetErrorMnemonic() function fills in the buf argument with a short textual representation of the number in the IM_Error argument. For example, 0x320004 (3,276,804) is converted to AcctNoSuchUser. The bufsize argument indicates the size of the storage available in buf. A buffer size of 64 bytes is sufficient to hold any mnemonic string.

IM_Errno2Mnemonic() works the same way as IM_GetErrorMnemonic(), but it accepts the number from an IM_Error object.

5.7 String Arrays

Interface: im_stringarray.h

```
typedef struct IM_StringArray
{
    IM_Header _header;
    int       num;
    char**    strings;
} IM_StringArray;

void IM_InitStringArray(IM_StringArray*);
void IM_FreeStringArray(IM_StringArray*);
void IM_ZeroStringArray(IM_StringArray*);
int  IM_CopyStringArray(IM_StringArray* dst, const IM_StringArray*
src);
```


5.8 Domains

Interface: im_domain.h

```
typedef enum IM_DomainType
{
  IM_DOMAIN_ NO_CHANGE=-1,
  IM_DOMAIN_UNKNOWN=0,
  IM_DOMAIN_DELETED =  'D',
  IM_DOMAIN_LOCAL =    'L',
  IM_DOMAIN_NON_AUTH = 'N',
  IM_DOMAIN_REWRITE=   'R'
} IM_DomainType;
```

The information about domains is conveyed via the structure IM_Domain:

```
typedef struct IM_Domain
{
      IM_Header          _header;
      IM_DomainType type;
      char*              name;
      char*              relayHost;
      char*              rewriteName;
      char*              wildcardAccount;
} IM_Domain;
```

The fields of this structure and their syntax are described below:

`_header`	Internal use.
`type`	The type of the domain, see IM_DomainType.
`name`	The name of the domain. Can be any valid domain name up to 64 characters `long`.
`relayHost`	Host name to use for SMTP relay for non-authoritative domains. May be specified as <host>.<domain>, for a total length of (32 + 1 + 64 = 97). Used with domains of type IM_DOMAIN_NON_AUTH only.
`rewriteName`	Change the domain portion of the destination address of all mail received by 'name' domain to 'rewriteName' domain. Can be any valid domain name up to 64 characters long. Used with domains of type IM_DOMAIN_REWRITE only.
`wildcardAccount`	Account within the 'name' domain to use as the destination mailbox for unknown recipients. Can be any valid account name up to 64 characters long. Used with domains of type IM_DOMAIN_LOCAL only. You can only use this field for an update, it will fail if it has any value during creation.

The functions below manipulate the IM_Domain structure:

```
void IM_InitDomain(IM_Domain* domain);
void IM_FreeDomain(IM_Domain* domain);
int  IM_CopyDomain(IM_Domain* target, const IM_Domain* origin);
```

The functions below access and modify information in the InterMail Directory through the IM_Domain structure:

```
int  IM_ReadDomain  (IM_Domain* domain, IM_Error* error);
int  IM_CreateDomain(IM_Domain* domain, IM_Error* error);
int  IM_DeleteDomain(IM_Domain* domain, IM_Error* error);
int  IM_UpdateDomain(IM_Domain* domain, IM_Error* error);
int  IM_ReadDomains(char* from, char* till, int max,
         IM_StringArray* domains, IM_Error* error);
int  IM_ReadSubDomains(char* topDomain, char* from, char* till, int max,
         IM_StringArray* domains, IM_Error* error);
```

IM_ReadDomain

```
int  IM_ReadDomain  (IM_Domain* domain, IM_Error* error);
```

The ‘name’ field of the ‘domain’ argument must point to the name of an SMTP domain, e.g. “my.company.com”. The function returns an error if the specified domain does not exist in the Directory database.The “type” field is set according to the type of the domain, and other fields are set depending on the type.

IM_CreateDomain

```
int  IM_CreateDomain(IM_Domain* domain, IM_Error* error);
```

The ‘name’ field of the ‘domain’ argument must point to the name of an SMTP domain. The ‘type’ field of the ‘domain’ argument must be a valid type for domain creation, either IM_DOMAIN_LOCAL, IM_DOMAIN_REWRITE, or IM_DOMAIN_NON_AUTH. This function creates the specified domain in the Directory database.

Three of the parameters in IM_Domain are individually used with three different domain types. ‘relayHost’ must be specified with domain type IM_DOMAIN_NON_AUTH. ‘rewriteName’ must be specified with domain type IM_DOMAIN_REWRITE. The ‘relayHost’ and ‘rewriteName’ parameters cause an error if specified with an incorrect domain type. ‘wildcardAccount’ is used with domain type IM_DOMAIN_LOCAL, but will cause an error on domain creation because the domain must first be created before any accounts within the domain (including the wildcard account) can be created. After the domain is created and the account is created, the wildcard account can be specified using IM_UpdateDomain.

IM_DeleteDomain

```
int  IM_DeleteDomain(IM_Domain* domain, IM_Error* error);
```

The ‘name’ field of the ‘domain’ argument must point to the name of an SMTP domain. This function changes the domain type of the specified domain to IM_DOMAIN_DELETED in the Directory database.

IM_UpdateDomain

```
int  IM_UpdateDomain(IM_Domain* domain, IM_Error* error);
```

The ‘name’ field of the ‘domain’ argument must point to the name of an SMTP domain and the ‘type’ field must be IM_DOMAIN_DELETED, IM_DOMAIN_LOCAL, IM_DOMAIN_REWRITE, or IM_DOMAIN_NON_AUTH. This function changes the domain type, relayHost, rewriteName, and/or wildcardAccount of the specified domain as dictated by the values in the ‘domain’ argument. Possible uses of IM_UpdateDomain are to:

- delete a domain
- change or add a wildcard account to a local domain
- change a relay host for a non-authoritative domain
- change a rewrite domain
- change a domain type from non-authoritative to local

As with IM_CreateDomain, three of the parameters in IM_Domain are individually used with three different domain types. ‘relayHost’ must be specified if updating to domain type IM_DOMAIN_NON_AUTH. ‘rewriteName’ must be specified if updating to domain type IM_DOMAIN_REWRITE. ‘wildcardAccount’ may optionally be specified if updating to domain type IM_DOMAIN_LOCAL. The ‘relayHost’, ‘rewriteName’, and ‘wildcardAccount’ parameters are ignored if specified with incorrect domain types.

There are some limits on changing domain types using IM_UpdateDomain. Setting any domain to type ‘deleted’ is a valid operation, however consistency checking may show account or rewrite domain dependencies that force other changes before the delete operation will be successful. Setting a deleted domain to any other domain type is a valid operation, provided any required parameters are included and valid. Switching a domain from local to non-authoritative (or visa-versa) is also valid. However, switching a domain from local or non-authoritative to rewrite (or visa-versa) is not allowed at this time. Delete the domain first, then set the domain to the desired type.

Note: *When `IM_UpdateDomain` specifies a wildcard account, the user must use only the local portion of the SMTP address.*

IM_ReadDomains

```
int  IM_ReadDomains(char* from, char* till, int max,
     IM_StringArray*, IM_Error*);
```

Fetch the domain names defined in the InterMail database. The “from” and “till” arguments specify the lower and upper bounds of the range of names to return, where “from” is inclusive, “till” is exclusive. If either “from” or “till” is NULL, it defaults to the lexically-lowest or lexically-greatest domain name, respectively. If “max” is –1, then all matching domains are returned, otherwise no more than “max” names are returned.

It is possible to iterate through all domains in the Directory by repeatedly calling IM_ReadDomains() and passing NULL for the “till” argument, and the last domain name from the previous invocation as the “from” argument. Since the “from” argument is inclusive, it is necessary to skip the first domain in the results on the second and subsequent invocations.

On success, the results are stored in the IM_StringArray argument.

IM_ReadSubDomains

```
int  IM_ReadSubDomains(char* topDomain, char* from, char* till,
        int max, IM_StringArray*, IM_Error*);
```

Fetch the subset of the domain names defined in the InterMail database that are contained by "topDomain," e.g. if "topDomain" is "software.com", then this function will return domains like "east.software.com" and "sales.west.software.com", but not "hardware.com". The "from" and "till" arguments specify the lower and upper bounds of the range of names to return, where both "from" and "till" are exclusive (note the difference from IM_ReadDomains(), above). If either "from" or "till" is NULL, it defaults to the lexically-lowest or lexically-greatest domain name, respectively. If "topDomain" is NULL, all domains are matched. No more than "max" names are returned, and "max" must be greater than zero.

Starting with NULL "from" and "till" arguments, it is possible to iterate through all sub-domains of a specified domain by repeatedly calling IM_ReadSubDomains() and passing NULL for the "till" argument, and the last domain name from the previous invocation as the "from" argument. Since "from" is exclusive, there is no need to skip any values in the results.

On success, the results are stored in the IM_StringArray argument. If the same IM_StringArray object is used repeatedly, the strings it points to will be de-allocated and re-initialized each time IM_ReadSubDomains() is called.

5.9 Accounts

Interface: im_account.h

```
typedef enum IM_AcStatus
{
    IM_ACSTATUS_NO_CHANGE=-1,  /* Used to imply no change desired */
    IM_ACSTATUS_UNKNOWN=0,

    IM_ACSTATUS_ACTIVE,        /* Active */
    IM_ACSTATUS_SUSPENDED,     /* Suspended */
    IM_ACSTATUS_DELETED,       /* Deleted */
    IM_ACSTATUS_MAINTENANCE,   /* Maintenance mode */
    IM_ACSTATUS_LOCKED,        /* Locked mode */
    IM_ACSTATUS_PROXY          /* Used for account migration */
} IM_AcStatus;

typedef enum IM_PwHashType
{
    IM_PWHASH_NO_CHANGE=-1,    /* Used to imply no change desired */
    IM_HASH_UNKNOWN=0,
    IM_PWHASH_CLEAR,           /* No hash scheme */
    IM_PWHASH_MD5_PO,          /* MD5 hash scheme (proprietary
                                  extensions) */
    IM_PWHASH_UNIX             /* UNIX crypt() style hashing */
} IM_PwHashType;
```



```
typedef enum IM_LocalDelivery
{
    IM_DELIVERY_NO_CHANGE=-1,  /* Used to imply no change desired */
    IM_DELIVERY_UNKNOWN=0,
    IM_DELIVERY_ENABLED,       /* Delivery to account mailbox
                                  enabled */
    IM_DELIVERY_DISABLED       /* Delivery to account mailbox
                                  disabled */
} IM_LocalDelivery;

typedef enum IM_ForwardFlag
{
    IM_FORWARDING_NO_CHANGE=-1, /* Implies no change desired */
    IM_FORWARDING_UNKNOWN=0,
    IM_FORWARDING_ENABLED,      /* forwarding is enabled. */
    IM_FORWARDING_DISABLED      /* forwarding is disabled. */
} IM_ForwardFlag;

typedef enum IM_ReplyType
{
    IM_REPLY_NO_CHANGE=-1,      /* Implies no change desired */
    IM_REPLY_UNKNOWN=0,

    IM_REPLY_NONE= 'N',        /* No auto-reply mode is engaged.
    IM_REPLY_AUTO= 'R',           See InterMail documentation for */
    IM_REPLY_ECHO= 'E',           a detailed description of       */
    IM_REPLY_VACATION= 'V'        the reply modes.                */
} IM_ReplyType;

typedef struct IM_Account
{
    IM_Header        _header;
    void*            _internalID;
    char*            smtpAddress;
    char*            popAddress;
    char*            mssHost;
    char*            smtpHost;
    char*            popHost;
    char*            emailIntID;
    char*            password;
    char*            plainPassword;
    char*            cosName;
    IM_ServiceDef    acCos;
    IM_ServiceDef    resultCos;
    IM_PwHashType    hash;
    IM_LocalDelivery local;
    IM_ForwardFlag   forward;
    IM_ReplyType     reply;
    IM_AcStatus      status;
  } IM_Account;
```

The fields of this struct and their syntax are described below:

`_header`	Internal use.
`_internalID`	Internal use.
`smtpAddress`	SMTP address, two strings up to 64 characters each, separated by an "@" symbol. The first string is the "local" portion of the address. The second string must be a domain name known to the InterMail Directory.
`popAddress`	Login name for POP, IMAP, WebMail, and InterManager, up to 129 characters.
`mssHost`	Hostname of up to 32 characters. Must correspond to the "logical hostname" of a system running an InterMail MSS. This field must be NULL if status is IM_ACSTATUS_PROXY.
`smtpHost`	Hostname of up to 64 characters, used when an account is in proxy mode to forward SMTP requests to an alternate server. This field must be NULL if status is not IM_ACSTATUS_PROXY.
`popHost`	Hostname of up to 64 characters, used when an account is in proxy mode to forward POP requests to an alternate server. This field must be NULL if status is not IM_ACSTATUS_PROXY.
`emailIntID`	Number with up to 38 decimal digits represented as a string
`status`	Status of the account, see IM_AcStatus.
`local`	Local delivery option, see IM_LocalDelivery.
`cosName`	COS name to which the account subscribes.
`acCos`	Account specific COS preferences (may not correspond to the actual COS for the account, see `resultCos`).
`resultCos`	Resultant COS after applying overriding rules on the subscribed COS attributes and the account specific COS preferences
`hash`	Type of password hashing scheme used, see IM_PwHashType.
`password`	Encrypted password
`plainPassword`	Unencrypted password
`forward`	Whether forwarding is enabled for the account, see IM_ForwardFlag.
`reply`	Auto-reply mode for the account, see IM_ReplyType.

These functions manipulate the IM_Account structure:

```
void IM_InitAccount(IM_Account* account);
void IM_FreeAccount(IM_Account* account);
```

Note: *`IM_InitAccount()` must be called before attempting to use any function which uses `IM_Account`.*

The following functions use the information in `IM_Account` to access or change the information in the Directory:

```
int IM_CreateAccount(IM_Account*, IM_Error*);
int IM_UpdateAccount(IM_Account*, IM_Error*);
int IM_UpdateAccountAddr(IM_Account*, IM_Account*, IM_Error*);
int IM_DeleteAccount(IM_Account*, IM_Error*);
int IM_ReadAccount(IM_Account*, IM_Error*);

int IM_CreateAlias(IM_Account* smtp, const char* alias, IM_Error*);
int IM_DeleteAlias(const char* alias, IM_Error*);
int IM_ReadAccountAliases(IM_Account*, IM_StringArray*, IM_Error*);
int IM_ReadAlias(const char* alias, IM_Account*, IM_Error*);
int IM_ReadPOP3(const char* pop3, IM_Account*, IM_Error*);

int IM_CreateForward(IM_Account* from, const char* smtpTo,
    IM_Error*);
int IM_DeleteForward(IM_Account* from, const char* smtpTo,
    IM_Error*);
int IM_ReadAccountForwards(IM_Account*, IM_StringArray*, IM_Error*);
int IM_EnableAccountForwards(IM_Account* account, IM_Error* error);
int IM_DisableAccountForwards(IM_Account* account, IM_Error* error);

int IM_HashPassword(char* password, char* hashPassword, int len,
    IM_PwHashType);
int IM_CheckPassword(char* password, char* hashPassword,
    IM_PwHashType);

int IM_ResetAcCos(IM_Account*, IM_Error*);
int IM_UpdateAcCos(IM_Account*, char* name, char* value, IM_Error*);
int IM_DeleteAcCos(IM_Account*, char* name, IM_Error*);

int IM_ReadAccounts(char* domain, char* from, char* till, int max,
    IM_StringArray* addresses, IM_Error*);
```

IM_CreateAccount

```
int IM_CreateAccount(IM_Account*, IM_Error*);
```

This function creates an account in the Directory database with the information in the IM_Account argument. The following fields are used:

`smtpAddress`	required
`mssHost`	required (if account status is not IM_ACSTATUS_PROXY)
`smtpHost`	required (if account status is IM_ACSTATUS_PROXY)
`popHost`	required (if account status is IM_ACSTATUS_PROXY)
`emailIntID`	required
`status`	optional – defaults to IM_ACSTATUS_ACTIVE
`popAddress`	optional – defaults to the local portion of smtpAddress.
`local`	optional – local delivery is enabled by default.
`cosName`	optional – defaults to the "default" COS, *not* recommended.
`acCos`	optional – no account-specific COS attributes are set by default. See the helper functions IM_ResetAcCos(), IM_UpdateAcCos(), and IM_DeleteAcCos() below.
`hash`	optional – defaults to IM_PWHASH_CLEAR.
`plainPassword` `password`	optional – if `plainPassword` is set it is assumed to be the cleartext version of the `password`. The `password` will be encrypted according to the hash field when it is stored in the Directory. Otherwise, if password is set it is assumed to be the encrypted version of the password as determined by hash. If neither password field is specified, the account will have no password and the account's user will be unable to access any service requiring authentication, e.g. POP, IMAP, WebMail, or InterManager. The password may be specified later using IM_UpdateAccount().

Note on COS values

The C-API supports a logical overlap between some fields in the IM_Account structure and corresponding key-value pairs in an account's COS settings. After a successful call to IM_ReadAccount(), the local-delivery (`account->local`), forwarding (`account->forward`), and reply-mode (account->reply) values can also be read from `resultCos` field using the COS attribute names: "pref_localdelivery", "pref_forwarding", and "pref_replymode". As with other COS attributes, all values are stored as strings. "pref_localdelivery" and "pref_forwarding" are booleans, so their values will be either `"␣"` or `"␣"`. "pref_replymode" will use one of the characters from IM_ReplyType.

For IM_CreateAccount(), “pref_localdelivery” can be specified in the `acCos` field for a new account. For IM_UpdateAccount(), both “pref_localdelivery” and “pref_forwarding” can be updated using attributes in the `acCos` field. In both functions, attributes specified in `acCos` take precedence over the corresponding fields in the IM_Account structure.

The “pref_replymode” setting cannot be altered using the `acCos` field since this feature generally requires more information than a simple mode value. Use the functions IM_CreateReply(), IM_UpdateReply(),and IM_DeleteReply()instead.

Note: *The `IM_CreateAccount` function returns an error when an account is created with a `PWHASH` value of `IM_PWHASH_NO_CHANGE`. The `IM_PWHASH_NO_CHANGE` constant is only valid when used with `IM_UpdateAccount()`.*

IM_UpdateAccount

```
int IM_UpdateAccount(IM_Account*, IM_Error*);
```

This function updates account information in the database. The account argument must have the smtpAddress field defined, as this determines which account will be updated. Other fields can be set in order to change their values for the account in the Directory. To leave a field unchanged, set the field to NULL for string (char*) values, or to an appropriate NO_CHANGE enumeration value. Often it is best to start with a freshly initialized IM_Account structure when performing updates so that no fields are inadvertently set.

If the password or plainPassword field is set, they each obey the same semantics as in IM_CreateAccount().

The resultCos field cannot be updated.

To add or modify per-account COS attributes, set the attribute names and their desired values in the acCos field. To delete a per-account attribute, set the name of the attribute in acCos, but leave the value NULL. See the helper functions IM_ResetAcCos(), IM_UpdateAcCos(), and IM_DeleteAcCos() below.

IM_HashPassword

```
int IM_HashPassword(char* password, char* hashPassword, int len,
    IM_PwHashType);
```

Take the contents of the `password` and encrypt it according to the specified IM_PwHashType. Store the results in hashPassword, which is assumed to be large enough to hold at least `len` bytes. If hashPassword is not large enough to hold the result, an error is returned. A hashPassword size of 64 should be sufficient for most purposes.

Note: *It is not necessary to use this function with IM_CreateAccount(). If the `hash` field of an IM_Account is set, the plainPassword will be encrypted appropriately while creating the account.*

In addition, due to the way passwords are hashed, the same clear text password submitted twice may not result in the same hashed password output.

IM_CheckPassword

```
int IM_CheckPassword(char* password, char* hashPassword,
       IM_PwHashType);
```

Validates `password` using hashPassword and IM_PwHashType fetched from the InterMail directory via IM_ReadAccount(). The function returns `IM_SUCCESS` if the password matches, otherwise `IM_FAILURE`.

Note: *`IM_ReadAccount()` provides a more complete authentication capability when the caller sets the `plainPassword` field in the `IM_Account` argument.*

IM_ResetAcCos

int IM_ResetAcCos(IM_Account*, IM_Error*);

Clear all the values in the acCos field of an IM_Account. All associated string storage is freed.

This function has no effect on the account in the InterMail directory, only the information stored locally in the IM_Account structure is modified.

IM_UpdateAcCos

int IM_UpdateAcCos(IM_Account*, char* name, char* value, IM_Error*);

Insert or change the value of an attribute in the acCos field of an IM_Account. String storage associated with the previous value, if any, is freed. The `name` argument should not use any upper-case characters.

This function has no effect on the account in the InterMail directory, only the information stored locally in the IM_Account structure is modified.

IM_DeleteAcCos

int IM_DeleteAcCos(IM_Account*, char* name, IM_Error*);

Remove a single value from the acCos field of an IM_Account. The `name` argument should not use any upper-case characters.

This function has no effect on the account in the InterMail directory, only the information stored locally in the IM_Account structure is modified.

IM_UpdateAccountAddr

```
int IM_UpdateAccountAddr(IM_Account* old, IM_Account* new,
    IM_Error*);
```

This function updates the smtpAddress field of an account, which is not possible with IM_UpdateAccount(). The smtpAddress field of the first IM_Account argument (`old`) specifies the account whose address is to be changed. The second IM_Account argument (`new`) specifies the desired smtpAddress.

IM_DeleteAccount

```
int IM_DeleteAccount(IM_Account*, IM_Error*);
```

This function deletes the account from the database. The account argument must contain the smtpAddress field to uniquely identify the account to be deleted.

IM_ReadAccount

```
int IM_ReadAccount(IM_Account*, IM_Error*);
```

This function fills in the account argument with information from the database. The account argument must specify one of the following fields to identify the account to be read: smtpAddress, popAddress, or emailIntId. If more than one of these fields is set, the API uses the first one it finds in the ordering listed here.

If the plainPassword field is set, then the function call is treated as a request for authentication. Under these circumstances, IM_ReadAccount() will return an IM_DIR_BAD_PASSWORD error if the password does not match the (possibly encrypted) copy in the account, or IM_DIR_GENERIC_ERROR error if the account is in a state that does not allow user-level access.

IM_ReadAlias

```
int IM_ReadAlias(const char* alias, IM_Account*, IM_Error*);
```

This function looks up an account using an e-mail alias. If an account is found, the account object is filled in just like IM_ReadAccount().

IM_ReadPOP3

```
int IM_ReadPOP3(const char* pop3, IM_Account*, IM_Error*);
```

This function looks up an account using a POP login name. If an account is found, the account object is filled in just like IM_ReadAccount().

IM_CreateAlias

```
int IM_CreateAlias(IM_Account* smtp, const char* alias, IM_Error*);
```

This function creates an alias for the account identified by the ‘smtp’ argument. This argument must contain the smtpAddress field to uniquely identify the account. The ‘alias’ argument must contain a fully qualified smtpAddress to be used as the alias to this account.

IM_DeleteAlias

```
int IM_DeleteAlias(const char* alias, IM_Error*);
```

This function deletes an alias identified by the ‘alias’ argument.

IM_ReadAccountAliases

```
int IM_ReadAccountAliases(IM_Account*, IM_StringArray*, IM_Error*);
```

This function reads all the aliases corresponding to an account in the database identified by the account argument. The account argument must contain the smtpAddress field to uniquely identify the account whose forwards have to be read. The forwards are returned in the IM_StringArray argument.

IM_CreateForward

```
int IM_CreateForward(IM_Account* from, const char* smtpTo,
    IM_Error*);
```

This function creates a forwarding address for mail destined for an account. The ‘from’ argument must contain the smtpAddress field filled in to uniquely identify the account. The ‘smtpTo’ field identifies a forwarding address which is the destination.

IM_DeleteForward

```
int IM_DeleteForward(IM_Account* from, const char* smtpTo,
    IM_Error*);
```

This function deletes a forwarding association between the account identified by the ‘from’ argument and the forward address identified by the ‘smtpTo’ argument. The ‘from’ account type must contain the smtpAddress field to uniquely identify the account.

IM_ReadAccountForwards

```
int IM_ReadAccountForwards(IM_Account*, IM_StringArray*, IM_Error*);
```

This function reads all the forwarded addresses corresponding to an account. The account argument must have the smtpAddress filled in to uniquely identify the account. The IM_StringArray argument contains the forwarded addresses upon return.

IM_EnableAccountForwards

```
int IM_EnableAccountForwards(IM_Account*, IM_Error* error);
```

This function enables forwarding for an account identified by the IM_Account argument. The smtpAddress of this argument must be filled in to uniquely identify the account.

IM_DisableAccountForwards

```
int IM_DisableAccountForwards(IM_Account*, IM_Error*);
```

This function disables forwarding for an account identified by the IM_Account argument. The smtpAddress of this argument must be filled in to uniquely identify the account.

IM_ReadAccounts

```
int IM_ReadAccounts(char* domain, char* from, char* till, int max,
    IM_StringArray* addresses, IM_Error*);
```

Fetch the subset of SMTP addresses defined in the InterMail Directory that are in “domain”. For example, if the following accounts exist: “jack@ software.com”, “jill@software.com”, “jill@west.software.com”, and “jane@hardware.com”, then this function will return “jack” and “jill” if the “domain” argument is “software.com”.

The “from” and “till” arguments specify the lower and upper bounds of the range of addresses to return, where “from” and “till” are both exclusive (like IM_ReadSubDomains(), note the difference from IM_ReadDomains(), above). If either “from” or “till” is NULL, it defaults to the lexically-lowest or lexically-greatest address, respectively. The “domain” argument must be specified. No more than “max” names are returned, and “max” must be greater than zero.

Starting with NULL values for the “from” and “till” arguments, it is possible to iterate through all addresses in a domain by repeatedly calling IM_ReadAccounts() and passing NULL for the “till” argument, and the last address from the previous invocation as the “from” argument. Since “from” is exclusive, there is no need to skip any values in the results.

On success, the results are stored in the IM_StringArray argument. If the same IM_StringArray object is used repeatedly, the strings it points to will be de-allocated and re-initialized each time IM_ReadAccounts() is called.

5.10 Mailboxes

Interface: im_mbox.h

```
typedef struct IM_Mbox
{
    IM_Header _header;
    void*     _internalID;
    char*     host;
    char*     name; /* String decimal digits, same as emailIntID */
    int       msgsStored;
    int       bytesStored;
    char*     inbox;
} IM_Mbox;

void IM_InitMbox  (IM_Mbox*);
int  IM_FreeMbox  (IM_Mbox*);

int  IM_ReadMbox  (IM_Mbox*, IM_Error*);
int  IM_CreateMbox(IM_Mbox*, const char* welcomeMsgId, IM_Error*);
int  IM_DeleteMbox(IM_Mbox*, IM_Error*);
int IM_SetMSSConnPoolSize(int size, IM_Error *);
```

IM_ReadMbox

```
int  IM_ReadMbox(IM_Mbox*, IM_Error*);
```

Fetch information about an existing mailbox.

The caller must set the host and name fields of the IM_Mbox argument.

IM_CreateMbox

```
int  IM_CreateMbox(IM_Account*, const char* welcomeMsgId, IM_Mbox*,
     IM_Error*);
```

Create a new mailbox using fields from the IM_Account argument and the welcome message-ID. The IM_Account's mssHost and emailIntID fields must have non-NULL values, and the IM_Account quota fields must be set appropriately. Normally, this should be done by calling IM_ReadAccount(), although in some special circumstances it is reasonable to assign these fields directly.

The welcomeMsgId parameter specifies the Message ID of a message stored in the database that the user would like to be automatically inserted into the newly created mailbox (i.e. this message usually introduces the user to the system and usage policies). If the welcome message-ID is NULL or "none", no welcome message will be inserted into the mailbox.

On success, the IM_Mbox argument will be initialized with the newly created mailbox information.

IM_DeleteMbox

```
int  IM_DeleteMbox(IM_Mbox*, IM_Error*);
```

Delete a mailbox.

The caller must set the host and name fields of the IM_Mbox argument.

IM_SetMSSConnPoolSize

```
int IM_SetMSSConnPoolSize(int size, IM_Error *);
```

This function sets the size of the MSS connection pool. The default size is 0, meaning no caching is done. If the connection pool size is set to "n," then the last "n" connections made to an MSS will be cached.

When using an IM_Mbox, if the C-API needs to make a connection to an MSS, it will check its connection pool first to see if there is an existing connection available. If there is, it will use it, otherwise it will make a new one.

When an IM_Mbox is destroyed, the MSS connection associated with it (if any) is placed in the pool, unless that would cause the size of the pool to exceed the maximum specified by this function, in which case the connection is destroyed.

5.11 Folders

Interface: im_folder.h

```
typedef enum IM_MsgFlagsIndex
{
    IM_MSGFLAG_RECENT=0,
    IM_MSGFLAG_SEEN,
    IM_MSGFLAG_ANSWERED,
    IM_MSGFLAG_FLAGGED,
    IM_MSGFLAG_DRAFT,
    IM_MSGFLAG_DELETED
} IM_MsgFlagsIndex;
```



```
typedef struct IM_Folder
{
    IM_Header  _header;
    void*      _internalID;
    char*      pathname;
    int        numFolders;
    int        readOnly;
    int        clearRecent;
    char**     names;
    int        numMsgs;
    int        UIDValidity;
    int*       msgRefs;
    int*       msgUIDs;
    int*       msgSizes;
    char**     msgIds;
    char**     msgFlags;
} IM_Folder;
```

The fields of this struct and their syntax are described below:

`_header`	Internal use.
`_internalID`	Internal use.
`pathname`	Location of the folder within the mailbox.
`numFolders`	Number of sub-folders contained by this folder.
`names`	Names of the sub-folders contained by this folder.
`numMsgs`	Number of messages in this folder, also the size of the arrays pointed to by msgRefs, msgUIDs, msgSizes, msgIds, and msgFlags.
`msgRefs`	Internal index for each message in this folder, required to use IM_Message.
`msgUIDs`	IMAP UID's for each message in this folder. All the UID's will be zero if connected to an older InterMail MSS without IMAP support.
`msgSizes`	Size in bytes of each message in this folder.
`msgIds`	POP UIDL strings for each message in this folder.
`msgFlags`	IMAP message flags for the messages in this folder. Each flag string currently contains six characters, each a 'T' or an 'F'. Each location in the array is identified by a constant in enum IM_MsgFlagsIndex.
`clearRecent`	Specifies whether the "recent" flags should be cleared when doing an IM_ReadFolder (default: FALSE)
`UIDValidity`	Read-only field returned by IM_ReadFolder; this is the UIDVALIDITY value from IMAP.
`readOnly`	Read-only field returned by IM_ReadFolder. Set to FALSE if a POP client has the folder open, in which case the caller should not make changes to the folder. This is also present in the Folder object of the Perl API.

```
void IM_InitFolder  (IM_Folder*);
int  IM_FreeFolder  (IM_Folder*);

int  IM_ReadFolder  (IM_Mbox*, IM_Folder*, IM_Error*);
int  IM_CreateFolder(IM_Mbox*, IM_Folder*, IM_Error*);
int  IM_DeleteFolder(IM_Mbox*, IM_Folder*, IM_Error*);
int  IM_RenameFolder(IM_Mbox*, IM_Folder*, const char* dstname,
     IM_Error*);
int  IM_DeleteFolderMsgs(IM_Mbox*, IM_Folder*, int count,
     int* msgArray, IM_Error*);
int  IM_ScanFolderMsgs(IM_Mbox*, IM_Folder*, int count,
     IM_Msg** msgArray, IM_Error*);
int  IM_MoveMsgs(IM_Mbox*, IM_Folder* src, IM_Folder* dst,
     int count, int* msgRef, IM_Error*);
int  IM_CopyMsgs(IM_Mbox*, IM_Folder* src, IM_Folder* dst,
     int count, int* msgRef, IM_Error*);
```

To read, create, rename, or delete a folder, the caller must initialize an IM_Mbox object as described above, and also set the `IM_Folder::pathname` field.

IM_ReadFolder() will fill in all remaining fields except clearRecent. The information in these fields is necessary in order to navigate to sub-folders or to access messages in the folder. In particular, values from the msgRefs array must be used to initialize an IM_Message object.

IM_ReadFolder

```
int  IM_ReadFolder(IM_Mbox*, IM_Folder*, IM_Error*);
```

Fetch information about a folder.

The caller must set the host and name fields of the IM_Mbox argument, and the pathname field of the IM_Folder argument.

IM_CreateFolder

int IM_CreateFolder(IM_Mbox*, IM_Folder*, IM_Error*);

Create a new folder in a mailbox.

The caller must set the host and name fields of the IM_Mbox argument, and the pathname field of the IM_Folder argument.

IM_DeleteFolder

```
int  IM_DeleteFolder(IM_Mbox*, IM_Folder*, IM_Error*);
```

Delete a folder from a mailbox.

The caller must set the host and name fields of the IM_Mbox argument, and the pathname field of the IM_Folder argument.

IM_RenameFolder

int IM_RenameFolder(IM_Mbox*, IM_Folder*, const char* dstname,
IM_Error*);

Renames the specified folder.

IM_DeleteFolderMsgs

int IM_DeleteFolderMsgs(IM_Mbox*, IM_Folder*, int count,
int* msgArray, IM_Error*);

Deletes ones or more messages from a folder. The caller must set the host and name fields on the "IM_Mbox" argument, and the pathname field of the "IM_Folder" argument. "msgArray" is an array of integers; the length of this array is given by the `count` argument. The integers specify a list of message indices to delete (0 being the first message in the folder). Normally IM_ReadFolder would be called first so the caller can select which messages to delete. Using this function is more efficient than deleting the messages one by one.

Note: *This function was formerly called "IM_DeleteFolderMessages", and this name is still supported for backwards compatibility.*

IM_ScanFolderMsgs

int IM_ScanFolderMsgs(IM_Mbox*, IM_Folder*, int count,
IM_Msg** msgArray, IM_Error*);

Scans the headers of one or more messages in a folder, so that future calls to IM_ReadMsgHeader() will be faster. The call must set the host and name fields of the IM_Mbox argument, and the pathname field of the "IM_Folder" argument. "msgArray" is an array of pointers to "IM_Msg" objects; the length of this array is given by the count argument. Normally IM_ReadFolder() would be called first so the caller can select which messages to scan. This function is useful when the caller needs to look at the headers of a large number of messages; it's more efficient to load the headers with this one call than to have them loaded one by one.

Note: *This function was formerly called "IM_ScanFolderMessages", and this name is still supported for backwards compatibility.*

IM_MoveMsgs

```
int  IM_MoveMsgs(IM_Mbox*, IM_Folder* src, IM_Folder* dst,
     int count, int* msgRef, IM_Error*);
```

Move "count" messages specified by the "msgRef" array from the "src" folder to the "dst" folder.

IM_CopyMsgs

```
int  IM_CopyMsgs(IM_Mbox*, IM_Folder* src, IM_Folder* dst,
     int count, int* msgRef, IM_Error*);
```

Copy "count" messages specified by the "msgRef" array from the "src" folder to the "dst" folder.

5.12 Messages

Interface: im_msg.h

```
typedef struct IM_Msg
{
    IM_Header  _header;
    void*      _internalID;
    int        msgRef;
    int        seen;
    int        size;
    char*      msgId;
    time_t     arrivalTime;
    char*      arrivalTimeString;
    char*      bodyBuffer;
} IM_Msg;

/* Caller must set IM_Msg::msgRef. */
void IM_InitMsg  (IM_Msg*);
int  IM_FreeMsg  (IM_Msg*);

int  IM_ReadMsg  (IM_Mbox*, IM_Folder*, IM_Msg*, IM_Error*);
int  IM_CreateMsg(IM_Mbox*, IM_Folder*, const char* from,
     cons char* text, IM_Msg*, IM_Error*);
int  IM_DeleteMsg(IM_Mbox*, IM_Folder*, IM_Msg*, IM_Error*);
int  IM_ReadMsgHeader(IM_Mbox*, IM_Folder*, IM_Msg*,
     const char* hdrLabel, char** hdrValue, IM_Error*);
int  IM_ReadMsgBody(IM_Mbox*, IM_Folder*, IM_Msg*,
int  offset, int size, char** text, IM_Error*);
int  IM_UpdateMsgFlags(IM_Mbox*, IM_Folder*, IM_Msg*,
     const char* flags, IM_Error*);
```

IM_ReadMsg

```
int  IM_ReadMsg(IM_Mbox*, IM_Folder*, IM_Msg*, IM_Error*);
```

Fetch information about a message.

This function will fill in all remaining fields in the IM_Msg object, but that does not include any of the actual message contents. To search for and retrieve specific headers from the message, call IM_ReadMsgHeader(). To retrieve all or part of the message text, call IM_ReadMsgBody().

The caller must set the host and name fields of the IM_Mbox argument, and the pathname field of the IM_Folder argument, and the msgRef field of the IM_Msg argument.

IM_CreateMsg

```
int  IM_CreateMsg(IM_Mbox*, IM_Folder*, const char* from,
     const char* text, IM_Msg*, IM_Error*);
```

Create a new message in a folder.

The caller must set the host and name fields of the IM_Mbox argument, and the pathname field of the IM_Folder argument. The from argument is used in the message "envelope". The text argument should point to the full text of an RFC822-formatted message, including headers. No syntax checking is performed on this text. On success, the msgRef field of the IM_Msg argument will be set.

IM_DeleteMsg

int IM_DeleteMsg(IM_Mbox*, IM_Folder*, IM_Msg*, IM_Error*);

Delete a message from a folder.

The caller must set the host and name fields of the IM_Mbox argument, and the pathname field of the IM_Folder argument, and the msgRef field of the IM_Msg argument.

IM_ReadMsgHeader

```
int  IM_ReadMsgHeader(IM_Mbox*, IM_Folder*, IM_Msg*,
     const char* hdrLabel, char** hdrValue, IM_Error*);
```

Fetch the value of a header from a message.

The caller must set the host and name fields of the IM_Mbox argument, and the pathname field of the IM_Folder argument, and the msgRef field of the IM_Msg argument. The hdrLabel argument should point to the name of a header field, e.g. "From:". On success, *hdrValue will point to a newly malloc'ed buffer containing the value of the header as a null-terminated string.

IM_ReadMsgBody

```
int  IM_ReadMsgBody(IM_Mbox*, IM_Folder*, IM_Msg*,
     int offset, int size, char** text, IM_Error*);
```

Fetch all, or part of, the textual contents of a message.

The caller must set the host and name fields of the IM_Mbox argument, and the pathname field of the IM_Folder argument, and the msgRef field of the IM_Msg argument. The offset and size arguments determine which portion of the message to fetch. Callers can determine the size of a message before they retrieve it by examining the msgSizes array from the containing folder.

On success, *text will point to an array containing the requested section of the message. If the bodyBuffer field of the IM_Mbox argument is not NULL, it is assumed to point to a char array of at least size + 1 bytes, and IM_ReadMsgBody will store the requested text in this array as a null-terminated string and set *text to bodyBuffer. Otherwise, *text will point to a newly malloc'ed buffer containing the requested text as a null-terminated string.

IM_UpdateMsgFlags

```
int  IM_UpdateMsgFlags(IM_Mbox*, IM_Folder*, IM_Msg*,
      const char* flags IM_Error*);
```

Updates the message flags of a message. The caller must set the host and name fields of the IM_Mbox argument, the pathname field of the IM_Folder argument, and the msgRef field of the IM_Msg argument. The flags argument is a pointer to a string of characters which specify which flags to change. Each character position represents a flag; a 'T' character sets the flag, an 'F' character clears it, and a '-' character leaves it alone. This string must be no longer than 6 characters; it may be shorter if only the first few flags need to be changed. For example, a flags string of “-TF” will cause. The seen flag (position 1) to be set, and the answered flag (position 2) to be cleared. The recent flag (position 0) will be left alone, as will all other flags.

5.13 Reply

Interface: im_reply.h

```
typedef struct IM_Reply
{
    IM_Header    _header;
    IM_ReplyType mode;  /* Types defined in im_account.h */
    char*        host;  /* the host name for the information */
    char*        text;  /* text contents */
    int          type;  /* type of contents */
} IM_Reply;
```

The fields of this struct and their syntax are described below:

`_header`	Internal use
`mode`	If type is 1, the auto-reply mode to use for the account, see im_account.h. If type is *not* 1, mode should be set to IM_REPLY_UNKNOWN.
`host`	Logical hostname of the MSS where text and auto-reply state will be stored.
`text`	If type is 1, the string to use for auto-reply message. Otherwise, the text of the account’s property identified by type.
`type`	type is initialized to 1 and need not be changed for normal auto-reply operations. Other values for type can be used to define arbitrary textual properties on an account. Some non-negative values for type are used by other parts of InterMail. User applications should use *negative* values for type to avoid any conflicts.

```
void IM_InitReply(IM_Reply*);
void IM_FreeReply(IM_Reply*);

int  IM_CreateReply(IM_Account*, IM_Reply*, IM_Error*);
int  IM_ReadReply  (IM_Account*, IM_Reply*, IM_Error*);
int  IM_UpdateReply(IM_Account*, IM_Reply*, IM_Error*);
int  IM_DeleteReply(IM_Account*, IM_Error*);
```


IM_CreateReply

```
int  IM_CreateReply(IM_Account*, IM_Reply*, IM_Error*);
```

If type is 1, this function defines the auto-reply text and initializes the auto-reply state for an account. The account argument must contain the smtpAddress which uniquely identifies the account. The reply argument must contain all the fields filled in. Note that the host field need not match the mssHost of the account, although this is a common arrangement.

If type is not 1, this function creates or updates the value of the account property specified by the type field.

IM_ReadReply

```
int  IM_ReadReply(IM_Account*, IM_Reply*, IM_Error*);
```

If type is 1, this function fetches the mode and text of the auto-reply message from the MSS corresponding to the account argument. The account argument must have the smtpAddress filled in.

If type is not 1, this function fetches the text of the account property associated with type.

IM_UpdateReply

```
int  IM_UpdateReply(IM_Account*, IM_Reply*, IM_Error*);
```

If type is 1, this function updates an account's auto-reply mode, text, and host. The account argument must have the smtpAddress filled in. If the host or text field of the reply argument is NULL, these are not updated. If the mode field is IM_REPLY_NO_CHANGE, the mode is not updated.

If type is not 1, this function updates the text of the account property associated with type.

IM_DeleteReply

```
int  IM_DeleteReply(IM_Account*, IM_Error*);
```

If type is 1, this function clears the auto-reply state for an account. The account argument must have the smtpAddress filled in.

If type is not 1, this function clears the text of the account property associated with type.

5.14 Class of Service

```
 typedef struct IM_ServiceAttr
{
char*                   name;
char*                   value;
} IM_ServiceAttr;
```


Note: *All COS attribute names are converted to lower-case in the Integrated Servicesl Directory, so it is prudent to avoid using upper-case characters in the name field. Any of the following variations may be used to set the POP server access for an account, "pref_POP", "pref_Pop", or "pref_pop", but when the COS information is retrieved from the Directory, it will revert to the lower-case only version, i.e. "pref_pop".*

```
typedef struct IM_ServiceDef       /*Located in im_defs.h */
{
int                           num;
IM_ServiceAttr*   attrs;
} IM_ServiceDef;

typedef struct IM_Cos
{
    IM_Header           _header;
    char*               name;
    IM_ServiceDef       serviceDef;
} IM_Cos;

typedef struct IM_CosAttrDef
{
    IM_Header           _header;
    int                 id;
    char*               name;
    IM_AttributeRule    rule;
    IM_AttributeSyntax  syntax;
} IM_CosAttrDef;
```

id is assigned by the database.

name is the Attribute name.

Note: *The same considerations from `IM_ServiceAttr` about upper-case characters in COS attribute names apply to the `name` field in `IM_CosAttrDef`*

rule can take one of the following values:

`IM_ATTR_RULE_COS_OVERRIDES`	Cos Overrides
`IM_ATTR_RULE_ACCOUNT_OVERRIDES`	Account Overrides
`IM_ATTR_RULE_LESSER`	Choose Lesser
`IM_ATTR_RULE_GREATER`	Choose Greater

syntax can take one of the following values:

`IM_ATTR_SYNTAX_STRING`	Null-Terminated String
`IM_ATTR_SYNTAX_NUMERIC`	Integer
`IM_ATTR_SYNTAX_BOOLEAN`	Boolean (True/False)

Valid values for a Boolean attribute are “1” (for true) and “0” (for false).

```
typedef struct IM_CosAttrDefArray
{
    IM_Header            _header;
    int                  num;
    IM_CosAttrDef*       attrDefs;
} IM_CosAttrDefArray;

void IM_InitCos(IM_Cos*);
int  IM_FreeCos(IM_Cos*);
void IM_InitCosAttrDef(IM_CosAttrDef*);
int  IM_FreeCosAttrDef(IM_CosAttrDef*);
void IM_InitCosAttrDefArray(IM_CosAttrDefArray*);
int  IM_FreeCosAttrDefArray(IM_CosAttrDefArray*);

int  IM_CreateCos(IM_Cos*, IM_Error*);
int  IM_DeleteCos(IM_Cos*, IM_Error*);
int  IM_ReadCosNames(IM_StringArray*, IM_Error*);
int  IM_ReadCos(IM_Cos*, IM_Error*);

int  IM_SetCosAttribute(IM_Cos*, char* attrName, char* attrValue,
     IM_Error*);
int  IM_UnsetCosAttribute(IM_Cos*, char* attrName, IM_Error*);

int  IM_CreateCosAttribute(IM_CosAttrDef*, IM_Error*);
int  IM_UpdateCosAttribute(IM_CosAttrDef*, IM_Error*);
int  IM_DeleteCosAttribute(IM_CosAttrDef*, IM_Error*);
int  IM_ReadCosAttributes(IM_CosAttrDefArray*, IM_Error*);
```

IM_CreateCos

```
int IM_CreateCos(IM_Cos*, IM_Error*);
```

This function creates a new COS name in the database. Only cosName field of the IM_Cos structure needs to be set. It returns error if the COS name already exists in the database.

IM_DeleteCos

```
int IM_DeleteCos(IM_Cos*, IM_Error*);
```

This function deletes a COS name and all its associated attributes from the database. Only cosName field of the IM_Cos structure needs to be set. It returns error if the COS name does not exist.

IM_ReadCosNames

```
int IM_ReadCosNames(IM_StringArray*, IM_Error*);
```

This function reads all COS names from the database. The results are returned in the IM_StringArray pointer.

IM_ReadCos

```
int IM_ReadCos(IM_Cos*, IM_Error*);
```

This function reads all the attributes associated with a COS name from the database. Only cosName field of the IM_Cos structure needs to be set. It returns error if the COS name does not exist.

IM_SetCosAttribute

```
int IM_SetCosAttribute(IM_Cos*, char* attrName, char* attrValue,
    IM_Error*);
```

This function assigns a value to an attribute for a COS. Only ‘name’ field in the IM_Cos structure needs to be specified. It overwrites the attribute value if the attribute already has an assigned value.

It returns an error in the following conditions:

- COS name does not exist.
- Unknown attribute name
- No attribute value passed

IM_UnsetCosAttribute

```
int IM_UnsetCosAttribute(IM_Cos*, char* attrName, IM_Error*);
```

This function deletes an attribute from a COS. Only ‘name’ field in the IM_Cos structure needs to be specified.

It returns an error in the following conditions:

- COS name does not exist.
- Unknown attribute name
- Attribute does not have an assigned value

IM_CreateCosAttribute

```
int IM_CreateCosAttribute(IM_CosAttrDef*, IM_Error*);
```

This function creates a new COS attribute definition in the database. ‘id’ field of IM_CosAttrDef structure is ignored. ‘name’, ‘syntax’ and ‘rule’ fields of IM_CosAttrDef structure must be specified. It returns an error if the attribute name already exists, or if the name does not begin with “pref_” or “perm_”.

IM_UpdateCosAttribute

```
int IM_UpdateCosAttribute(IM_CosAttrDef*, IM_Error*);
```

This function modifies the rule associated with an attribute. ‘id’ field in the IM_CosAttrDef structure is ignored. ‘name’ and ‘rule’ fields must be specified. It returns error if the attribute does not exist.

IM_DeleteCosAttribute

```
int IM_DeleteCosAttribute(IM_CosAttrDef*, IM_Error*);
```

This function deletes a COS attribute definition from the database if it is not used in a COS definition otherwise it returns an error. Only ‘name’ field of IM_CosAttrDef structure needs to be specified. It returns an error if the attribute does not exist.

IM_ReadCosAttributes

int IM_ReadCosAttributes(IM_CosAttrDefArray*, IM_Error*);

This function reads all COS attribute definitions from the database. The data is returned in the IM_CosAttrDefArray argument.

5.15 Configuration Information

Interface: im_config.h

```
typedef struct IM_Config
{
   IM_Header _header;
   char*     name;
   char*     host;
   char*     prog;
   char*     defvalue;
   char*     value;
} IM_Config;
```


The fields of this struct and their syntax are described below:

`_header`	internal use.
`name`	Name of the configuration variable, e.g. “netTimeout”.
`host`	Logical hostname. For example, if name is “netTimeout” and host is “mtahost”, then this IM_Config refers to the configuration variable “/mtahost/common/netTimeout”, or “/*/common/netTimeout” if that's not present. If host is NULL, then the local host is assumed.
`prog`	Name of the InterMail program requested, e.g. “mta”. This doesn't have to be the name of an actual program, as long as there is a config key associated with this program in the config.db. For example if prog equals “progname” and name is “netTimeout”, then “/*/progname/netTimeout” will be retrieved, or “/*/common/netTimeout” if that's not present.
`defvalue`	the default value, the value to return if the configuration variable is not found and if there is no wildcard key (like /*/common/...) which applies. This can be NULL if no default is desired.
`value`	returned config parameter.

```
void IM_InitConfig(IM_Config*);
void IM_FreeConfig(IM_Config*);
int  IM_ReadConfig(IM_Config*, IM_Error*);
```

IM_InitConfig

```
void IM_InitConfig(IM_Config*)
```

This function initializes an IM_Config structure.

IM_FreeConfig

```
void IM_FreeConfig(IM_Config*)
```

This function frees the `IM_InitConfig` structure.

IM_ReadConfig

```
int IM_ReadConfig(IM_Config*, IM_Error*)
```

IM_ReadConfig uses the information in IM_Config to access information in the InterMail configuration database. It fills in the value field with information from the configuration database. The name, field (and optionally the host and prog fields) must be specified to identify the desired config value. It fills the value field with the defvalue argument if it fails to find the information in the configuration database.

5.16 MIME Information

Interface: im_msg.h

```
typedef struct IM_MimeInfo
{
    IM_Header      _header;
    void*          _internalID;
    IM_Msg*        _msg;
    int            isMultiPart;
    int            isMessagePart;
    int            childCount;
    int            headerOffset;
    int            bodyOffset;
    int            headerLength;
    int            totalSize;
    int            numLines;
    char*          contentTransferEncoding;
    char*          mainContentType;
    char*          subContentType;
    char*          contentId;
    char*          contentDescription;
    char*          boundary;
    char*          contentMD5;
    IM_StringArray contentDisposition;
    IM_StringArray contentLanguage;
    IM_StringArray parameters;
} IM_MimeInfo;
```

IM_MimeInfo describes a message or a particular body part of that message. A message consists of a “top-level” body part which describes the entire message; this body part may have subparts, which may in turn have other subparts.

`_header`	Internal use.
`_internalID`	Internal use.
`_msg`	Internal use.
`isMultiPart`	Set to TRUE (1) if this is a multipart body part, otherwise FALSE (0).
`isMessagePart`	Set to TRUE if this is a message/rfc822 body part
`childCount`	The number of child body parts this body part has, if isMultiPart is TRUE.
`headerOffset`	The offset in bytes of the body part header from the start of the message. For the top-level body part, this means the offset to the start of the message header; it's always 0. For other body parts, this means the offset to the start of the MIME header.

`bodyOffset`	The offset of the body part data from the start of the message. For the top-level body part; this means the offset to the start of the body; for other body parts, it means the offset to the line right after the MIME header (right after the blank line, that is).
`headerLength`	The length of the header in bytes.
`totalSize`	The total size of this body part in bytes, including the header.
`numLines`	The number of lines in the body part data. If the body part is encoded, this refers to the number of lines in the text after encoding.
`contentTransferEncoding`	The value of the Content-Transfer-Encoding header for this bodypart, or an empty string if there isn't one.
`mainContentType`	The first half of the media type from the Content-Type: header for this bodypart.
`subContentType`	The media subtype from the Content-Type: header. For example, a text/plain bodypart will have a mainContentType of “text” and a subContentType of “plain”.
`contentId`	The value of the Content-Id header, or an empty string if there isn't one.
`contentDescription`	The value of the Content-Description header, or an empty string if there isn't one.
`boundary`	The value of the boundary parameter in the Content-Type header, or an empty string if there isn't one.
`contentMD5`	The value of the Content-MD5 header, or an empty string if there isn't one.
`contentDisposition`	A string array describing the Content-Disposition header. If the Content-Disposition header is not present, this array is empty. Otherwise it contains a disposition type string followed by a series of attribute/value pairs. See RFC 1806 for details. Example: “Content-Disposition: attachment; filename=x” is returned as the three-element array (“attachment”, “filename”, “x”).
`contentLanguage`	A string array describing the Content-Language header. If the Content-Language header is not present, this array is empty. Otherwise it contains a list of language tags. See RFC1766 for details.
`parameters`	A string array describing the Content-Type parameters. It contains a series of attribute/value pairs. For example a “Content-Type: text/plain; name=value, name2=value” header would cause the “parameters” array to contain the four-element list (“name”, “value”, “name2”, “value2”).

The functions below manipulate the IM_MimeInfo structure.

```
void IM_InitMimeInfo(IM_MimeInfo*);
int  IM_FreeMimeInfo(IM_MimeInfo*);
```

The function below retrieves MIME information in a given message:

```
int  IM_ReadMsgMimeInfo(IM_Mbox*, IM_Folder*, IM_Msg*,
     IM_MimeInfo* parent, int index, IM_MimeInfo* output,
     IM_Error*);
```

IM_InitMimeInfo

```
void IM_InitMimeInfo(IM_MimeInfo*);
```

This function initializes the IM_MimeInfo structure.

IM_FreeMimeInfo

```
int  IM_FreeMimeInfo(IM_MimeInfo*);
```

This function frees the IM_MimeInfo structure.

IM_ReadMsgMimeInfo

```
int  IM_ReadMsgMimeInfo(IM_Mbox*, IM_Folder*, IM_Msg*,
     IM_MimeInfo* parent, int index, IM_MimeInfo* output,
     IM_Error*);
```

The caller must set the host and name fields of the IM_Mbox argument, and the pathname field of the IM_Folder argument, and the msgRef field of the IM_Msg argument. These arguments specify which message to fetch information about and where to find it.

If the parent argument is NULL, then the index argument is ignored and information about the message's top-level body part (i.e. the message as a whole) is returned.

If the parent argument is non-NULL and it points to an IM_MimeInfo describing a multi-part body part, then the index argument must specify the index of a child body part. This index must be between 0 and parent->childCount - 1. Information about that child body part will be returned.

If the parent argument is non-NULL and it points to an IM_MimeInfo structure describing a message/rfc822 body part, then the index argument is ignored, and information about the body part contained by the message will be returned.

On success, the structure pointed to by the output argument will contain the requested information about the message.

Here is a sample piece of code that fetches information about the complete body part tree for a message.

```
void
dumpSubMimeInfo(IM_Mbox* mbox, IM_Folder* folder, IM_Msg* msg,
                IM_MimeInfo* minfo, int level, IM_Error* err)
{
    char pad[100];
```

```
if (level >= 100)
        return;

    memset(pad, ' ', level);
    pad[level] = '\0';

    printf("%sIM_MimeInfo:\n"
        "  %s%s childCount=%d hoff=%d boff=%d hlen=%d"
           " totsize=%d lines=%d\n"
        "  %scte=\"%s\" ct=\"%s/%s\" cid=\"%s\" cdesc=\"%s\"\n"
        "  %sbound=\"%s\" cmd5=\"%s\"\n",
        pad,
        pad, minfo->isMultiPart ? "multi" :
             minfo->isMessagePart ? "msg" : "single",
             minfo->childCount, minfo->headerOffset,
             minfo->bodyOffset, minfo->headerLength,
             minfo->totalSize, minfo->numLines,
        pad, minfo->contentTransferEncoding, minfo->mainContentType,
             minfo->subContentType, minfo->contentId,
             minfo->contentDescription,
        pad, minfo->boundary, minfo->contentMD5);

    if (minfo->isMessagePart || minfo->isMultiPart) {
        int i;
        int count;
        IM_MimeInfo child_minfo;

        IM_InitMimeInfo(&child_minfo);
        count = (minfo->isMultiPart ? minfo->childCount : 1);
        for (i = 0; i < count; i++) {
            IM_ReadMsgMimeInfo(mbox, folder, msg, minfo, i,
            &child_minfo, err);
            dumpSubMimeInfo(mbox, folder, msg, &child_minfo,
            level+1, err);
        }
        IM_FreeMimeInfo(&child_minfo);
    }
}

void
dumpMimeInfo(IM_Mbox *mbox, IM_Folder *folder, IM_Msg *msg, IM_Erro
*err)
{
    IM_MimeInfo minfo;
    IM_InitMimeInfo(&minfo);
    if (IM_ReadMsgMimeInfo(mbox, folder, msg, NULL, 0, &minfo, err)
          == IM_FAILURE)
        fprintf(stderr, "ReadMsgMimeInfo failed!\n");
    else
        dumpSubMimeInfo(mbox, folder, msg, &minfo, 0, err);
    IM_FreeMimeInfo(&minfo);
}
```

5.17 Log Messages

This class can create new InterMail log files, append entries to a log file and fetch the text that would be generated for an entry. It works together with the IM_LogContext class.

Interface: im_log.h

```
typedef enum IM_Severity
{
    IM_SEVERITY_UNKNOWN = 0,
    IM_SEVERITY_NOTIFICATION = 1,
    IM_SEVERITY_WARNING = 2,
    IM_SEVERITY_ERROR = 3,
    IM_SEVERITY_URGENT = 4,
    IM_SEVERITY_FATAL = 5
    IM_SEVERITY_MAX = IM_SEVERITY_FATAL
} IM_Severity;

typedef struct IM_LogMsg
{
  IM_Header       _header;
  IM_Severity     severity;
  int             msgId;
  char*           type;
  IM_StringArray  args;
  char*           text;
  char*           logName;
} IM_LogMsg;
```

The information about log messages is conveyed via the structure IM_LogMsg:

`_header`	Internal use
`severity`	The severity of the condition that is prompting the log message.
`type`	A short string, corresponding to an InterMail message number, which describes the type of message to log; for example, “PopConnMade”.
`msgId`	An InterMail message number identifying the message. For example, the message id for PopConnMade is 4325385. In almost all cases the caller should specify “type” rather than “msgId” to identify a message.
`args`	A list of arguments to include in the log entry. Most message numbers expect a pre-determined list of arguments, e.g. `PopConnTimedOut` expects the first args value to be the number of seconds that passed before the client timed out.
`logName`	Name of the log file. This is only used when the first log entry is written, or when the log file is created.

These functions manipulate the IM_LogMsg structure:

```
void IM_InitLogMsg(IM_LogMsg*, IM_Error*);
void IM_FreeLogMsg(IM_LogMsg*);
int  IM_CreateLogFile(IM_LogMsg*, IM_Error*);
int  IM_WriteLogMsg(IM_LogMsg*, IM_Error*);
int  IM_ReadLogMsgText(IM_LogMsg*, IM_Error*);
```

Example:

```
IM_LogMsg lg;
IM_Error er;
IM_InitError(&er);
IM_InitLogMsg(&lg);
lg.logName = strdup("myprog");
lg.messageType = strdup("ProcSomethingBad");
lg.args.num = 3;
char *args[3];
args[0] = "arg1";
args[1] = "arg2";
args[2] = "arg3";
lg.args.strings = args;
lg.severity = IM_SEVERITY_URGENT;
IM_WriteLogMsg(&lg, &er);
IM_GetLogMsgText(&lg, &er);
printf("log message looks like: %s\n", lg.text);
lg.args.strings = 0;
IM_FreeLogMsg(&lg);
```

IM_InitLogMsg

```
void IM_InitLogMsg(IM_LogMsg* logmsg, IM_Error* error);
```

Initializes the IM_LogMsg structure. IM_InitLogMsg() must be called before attempting to use any function which uses IM_LogMsg.

IM_FreeLogMsg

```
void IM_FreeLog Msg(IM_LogMsg* logmsg);
```

Free the IM_LogMsg structure.

IM_CreateLogFile

```
int IM_CreateLogFile(IM_LogMsg* logmsg, IM_Error *error);
```

Creates or opens the log file, if it hasn't been created or opened already, using “logName” as the base part of the filename. For example if “logName” is “mta”, then this function will attempt to open the file “mta.server-hostname.current-date-and-time.log”, creating it if necessary. All further log messages will go to that log file. If a log file has already been opened this function does nothing.

IM_WriteLogMsg

```
int IM_WriteLogMsg(IM_LogMsg* logmsg, IM_Error* error);
```

Writes a log message to the log file. If the log file hasn't been opened, then “logName” specifies the base part of the filename.

IM_ReadLogMsgText

```
int IM_ReadLogMsgText(IM_LogMsg* logmsg, IM_Error* error);
```

Formats a log message and returns it in the “text” field of the logmsg.

5.18 Log Context

This structure is used to describe context information, which is used when generating log messages. Any log message that is generated will have context information appended to it. It is not necessary to set the log context, but it can be convenient to ensure that log messages always contain consistent information. The format of the context strings is not very important; they will be written to the log file as-is.

Interface: im_log.h

```
typedef struct IM_LogContext
{
  IM_Header _header;
  char*     mailUser;
  char*     mailHost;
  char*     mailbox;
  char*     from;
  char*     folder;
  char*     msgId;
  char*     origMsgId;
  char*     cmd;
  int       port;
  int       size;
  char*     destHost;
  char*     fromHost;
} IM_LogContext;
```


The information about log context is conveyed via the structure IM_LogContext:

`_header`	For internal use.
`mailUser`	The current username or recipient address. If this is set, then it will appear as “user=xxx” in the log file.
`mailHost`	The current mail server we are talking to or trying to connect to. If this is set, then it will appear as “mss=xxx” in the log file.
`mailbox`	The current mailbox id. If this is set, then it will appear as “mbox=xxx”.
`from`	The current from address. If this is set, then it will appear as “from=xxx”.
`folder`	The current folder name. If this is set, then it will appear as “fldr=xxx”.
`msgId`	The message ID of the current message. If this is set, then it will appear as “msgid=xxx”.
`origMsgId`	The original message ID of the current message. If this is set, then it will appear as “origMsgid=xxx”.
`cmd`	The current client command. If this is set, then it will appear as “cmd=xxx”.
`port`	Current port number. If this is set, then it will appear as “port=xxx”.
`size`	The size in bytes of the current message. If this is set, then it will appear as “size=xxx”.
`fromHost`	The current client host. If this is set, then it will appear as “fromhost=xxx”
`destHost`	The current destination host. If this is set, then it will appear as “desthost=xxx”.

These functions manipulate the IM_LogContext structure:

```
void IM_InitLogContext(IM_LogContext*, IM_Error*);
void IM_FreeLogContext(IM_LogContext*);
int  IM_UpdateLogContext(IM_LogContext*, IM_Error*);
int  IM_ReadLogContext(IM_LogContext*, IM_Error*);
```

For instance, if a “`PopConnMade`” log message is written to the log with no arguments, but with the `mailUser` context variable set to “`tmpusr7001`”, `mailHost` set to “`calgary`”, `port` set to `5681`, and `mailbox` set to `7001`, then the following text will be generated in the log file (as a single line):

```
19980518 125545971-0700 calgary popserv 23904 10 12 Note;PopConnMade
(66/9)user=tmpusr7001:mss=calgary:port=5681:mbox=7001
```


IM_InitLogContext

```
void IM_InitLogContext(IM_LogContext* logc, IM_Error* error);
```

Initializes the IM_LogContext structure. IM_InitLogContext() must be called before attempting to use any function which uses IM_LogContext.

IM_FreeLogContext

```
void IM_FreeLogContext(IM_LogContext* logc);
```

Free the IM_LogContext structure.

IM_UpdateLogContext

```
int IM_UpdateLogContext(IM_LogContext* logctx, IM_Error* error);
```

Update the log context to match the information in "`logctx`". Fields become disabled if they are updated to a NULL value. Subsequent log messages will include the updated log context information.

IM_ReadLogContext

```
int IM_ReadLogContext(IM_LogContext* logctx, IM_Error* error);
```

Read the current log context and save the context information in "`logctx`".

5.19 Compiling, Linking, and Running

InterMail header files are installed in the `include` directory at the top of the InterMail product directory. InterMail libraries are installed in the `lib` directory at the top of the InterMail product hierarchy. Additional required libraries come from the Oracle installation and your local operating system.

The API will connect directly to an Oracle database server when accessing some InterMail Directory information. This means all applications that use the API are Oracle clients, and there is a *long* list of Oracle libraries that are required to support this capability.

***Note**: The repeated Oracle libraries in the link command are necessary due to circular dependencies in the Oracle library set.*

Some of the libraries that are accessed by `libim.so` will spawn threads to do some of their work, so all applications that use the API are inherently multi-threaded. Most platforms have special requirements for compiling and linking multi-threaded applications, and examples are given below for particular operating systems.

Currently, some portions of the InterMail C-API are not re-entrant, so unpredictable errors may occur if functions in this library are called simultaneously by more than one thread in the same process. If the InterMail C-API is used by multiple threads in a single process, you should implement a single mutex that is always locked before any of the API functions are invoked.

Users of the API should always have suitable values for the environment variables $INTERMAIL, $ORACLE_HOME, and $TNS_ADMIN. A great deal of information about the InterMail software is determined by a configuration database stored under the $INTERMAIL directory, so it is imperative that applications use the correct location. Likewise, the $ORACLE_HOME and $TNS_ADMIN directories have a significant impact on the behavior of the Oracle libraries used by the API. The shell startup files (.profile, .cshrc) for the `imail` user are set up by the InterMail installation to initialize these variables appropriately.

The $INTERMAIL environment variable is mandatory when executing code linked with the API. The $ORACLE_HOME and $TNS_ADMIN can be derived from the InterMail configuration database, so under normal circumstances you won't need to set them before executing your application. These Oracle environment variables are still necessary for compiling and linking. If the values are not set, or if the directories cannot be found, then the library functions will fail promptly, but if there are multiple installations of InterMail or Oracle on a single machine, then inconsistent behavior may result if the environment variables point to the wrong installation directory.

While it may be possible to use a "C" compiler, the C++ compiler is recommended, and the C++ linker is required because libomu, which you must link with, is a C++ library.

The header files are compatible with both C and C++ source files.

Note that your Makefile or compile command line should specify where to find these files. They are located below the directory specified by the $INTERMAIL variable, in the include subdirectory. A Makefile is strongly recommended, given the number files to include and link.

5.20 Sun Microsystems Solaris 2.5.1 and 2.6

To compile files for multi-threaded applications on Solaris with the SPARCWorks C or C++ compiler, use the "`-mt`" flag.

```
cc -mt -I$INTERMAIL/include -c myfile.c
```

To link an application with the InterMail API on Solaris, use the following command as an example.

```
CC -mt -o application object-files… \
       -L$INTERMAIL/lib -lim –lomu \
       -L$ORACLE_HOME/lib -lsql \
       -lsqlnet -lncr -lsqlnet -lclient -lcommon -lgeneric \
       -lsqlnet -lncr -lsqlnet -lclient -lcommon -lgeneric \
       -lepc -lnlsrtl3 -lc3v6 \
       -lcore3 -lnlsrtl3 -lcore3 -lnlsrtl3 \
       -ldl -lsocket -lnsl -lm
```


5.21 Digital UNIX

To compile files for multi-threaded applications on Digital UNIX, use the "`-pthread`" flag.

```
cxx -pthread -I$INTERMAIL/include -c myfile.c
```

To link an application with the InterMail API on Digital UNIX, use the following command as an example.

```
cxx -pthread -o application object-files… \
       -L$INTERMAIL/lib -lim -lomu \
       -L$ORACLE_HOME/lib -lsql \
       -lsqlnet -lncr -lsqlnet -lclient -lcommon -lgeneric \
       -lsqlnet -lncr -lsqlnet -lclient -lcommon -lgeneric \
       -lepc -lnlsrtl3 -lc3v6 \
       -lcore3 -lnlsrtl3 -lcore3 -lnlsrtl3 \
       -lm
```

5.22 Silicon Graphics IRIX 6

To compile files for multi-threaded InterMail applications on Silicon Graphics IRIX, use the –32 and –signed flags and the following pre-processor definitions.

```
cc –32 –signed -D_SGI_REENTRANT_FUNCTIONS -D_SGI_MP_SOURCE \
    -I$INTERMAIL/include -c myfile.c
```

To link an application with the InterMail API on Silicon Graphics IRIX, use the following command as an example. Note that the C++ linker is used, even though your source code may be written in C.

```
CC -o application object-files… \
       -L$INTERMAIL/lib -lim -lomu \
       -L$ORACLE_HOME/lib -lsql \
       -lsqlnet -lncr -lsqlnet -lclient -lcommon -lgeneric \
       -lsqlnet -lncr -lsqlnet -lclient -lcommon -lgeneric \
       -lepc -lnlsrtl3 -lc3v6 \
       -lcore3 -lnlsrtl3 -lcore3 -lnlsrtl3 \
       -lm -lpthread
```


5.23 File Summary

InterMail API header files:

```
im_account.h
im_config.h
im_cos.h
im_defs.h
im_domain.h
im_error.h
im_folder.h
im_log.h
im_mbox.h
im_msg.h
im_reply.h
im_stringarray.h
im_types.h
```

InterMail API libraries:

```
libim.so  (link to libim.4.x)
libomu.so (link to libomu.4.x)
```

6

InterMail Perl API

This chapter describes the InterMail Perl API (SwCom::Mail), which operates on the following objects:

- Class of Service Attributes
- Classes of Service
- Domains
- Accounts
- Auto-Reply Messages
- Mailboxes
- Folders
- Messages

6.1 Introduction

The programming interface relies on the object-oriented features of Perl 5, and uses a Perl binary distribution provided by Software.com.

The Perl classes making up SwCom::Mail are built upon the corresponding objects in the InterMail C API and depend strongly on the conventions used in it. Users of SwCom::Mail are encouraged to read Chapter 5 for an understanding of the fundamental concepts of InterMail administrative objects and the requirements for parameters of various function calls.

SwCom::Mail closely mimics the InterMail C API. Because of the differences between C and Perl, some SwCom::Mail functions deviate from their C counterparts, but they usually follow an intuitive mapping when the object-oriented Perl supports a simpler form of expression than the C syntax would allow.

A detailed description for each class and its methods is given in Section 6.2.

6.1.1 General Usage Notes

Due to various operating systems' restrictions, Perl scripts that use SwCom::Mail classes can only be executed by a specially built, InterMail Perl interpreter. The fundamental Perl source code was not changed for this version of the interpreter, but the compilation and linking options for programs that spawn threads differ from the configuration in the standard Perl distribution.

6.1.2 Hooking SwCom::Mail from Your Scripts

To use SwCom::Mail, place the following statements in the Perl script before using any SwCom::Mail classes, functions, or constants:

```
use SwCom::Mail::All;
im_init();
```

This will import the necessary definitions, including the definitions of the symbolic constants which enumerate the types of domains, accounts, password encodings, etc., and then initialize the InterMail library.

Note: *In order to allow InterMail services to use this script, the user must first initialize the library by calling the* `IM_init()` *function.*

Most InterMail objects contain fields that are implemented as enumerations, or symbolic constants. Consult the C API documentation for the names of relevant constants and use them in accordance with their “C API purpose”.

For example, this statement in Perl:

```
print '▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫'▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫"▫▫"▫
```

will produce the following output:

```
IM_DOMAIN_NON_AUTH = 78
```

in accordance with the C API definition in `IM_domain.h`:

```
typedef enum IM_DomainType {
  ...
  IM_DOMAIN_NON_AUTH = '▫'▫
  ...
} IM_DomainType;
```

The symbolic constants in the InterMail API are implemented as dynamically defined subroutines in Perl. This may cause some confusion if these constants are combined in arithmetic expressions. Currently, you never need to combine these constants to use the Perl API, but should you need to mix these constants in an expression, prefix them with “`&`” or append “`()`” to them. Using either of these conventions informs the Perl interpreter that these symbols should be treated as subroutines, which triggers the necessary “AUTOLOAD” functionality to define them.

6.1.3 Underlying Libraries and Versioning

A prerequisite to execute a Perl script using SwCom::Mail is to have all underlying libraries on your system at locations where they can be found by the run-time linker. This includes:

- system libraries, such as `libc.so`, `libC.so`, `libpthread.so`, etc. Theoretically, these libraries should always be in the right directories, e.g. in `/usr/lib` or `/usr/shlib` (depending on the platform).
- InterMail shared libraries: currently there are two of them, `libomu.so` and `libim.so`. For each of these libraries, the containing directory (usually `$INTERMAIL/lib`) should be part of the environment variable `$LD_LIBRARY_PATH`.
- Perl modules making up the SwCom::Mail library; these should be at a location where the Perl interpreter can find them. The Perl interpreter installed with InterMail (typically, `$INTERMAIL/perl/bin/perl`) knows how to find the standard version of these libraries in the normal location. You may use a custom version of SwCom::Mail by setting the environment variable `$PERL5LIB` so that your private library's location is searched before the one in the Perl package.

The SwCom::Mail library provides a Perl interface to the InterMail C API (`libim.so`), which itself works as a gateway to InterMail foundation classes (`libomu.so`), database libraries and so forth. A user of SwCom::Mail does not need to worry about software layers underneath the C API library, which encapsulates the complexity of InterMail internals and screens users from the ongoing changes in them. From time to time, the C API will change in a way that impacts the Perl API, which may cause some older scripts to fail with the updated InterMail. The release notes will describe all known incompatibilities.

The version of InterMail for which SwCom::Mail was built is available in Perl by calling the function `IM_version()`. A user may want to check the InterMail version before doing anything else in a script:

```
use SwCom::Mail::All;
IM_init();
my $version = IM_version();
if ($version =~ /^4\./) {
  print ?Version=$version\n?;
      # prints something like: “Version=4.0-1.5”
} else {
  die ?Script assumes InterMail 4.xx; current version is
$version\n?;
}
```

The SwCom::Mail library described in this chapter corresponds to Version 4.0 of InterMail.

6.1.4 Error Handling

Most of the methods return a non-zero integer (1, to be accurate) for success, and 0 otherwise. This is a common (but not universal) convention. In a few cases it is more convenient and natural to return a list on success, and undef for failure.

Detailed error information from the InterMail C API can be examined by calling the following Perl functions:

- `IM_errnum()`
- `IM_errstr()`
- `IM_errmnemonic()`

`IM_errnum()` returns a non-zero value if an InterMail operation failed. In this case, the `IM_errnum()`'s return code is the InterMail's error number, the full or mnemonic description of the reason why the operation failed, should be available as a string returned by the functions `IM_errstr()` or `IM_errmnemonic()`, respectively.

It is useful to use the following definition of an error handling Perl subroutine, which will be repeatedly used in examples:

```
sub print_status
{
      my ($n,$s);
      if ($n = IM_errnum()) {
            $s = IM_errstr() || IM_errmnemonic() ||
                 '(no further information)';
            print ?** Error $n: $s\n?
            return;
      }
      print ?** OK\n?;
}
```

6.1.5 Classes and Objects

This section illustrates the general ideas and conventions of using SwCom::Mail objects, using the Domain object as an example.

It is important to understand the difference between creating a domain as a record in a directory database and creating a Perl object of the type Domain. The latter works just as a place-holder for information to be transferred to or from C functions that operate on a directory database and may, in particular, actually create a domain as a new record in the database.

To create a Perl object of the type Domain, one should invoke a class constructor, through a method new:

```
$domain=new Domain(
  type        => IM_DOMAIN_LOCAL,
  name        => 'oops.com',
  ...
);
```

To create a new domain record in the directory database, call the method `Create()` on a Perl object previously constructed as shown above:

```
$domain->Create();
```

Note that Perl objects and the InterMail objects they refer to may be (and in practice, often will be) out of sync. One will likely create and use "incomplete" Perl objects— the ones in which some fields are undefined or are not valid. The only way to obtain a "snapshot" of a real InterMail object is to invoke the method `Read()` from the corresponding Perl class. (Theoretically, by the time the information about field values is passed to a user script by a `Read()` method, the underlying InterMail object may change due to another process's activity. But this is a general problem with concurrent manipulation of database records and is not specific to SwCom::Mail.)

The same is true for objects of all other SwCom::Mail classes and their corresponding InterMail entities.

All class constructors (`new()` subroutines) use *call-by-pairs* format, where one can conveniently assign a value to all or some attributes of the object. It is not necessary to set every field of an object for all operations—most methods use only a small subset of an object's fields. In some cases (the `Read()` method) most of the fields are assigned a (new) value as a result of a method call.

One can also directly assign values to an object's fields, like this:

```
$domain->relayHost('hostess');
```

The value of an object's field can be accessed this way:

```
$host=$domain->relayHost();
```

Note that it is possible to set or get the value of an object's field by using this syntax:

```
$domain->{relayHost} = 'hostess';
$host = $domain->{relayHost};
```

This syntax is discouraged, since the misspelling of a field name will go unnoticed until long after the point where the error occurred. Setting/getting the value of a field through a function calls is safe—for our example above, the class Domain must have a field named "relayHost", otherwise these functions would fail immediately.

All SwCom::Mail classes define the method `Dump()` for returning the information on the field values as a string, in a format that is suitable for direct printing on a stream. One can use the following construction, for example:

```
print STREAM object->Dump();
```

Note: *For a field that takes an integer value from a finite set, corresponding to a C enumeration, the value of the field will be printed not as an integer but as the name of the enumeration constant.*

This chapter uses the term instance method (function) for functions that can only be invoked with an object, like this:

```
object_reference->method(...)
```

Class methods (functions) do not need an object to operate upon and can be invoked this way:

```
class_name::method(...)
```

Class methods resemble static methods in C++.

6.2 Class Reference

This section describes the attributes and methods of all classes in the InterMail Perl API.

6.2.1 CosAttribute

The class CosAttribute (Class of Service Attribute) has the following fields:

`id`	Integer
`name`	String. All CosAttribute names must begin with a prefix of “perm_” or “pref_”.
`rule`	`enum IM_AttributeRule` (see Chapter 5)
`syntax`	`enum IM_AttributeSyntax` (see Chapter 5)

A new `CosAttribute` object can be created this way:

```
$cos = new CosAttribute(
    name   => '␣␣␣␣␣␣␣␣␣␣'␣
    rule   => IM_ATTR_RULE_COS_OVERRIDES,
    syntax => IM_ATTR_SYNTAX_NUMERIC,
};
```

InterMail is installed with a large set of pre-defined `CosAttributes`. Most of the time, you will be referencing the existing attribute definitions, rather than creating new ones.

Note: *It is best to avoid upper-case characters when accessing or defining the name fields of `CosAttribute` and `Cos` objects. All upper-case letters in name fields will be converted to lower-case before they are stored, or queried, in the database. This means it is possible to look up `CosAttributes` and `Cos` objects using mixed case, but the return values from `CosAttribute::ReadAll()` will use no upper-case. For `Cos::Read()` and `Account::Read()`, the keys of the `serviceDef, acCos, and resultCos` hashes will likewise be all lower-case.*

The class `CosAttribute` defines the following methods:

ReadAll ( )

Prerequisites

None. This is a class method, which must be run as follows:

```
my @attr_list = CosAttribute::ReadAll();
```

Implementation

Calls `IM_ReadCosAttributes()`.

Return Values

Returns a list of references to `CosAttribute` objects on success, undef on failure.

Create ( )

Prerequisites

The name, rule, and syntax fields must be defined. The name field must not conflict with any existing `CosAttribute`.

Implementation

Calls `IM_CreateCosAttribute()`.

Return Values

Returns “1” on success, “0” on failure.

Delete ( )

Prerequisites

The name field must refer to an existing `CosAttribute`.

Implementation

Calls `IM_DeleteCosAttribute()`.

Return Values

Returns “1” on success, “0” on failure.

Update ( )

Prerequisites

The name field must refer to an existing `CosAttribute`.

Implementation

Calls `IM_UpdateCosAttribute()`.

Return Values

Returns “1” on success, “0” on failure.

Example

```
my @attr_list = CosAttribute::ReadAll();
my $attr = $attr_list[0] if defined(@attr_list);

### For fast lookup of attributes by name:
my %attr_dict;
foreach $attr (@attr_list) {
    $attr_dict{$attr->name()} = $attr;
}
$attr = $attr_dict{’basic’};

# ..........................................................

my $attr = new CosAttribute(
    name   => '□□□□□□□□□□□□□□□□□□□□'□
    rule   => IM_ATTR_RULE_COS_OVERRIDES,
    syntax => IM_ATTR_SYNTAX_STRING);
$attr->Create();

$attr->rule(IM_ATTR_RULE_ACCOUNT_OVERRIDES);
$attr->Update();

$attr->Delete();
```

6.2.2 Cos

The class Cos (Class of Service) has the following fields:

`name`	String
`serviceDef`	Reference to associative array of Class of Service (COS) values

A new Cos object can be created this way:

```
my $cos = new Cos(
    name       => '□□□□□'□
    serviceDef => {
        pref_quotatotkb     => '□□□□□□□
        pref_quotathreshold => '□□'□
        pref_pop            => '□'□
        pref_imap           => '□'□
        pref_intermanager   => '□'
    }
);
```

Note: *The value for the "serviceDef" field is a reference to an anonymous hash.*

See the note under `CosAttribute` *about upper-case characters in name fields.*

The current values of the fields will be returned as a formatted string by the Dump() function.

The class Cos defines the following methods to access the InterMail facilities:

ReadNames ( )

Prerequisites

None. This is a class method, which must be run as follows:

```
my @name_list = Cos::ReadNames();
```

Implementation

Calls `IM_ReadCosNames()`.

Return Values

Returns a list of names of Cos objects on success, undef on failure.

Create ( )

Prerequisites

The name field must be defined and not already in use by a previous created Cos.

Implementation

Calls `IM_CreateCos()`.

Return Values

Returns "1" on success, "0" on failure.

Read ( )

Prerequisites

The name field must refer to an existing Cos.

Implementation

Calls `IM_ReadCos()`.

Return Values

The "serviceDef" field is set to the current state of the Cos. Returns "1" on success, "0" on failure.

SetAttribute (*attr_name, attr_value*)

Prerequisites

The name field must refer to an existing `Cos`. The `attr_name` argument must correspond to an existing `CosAttribute`, and `attr_value` must be defined.

Implementation

Calls `IM_SetCosAttribute()`.

Return Values

Returns "1" on success, "0" on failure. The attribute is added to the set defined by the Cos if necessary, and the value is set accordingly.

UnsetAttribute (attr_name)

Prerequisites

The name field must refer to an existing Cos. The `attr_name` argument must correspond to an existing `CosAttribute`.

Implementation

Calls `IM_UnsetCosAttribute()`.

Return Values

Returns "1" on success, "0" on failure. The attribute is removed from the set defined by the Cos.

Delete ( )

Prerequisites

The name field must refer to an existing Cos.

Calls `IM_DeleteCos()`.

Return Values

Returns “1” on success, “0” on failure.

Example

```
my @name_list = Cos::ReadNames();

my $cos = new Cos(name => $name_list[0]);
$cos->Read();

my %serviceDef = %{$cos->serviceDef()};
my ($name, $value);
while (($name, $value) = each %serviceDef) {
    print "Attribute \"$name\" has value \"$value\"\n";
}
# ..........................................................
my $serviceDef = {
    pref_quotaTotKb => ’10000’,
    pref_quotaBounceNotify => ’1’
};
$cos = new Cos(name => ’basic’, serviceDef => $serviceDef);
$cos->Create();

$cos->SetAttribute('□□□□□□□□□□□□□□□'□□'□'□□
$cos->SetAttribute('□□□□□□□□□'□□'□'□□
$cos->UnsetAttribute('□□□□□□□□□□□□□□□□□□□□□□'□□
$cos->Delete();
```

6.2.3 Domain

The class Domain has the following attributes (fields):

`type`	Enumerated integer (see Chapter 5)
`name`	String
`relayHost`	String
`rewriteName`	String
`wildcardAccount`	String

A new Domain object can be created this way:

```
$domain=new Domain(
  type        => IM_DOMAIN_LOCAL,
  name        => '□□□□□□□'
);
```

The current values of the fields will be returned as a formatted string by the `Dump()` function.

The class Domain defines the following methods to access the InterMail facilities (see Chapter 5 for the requirements to the parameters and parameter combinations for various operations):

List (*from, till, max*)

Prerequisites

This is a class function that should not be invoked on a Domain object. Instead, use the following form:

```
my @domain_list = Domain::List($from, $till, $max);
```

All arguments must be specified, but from and till can be empty strings, indicating the lexically least and greatest domain names, respectively. The max argument should be a number.

Implementation

Calls `IM_ReadDomains()`.

Return Values

Returns a list of names on success, undef on failure.

The domain names that are lexically greater than or equal to from, and lexically less than till, are returned in a list. If from is empty or undefined, there is no lower lexical bound on the match. If till is empty or undefined, there is no upper lexical bound on the match. The comparison with from is inclusive, while till is compared exclusively, meaning domain names that match till exactly will not be returned. At most max names will be returned, and domain names will be returned in lexically increasing order. If max is –1, then all matching domain names will be returned.

It is possible to iterate through all domains using code similar to the following:

```
my $max = 100;
my $from = ''▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫
my $till = ''▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫
my @list;

while (@list = Domain::List($from, $till, $max)) {
    shift(@list) if $list[0] eq $from; # Skip first if we've seen
it.
    last unless @list;     # All done if list is now empty.
    my $name;
    foreach $name (@list) {
        print "▫▫▫▫▫▫▫▫▫▫"▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫
    }
    $from = pop(@list);    # Get the next set of names.
}
```


ListSubDomains (*topDomain, from, till, max*)

Prerequisites

This is a class function that should not be invoked on a Domain object. Instead, use the following form:

```
my @subdomains = Domain::ListSubDomains($topDomain, $from, $till,
$max);
```

All arguments must be specified, but from and till can be empty strings. The topDomain and max arguments must be defined, and max must be a number greater than zero.

Implementation

Calls `IM_ReadSubDomains()`.

Return Values

Returns a list of domain names on success.

The domain names contained by `topDomain` that are lexically greater than “from” and lexically less than “till” are returned. If “from” is empty or undefined, there is no lower lexical bound on the match. If “till” is empty or undefined, there is no upper lexical bound on the match. Both “from” and “till” are exclusive bounds, meaning domain names that match either “from” or “till” exactly will not be returned. At most “max” names will be returned, and domain names will be returned in lexically increasing order.

It is possible to iterate through all the sub-domains contained by a domain using code similar to the following:

```
my $topdomain = '▫▫▫▫▫▫▫▫▫▫▫▫▫'▫
my $max = 100;
my $from = ''▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫
my $till = ''▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫
my @list;

while (@list = Domain::ListSubDomains(
                    $topdomain, $from, $till, $max)) {
    my $name;
    foreach $name (@list) {
        print "▫▫▫▫▫▫▫▫▫▫"▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫
    }
    $from = pop(@list);    # Get the next set of names.
}
```

Create ( )

Prerequisites

The fields name must be defined and valid.

Implementation

Calls `IM_CreateDomain()`.

Return Values

Returns “1” on success, “0” on failure.

Read ( )

Prerequisites

The field name must be defined and valid.

Implementation

Calls `IM_ReadDomain()`.

Return Values

Returns “1” on success, “0” on failure.

Update ( )

Prerequisites

The field name must be defined and valid.

Implementation

Calls `IM_UpdateDomain()`.

Return Values

Returns “1” on success, “0” on failure.

For those fields of the Perl object that are defined, the values will be used to rewrite the values of the record in the InterMail database.

No modification to the fields of the object is done in the result of calling this method. Use `Read()` to update the fields after calling `Update()`.

Note: *When `Update()` specifies a wildcard account, the user must use only the local portion of the SMTP address.*

Delete ( )

Prerequisites

The field name must be defined and valid.

Implementation

Calls `IM_DeleteDomain()`.

Return Values

Returns “1” on success, “0” on failure.

Example

```
$domain = new Domain(
  type        => IM_DOMAIN_LOCAL,
  name        => '▫▫▫▫▫▫▫'▫
);

$domain->type(IM_DOMAIN_REWRITE);
$domain->name('▫▫▫'▫▫
$domain->rewriteName('▫▫▫'▫▫
print '▫▫▫▫▫▫▫▫▫▫▫▫▫'▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫"▫▫"▫
print $domain->Dump();

$domain->Create();
print_status();
if ($domain->Read()) {
  print $domain->Dump();
}
$domain->rewriteName('▫▫▫▫▫▫▫'▫▫
$domain->Update();
print_status();
$domain->Delete();
print_status();
```

6.2.4 Account

The class Account has the following attributes (fields):

`smtpAddress`	String (must be a valid RFC822 mail address)
`popAddress`	String
`mssHost`	String
`smtpHost`	String
`popHost`	String
`emailIntID`	String (of decimal digits)
`password`	String
`plainPassword`	String
`hash`	`enum IM_PwHashType` (see Chapter 5)
`status`	`enum IM_AcStatus` (see Chapter 5)
`local`	`enum IM_LocalDelivery` (see Chapter 5)
`cosName`	String
`acCos`	Reference to associative array of Class of Service (COS) values
`resultCos`	Reference to associative array of Class of Service (COS) values (*read-only*)

A new Account object can be created in the following manner:

```
$account=new Account(
  smtpAddress     => '□□□□□□□□□□□□'□
  popAddress      => '□□□□□□□□□□'□
  mssHost         => '□□□□□□□'□
  emailIntID      => '□□□□□□□□□□□□□□□□'□
  plainPassword   => '□□□□□□□□'□
  hash            => IM_PWHASH_CLEAR,
  status          => IM_ACSTATUS_ACTIVE,
  local           => IM_DELIVERY_ENABLED,
  acCos           => { pref_quotaTotKB     => '□□□□□'□
                       pref_quotaThreshold => '□□'
                     }
);
```

The class Account defines the following methods to access the InterMail facilities (see Chapter 5 for the requirements to the parameters and parameter combinations for various operations):

Create ( )

Prerequisites

The fields `smtpAddress` and `mssHost` must be defined and valid; the field `emailIntID` must be set to an "integer-like" string (i.e. a string consisting of decimal digits only).

If the account is to be accessible via POP or IMAP, the `plainPassword` or `password` field must be set. If `plainPassword` is set, it specifies the desired password in plaintext, and it takes precedence over password. Otherwise, password is assumed to be encrypted as specified by hash. If hash is not set, it defaults to `IM_PWHASH_CLEAR`.

To create an account in proxy mode, specify a status of `IM_ACSTATUS_PROXY`, and define values for both `smtpHost` and `popHost`, but do not define `mssHost`. If the account is not in proxy mode, define `mssHost`, but do not define `smtpHost` or `popHost`.

Implementation

Calls `IM_CreateAccount()`.

Return Values

Returns "1" on success, "0" on failure.

If the call succeeds, an account record will be created in the directory, with the parameters specified by the fields of the Perl object, or, for non-mandatory fields (see Prerequisites above), set to default values.

Read ( )

Prerequisites

One of the following fields must be defined: `smtpAddress` , `popAddress`, `emailIntID`. If more than one is set, the precedence follows the order shown. If `plainPassword` is set, the call is treated as an authentication request, and it will return an error if the password is incorrect.

Implementation

Calls `IM_ReadAccount()`.

Return Values

Returns “1” on success, “0” on failure.

If the call succeeds, all fields of the Perl object will be set from the InterMail Directory.

Update ( )

Prerequisites

The field `smtpAddress` must be defined and valid.

Implementation

Calls `IM_UpdateAccount()`.

Return Values

Returns “1” on success, “0” on failure.

For those fields of the Perl object that are defined, the values will be used to rewrite the values of the record in the InterMail database. There are constraints on the values of certain fields during an update that are described in the InterMail C API documentation.

For the acCos field, every key that is defined in the associative array will be used in the update. Keys that have an undefined value will result in the removal of the corresponding COS attribute from the account. Other keys will add or modify COS attributes to the account.

No modification to the fields of the object is done in the result of calling this method. Use `Read()` to update the fields after calling `Update()`.

Generally, it is not recommended that the `Update()` function be applied to objects that have been `Read()`.The problem is that every single field of the object is being updated, not just the fields changed after the `Read()`. Instead, create a separate object for the call to `Update()`, and set the fields needed to identify the object and the fields that need to be changed. For example:

```
$account=new Account(smtpAddress => "joe.bloe@software.com");
$account->Read();

$upaccount = new Account(
   smtpAddress => $account->smtpAddress(),
   mssHost => "mss2");
$upaccount->Update();
```

This will work assuming that the account was not previously in proxy mode. If it was in proxy mode, you should change the status at the same time that you specify the `mssHost`, as follows:

```
$upaccount = new Account(
   smtpAddress => $account->smtpAddress(),
   status => IM_ACSTATUS_ACTIVE,
   mssHost => "mss2");
$upaccount->Update();
```


UpdateAddr (*newSmtpAddress*)

Prerequisites

The field `smtpAddress` of the object must be defined and valid. The `newSmtpAddress` argument must be of the form “name@domain”, where domain is a valid domain in the InterMail directory, and the combination of name and domain has not already been used by another account or alias to an account.

Implementation

Calls `IM_UpdateAccountAddr()`.

Return Values

Returns “1” on success, “0” on failure.

Only the primary SMTP address of the account is altered by this function.

Delete ( )

Prerequisites

The field `smtpAddress` must be defined and valid.

Implementation

Calls `IM_DeleteAccount()`.

Return Values

Returns “1” on success, “0” on failure.

CreateAlias (*addr*)

Prerequisites

The field `smtpAddress` must be defined and valid. The `addr` argument must be a valid e-mail address, ending with an “@” followed by a domain name that is defined in the InterMail Directory.

Implementation

Calls `IM_CreateAlias();`.

Return Values

Returns “1” on success, “0” on failure.

DeleteAlias (*addr*)

Prerequisites

The `addr` argument must refer to an existing alias for this account.

Implementation

Calls `IM_DeleteAlias();`.

Return Values

Returns “1” on success, “0” on failure.

ReadAliases ( )

Prerequisites

The field `smtpAddress` must be defined and valid.

Implementation

Calls `IM_ReadAccountAliases()`.

Note: *If the Perl variable $IM_ReadAccountAliases_master is true, then IM_ReadAccountAliases_master() is called instead. This function gets its results from the master database, which avoids an apparent inconsistency that occurs when the Directory cache does not immediately register changes resulting from a call to CreateAlias(). Use this feature with caution, since it can impact the performance of the InterMail Directory.*

Return Values

On success returns the list of existing aliases for this account. If no aliases are found, or if `IM_ReadAccountAliases()` fails, an empty list is returned.

```
$addr = '▫▫▫▫▫▫▫▫▫▫▫▫'▫
$account->CreateAlias($addr);
print_status();
$IM_ReadAccountAliases_master = 1;   # (see special note above)
@read_aliases = $account->ReadAliases();
if (@read_aliases) {
  print "▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫"▫
} else {
  print "▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫"▫
  print_status();
}
Account::DeleteAlias($addr);
```

CreateForward (*addr*)

Prerequisites

The field `smtpAddress` must be defined and valid. The `addr` argument must be a valid e-mail address, ending with an “@” followed by a domain name. The domain need not exist in the InterMail Directory. At least one forwarding address must exist for this account.

Implementation

Calls `IM_CreateForward(addr);`.

Return Values

Returns “1” on success, “0” on failure.

DeleteForward (*addr*)

Prerequisites

The field `smtpAddress` must be defined and valid. The `addr` argument must refer to an existing forwarding address for this account.

Implementation

Calls `IM_DeleteForward(addr);`

Return Values

Returns “1” on success, “0” on failure.

ReadForwards ( )

Prerequisites

The field `smtpAddress` must be defined and valid.

Implementation

Calls `IM_ReadAccountForwards()`.

Return Values

On success returns the list of existing forward addresses for this account. If no such addresses exist, or if `IM_ReadAccountForwards()` fails, an empty list is returned.

```
$addr="bad@boy.com";
$account->CreateForward($addr);
print_status();
@read_forwards=$account->ReadForwards();
if (@read_forwards) {
  print "□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□"□
} else {
  print "□□□□□□□□□□□□□□□□□□□"□
  print_status();
}
$account->DeleteForward($addr);
```


Note: *Although the methods to manipulate account aliases and forwards look very similar, there is a significant difference between them:* `DeleteAlias()` *is a class method, whereas* `DeleteForward()` *is an instance method.*

EnableForwarding ( )

Prerequisites

The field `smtpAddress` must be defined and valid. At least one forwarding address must exist for this account.

Implementation

Calls `IM_EnableAccountForwards()`.

Return Values

Returns “1” on success, “0” on failure.

DisableForwarding ( )

Prerequisites

The field `smtpAddress` must be defined and valid.

Implementation

Calls `IM_DisableAccountForwards()`.

Return Values

Returns “1” on success, “0” on failure.

```
$account->EnableForwarding();
...
$account->DisableForwarding();
```

ReadAlias ( )

Prerequisites

The field `smtpAddress` must be defined and valid.

Implementation

Calls `IM_ReadAlias()`.

Return Values

Returns “1” on success, “0” on failure.

One field, `smtpAddress` is mandatory and it should be set to a meaningful SMTP address, which will be considered to be an alias by this method. If an account with this alias exists, the method will reset this field to the account's primary SMTP address. No other change to the account fields will be made -- no information will be passed back from the directory to the Perl object, except for the SMTP address, and no existing field will be cleared.

ReadPOP3 ( )

Prerequisites

The field `popAddress` must be defined and valid.

Implementation

Calls `IM_ReadPOP3()`.

Return Values

Returns “1” on success, “0” on failure.

One field, `popAddress` is mandatory and it should be set to a meaningful POP address. If an account with this POP address exists, the method will set the field `smtpAddress` to the account's primary SMTP address. No other change to the account fields will be made -- no information will be passed back from the directory to the Perl object except for the SMTP address and no existing field will be cleared.

```
# Alias and pop address for the account cat@pet.com
$alias=“kitty@pet.com”;
$pop=“katze”;

$account=new Account(popAddress => “hound”);
$account->ReadPOP3();
# The fields will now be:
#  smtpAddress=>“dog@pet.com”, popAddress=>“hound”

$account->smtpAddress(“kitty@pet.com”);

$account->ReadAlias();
# The fields will be now:
#   smtpAddress=>“cat@pet.com”, popAddress=>“katze”
```

List (*domain*, *from*, *till*, *max*)

Prerequisites

This function does not operate on an Account object, so it must be invoked as `Account::List(…)`. The domain and max arguments must be defined and max must be greater than zero.

Implementation

Calls `IM_ReadAccounts()`.

Return Values

Returns a list of SMTP addresses (“local” portion only) on success.

The SMTP addresses of accounts in domain that are lexically greater than from and lexically less than till are returned. If from is empty or undefined, there is no lower lexical bound on the match. If till is empty or undefined, there is no upper lexical bound on the match. Both from and till are exclusive bounds, meaning names that match either from or till exactly will not be returned. Only addresses that match the specified domain exactly will be returned. At most max names will be returned, and names will be returned in lexically increasing order.

It is possible to iterate through all the names in a domain using code similar to the following:

```
my $domain = '▫▫▫▫▫▫▫▫▫▫▫▫▫'▫
my $max = 100;
my $from = ''▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫
my $till = ''▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫
my @list;

while (@list = Account::List($domain, $from, $till, $max)) {
    my $name;
    foreach $name (@list) {
        print "▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫"▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫
    }
    $from = pop(@list); # Get the next set of up to $max names.
}
```

HashPassword (*password*, *hashType*)

Prerequisites

This function does not operate on an Account object, so it must be invoked as `Account::HashPassword(…)`. The `password` and `hashType` arguments must be defined, and `hashType` must correspond to one of the `IM_PWHASH*` constants.

Implementation

Calls `IM_HashPassword()`.

Return Values

Returns the specified password in hashed form according to `hashType`.

CheckPassword (*password*, *hashedPassword*, *hashType*)

Prerequisites

This function does not operate on an Account object, so it must be invoked as `Account::CheckPassword(…)`. The password, `hashedPassword`, and `hashType` arguments must be defined, and `hashType` must correspond to one of the `IM_PWHASH*` constants.

Implementation

Calls `IM_CheckPassword()`.

Return Values

Returns 1 if password matches `hashedPassword` when hashed according to `hashType`.

6.2.5 Mailbox

The class Mailbox has the following attributes (fields):

`account`	Reference to an Account object
`name`	String (of decimal digits). Field is the same as `emailIntID` except for administrative accounts
`host`	String
`msgsStored`	Integer (read-only)
`bytesStored`	Integer (read-only)
`inbox`	String

Some `Mailbox` methods use an associated Account object, which need not refer to an existing InterMail account, although most of the time it will. The Mailbox methods will use various fields of the Account object, depending on the operation, such as `emailIntID` and `mssHost`. The caller must ensure that the necessary fields of the Account are properly initialized, either by calling `Account::Read()`, or by setting the fields directly.

This may be necessary with administrative mailboxes; they do not have an InterMail directory account associated with them. Still, to access such a mailbox, one needs to specify some information that makes sense only in connection with an account.

Thus the `Mailbox` constructor looks different than other `SwCom::Mail` constructors: it takes a single argument (not in the call-by-pairs fashion), which should be an Account object. But the constructor is "intelligent" enough be able to process the call with the call-by-pairs syntax, where the only argument that matters is the account one.

The sequence of steps in manipulating a Mailbox object should be therefore as follows:

```
$account= ... # an account to tie the mailbox to.
$mailbox = new Mailbox($account);

# Or use the alternative syntax:
# $mailbox = new Mailbox(account => $account);
```

Make sure that the required fields of the $account are set prior to invoking functions on a Mailbox.

This will set the mailbox's name field to the "owning" account's `emailIntID`, and its host field to the account's `mssHost` value.

Note that the only "active" field of a Mailbox object is account—although all the rest can be set by a user, this will have no influence on any operation on the object. Instead, those fields will be set to the "real" values as a result of invoking a method of the class.

This how you could start working with a “real” account:

```
# Construct an account object without specifying
# any information but the "▫▫▫"▫
$account=new Account(smtpAddress=>“pig@farm.com”);

# Get the account data from InterMail database:
$account->Read(); # sets emailIntID, host, quotas etc

# Construct a mailbox object:
$mailbox=new Mailbox($account);

# Notice that $mailbox”s fields, other than “account”,
# are undefined yet:
print $mailbox->name(); # prints ""
print $mailbox->host(); # prints ""

if (want_create()) { # Create a "▫▫▫▫"▫▫▫▫▫▫▫▫▫
  $mailbox->Create();
} else { # will read an existing mailbox data
  $mailbox->Read();
}

# $mailbox”s fields have meaningful values now:
print $mailbox->name(); # prints "▫▫▫▫▫▫▫▫▫▫▫"
print $mailbox->host(); # prints "▫▫▫▫"
```

And this how you could deal with a “fake” account (also, we demonstrate an alternative, “shortcut” way to construct a `Mailbox`):

```
# Construct a mailbox object:
$mailbox=new Mailbox(new Account(
  # This is as if we had set the mailbox”s name and host directly:
  emailIntID=>“1234567890”, mssHost=>“lost”));

$mailbox->Read();

# $mailbox”s fields have meaningful values now:
print $mailbox->name(); # prints "▫▫▫▫▫▫▫▫▫▫▫"
print $mailbox->host(); # prints "▫▫▫▫"
```

The class `Mailbox` defines the following methods to access the InterMail facilities (see Chapter 5 for the requirements to the parameters and parameter combinations for various operations):

Create (*messageId*)

Prerequisites

The account field must reference a valid Account object, with `emailIntID` and `mssHost` set.

Implementation

Calls `IM_CreateMailbox()`.

Return Values

Returns “1” on success, “0” on failure.

Creates a mailbox and, optionally, inserts a message identified by `messageId`. If the second argument is absent or is the string ″none″, no message will be put in the newly created mailbox.

If the call is successful, fills in the fields of the object.

Read ( )

Prerequisites

The account field must reference a valid Account object, with the `emailIntID` and `mssHost` fields set.

Implementation

Calls `IM_ReadMailbox()`.

Return Values

Returns “1” on success, “0” on failure.

If successful, the method fills in all the fields of the object; the name field will be set to the “owning” account's `emailIntID` and the host field to the account's `mssHost` field.

Delete ( )

Prerequisites

The account field must reference a valid Account object, with the `emailIntID` and `mssHost` fields set.

Implementation

Calls `IM_DeleteMailbox()`.

Return Values

Returns “1” on success, “0” on failure.

Deletes the mailbox record in the InterMail database.

Example

```
# Construct an account object without specifying
# any information but the ″▫▫▫″▫
$account=new Account(smtpAddress=>“pig@farm.com”);
```



```
# Get the account data from InterMail database:
$account->Read(); # sets emailIntID. host, etc

# Construct a mailbox object:
$mailbox=new Mailbox($account);

# Create a "□□□□"□□□□□□□□□
$mailbox->Create();
print_status();

if ($mailbox->Read()) {
  print $mailbox->Dump();
}
print_status();

$mailbox->Delete()
print_status();
```

Notes

The need to create a “fake” account may arise for administrative mailboxes. These administrative accounts do not have an SMTP address, therefore, the only way to refer to them is by specifying the host where they (their mailboxes) reside, and their `emailIntID`.

The `emailIntID` of a fake account does not have to be a string of decimal digits, as required for real accounts. As a matter of fact, it will usually be something like ″admin″.

One will probably not use `SwCom::Mail` to create or delete administrative mailboxes -- there are special InterMail tools to do it. The only operations normally required for these special mailboxes would be adding and removing messages and folders.

An even more likely scenario of working with an administrative mailbox is to invoke the `Read()` method just to “synchronize” the Perl object with the InterMail database record. *After* invoking the `Read()` method, the Mailbox object can be used to access the mailbox's folders and messages.

A typical example of using a Mailbox object to access administrative data is as follows:

```
# Note that we do not use the account SMTP address, but
# rather its emailIntID, which is quite a different thing:
$mailbox = new Mailbox(
      new Account(emailIntID => "□□□□□"□□□□□□□□□□□□"□□□□□□□"□□□
$mailbox->Read();

# $mailbox”s fields have meaningful values now:
print $mailbox->name(); # prints "□□□□□"
print $mailbox->host(); # prints "□□□□□□□"
print $mailbox->inbox(); # prints "□□□□□"

# But the really interesting information is inside folders:
$folder = new Folder(
  mailbox   => $mailbox,
  pathname  => '□□□□□□□'□
);

# One can work with the folder now:
$folder->Read();
```

im_setMSSConnPoolSize (*size*)

Prerequisites

The `size` argument must be a number greater than or equal to zero.

Implementation

Calls `IM_SetMSSConnPoolSize()`.

Return Values

None.

This function sets the size of the MSS connection pool. The default size is 0, meaning no caching is done. If the connection pool size is set to “n”, then the last “n” connections made to an MSS will be cached.

When using a Mailbox, if the underlying C-API needs to make a connection to an MSS, it will check its connection pool first to see if there is an existing connection available. If one is available, it will use it, otherwise it will make a new one.

When an `IM_Mbox` is destroyed, the MSS connection associated with it (if any) is placed in the pool, unless that would cause the size of the pool to exceed the maximum specified by this function, in which case the connection is destroyed.

6.2.6 Folder

The class Folder has the following attributes (fields):

`mailbox`	Reference to a Mailbox object
`pathname`	String
`folders`	Reference to an array of strings (names of subfolders)
`messages`	Reference to an array of Message objects
`readOnly`	integer
`clearRecent`	integer
`UIDValidity`	integer

A new Folder object can be created in one of two ways:

1. By specifying the mailbox it belongs to and the full “path” to this folder:

```
$mailbox = ... # get a reference to an existing mailbox.
$folder = new Folder(
  mailbox   => $mailbox,
  pathname  => '▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫'
);
```

2. By specifying the folder's parent, be it a mailbox (for the top-level folders) or another folder, and a relative pathname from the parent:

```
$folder_f1 = new Folder(parent => $mailbox, name => '▫▫'▫▫
$folder_f1_f2 = new Folder(parent => $folder_1, name => '▫▫'▫▫
```


When the constructor is invoked this way, the mailbox and pathname fields are built on the fly, using the parent's information.

The current values of the fields can be printed on the standard output stream as follows:

```
print $folder->Dump();
```

The class Folder defines the following methods to access the InterMail facilities (see Chapter 5 for the requirements to the parameters and parameter combinations for various operations):

Create ( )

Prerequisites

The fields mailbox and pathname must be defined and valid.

Implementation

Calls `IM_CreateFolder()`.

Return Values

Returns "1" on success, "0" on failure.

Read ( )

Prerequisites

The fields mailbox and pathname must be defined and valid.

Implementation

Calls `IM_ReadFolder()`.

Return Values

Returns "1" on success, "0" on failure.

On successful return from this function:

1. The field folders will be a reference to an array of strings—names of this folder's subfolders.
2. The field messages will be a reference to an array of Message objects. The information contained in the elements of this array will be incomplete—only the fields `msgRef`, `msgId` and size will be filled in. Use the `Message::Read()` function to retrieve all the data for a specific message.

Rename (*new_name*)

Prerequisites

The fields `mailbox` and `pathname` must be defined and valid.

Implementation

Calls `IM_RenameFolder()`.

Return Values

Returns “1” on success, “0” on failure.

Renames the folder to the `new_name`.

Delete ( )

Prer**e**quisites

The fields `mailbox` and `pathname` must be defined and valid.

Implementation

Calls `IM_RenameFolder()`.

Return Values

Returns “1” on success, “0” on failure.

DeleteMessages (*msgs, ...*)

Prerequisites

The fields `mailbox` and `pathname` must be defined and valid.

Implementation

Calls `IM_DeleteFolderMsgs()`.

Return Values

Returns “1” on success, “0” on failure.

Deletes one or more messages from a folder. One or more Message objects (from the Folder’s messages array) should be passed as arguments. It is also possible to pass one or more arrays of Messages instead of individual Messages. This function is more efficient than deleting the messages one by one.

ScanMessages (*msgs*, ...)

Prerequisites

The fields `mailbox` and `pathname` must be defined and valid.

Implementation

Calls `IM_ScanFolderMsgs()`.

Return Values

Returns “1” on success, “0” on failure.

Scans the headers of one or more messages in a folder so that future calls to `ReadHeader()` will be faster. One or more Messages (from the Folder’s messages array) should be passed as arguments. It is also possible to pass one or more arrays of Messages’s instead of individual Messages’s. This function is useful when the caller needs to look at the headers of a large number of messages; it's more efficient to load the headers with this one call than to have them loaded one by one.

Example

```
$mailbox = ... # get a reference to an existing mailbox.
$folder_f1 = new Folder(parent => $mailbox, name => '□□'□□
$folder_f1_f2 = new Folder(parent => $folder_f1, name => '□□'□□

$folder_f1->Create();
print_status();
$folder_f1_f2->Create()
print_status();

$folder_f1->Read();
print_status();
print $folder_f1->Dump();

print "□□□□□□□□□□□□□□□□□□□□□□"□
foreach $foldername (@{$folder_f1->folders()}) {
  print "□□□□□□□□□□□□□□"□
}

print "□□□□□□□□□□□□□□□□□□□□"□
foreach $msg (@{$folder_f1->messages()}) {
  print $msg->Dump(), "□□"□

  # or do it field by field:
  # my $id = $msg->msgId();
  # my $size = $msg->size();
  # my $msgRef = $msg->msgRef();
  # print "□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□"□
}

$folder_f1_f2->Rename('□□□□□□□'□□
print_status();
print $folder_f1_f2->Dump();
$folder_f1_f2->Delete();
```

MoveMessages (*dstFolder*, *msgRefs*, …)

Prerequisites

The fields `mailbox` and `pathname` must be defined and valid. The `dstFolder` argument must be a reference to another Folder in the same Mailbox as the Folder on which this function is invoked.

Implementation

Calls `IM_MoveMsgs()`.

Return Values

Returns “1” on success, “0” on failure.

Moves messages from a folder to another folder. One or more msgRef’s (from the Message objects in the Folder’s messages array) should be passed as arguments. It is also possible to pass one or more arrays of msgRef’s instead of individual msgRef’s.

CopyMessages (*dstFolder*, *msgRefs*, …)

Prerequisites

The fields `mailbox` and `pathname` must be defined and valid. The `dstFolder` argument must be a reference to another Folder in the same Mailbox as the Folder on which this function is invoked.

Implementation

Calls `IM_CopyMsgs()`.

Return Values

Returns “1” on success, “0” on failure.

Copies messages from a folder to another folder. One or more msgRef’s (from the Message objects in the Folder’s messages array) should be passed as arguments. It is also possible to pass one or more arrays of msgRef’s instead of individual msgRef’s.

6.2.7 Message

The class Message has the following attributes (fields):

`mailbox`	Reference to a Mailbox object
`folder`	Reference to a Folder object
`msgRef`	Integer
`msgId`	String
`size`	Integer
`seen`	Integer
`arrivalTime`	Integer
`arrivalTimeString`	String
`UID`	Integer
`flags`	String

The Message constructor requires a reference to the Folder object that contains the message, either as a single argument:

```
$folder = ... # get a reference to an existing folder.
$message = new Message($folder);
```

or through the call-by-pairs syntax:

```
$folder = ... # get a reference to an existing folder.
$message = new Message(folder => $folder);
```

The field mailbox will be set by the constructor to point to the Mailbox where the folder resides, and should not be set directly by the client code.

The class Message defines the following methods to access the InterMail facilities (see Chapter 5 for the requirements to the parameters and parameter combinations for various operations):

Create (*from*, *text*)

Prerequisites

The fields `mailbox` and `folder` must be defined and valid. The “from” and “text” arguments must be specified, and text must be formatted according to RFC822.

Implementation

Calls `IM_CreateMsg()`.

Return Values

Returns “1” on success, “0” on failure.

If the call is successful fills in the field `msgRef`.

Read ( )

Prerequisites

The fields `mailbox`, `folder` and `msgRef` must be defined and valid.

Implementation

Calls `IM_ReadMsg()`.

Return Values

Returns “1” on success, “0” on failure.

Fills in the values of all fields of the object by retrieving the information from the InterMail database.

ReadHeader (*field*)

Prerequisites

The fields `mailbox`, `folder` and `msgRef` must be defined and valid.

Implementation

Calls `IM_ReadMsgHeader()`.

Return Values

If the call is successful (in particular, the requested field should exist in this message), the body of the field is returned, as a string. Otherwise, the undef value is returned.

ReadContent (*offset*, *size*)

Prerequisites

The fields `mailbox`, `folder` and `msgRef` must be defined and valid.

Implementation

Calls `IM_ReadMsgBody()`.

Return Values

If the call is successful, the requested chunk of the text of the message (which includes “headers” is returned, as a string. Otherwise, the undef value is returned.

Delete ( )

Prerequisites

The fields `mailbox`, `folder` and `msgRef` must be defined and valid.

Implementation

Calls `IM_DeleteMsg()`.

Return Values

Returns “1” on success, “0” on failure.

UpdateFlags (*flags*)

Prerequisites

The fields `mailbox`, `folder` and `msgRef` must be defined and valid.

Implementation

Calls `IM_UpdateMsgFlags()`.

Return Values

Returns “1” on success, “0” on failure.

Updates the message flags of a message. The flags argument is a string of characters which specify which flags to change. Each character position represents a flag; a 'T' character sets the flag, a 'F' character clears it, and a '-' character leaves it alone. This string must be no longer than 6 characters; if may be shorter if only the first few flags need to be changed.

Example

```
$folder = ... # get a reference to an existing folder.
$message = new Message(folder => $folder);
$from = “cat@house.com”;
$text = <<'□□□□□□□□□□□□'
From: cat
To: mouse

Care to join me for dinner?
EndOfMessage

$folder->Read();
print $folder->Dump();  # lists messages (list_1)
$message->Create($from, $text);
$folder->Read();
print $folder->Dump();  # lists messages (list_2)

# All operations on messages, except for “Create()”,
$message->msgRef(123);
$message->Read();
print $message->Dump();

$headertext = $message->ReadHeader('□□□□'□□
print "□□□□□□□"□□□□□"□□□"□□□□□□□□□□□□"□□"□□□□□□□□□□□□□□□□

$bodytext = $message->ReadContent(0, 64);
print "□□□□□□□"□□□□□□□□□□"□□"□□□□□□□□□□□□□
$message->Delete();
```

6.2.8 Reply

The class Reply has the following attributes (fields):

`account`	Reference to an Account object
`mode`	Reference to a Folder object
`msgRef`	`enum IM_ReplyType` (see C API documentation)
`host`	String
`text`	String
`type`	Reserved for future use

A new Reply object can be created this way:

```
$account = ... # get a reference to an existing account.
$reply = new Reply(
  account => $account,
  mode    => IM_REPLY_VACATION,
  host    => '▫▫▫▫▫▫▫▫'▫
  text    => '▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫'
);
```

Create ( )

Prerequisites

All the fields must be defined and valid. The account field must reference a Perl object with a correctly set `emailIntID` field.

Implementation

Calls `IM_CreateReply()`.

Return Values

Returns “1” on success, “0” on failure.

Read ( )

Prerequisites

The field account must be defined and valid. The account field must reference a Perl object with a correctly set `emailIntID` field.

Implementation

Calls `IM_ReadReply()`.

Return Values

Returns “1” on success, “0” on failure.

Update ( )

Prerequisites

The field account must be defined and valid. The account field must reference a Perl object with a correctly set `emailIntID` field.

Implementation

Calls `IM_UpdateReply()`.

Return Values

Returns “1” on success, “0” on failure.

For those fields of the Perl object that are defined, the values will be used to rewrite the values of the record in the InterMail database.

No modification to the fields of the object is done in the result of calling this method. Use `Read()` to update the fields after calling `Update()`.

Delete ( )

Prerequisites

The field `account` must be defined and valid.

Implementation

Calls `IM_DeleteReply()`.

Return Values

Returns “1” on success, “0” on failure.

Example

```
$account = ... # get a reference to an existing account.
$reply = new Reply(
  account => $account,
  mode    => IM_REPLY_VACATION,
  host    => '▫▫▫▫▫▫▫'▫
  text    => '▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫'
);

print '▫▫▫▫▫▫▫▫▫▫▫▫'▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫"▫▫"▫
print $reply->Dump();
$reply->Create();
print_status();
if ($reply->Read()) {
  print $reply->Dump();
}
$reply->text('▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫'▫▫
$reply->Update();
print_status();
$reply->Delete();
```


Notes

There is an interesting difference between the class Reply and the rest of `SwCom::Mail` classes. For the latter ones, an InterMail record does not exist until one invokes the method `Create()` on an object of the class. The Reply class is different because it is essentially an attribute of an Account, and all accounts have an auto-reply state, even if no Reply object has ever been created for them. An account's auto-reply type defaults to `IM_REPLY_NONE`, therefore `Reply::Read()` will always succeed for Reply objects that refer to existing InterMail accounts (except for internal InterMail errors). Be careful not to invoke the method `Read()` on a Reply object if you intend to call `Create()` afterwards: `Read()` will change the Perl object even when "there was no auto-reply" for the account.

6.2.9 ConfigItem

The class ConfigItem has the following attributes (fields):

`name`	String
`host`	String
`prog`	String
`defvalue`	String
`value`	String

A new ConfigItem object can be created this way:

```
$config = new ConfigItem(
  name => 'netTimeout',
  host => 'calgary',
  prog => 'mta',
  defvalue => '␣␣'
);
```

Read ( )

Prerequisites

The name field must be set.

Implementation

Calls `IM_ReadConfig()`.

Return Values

Returns "1" on success, "0" on failure.

6.2.10 MimeInfo

The class MimeInfo has the following attributes (fields):

`isMultiPart`	Integer
`isMessagePart`	Integer
`childCount`	Integer
`headerOffset`	Integer
`bodyOffset`	Integer
`headerLength`	Integer
`totalSize`	Integer
`numLines`	Integer
`contentTransferEncoding`	String
`mainContentType`	String
`subContentType`	String
`contentID`	String
`contentDescription`	String
`boundary`	String
`contentMD5`	String
`contentDisposition`	String
`contentLanguage`	Array of strings
`parameters`	Array of strings

ReadMimeInfo (*mime*, [*parent*, [*index*]])

Prerequisites

Note: *This is a member function of the Message class, not the MimeInfo class. See sample code below.*

The `mime` argument must be defined and valid. The `parent` argument is optional; if present, it must be defined and valid and point to a `MimeInfo` object describing a multipart or message/rfc822 message. If it points to a multipart, the index argument must be present and must be defined and valid.

Implementation

Calls `IM_ReadMimeInfo()`.

Return Values

Returns “1” on success, “0” on failure.

The following is a sample piece of code which dumps a message's MIME tree:

```
# Load the message
$message = new Message(folder=>..., msgref=>...);
$message->Read();
$mime = new MimeInfo;
$message->ReadMimeInfo($mime);
dumpit('toplevel', $message, $mime);

sub dumpit
{
   my ($id, $message, $mime) = @_;
   print "================================\n";
   print "$id\n";
   print $mime->Dump(), "\n";
   if ($mime->{isMessagePart}) {
       $cmime = new MimeInfo;
       $message->ReadMimeInfo($cmime, $mime);
       dumpit($id . '.body', $message, $cmime);
   } elsif ($mime->{isMultiPart}) {
       $cmime = new MimeInfo;
       my $i;
       for ($i = 0; $i != $mime->{childCount}; $i++) {
           $message->ReadMimeInfo($cmime, $mime, $i);
           dumpit($id . '.' . $i, $message, $cmime);
       }
   }
}
```

6.2.11 LogMsg

Objects of type LogMsg have the following attributes:

`severity`	`enum IM_Severity`
`msgId`	Integer referring to a valid InterMail error number (e.g. 0x490013)
`type`	String referring to a valid InterMail error mnemonic, e.g. 'NioConnTimeout'
`args`	Reference to an array of strings
`text`	String
`logName`	String

A new LogMsg object can be created this way:

```
$logmsg = new LogMsg(
  severity => IM_SEVERITY_ERROR,
  type => '□□□□□□□□□□□□□□'□
  args => [ $filename ]
);
```


CreateFile ( )

Prerequisites

`logName` must be set to a valid filename containing no "/" characters. Other attributes are ignored.

Implementation

Calls `IM_CreateLogFile()`.

Return Values

Returns "1" on success, "0" on failure.

Write ( )

Prerequisites

Either `msgID` or type must be set, and severity must be set. The `args` and `logName` attributes are used if set.

Implementation

Calls `IM_WriteLogMsg()`.

Return Values

Returns "1" on success, "0" on failure.

This function can be used as a member function or a class function. When invoked as a member function on a LogMsg object, it uses the attributes stored in the object, for example:

```
$logmsg = new LogMsg(severity => IM_SEVERITY_ERROR,
                     type => '▫▫▫▫▫▫▫▫▫▫▫▫▫▫'▫
                     args => [ $filename ]);
$logmsg->Write();
```

Often there is no need for the LogMsg object beyond the creation of the log file entry, so we support a shortcut that accomplishes the same action as the previous code in a single statement:

```
LogMsg::Write(severity => IM_SEVERITY_ERROR,
              type => '▫▫▫▫▫▫▫▫▫▫▫▫▫▫'▫
              args => [ $filename ]);
```

ReadText ( )

Prerequisites

Either `msgID` or type must be set, and severity must be set. The `args` attribute is used if set.

Implementation

Calls `IM_ReadLogMsgText()`.

Return Values

Returns "1" on success, "0" on failure. The text attribute is filled in with the text that would be used in a log file entry (without the leading timestamp and process information).

6.2.12 LogContext

Objects of type LogContext have the following attributes:

`mailUser`	String: an SMTP address or POP login name
`mailHost`	String: logical hostname for an MSS machine
`mailbox`	String of decimal digits
`from`	String: an SMTP address
`folder`	String: pathname to a folder in a mailbox
`msgId`	String: contents of the Message-ID header in an RFC822 message
`origMsgId`	String: contents of the Message-ID header in an RFC822 message before it was modified by the MSS
`size`	Integer: size of an RFC822 message in bytes
`cmd`	String: identifies the current operation
`destHost`	String: logical hostname for an MSS machine
`fromHost`	String: logical hostname for an MSS machine
`port`	Integer: port number used to connect to an MSS

A new LogContext object can be created this way:

```
$logmsg = new LogContext(
  mailUser => $userName,
  folder => $folderName,
  cmd => $commandName
);
```

Update ( )

Prerequisites

None.

Implementation

Calls `IM_UpdateLogContext()`.

Return Values

Returns “1” on success, “0” on failure.

The current values in the LogContext object will be saved in the global logging context for use when generating log file entries. Missing or empty attributes will cause those values to be omitted from log file entries.

Read ( )

Prerequisites

None.

Implementation

Calls `IM_ReadLogContext()`.

Return Values

Returns “1” on success, “0” on failure.

The values from the global logging context will be stored into the LogContext object, replacing any values that were previously stored there.

7

InterManager Perl API

This chapter describes the InterManager Perl API that is used to support the InterManager Web interface as well as support the batch-load utility, which provisions InterManager data. This chapter includes the following information:

- An overview of the semantics of the API libraries.
- Descriptions of the data types used in the API.
- Listing of individual API functions.
- Description of compiling and linking to the API libraries.

7.1 Introduction

The InterManager Perl API provides an object-oriented interface to the data that can be manipulated by Software.com's InterManager product. This includes LDAP and InterMail Directory information about organizations, people, and e-mail accounts.

Note: *To maintain compatibility with the InterMail Perl API, the original attribute names set upon objects are recognized (eg.* `smtpAddress` *for the Account object and so forth). Note that the object* `Read()` *methods will only return InterManager style attribute names. (These are currently all lower cased.)*

7.2 Overview of Classes

This table should give you a quick overview of the classes used by the library. Indentation shows class inheritance. Those marked with '*' are internal to the API implementation.

`SwCom::Error`	Access to errors
`SwCom::WebSession`	A WWW session
`SwCom::IMgr::Session`	An LDAP session
`SwCom::IMgr::Entry`	Base class for LDAP entries
`SwCom::IMgr::Root`	LDAP Root
`SwCom::IMgr::Org`	LDAP Organization
`SwCom::IMGR::OrgUnit`	LDAP Organization Unit
`SwCom::IMgr::Person`	LDAP Person
`SwCom::IMgr::MailGroup`	Mail Group
`SwCom::IMgr::MailTemplate`	Mail Template

`SwCom::IMgr::MailCOS`	Class Of Service
`SwCom::IMgr::Provider`	Mail Service Provider
`SwCom::IMgr::AdminGroup*`	Administrative Group
`SwCom::IMgr::Search*`	LDAP entry search engine
`SwCom::Mail::Domain`	Mail API Domain Name object
`SwCom::IMgr::Domain`	InterManager Domain Name object

7.3 InterManager Perl API Classes

The following sections describe the individual API classes contained in the InterManager Perl API.

7.3.1 SwCom::Error

Description

Provides access to InterManager errors. The `SwCom::Error` class contains information about why an operation was unsuccessful.

Synopsis

```
my $err = new SwCom::Error($type, $number, $string);
print $err->Type();
print $err->String();
die if ($err->Number() > 5);
my $err2 = new_from_error SwCom::Error($err);
```

Methods

- **new SwCom::Error(*type, number, string*)**
  Constructor for the Error object. Initializes the Error object with the supplied arguments. Returns a reference to a `SwCom::Error` object.
- **new_from_error SwCom::Error(*Error*)**
  Constructor for the Error object. Returns a reference to a `SwCom::Error` object. Constructs a new Error object from the given Error object.
- **Type()**
  This method returns a string describing the source of the error, thus allowing for overlapping error numbers. Valid types are: 'LDAP', 'IM'.
- **String()**
  This method returns a string describing the error.
- **Number()**
  This method returns an error id number.

7.3.2 SwCom::WebSession

Description

Manages WWW administration sessions. The `SwCom::WebSession` class provides functions for managing WWW administrative sessions. After a session is created, it is identified by a unique token that gets passed from HTTP request to HTTP request. These tokens are time sensitive and can either expire, or be explicitly killed with the `Delete()` method.

Synopsis

```
$ws = new SwCom::WebSession;
die unless $ws->Create($ipaddr, $login, $passwd, $md5passwd);
die unless $ws->Validate($ipaddr, $token);
$token = $ws->GetToken();
if ($ws->Delete()) { print "Log out successful"; }
$ldap_session = $ws->GetLDAPSession();
$err = $ws->GetError();
$ws->SetError($err);
```

Methods

- **new SwCom::WebSession()**
  `SwCom::WebSession` constructor. Returns a reference to a new WebSession object.
- **Create($ipaddr, $login, $password, $md5passwd)**
  Method used to authenticate a new user and activate a new Web Session for the user. It sets up an `SwCom::IMgr::Session` object representing the active Web Session. If `$password` is defined, the user is authenticated by comparing the plain text password stored in their account. If the `$md5passwd` is defined, the user is authenticated by comparing the hashed password value read from their account. Both may be supplied.

  Returns 1 if successful, 0 otherwise.
- **Update()**
  Modifies the values persisted for the active `SwCom::WebSession` object. The values are stored in a hash upon the object. The valid keys are token, `ipaddr`, `login`, `passwd`, `encpasswd`, `smtpAddress`, `popAddress`, `mssHost`, `emailIntID`, and services. If the key 'token' is undefined, a new token will be calculated and stored for the active session. The service is a hash of key/value pairs, where each key is a valid Class of Service attribute.

  Returns 1 if successful.
- **Validate($ipaddr, $token)**
  This method verifies that a given token exists and has not timed out. It sets up an `SwCom::IMgr::WebSession` object for the active session identified by the token.

  Returns a new `SwCom::IMgr::WebSession` object if successful, 0 otherwise.
- **GetToken()**
  Returns a unique ASCII string that identifies the WebSession. This string must be passed in every HTTP request of a web session.

- **Delete()**
  This method is used to explicitly end a running WebSession. The token identified with the Web session is invalidated. The LDAP session is closed (if necessary) by "unbinding" the user. This function is useful for implementing 'logout' buttons.

  Returns 1 on success, 0 otherwise.

- **GetLDAPSession()**
  This method returns a reference to a `SwCom::IMgr::Session` object. This reference can be used to construct objects derived from `SwCom::IMgr::Entry`.

- **GetError()**
  Returns a reference to a `SwCom::Error` object that describes the reason for most recent failed operation. Not reset on success.

- **SetError($err) | SetError (*type, number, string*)**
  Sets the Error object within the WebSession object. Input could either be an Error object or error type, number and string.

7.3.3 SwCom::IMgr::Session

Description

Manages LDAP sessions. This class provides a few simple methods for managing LDAP sessions within the InterManager API. Normally, the WebSession class will manage the creation of a `SwCom::IMgr::Session` object.

Synopsis

```
$ldap_session = new SwCom::IMgr::Session($ldap_host, $login,
$passwd);
$ld = $ldap_session->Connect();
$ld = $ldap_session->Bind();
$ld = $ldap_session->RootBind();
$ldap_session->Unbind();
$ldap_session->Log($text);
$ldap_session->SetErrorDst($objref);
$ldap_session->SaveError(errtype, errnum, errstr);
$ldap_session->SaveLDAPError($err);
my $err_num = $ldap_session->GetLDAPError();
my $err  = $ldap_session->GetError();
```

Methods

- **new SwCom::IMgr::Session(*ldap_host, login, passwd*)**
  Returns a reference to a new `SwCom::IMgr::Session` object. The user of the session is identified via the `login` and `passwd` parameters. These are identical to those supplied in `SwCom::WebSession::Create()`. See `SwCom::WebSession::GetLDAPSession()`.

- **Connect()**
  Opens a connection to the LDAP server if one is not already open for the current Web session. Saves the LDAP session descriptor in the current object. The LDAP session

descriptor is often used when constructing InterManager objects that are written to the DIT. This function is called automatically by `Bind()`.

Returns an LDAP session descriptor on success, otherwise undef.

- **Bind()**
  Binds to the LDAP server using the login and password supplied to the constructor. Returns the LDAP session descriptor on success, otherwise undef.
- **RootBind()**
  Binds to the LDAP server as "Root". Returns the LDAP session descriptor on success, undef otherwise.
- **Unbind()**
  Unbinds from the LDAP host. Returns 1 on success, 0 otherwise. See `SwCom::WebSession::Delete()`.
- **SetErrorDst (*ErrorDst*)**
  Stores a reference to an object, whereto all errors are to be sent. `ErrorDst` is a reference to the recipient object and it must implement the `GetError()` `SetError()` methods. (See `WebSession::GetError()` and `WebSession::SetError()`). Prior to storing the new object reference, the `SetError()` method is called upon it to verify it works.

  Returns the old Error object.

  Note: *The destination object is initially the `SwCom::IMgr::WebSession` object that created the current LDAP session object. The WebSession will in turn store errors it receives in a `SwCom::Error` object.*

- **SaveLDAPError(*err*)**
  Saves the supplied LDAP error code in the current object and sets the error type to 'LDAP'. Sets the error in the error destination object if there is one. See `SwCom::IMgr::Session::SetErrorDst()`. Calls SwCom::IMgr::Session::Log() to print the error string to STDERR. Does not return anything.
- **GetLDAPError()**
  Returns the error number stored in the Session object.

  Note: *It does not matter if the error code is and LDAP error or not.*

- **SaveError($err) | SaveError(*type, number, string*)**
  Saves the supplied error within the current Session object. Sets the error in the error destination object if there is one. See `SwCom::IMgr::Session::SetErrorDst()`. Calls `SwCom::IMgr::Session::Log()` to print the error string to STDERR. Does not return anything.
- **GetError()**
  Returns the Error object from the Session object.
- **Log(text)**
  Prints the text message along with the localtime to STDERR.

7.3.4 SwCom::IMgr::Entry

Description

Generic class for LDAP entries. A foundation class implementing common methods and utilities used by most objects that are written as LDAP entries. Objects of type `SwCom::IMgr::Entry` should not normally be created, except during the constructor of a derived class.

In most cases, the behavior of a particular LDAP entry is determined by its attributes. The InterManager API provides access to attributes, and it is possible to do quite a lot with the generic Create, Read, Add, SetAttributes, Remove, Update, and Delete operations. There are very few calls in this API designed to access or control a specific attribute.

Detailed information about LDAP attributes entries and hierarchies are present in the DIT documentation.

Attempts to use entries or attributes in a manner that contradicts the InterManager DIT, or to access entries or attributes without the necessary access privileges, will be rejected by the API.

Synopsis

```
my $entry = new SwCom::IMgr::Entry($class, $imgrSession, $dn);
my $parent = $entry->GetParent();
my $dn = $entry->GetDN();
my $parent_dn = $entry->GetParentDN ();
my $org = $entry->Org();
return undef unless $entry->Create({attr1 => [$a1v1, a1v2, ...],
...});
print "Entry Deleted" if ($entry->Delete());
print "Entry updated" if ($entry->Update(\%attrs));
my %attrs = %{$entry->Read( [attr1, attr2, ...] )};
return undef unless $entry->Rename($new_rdn, $delete_old_rdn);
return undef unless $entry->ChangeDN($dn);

my $domain = SwCom::IMgr::Entry::DNToDomain($dn);
my $dn = SwCom::IMgr::Entry::DomainToDN:: ($domain_name);
my $domain_dn = SwCom::IMgr::Entry::DNToDomainDN($dn);

return $this->_LDAPError($ld);
```


Methods

- **new SwCom::IMgr::Entry(*IMgrSession*, *dn*);**
  Constructor for SwCom::IMgr::Entry. Returns a reference to a SwCom::IMgr::Entry object. This method is not normally used outside the InterManager API.
- **GetDN()**
  Returns the distinguished name (DN) of the LDAP entry. Call this routine using its fully qualified name.
- **GetParentDN()**
  Returns DN of the parent LDAPentry. Call this routine using its fully qualified name.
- **GetParent()**
  Constructs parent's entry from the parent's DN. Returns the parent `SwCom::IMgr::Entry` object.
- **DNToDomain(*dn*)**
  Extracts the domain name from the DN. Presumes the dn is of an entry residing under an Organization within the DIT. The primary domain name is used to construct the Organization's DN. It is that primary domain name that is extracted by this routine. Returns a domain name. Do not pass a `$self pointer` to this routine. Call this routine using its fully qualified name. For example,. if DN is:

  `uid=abc@xyz.com,ou=engg,dc=venus,dc=software,dc=com`

  it returns "venus.software.com".
- **DomainToDN(*domain_name*)**
  Constructs the DN from the complete domain name. Do not pass a `$self` pointer to this routine. Call this routine using its fully qualified name. Returns a DN. For example, if domain_name is:

  `venus.software.com,"`

  it returns "dc=venus,dc=software,dc=com."
- **DNToDomainDN(*dn*)**
  Extracts the DN that comprises only "dc=" relative DN (RDN) components. Do not pass a `$self` pointer to this routine. Call this routine using its fully qualified name. Returns a DN. For example, if DN is:

  `uid=abc@xyz.com,ou=engg,dc=venus,dc=software,dc=com`

  it returns "dc=venus,dc=software,dc=com".
- **Org()**
  For Entries residing within an Organization, returns a reference to their `SwCom::IMgr::Org` object.
- **Create(attr1 => [a1v1, a1v2, ...], ...)**
  Creates the LDAP entry in the directory. If exactly one argument follows `$self`, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in. Returns 1 on success, undef otherwise.
- **Delete()**
  Deletes the LDAP entry from the directory. Returns 1 on success, undef otherwise.

- **Add(attr1 => [a1v1, a1v2, ...], …)**
  Adds the specified attributes to the LDAP entry. If exactly one argument follows `$self`, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in. Returns 1 on success, undef otherwise.

- **Remove(attr1 => [a1v1, a1v2, ...], …)**
  Removes the specified attributes from the entry. If exactly one argument follows `$self`, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in. Returns 1 on success, undef otherwise.

- **SetAttributes(attr1 => [a1v1, a1v2, ...], …)**
  Sets the attributes (keys of hash) to the specified values in the entry. If exactly one argument follows `$self`, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in. Returns 1 on success, undef otherwise.

- **Update(attr1 => {op => [a1v1, a1v2, ...]]}, ...)**
  Updates the LDAP entry using the passed attr/value-array pairs. If exactly one argument follows self, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in. The op key in the second hash above may be: ‘a’, 'd', or ‘r’, representing “add”, “delete”, or “replace” respectively. This method is not intended to be called. Use instead the Add, Remove, and SetAttributes methods. They are implemented in terms of the Update method. Returns 1 on success, undef otherwise.

  To replace all attribute values, use the following form:

  ```
  {attr1 => {’r’ => [a1v1, a1v2, ...]}, ...}
  ```

  To add new attribute values:

  ```
  {attr1 => {’a’ => [a1v1, a1v2, ...]}, ...}
  ```

  To remove select attribute values:

  ```
  {attr1 => {’d’ => [a1v1, a1v2, ...]}, ...}
  ```

  Example set of mixed operations:

  ```
  { oldbooks  => {’d’ => [’Horton Hears a Who’, ’The Bird Snatcher’]},
     newbooks  => {’a’ => [’Green Eggs and Ham’]},
     library   => {’r’ => [’0002’, ’0003’, ’0004’]},
  }
  ```

- **Read (*attr1, attr2,* ...)**
  Reads the LDAP entry returning its attr/value-array pairs in a hash reference. The attribute list is optional. If used, only specified attributes are retrieved from the entry. Otherwise, all attributes are retrieved.

> ***Note:*** *As attributes may be multi-valued, the returned hash stores the attribute values as arrays (even single valued attributes). Any attributes read that don't yet have values on the LDAP entry are a valid key in the returned hash, mapped to the undef value. In case of error, it returns undef.*

- **Rename (*new_rdn*, delete_old_rdn=1)**
  Functionally equivalent to the `ldap_modrdn()` function, this routine alters the most specific component of the object's DN. The `$delete_old_rdn` flag is optional. If it evaluates to TRUE, the old RDN attribute value is deleted. If it is not specified, it defaults to TRUE. Returns 1 on success, undef otherwise.
- **ChangeDN ($new_dn)**
  It is functionally equivalent to the `ldap_rename()` function. The entry's DN is changed to the new DN. The entry must be a leaf in the DIT. Returns 1 on success, undef otherwise.

7.3.5 SwCom::IMgr::Root

Description

The LDAP root. A way to specify the LDAP root (DN = "") for functions that accept LDAP entry arguments. There are no valid operations on the LDAP root itself.

> ***Note:*** *As all methods that take a DN as parameter allow root to be specified as "", there has been no use of this object.)*

Synopsis

```
$ldap_top = new SwCom::IMgr::Root(IMgrSession);
```

Methods

- **new SwCom::IMgr::Root(IMgrSession);**
  Constructor for IMgr::Root. Returns a reference to a SwCom::IMgr::Root object. See also `SwCom::IMgr::Entry`.

7.3.6 SwCom::IMgr::Org

Description

LDAP Organizations. This object corresponds to an LDAP entry that represents the top level of an organization.

Synopsis

```
$org = new SwCom::IMgr::Org(IMgrSession, $o_name, $domain_name);
$org = new_from_dn SwCom::IMgr::Org(IMgrSession, $dn);

$name = $org->Name();
$domain_name = $org-> PrimaryDomainName ();

$org->SetProperties(attr1, …);

$org->Create();
$org->Delete();

@mailGroup_list = $org->GetMailGroups(cos_name1, …);

$org_unit = $org->OrgUnit($ou_name1, $ou_name2);
$ou_list = $org->GetOrgUnits($flag = 0);

@ou_names = $org->GetOrgUnitNames($flag = 0);
%ou_names = $org->GetOrgUnits($flag = 0);

my %attrs = %{ $org->Read('attr', …) };

$org->Add('attr' => val, ...);
$org->SetAttributes('attr' => val, ...);
$org->Remove('attr' => val, ...);

$org->AddAdmins($role, IMgrPerson ...);
$org->RemoveAdmins($role, IMgrPerson ...);
@adm_list = $org->GetAdmins($role ...);
```

Methods not called outside of InterManager API.

```
$org->IsSite();
$org->IsCustomer();
$dn = $org->MailGroupParentDN();
$dn = $org-> MailTemplateParentDN ();
```

Methods

- **new SwCom::IMgr::Org(IMgrSession, o_name, domain_name)**
  Constructor for the IMgr::Org. The ‘domain_name’ argument is optional. If `domain_name` is absent it searches for the `o_name` in the DIT and gets the DN. Otherwise it constructs the DN from `o_name` and `domain_name`. The caller must pass both `o_name` and `domain_name` if he/she will be calling Create method next. Returns a reference to a `SwCom::IMgr::Org` object. Returns undef on error.
- **new_from_dn SwCom::IMgr::Org(IMgrSession, $dn)**
  Constructor for the `IMgr::Org`. Returns a reference to a `SwCom::IMgr::Org` object. Caller knows DN. Not to be used if Create will be called next!
- **Name()**
  Returns the organization’s name. This could also be read via `$org->Read('o')`, but in general it is faster to access it via this method. Returns undef in case of error.

- **MailTemplateParentDN()**
  Returns the DN of the parent object under which default mail user template entries are located in an organization.

- **PrimaryDomainName()**
  Returns the primary domain name of the organization.

- **SetProperties (attr => val, ...)**
  Sets the properties to the corresponding values. If exactly one argument follows $self, it assumes that the argument is a reference to a hash of property name/value pairs. Otherwise, it assumes that property name/value pairs were passed in. Valid properties are CreateNonExistentDomains. Set this property to 1 before calling Create. If the domains allowed for an organization (including the primary domain) should be created if they do not already exist. Set this property to 0 if the domains are expected to exist prior to calling Create. It always returns 1.

- **Create(attr => val,…)**
  Creates the customer organization and the associated LDAP entries. If exactly one argument follows `$self`, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in. This routine will also set ACI rules governing the permissions various types of users have to read/write entries under the organization in the DIT. For example, all organization administrators will be able to create new organizational units and users; ordinary users will not. Returns true on success, undef otherwise. See `SwCom::IMgr::Entry::Create`, `SwCom::IMgr::Entry::Set` Properties.

- **Delete()**
  Deletes the organization and the associated LDAP entries. The Person entries in the organization's subtree must be deleted before calling this method. The caller must afterwards delete the primary domain name for the Org. Returns 1 on success, undef otherwise. (See `SwCom::IMGR::Entry::Delete`.)

- **Read('attr', ...)**
  Reads the listed attributes on the current organization and returns a reference to hash of the attribute names mapped to arrays of values. If no attributes are passed in, it reads all attributes. Any attributes read that don't yet have values on the LDAP entry are a valid key in the returned hash, mapped to the undef value. Returns undef on error. See `SwCom::IMgr::Entry::Read`.

- **Add('attr' => [val, ...], ...)**
  Adds the specified attribute values to the organization entry. If exactly one argument follows `$self`, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in. Returns 1 on success, undef otherwise.

  Restrictions—values can not be added to the following attributes: objectclass, dc, o, businesscategory.

- **Remove('attr' => [val, ...], ...)**
  Removes the specified attribute values from the organization entry. If exactly one argument follows `$self`, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in. Returns 1 on success, undef otherwise.

Restrictions—the Org's primary domain name can not be removed from the 'alloweddomains' attribute. Also, a user can not remove values from the following attributes: objectclass, dc, o, businesscategory.

- **SetAttributes('attr' => [val, ...], ...)**
  Replaces all values of the specified attributes on the organization entry. If exactly one argument follows `$self`, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in. Returns 1 on success, undef otherwise.

  Restrictions—a user can not replace values on the following attributes: objectclass, dc, o, businesscategory.

- **GetMailGroups(cos_name1, …)**
  Returns a list of the `SwCom::IMgr::MailGroup` objects for the organization. The mail groups are associated with a MailCOS. If one or more COS names are supplied, only return the mail groups associated with those Cos's. In case of error it returns undef.

  See also `SwCom::IMgr::Entry`

- **AddAdmins(role, person1, ...);**
  Grants an administrative role over this organization to the persons. Currently, parameter 'role' is not being used. The persons passed in are `SwCom::IMgr::Person` objects. All persons are added to the organizational administrator group. Returns 1 on success, undef otherwise.

- **RemoveAdmins(role, person1, ...);**
  Removes persons from the organization's administrator group object, i.e. revokes an administrative role for persons over this organization. Currently 'role' is ignored. The persons passed in are `SwCom::IMgr::Person` objects. All persons are removed from the organizational administrator group. Returns 1 on success, undef otherwise.

- **GetAdmins(role...)**
  Returns a list of `SwCom::IMgr::Person` objects corresponding to the organization administrators. Currently 'role' is ignored. All persons returned are from the organizational administrator group. In case of error it returns undef.

- **OrgUnit(ou_name)**
  Returns a reference to an SwCom::IMgr::OrgUnit object, identified by its name, located beneath the current organization object in the LDAP hierarchy. In case of error it returns undef.

  For example, `$org->OrgUnit(` "dev") under the organization unit "eng", under an organization object with DN `dc=worldnet, dc=att, dc=com` would return a `SwCom::IMgr::OrgUnit` object with the DN `ou=dev,ou=eng,dc=worldnet,dc=att,dc=com`.

- **GetOrgUnitNames(flag = 0);**
  Returns names of the organizational units beneath the organization object. If flag is 1, return the names of the organizational units that are the first level children under the organization within the DIT.

If flag is 0, names of all the organizational units beneath the organization object are returned. Entries are returned as an array of comma delimited OU names, such that the hierarchical relationship of OU's is preserved.

If flag is 0 (default) and assuming org units: a/m, a/n, a/n/x, a/b returns an array of arrays listing children organizational units.

```
ie: ( [a],
[a, m],
[a, n],
[a, n, x],
[a, b],
)
```

In case of error it returns undef.

- **GetOrgUnits(flag = 0)**
  Returns names of the organizational units beneath the organization object. If flag is 1, return the names of the organizational units that are the first level children under the organization within the DIT. If flag is 0, names of all the organizational units beneath the organization object are returned.

  Entries are returned as a reference to a hash of hashes `{{ou1 => {}}, ...}` preserving the hierarchy. Every key/value is: `<OU name>/<hash of children>`. If an OU does not have children, the value is the empty hash `{}`. Note that if flag is 1, all child hash values are the empty hash `{}`.

  For example, assuming org units: `a/m, a/n, a/n/x, a/b` the function returns a reference to a hash of hashes mapping org units to their children.

  ```
  ie: { a => { m => {},
        n => { x => {},
      },
    b => {},
  }
  ```

  In case of error it returns undef.

- **Search(*Session, attr, value, op*)**
  Returns an array of organization names based on the search expression ‘attr’ ‘op’ ‘value’. ‘op’ maybe 'starts_with' (default), or 'is'. This routine may not be called through a $self pointer. It should be called using the fully qualified name of the function. In case of error it returns undef.

- **IsSite()**
  Returns 1 if the organization is a site (ISP), else 0. In case of error, it returns undef. This method is not called outside of InterManager API.

- **IsCustomer()**
  Return 1 if the organization is an ISP customer organization, else 0. In case of error, it returns undef. This method is not called outside of InterManager API.

- **MailGroupParentDN()**
  Returns the DN of the parent object under which the mail group entries are located in an organization. This method is not called outside of InterManager API.

7.3.7 SwCom::IMgr::OrgUnit

Description

LDAP Organization Units. An LDAP entry that represents an organizational unit beneath an organization. The `SwCom::IMgr::OrgUnit` helps to organize and identify organizational unit entries in the LDAP hierarchy.

Synopsis

```
$org_unit = new SwCom::IMgr::OrgUnit(IMgrSession, $org, $ou_name,
...);
$org_unit = new_from_org SwCom::IMgr::OrgUnit(IMgrOrg, $ou_name,
...);
$org_unit = new_from_ou SwCom::IMgr::OrgUnit(IMgrOrgUnit, $ou_name,
...);
$org_unit = new_from_dn SwCom::IMgr::OrgUnit($dn);

$name = $org_unit->Name();
$org_name = $org_unit->OrgName();

$org_unit->Create(attr => [val]. …);
$org_unit->Delete();

my %attrs = %{ $org_unit->Read('attr', …) };

$org_unit->AddAdmins($role, IMgrPerson ...);
$org_unit->RemoveAdmins($role, IMgrPerson ...);
@adm_list = $org_unit->GetAdmins($role...);
```

Methods not called outside of InterManager API.

```
$org_unit->IsConsumerOrgUnit();
$org_unit->IsBusinessOrgUnit();
```

Methods

- **Constructors**
  Constructors for the `SwCom::IMgr::OrgUnit`. Return a reference to a `SwCom::IMgr::OrgUnit` object.

  ```
  $org_unit = new SwCom::IMgr::OrgUnit(IMgrSession, org_name, ou_name,
  ...);
  ```

  This form requires a SwCom::IMgr::Session, the org name and at least one ou_name.

  ```
  $org_unit = new_from_org SwCom::IMgr::OrgUnit(IMgrOrg,
  $ou_name, ...);
  ```

  The second form, requires a valid instance of `SwCom::IMgr::Org` object and at least one `ou_name`.

  ```
  $org_unit = new_from_ou SwCom::IMgr::OrgUnit(IMgrOrgUnit,
  $ou_name, ...);
  ```


The third form, requires a valid instance of SwCom::IMgr::OrgUnit object and at least one ou_name.

```
$org_unit = new_from_dn (IMgrSession, $dn)
```

The fourth form, requires a SwCom::IMgr::Session and the full DN of an organization unit.

In the first three forms, more than 1 OU name may be specified to get nested OrgUnits. OU names should be listed in descending hierarchical order. The first OU name must be immediately beneath the organization for the first and second forms, and the organizational unit in the third form. The last OU name is the organizational unit to be instantiated. Also see `SwCom::IMgr::Org`, `SwCom::IMgr::Entry`

- **Name()**
  Returns the organizational unit name. This is faster than reading from the 'ou' attribute from the entry. In case of error it returns undef.

- **OrgName()**
  Returns the organizational unit's parent organization name. Faster than reading from the 'o' attribute from the entry. In case of error it returns undef.

- **Create(attr1 => [attr_val1], …)**
  Creates an organizational unit object. . If exactly one argument follows $self, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in. This routine will also set ACI rules governing the permissions various types of users have to read/write entries under the organizational unit in the DIT. (e.g. all organizational unit administrators will be able to create new organizational units and users beneath the current OU; ordinary users will not.) Returns 1 on success, undef otherwise.

  The caller must pass in `'businesscategory' => 'consumer'` if they want to create a consumer OU under a provider subtree.

  Restrictions—a user can not pass in the following attributes: `objectclass`, `ou`. One can not create a consumer OU underneath a customer organization subtree.

- **Delete()**
  Deletes the OU entry and the associated entries pertaining to an email based Organizational Unit in the DIT. ACI rules referring to these entries are deleted too. This method assumes that all children entries of type SwCom::IMgr::Person have already been deleted. Returns 1 on success, undef otherwise.

- **Read('attr', ...)**
  Reads the listed attributes on the current Organization Unit and returns a reference to hash of the attribute names mapped to arrays of values. If no attributes are passed in, reads all attributes. Any attributes that are read that do not yet have values on the LDAP entry are a valid key in the returned hash, mapped to the undef value. Returns undef on error. See `SwCom::IMgr::Entry::Read.`

- **AddAdmins(role, IMgrPerson, ...)**
  Grants an administrative role over this OU to persons. Currently 'role' is not being used. Creates the OU admin group if it does not already exist. All persons are added to the organizational unit administrator group. The persons passed in are `SwCom::IMgr::Person` objects. Returns 1 on success, undef otherwise.
- **RemoveAdmins(role, ImgrPerson, ...)**
  Removes persons from the OU administrators' group object, i.e. revokes an administrative role for persons over this OU. Currently 'role' is ignored. All persons are removed from the Organizational Unit administrative group. The persons passed in are `SwCom::IMgr::Person` objects. Returns 1 on success, undef otherwise.
- **GetAdmins(role, ...)**
  Returns a list of SwCom::IMgr::Person objects corresponding to the OU administrators. Currently 'role' is ignored. In case of error it returns undef.
- **IsConsumerOrgUnit()**
  Return 1 if this OU contains consumer accounts, else 0. A consumer account is for an email user not affiliated with a customer organization. This method is not called outside of InterManager API.
- **IsBusinessOrgUnit()**
  Return 1 if this OU contains business accounts, else 0. All OUs within customer organizations are business OUs. OUs within an provider's organization may be business or consumer OUs with the restriction that business OUs may not reside under a consumer OU. This method is not called outside of the InterManager API.

7.3.8 SwCom::IMgr::Person

Description

LDAP Person. An LDAP entry that represents a person associated with an organization or an organizational unit. If an e-mail account is associated with the end-user, the attributes of that account can also be accessed directly through the Person object.

Synopsis

```
$person = new_from_parent SwCom::IMgr::Person($parent, $person_name,
$cos_name);

$person = new_from_mailgroup SwCom::IMgr::Person($parent,
$person_name, IMgrMailGroup);

$person = new_from_name SwCom::IMgr::Person (IMgrSession,
$person_name, $parent, $immchild=0);

$person = new_from_dn(IMgrSession, $dn);

$name = $person->Name();

$smtp = $person->Smtp();

$person ->SetProperties('attr' => val, ...);
```



```
$person ->Create('attr' => val, ...);
$person ->Delete();

my %attrs = %{ $person->Read('attr', …) };

$mg = $person->GetMailGroup();
$mt = $person->GetMailTemplate();

$org->Add('attr' => val, ...);
$org->SetAttributes('attr' => val, ...);
$org->Remove('attr' => val, ...);

$person ->Move($org, $ou_name ...);
```

Methods

- **new_from_parent SwCom::IMgr::Person (parent, name, cos_name)**
  Constructor for the `SwCom::IMgr::Person`. Returns a reference to a `SwCom::IMgr::Person` object.

  Where:

`parent`	The LDAP entry structurally above the Person in the DIT (may be of type `SwCom::IMgr::Org` or `SwCom::IMgr::OrgUnit`).
`name`	The UID value (smtp address) for the Person.
`cos_name`	The COS to be associated with the mail account for the person.

- **new_from_mailgroup SwCom::IMgr::Person (parent, name, IMgrMailGroup)**
  Constructor for the `IMgrPerson`. Returns a reference to a `SwCom::IMgr::Person` object.

  Where:

`parent`	The LDAP entry structurally above the Person in the DIT (may be of type `SwCom::IMgr::Org` or `SwCom::IMgr::OrgUnit`).
`name`	The UID value (smtp address) for the Person.
`IMgrMailGroup`	A reference to a `SwCom::IMgr::MailGroup` object, representing the mail group of which the Person is to be made a member. This links the Person to their mail COS. (The mail COS establishes the various services supported for the Person's mailbox.)

- **new_from_name SwCom::IMgr::Person(IMgrSession, name, parent, immchild=0)**
  Constructor for the `SwCom::IMgr::Person`. Returns a reference to a `SwCom::IMgr::Person` object.

  Where:

`IMgrSession`	
`parent`	The LDAP entry structurally above the Person in the DIT (may be of type `SwCom::IMgr::Org` or `SwCom::IMgr::OrgUnit`).
`name`	The UID value (smtp address) for the Person.

Note: *If immchild is 1, the Person is directly underneath the parent. Otherwise, we must search for the Person. If 'parent' is supplied, search under the parent, otherwise search from the DIT root.*

- **new_from_dn SwCom::IMgr::Person(IMgrSession, dn)**
  Constructor for the SwCom::IMgr::Person. Returns a reference to a SwCom::IMgr::Person object. Construct a Person object given the full Distinguished Name.

- **Name()**
  Returns the 'uid' attribute value (happens to be the SMTP Address) for the person. In case of error it returns undef.

- **Smtp()**
  Returns the SMTP Address ('mail' attribute value) for the person. In case of error it returns undef.

- **GetMailGroup()**
  Returns the mail group object (`SwCom::IMgr::MailGroup`) for this person. In case of error it returns undef.

- **GetMailTemplate()**
  Returns the mail template object (`SwCom::IMgr::MailTemplate`) for this person. In case of error it returns undef.

- **SetProperties('attr' => val, ...)**
  Sets the properties to the specified values. If exactly one argument follows self, it assumes that the argument is a reference to a hash of property name/value pairs. Otherwise, it assumes that property name/value pairs were passed in.

  Valid properties are `AccountExists`, `LdapEntryExists` each which take values 1 or 0; `OnAccountExistence`, `OnLdapExistence` each which take values `fail_if_exists` or 'ok_if_exists'. These four properties are used to control the behavior of `Create()`. (See `SwCom::IMgr::Person::Create()`).

  If `AccountExists` is set to 1, `Create()` will verify that the account record for the Person exists. Otherwise, `Create()` will attempt to create the account record. If `LdapEntryExists` is set to 1, `Create()` will verify that the LDAP entry for the Person exists in the DIT. Otherwise, `Create()` will attempt to create the LDAP entry.

  `OnAccountExistence` determines what happens if `Create()` attempts to create the account record for the Person and it already exists. If `OnAccountExistence` is set to `fail_if_exists`, an error is returned (undef), otherwise if set to `ok_if_exists`.

  Likewise, `OnLdapExistence` determines what happens if `Create()` attempts to create the LDAP entry for the Person and it already exists. If `OnLdapExistence` is set to `fail_if_exists`, an error is returned (undef), otherwise if set to `ok_if_exists`. Always returns 1.

- **Create(attr1 => [attr_val1], …)**
  Creates a Person's object in the directory. The Person's parent entry must already exist in the directory. One of the following constructors must have been used to construct the Person object: `new_from_parent` or `new_from_mailgroup`. Depending on the property settings, the Person's LDAP entry and/or account record may be read to verify it already exists. (See `SetProperties()`).Returns 1 on success, undef otherwise.

  Restrictions—a user can not pass in the following attributes: `objectclass`, `mail`, `uid`, `mailgroup`, `allowedservices`; and the Mail API attributes: `smtpAddress`, `cosName`, `resultCos`.

- **Delete()**
  Deletes the Person object from the directory. The Person is removed from the mail group and any administrative groups of which they are a member. Returns 1 on success, undef otherwise.

- **Move(org, ou_name ...)**
  Moves the person from their current location in the DIT to the Organizational Unit underneath the Organization specified by org. If `ou_name` is not passed in, the Person is moved directly under the Organization. The `org` argument may be a reference to a `SwCom::IMgr::Org` object or the Organization's name. All OU names supplied may only be names (as opposed to object references). Returns 1 on success, undef otherwise.

- **Read('attr', ...)**
  Reads the listed attributes on the current Person object and returns a reference to hash of the attribute names mapped to arrays of values. If no attributes are passed in, reads all attributes. Any attributes read that don't yet have values on the LDAP entry are a valid key in the returned hash, mapped to the undef value. Returns undef on error. See also `SwCom::IMgr::Entry::Read`.

- **Add('attr' => [val, ...], ...)**
  Adds the specified attribute values to the Person entry. If exactly one argument follows `$self`, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in. Returns 1 on success, undef otherwise.

  Restrictions—a user can not add values to the following attributes: `objectclass`, `uid`, `mail`, `mailgroup`, `allowedservices`, `preferredservices`; and the InterMail API attributes: `smtpAddress`, `popAddress`, `mssHost`, `emailIntID`, `password`, `hash`, `status`, `acCos`, `resultCos`.

  Use `SetAttributes()` instead.

- **Remove('attr' => [val, ...], ...)**
  Removes the specified attribute values from the Person entry. If exactly one argument follows $self, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in. Use `SetAttributes()` instead. Returns 1 on success, undef otherwise.

  Restrictions—a user can not remove values from the following attributes: `objectclass`, `uid`, `mail`, `mailgroup`, `allowedservices`, `preferredservices`; and the InterMail API attributes: `smtpAddress`, `popAddress`, `mssHost`, `emailIntID`, `password`, `hash`, `status`, `acCos`, `resultCos`.

- **SetAttributes('attr' => [val, ...], ...)**
  Replaces all values of the specified attributes on the Person entry. If exactly one argument follows $self, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in. Returns 1 on success, undef otherwise.

  Restrictions—a user can not replace values on the following attributes: `objectclass`, `uid`, `allowedservices`.

- **Roles()**
  NOT YET IMPLEMENTED.

- **GetAdminAuthority()**
  Returns a hash mapping an administrative role to those entries upon which the person has that authority. Valid authoritative roles for the hash keys are 'site admin', 'csr admin', 'org admin', and 'ou admin'. The site, csr, and org admin roles map to arrays of entry names. Example: ('org admin' => ['Widget Services', 'ABC Printing, Inc.']). The ou admin role maps to an array of arrays as follows: ('ou admin' => [['Widget Services', 'Engineering', 'Sustaining'], ['Widget Services', 'Marketing', 'Foreign']],). Here, each sub-array contains the organization and organizational unit names as its elements. Element zero contains the organization name, consecutive elements contain the child organizational units, with the final organizational unit stored as the last element.

- **GetOrgUnitNames()**
  Return the Organizational Unit names structurally above the Person in as a list. Within the list, the OU's are in descending, hierarchical order. For example, if the Person's DN is:

  `uid=joe.bloe@venus.software.com,ou=dev,ou=west,ou=eng,dc=accordance,dc=com`

  the list returned is: eng, west, dev

- **Search(IMgrSession, @attrs, @ops, $value, $boolop= 'AND')**
  Returns a list of 'uids' for the Persons which match the search criteria. The array of attributes is related to the value by the parallel items in the 'ops' (operations) array. This relationship may be 'is' or 'starts_with'. So for instance, the triplet: <$attr[0] $ops[0] $value> forms one filter expression. 'boolop' may be 'AND' or 'OR'. 'boolop' is used to connect individual filters to construct a complex filter. The default for 'boolop' is 'AND'. In case of error it returns undef. See also `SwCom::IMgr::Entry`.

7.3.9 SwCom::IMgr::MailGroup

Description

LDAP Mail Group. This object coordinates the process of the email administration for a group of people. The mail group's primary purpose is to relate a set of mailboxes (owned by Persons) to a unique mail COS and to potentially restrict how many of mailboxes with this COS may be created. The mail COS determines the services actually supported upon a mailbox.

Attributes include a pointer to a mail COS object (see below), a pointer to a mail template object (optional), a quota for the maximum number of people that can be assigned to this group (and hence use this mail COS) etc. See also `SwCom::IMgr::Entry`.

Synopsis

```
$mg = new SwCom::IMgr::MailGroup(IMgrOrg, $site_name, $cos_name);
$mg = new_from_org  SwCom::IMgr::MailGroup(IMgrOrg, $cos_name);
$mg = new_from_cos SwCom::IMgr::MailGroup(IMgrOrg, IMgrMailCOS);
$mg = new_from_dn SwCom::IMgr::MailGroup(IMgrSession, $dn);

$name = $mg->Name();

$mg ->SetProperties('attr' => val, ...);

$mg ->Create('attr' => val, ...);
$mg ->Delete();

my %attrs = %{ $mg->Read('attr', …) };

$mt = $mg->GetMailTemplate();
$cos = $mg->GetMailCOS();

$mg->Add('attr' => val, ...);
$mg->SetAttributes('attr' => val, ...);
$mg->Remove('attr' => val, ...);
```

Methods

- **new SwCom::IMgr::MailGroup(IMgrOrg, prov_name, cos_name)**
  Constructor for the `SwCom::IMgr::MailGroup`. Returns a reference to a `SwCom::IMgr::MailGroup` object.

  Caller passes the SwCom::IMgr::Org object for the organization owning the mail group, the provider owning the COS, and the COS name associated with the mail group. The caller may use this constructor if they will be calling Create next.

  In case of error, it returns undef.

- **new_from_org SwCom::IMgr::MailGroup(IMgrOrg, $cos_name)**
  Constructor for the `SwCom::IMgr::MailGroup`. Returns a reference to a `SwCom::IMgr::MailGroup` object.

  Caller passes the `SwCom::IMgr::Org` object for the organization owning the mail group, and the COS name associated with the mail group. This constructor calculates the DN directly from the IMgrOrg's DN.

  Restrictions—this is not to be used if `Create()` will be called next

- **new_from_COS SwCom::IMgr::MailGroup(IMgrOrg, IMgrMailCos)**
  Constructor for the `SwCom::IMgr::MailGroup`. Returns a reference to a `SwCom::IMgr::MailGroup` object.

  Caller passes the `SwCom::IMgr::Org` object for the organization owning the mail group, and the `SwCom::IMgr::MailCos` object associated with the mail group. The caller may use this constructor if they will be calling `Create()` next. In case of error, it returns undef.

- **new_from_dn SwCom::IMGR::MailGroup(IMgrSession, $dn)**
  Constructor for the `IMgr::MailGroup`. Returns a reference to a `SwCom::IMgr::MailGroup` object. Caller knows the DN. In case of error, it returns undef.

  Restrictions—this is not to be used if `Create()` will be called next!

- **Name()**
  Returns the mail group's name. . In case of error returns an undef.

- **GetMailCOS()**
  Returns a reference to a `SwCom::IMgr::MailCOS` object associated with this mail group. In case of error it returns undef.

- **GetMailTemplate()**
  Returns `SwCom::IMgr::MailTemplate` object associated with this mail group. In case of error it returns undef.

- **GetMailTemplateDN()**
  Reads the mail template DN from the mail group object.

- **SetProperties('attr' => val, ...)**
  Sets the properties to the specified values. If exactly one argument follows self, it assumes that the argument is a reference to a hash of property name/value pairs. Otherwise, it assumes that property name/value pairs were passed in. Always returns 1.

  Valid properties are `LdapEntryExists`, which takes values 1 or 0, and `OnLdapExistence`, which takes values `fail_if_exists` or `ok_if_exists`. These properties are used to control the behavior of `Create()`. (See `SwCom::IMgr::MailGroup::Create()`).

  If `LdapEntryExists` is set to 1, `Create()` will verify that the LDAP entry for the mail group exists in the DIT. Otherwise, `Create()` will attempt to create the LDAP entry.

  `OnLdapExistence` determines what happens if `Create()` attempts to create the LDAP entry for the `MailGroup` and it already exists. If `OnLdapExistence` is set to `fail_if_exists`, an error is returned (undef), otherwise if set to `ok_if_exists`.

- **Create (’attr’ => [value ...], ...)**
  Creates a mail group entry in the directory. If exactly one argument follows self, it assumes that the argument is a reference to a hash of property name/value pairs. Otherwise, it assumes that property name/value pairs were passed in.

  The caller must construct the `SwCom::IMgr::MailGroup` object before calling this method using the following constructors: `new()`, `new_from_COS()`.

  Caller may pass in the max users that may own a mailbox using the associated COS. For example `'maxusers' => [number]`. Otherwise, the max users defaults to 0 (infinity). Returns 1 on success, undef otherwise.

  Restrictions—a user can not pass in the following attributes: `objectclass`, `cn`, `numusers`.

- **Delete()**
  Deletes a mail group entry from the directory. Fails if there are any members. Returns 1 in case of success and undef otherwise.

- **Read(’attr’, ...)**
  Reads the listed attributes on the current mail group object and returns a reference to hash of the attribute names mapped to arrays of values. If no attributes are passed in, reads all attributes. Any attributes that are read that do not yet have values on the LDAP entry are a valid key in the returned hash, mapped to the undef value. Returns undef on error. See also `SwCom::IMgr::Entry::Read`.

- **Add(’attr’ => [val, ...], ...)**
  Adds the specified attribute values to the mail group entry. If exactly one argument follows `$self`, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in. Returns 1 on success, undef otherwise.

  Restrictions—a user can not add values to the following attributes: `objectclass`, `cn`, `numusers`, `maxusers`.

- **Remove(’attr’ => [val, ...], ...)**
  Removes the specified attribute values from the mail group entry. If exactly one argument follows $self, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in. Returns 1 on success, undef otherwise.

  Restrictions—a user can not remove values from the following attributes: `objectclass`, `cn`, `numusers`, `maxusers`.

- **SetAttributes(’attr’ => [val, ...], ...)**
  Replaces all values of the specified attributes on the Person entry. If exactly one argument follows `$self`, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in. Returns 1 on success, undef otherwise.

  Restrictions—a user can not replace values on the following attributes: `objectclass`, `cn`, `numusers`.

- **AddMembers (person1, ...)**
  Adds one or more members to a mail group. Checks to make sure that the current member count is not exceeding the max allowed. Max may be 0, which is infinity. Specified persons are the `SwCom::IMgr::Person` objects. Returns 1 on success, undef otherwise.

- **RemoveMembers (person1, ...)**
  Removes one or more members from a mail group. Specified persons are the `SwCom::IMgr::Person` objects. Returns 1 on success, undef otherwise.

7.3.10 SwCom::IMgr::MailTemplate

Description

LDAP Mail Template. A mail template is used to establish any initial attribute value(s) for a newly created Person object. A mail template is referred to by a mail group and is associated with the mail COS also referred by the same mail group.

Upon creating the Person, all attribute values from the relevant mail template are read and placed upon the Person object. These attributes may not be passed into the `Create()` method for the Person. Some of these attributes may, however, be modified after the Person is created. See the `SetAttributes()`, `Add()`, and `Remove()` methods for more information.

Synopsis

```
$mt = new_from_org SwCom::IMgr::MailTemplate(IMgrOrg, $cos_name);
$mt = new_from_dn SwCom::IMgr::MailTemplate(IMgrSession, $dn);

$name = $mt->Name();
$cos = $mt->GetMailCOS();
$mg = $mt->GetMailGroup();

$mt ->SetProperties('attr' => val, ...);

$mt ->Create('attr' => val, ...);
$mt ->Delete();

my %attrs = %{ $mt->Read('attr', …) };

$mt->Add('attr' => val, ...);
$mt->SetAttributes('attr' => val, ...);
$mt->Remove('attr' => val, ...);
```

Methods

- **new_from_org SwCom::IMgr::MailTemplate(IMgrOrg, cos_name)**
  Constructor for the SwCom::IMgr::MailTemplate object. Returns a reference to a SwCom::IMgr::MailTemplate object.

  Caller passes the IMgrOrg object for the organization owning the mail template and the COS name associated with the template. Returns undef in case of an error.

- **new_from_dn SwCom::IMgr::MailTemplate(IMgrSession, dn)**
  Constructor for the SwCom::IMgr::MailTemplate object. Returns a reference to a SwCom::IMgr::MailTemplate object. Caller knows the DN. Returns undef in case of an error.

  Restrictions—this is not to be used if **Create()** will be called next.

- **Name()**
  Returns mail template's name.

- **GetMailCOS()**
  Returns the `SwCom::IMgr::MailCOS` object associated with the mail group referencing this template. Returns undef in case of an error.

- **GetMailGroup()**
  Returns the `SwCom::IMgr::MailGroup` object associated with the mail group referencing this template. Returns undef in case of an error.

- **SetProperties('attr' => val, ...)**
  Sets the properties to the specified values. If exactly one argument follows self, it assumes that the argument is a reference to a hash of property name/value pairs. Otherwise, it assumes that property name/value pairs were passed in.

  Valid properties are `LdapEntryExists`, which takes values 1 or 0, and `OnLdapExistence`, which takes values 'fail_if_exists' or 'ok_if_exists'. These properties are used to control the behavior of `Create()`. See also `SwCom::IMgr::MailTemplate::Create()`.

  If `LdapEntryExists` is set to 1, `Create()` will verify that the LDAP entry for the mail group exists in the DIT. Otherwise, `Create()` will attempt to create the LDAP entry.

  `OnLdapExistence` determines what happens if `Create()` attempts to create the LDAP entry for the `MailGroup` and it already exists. If `OnLdapExistence` is set to `fail_if_exists`, an error is returned (undef), otherwise if set to `ok_if_exists`.

  Always returns 1.

- **Create ('attr' => [value ...], ...)**
  Creates a mail template entry in the directory. If exactly one argument follows self, it assumes that the argument is a reference to a hash of property name/value pairs. Otherwise, it assumes that property name/value pairs were passed in.

  The caller must construct the `SwCom::IMgr::MailTemplate` object before calling this method using the `new_from_org()` constructor.

  Restrictions—a user can not pass in the following attributes: `objectclass`, `mup`, `mail`, `mailid`; and the Mail API attributes: `smtpAddress`, `emailIntID`.

  Returns 1 on success, undef otherwise.

- **Delete ()**
  Deletes a mail template entry from the directory. Fails if there are mail group entries referencing the mail template entry. Returns undef in case of error.

- **Read('attr', ...)**
  Reads the listed attributes on the current mail template object and returns a reference to hash of the attribute names mapped to arrays of values. If no attributes are passed in, reads all attributes. Any attributes that are read that do not yet have values on the LDAP entry are a valid key in the returned hash, mapped to the undef value. Returns undef on error. See also `SwCom::IMgr::Entry::Read`.

- **Add('attr' => [val, ...], ...)**
  Adds the specified attribute values to the mail template entry. If exactly one argument follows `$self`, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in. Returns 1 on success, undef otherwise.

  Restrictions—a user can not add values to the following attributes: `objectclass`, `mup`, `mail`, `mailid`; and the InterMail API attributes: `smtpAddress`, `emailIntID`.

- **Remove('attr' => [val, ...], ...)**
  Removes the specified attribute values from the mail template entry. If exactly one argument follows `$self`, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in. Returns 1 on success, undef otherwise.

  Restrictions—a user can not remove values from the following attributes: `objectclass`, `mup`, `mail`, `mailid`; and the Mail API attributes: `smtpAddress`, `emailIntID`.

- **SetAttributes('attr' => [val, ...], ...)**
  Replaces all values of the specified attributes on the mail template entry. If exactly one argument follows `$self`, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in. Returns 1 on success, undef otherwise.

  Restrictions—a user can not replace values on the following attributes: `objectclass`, `mup`, `mail`, `mailid`; and the Mail API attributes: `smtpAddress`, `emailIntID`.

7.3.11 SwCom::IMgr::MailCOS

Description

Mail class of service. It corresponds to an LDAP entry whose attributes: `preferredservices` and `allowedservices` (hashes of service/value settings) together control the availability of features to a group of end-users. Also referred to as a MailUserPolicy. See also `SwCom::IMgr::Entry`.

Synopsis

```
$cos = new SwCom::IMgr::MailCOS( IMgrSession, $prov_name, $cos_name
);
$cos = new_from_provider SwCom::IMgr::MailCOS( IMgrProvider,
$cos_name );
$cos = new_from_dn SwCom::IMgr::MailCOS( IMgrSession, $dn );

$name = $cos->Name();

$cos ->SetProperties('attr' => val, ...);

$cos ->Create('attr' => val, ...);
$cos ->Delete();

my %attrs = %{ $mt->Read('attr', …) };

$cos ->Add('attr' => val, ...);
$cos ->SetAttributes('attr' => val, ...);
$cos ->Remove('attr' => val, ...);
```

Methods

- **new (IMgrSession, prov_name, cos_name)**
  Constructor for the `SwCom::IMgr::MailCOS` object. Returns a reference to a `SwCom::IMgr::MailCOS` object. Caller passes name of provider owning the COS and the COS name. Returns undef in case of error.
- **new_from_provider (IMgrProvider, $cos_name)**
  Constructor for the `SwCom::IMgr::MailCOS`. Returns a reference to a `SwCom::IMgr::MailCOS` object. Caller passes in the `IMgrProvider` object for the provider owning the COS and the COS name. Returns undef in case of error.
- **new_from_dn (Session, $dn)**
  Constructor for the `SwCom::IMgr::MailCOS` object. Returns a reference to a `SwCom::IMgr::MailCOS` object. Caller knows the DN. In case of error it returns undef.

  Restrictions—this is not to be used if `Create()` will be called next.
- **Name()**
  Returns COS name.
- **SetProperties('attr' => val, ...)**
  Sets the properties to the specified values. If exactly one argument follows self, it assumes that the argument is a reference to a hash of property name/value pairs. Otherwise, it assumes that property name/value pairs were passed in.

  Valid properties are `MailCOSExists`, `LdapEntryExists` each which take values 1 or 0; `OnMailCOSExistence`, `OnLdapExistence` each which take values `fail_if_exists` or `ok_if_exists`. These four properties are used to control the behavior of `Create()`. (See `SwCom::IMgr::MailCOS::Create()`).

If `MailCOSExists` is set to 1, `Create()` will verify that the account record for the Person exists. Otherwise, `Create()` will attempt to create the account record. If `LdapEntryExists` is set to 1, `Create()` will verify that the LDAP entry for the Person exists in the DIT. Otherwise, `Create()` will attempt to create the LDAP entry.

`OnMailCOSExistence` determines what happens if `Create()` attempts to create the name in the COS table for the mail COS and it already exists. If `OnMailCOSExistence` is set to `fail_if_exists`, an error is returned (undef), otherwise if set to `ok_if_exists`.

Likewise, `OnLdapExistence` determines what happens if `Create()` attempts to create the LDAP entry for the mail COS and it already exists. If `OnLdapExistence` is set to `fail_if_exists`, an error is returned (undef), otherwise if set to `ok_if_exists`.

Always returns 1.

- **Create ('attr' => [value ...], ...)**
  Creates a mail COS entry in the directory and a new COS name in the COS name table. If exactly one argument follows `$self`, it assumes that the argument is a reference to a hash of property name/value pairs. Otherwise, it assumes that property name/value pairs were passed in.

  The caller must construct the `SwCom::IMgr::MailCOS` object before calling this method using one of the following constructors: `new()` or `new_from_provider()`.

  Returns 1 on success, undef otherwise.

- **Delete ()**
  Deletes the mail COS entry from the directory provided no mail group entries are currently referring to it. This includes the LDAP entry, as well as, removing COS name from the COS name table.

  Returns 1 in case of success undef otherwise.

- **Read('attr', ...)**
  Reads the listed attributes on the current mail COS object and returns a reference to hash of the attribute names mapped to arrays of values. If no attributes are passed in, reads all attributes. Any attributes that are read that do not yet have values on the LDAP entry are a valid key in the returned hash, mapped to the undef value.

  The `attr` parameter `@attrs`, lists the attributes the caller wishes to receive back. These are returned in a hash of `attr name/[attr values]` pairs. (The values are set in an array because LDAP allows multiple values for an attribute.) If `@attrs` is empty or undefined, all attributes of the MailCOS object are returned.

  Returns undef on error.

- **Add('attr' => [val, ...], ...)**
  Adds the specified attribute values to the mail COS entry. If exactly one argument follows `$self`, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in.

  Restrictions—a user can not add values to the following attributes: `objectclass`, `mup`, `mail`, `mailid`; and the Mail API attributes: `smtpAddress`, `emailIntID`.

  Returns 1 on success, undef otherwise.

- **Remove('attr' => [val, ...], ...)**
  Removes the specified attribute values from the mail COS entry. If exactly one argument follows `$self`, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in.

  Restrictions—a user can not remove values from the following attributes: `objectclass`, `mup`, `mail`, `mailid`; and the Mail API attributes: `smtpAddress`, `emailIntID`.

  Returns 1 on success, undef otherwise.

- **SetAttributes('attr' => [val, ...], ...)**
  Replaces all values of the specified attributes on the mail COS entry. If exactly one argument follows `$self`, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in.

  Restrictions—a user can not replace values on the following attributes: `objectclass`, `mup`, `mail`, `mailid`; and the Mail API attributes: `smtpAddress`, `emailIntID`.

  Returns 1 on success, undef otherwise.

7.3.12 SwCom::IMgr::Provider

Description

The following is the set of LDAP entries in DIT that represent an Internet Service Provider. Sub-entries created for an ISP include: mail COS entries, and Site, CSR, Org administrative groups. The mail template, mail group and OU administrative group entries are included if the ISP is to reside as an InterManager Organization in the DIT. (ie. they outsource email services to themselves.)

Synopsis

```
$site = new SwCom::IMgr::Provider(IMgrSession, $site_name,
$domain_name);

$site_list = SwCom::IMgr::Provider::List(IMgrSession);

$name = $site->Name();
$domain = $site->PrimaryDomainName();

$site->Create();
$site->Delete();

$org = $site->GetOrg($org_name = $site_name);

$site->AddAdmins($role, IMgrPerson ...);
$site->RemoveAdmins($role, IMgrPerson ...);
@adm_list = $site->GetAdmins($role...);

@group_list = $site->GetMailGroups($org_name);
@cos_list = $site->GetMailCOSs();
```

Methods not called outside of the InterManager API.

```
$site->IsSite();
$site->IsCustomer();
$dn = $site->MailCOSParentDN();
$dn = $site->MailGroupParentDN();
$dn = $site->MailTemplateParentDN();
$site->CreateOrgUnit();
```

Methods

- **new SwCom::IMgr::Provider(IMgrSession, prov_name, domain_name)**
  Constructor for the `SwCom::IMgr::Provider`. The `SwCom::IMgr::Provider` object represents an ISP in the DIT. The `domain_name` argument is optional. If `domain_name` is absent it searches for the `prov_name` in the DIT and gets the DN. Otherwise it constructs the DN from `prov_name` and `domain_name`.

  The caller must pass both `prov_name` and `domain_name` if he/she will be calling the `Create()` method next.

  Returns a reference to a `SwCom::IMgr::Provider` object.

- **List(IMgrSession)**
  Returns a reference to an array of the names of all ISPs in the DIT. The ISP Organization entry is distinguished by its `businesscategory` attribute, which contains the value `provider`. It must not be called with $self as the first arguments; call using the fully qualified names. Return undef in case of an error.

  Note: *Currently, the InterManager only supports one ISP in the DIT.*

- **Name()**
  Returns Provider's name (e.g. Widget Services).

- **PrimaryDomainName()**
  Return Provider's primary domain name (e.g. widget.abc.com).

- **Create('attr' => [value ...], …)**
  Creates the new provider Organization entry and all other relevant entries pertaining to an ISP in the DIT. If exactly one argument follows self, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in.

  Returns 1 on success, undef otherwise.

- **Delete();**
  Deletes the provider entry and other relevant entries pertaining to an ISP operation from the DIT. OUs, Persons, MailGroups, and MailTemplates must already have been removed.

  Returns true on success, undef otherwise.

- **GetOrg(org_name)**
  Returns a reference to an `SwCom::IMgr::Org` object specified by `org_name`. It is required that this organization be a customer of the provider, or be the provider itself. If `org_name` is undefined, returns a `SwCom::IMgr::Org` for the site's organization otherwise returns the `SwCom::IMgr::Org` for the specified organization.

- **AddAdmins(role, ImgrPerson, ...);**
  Grants an administrative role over this Provider to Persons. Role is 'site admin', 'csr admin', or 'org admin'. The people passed in are `SwCom::IMgr::Person` objects. Returns 1 on success, undef otherwise.

- **RemoveAdmins($role, ImgrPerson, ...);**
  Removes the specified people from the appropriate administrative group within the Provider - i.e. revokes an administrative role for the Persons. Role is 'site admin' , 'csr admin', or 'org admin'. The people passed in are `SwCom::IMgr::Person` objects.

  Returns 1 on success, undef otherwise.

- **GetAdmins(role, ...);**
  Returns the `SwCom::IMgr::Person` objects for the Persons with the specified administrative role(s) for the Provider. If roles aren't specified, the default is 'site admin.'

  Returns undef in case of an error.

- **GetMailGroups(org_name);**
  Returns a list of `SwCom::IMgr::MailGroup` objects corresponding to the mail groups defined for the specified Organization. If `org_name` is not specified, the default is the Provider's organizational name.

  Returns undef in case of an error.

- **GetMailCOSs()**
  Returns a list of `SwCom::IMgr::MailCOS` objects corresponding to all the mail COS objects owned by the Provider. These include the private COS's used by the ISP for its own internal mail users (ie. employees). These also include the COS's used for consumer accounts. (Consumer accounts are stored under the Provider's Organization subtree within the DIT.)

  Returns undef in case of an error.

- **GetMailCOSNames()**
  Returns a list of COS names corresponding to all the mail COS objects owned by the Provider. These include the private COS's used by the ISP for its own mail users (ie. employees). These also include the COS's used for consumer accounts. (Consumer accounts are stored under the Provider's Organization subtree within the DIT.)

  Returns reference to an array or undef in the of an error.

- **IsSite()**
  Returns 1.

- **IsCustomer()**
  Returns 0.

- **MailGroupParentDN()**
  Returns the DN of the entry (mail data DN) beneath which the mail group entries are stored in the Provider's subtree.

- **MailTemplateParentDN()**
  Returns the DN of the entry (mail data DN) beneath which the mail template entries are stored in the Provider's subtree.

- **MailCOSParentDN()**
  Returns the DN of the entry (mail policies DN) beneath which the mail COS entries are stored in the Provider's subtree.

- **CreateOrgUnit(ou, attrs)**
  Creates an Org Unit within the Provider's subtree in the DIT. The `ou` argument is an `SwCom::IMgr::OrgUnit` object. The `attrs` argument is a reference to hash of attribute name-value pairs. The `ou` could be of business or consumer type. If undef, it is assumed to be a business out. Returns 1 on success, undef otherwise.

7.3.13 SwCom::IMgr::AdminGroup

Description

The InterManager object that represents an administrative group of people. Note that this object is used internally when adding or removing administrators to/from Organizations, Organizational Units, and Providers.

Synopsis

```
$admin_group = new SwCom::IMgr::AdminGroup($parent, $name);

$admin_group = new_from_dn SwCom::IMgr::AdminGroup(IMgrSession,
$dn);

$admin_group->Create();
$admin_group->Delete();

$admin_group->AddAdmins(IMGRPerson ...);
$admin_group->RemoveAdmins(IMGRPerson ...);
@adm_list = $admin_group->GetAdmins();
```

Methods

- **new SwCom::IMgr::AdminGroup(parent, name)**
  Constructor for the `SwCom::IMgr::AdminGroup`. Returns a reference to a `SwCom::IMgr::AdminGroup`. The 'parent' argument is an `SwCom::IMgr::Entry` object.

- **Create('attr' => [value ...],…)**
  Creates an admin group. If exactly one argument follows self, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in.

  The caller is expected to fill in the 'description' describing what kind of admin group they are making: site admins', 'csr admins', 'org admins', 'ou admins'. Caller has previously established the name of the group via the constructor. Returns true on success, undef otherwise.

  Note: *The 'member' attribute requires DNs. Caller is expected to have resolved their members to DNs (perhaps by $person->GetDN()).*

- **Delete()**
  Deletes the admin group. Benign if admin group doesn't exist. Returns true on success, undef otherwise.

- **AddAdmins(person1, ...)**
  Adds the specified Persons to the admin group. The people passed in are `SwCom::IMgr::Person` objects. The admin group entry must already exist in the directory. Returns 1 on success, undef otherwise.

- **RemoveAdmins(person1, ...)**
  Removes the specified Persons from the admin group. The people passed in are `SwCom::IMgr::Person` objects. Returns 1 on success, undef otherwise.

- **GetAdmins()**
  Returns `SwCom::IMgr::Person` objects corresponding to the members belonging to the admin group. Returns undef in case of an error.

7.3.14 SwCom::IMgr::Domain

Description

This represents a domain. This class inherits most of its functionality from `SwCom::Mail::Domain`. See also `SwCom::Mail::Domain`.

Synopsis

```
$domain = new SwCom::IMGR::Domain(IMGRSession, %attrs);

$name = $domain->Name();

$domain ->SetProperties('attr' => val, ...);

$domain->Create('attr' => val, ...);
$domain->Delete();

my %attrs = %{ $domain->Read('attr', …) };

$domain->SetAttributes('attr' => val, ...);
```

Methods

- **new SwCom::IMgr::Domain(IMgrSession, %attrs);**
  Constructor for the `SwCom::IMgr::AdminGroup`. Returns a reference to a `SwCom::IMgr::Domain` object that has the specified attribute settings.
- **List(IMgrSession);**
  Lists all domains that have been created. A domain may be added to any or all Organizations. `$self` must not be passed in; only call this method using its fully qualified name.
- **Name()**
  Returns the domain name.

- **SetProperties('attr' => val, ...)**
  Sets the properties to the specified values. If exactly one argument follows self, it assumes that the argument is a reference to a hash of property name/value pairs. Otherwise, it assumes that property name/value pairs were passed in.

  Valid properties are `MailDomainExists`, which takes values 1 or 0; `OnMailDomainExistence`, which take values `fail_if_exists` or `ok_if_exists`. These properties are used to control the behavior of `Create()`. (See `SwCom::IMgr::Domain::Create()`).

  If `MailDomainExists` is set to 1, `Create()` will verify that a domain record for the domain exists. Otherwise, `Create()` will attempt to create the domain record.

  `OnMailDomainExistence` determines what happens if `Create()` attempts to create the name in the domain record and it already exists. If `OnMailDomainExistence` is set to `fail_if_exists`, an error is returned (undef), otherwise if set to `ok_if_exists`.

  Always returns 1.

- **Create('attr' => [value ...], ...))**
  Creates the domain. If exactly one argument follows self, it assumes that the argument is a reference to a hash of property name/value pairs. Otherwise, it assumes that property name/value pairs were passed in. All attributes are passed through to `SwCom::Mail::Domain::Create()`. Depending on the property settings, the domain record may be read to verify it already exists. (See `SetProperties()`).

- **Delete()**
  It deletes the domain from the Mail domain table. If any Organization lists the domain as an allowed domain, do not delete the domain and return an error. Returns 1 on success, undef otherwise.

Note: *This is the `alloweddomains` attribute of `SwCom::IMgr::Org`.*

- **SetAttributes('attr' => [val], ...)**
  Replaces all values of the specified attributes on the mail COS entry. If exactly one argument follows `$self`, it assumes that the argument is a reference to a hash of attribute name/value pairs. Otherwise, it assumes that attribute name/value pairs were passed in. Returns 1 on success, undef otherwise

  Restrictions—all attributes are single valued only.

- **Read()**
  Reads all attributes on the current Domain object and returns a reference to hash of the attribute names mapped to arrays of values. Any attributes that are read that do not yet have a value in the domain record are a valid key in the returned hash, mapped to an empty array: []. Returns undef on error.

8

Database Administration

The chapter discusses the tasks related to managing the Integrated Services Directory database, and includes the following information:

- Monitoring the Integrated Services Directory Oracle database.
- Expanding the database.

8.1 Monitoring the Database

The Integrated Services Directory database is stored in an Oracle relational database.. This database requires regular monitoring and maintenance in order to function at peak capacity. The following sections explain how to perform some routine monitoring functions, as well as how to check for problems which can adversely affect performance of the InterMail system, most notably index fragmentation and insufficient allocation of tablespace.

8.1.1 Indexes and Tables

There are two types of database constructs in the Integrated Services Directory: tables and indexes. It is important to understand these in order to perform monitoring and maintenance functions in InterMail.

- A table contains zero or more rows, with each row made up of a number of columns. Each column is of a specific data type.
- An index refers to one or more columns inside a table. Each row in the table has a corresponding entry in the index. An index entry for a row consists of a copy of the values of the indexed column(s) in the row, plus a pointer to the row, called a RowID.

Indexes are directly related to tables and are used to optimize the querying process. Rows in tables are completely unordered; in order to find a row with a particular value in a given column without an index, the table has to be scanned row by row starting from the beginning of the table. On the other hand, indexes are organized as balanced trees; the relevant index entry for a particular column value can be fetched relatively quickly. The RowID in the index entry will point to the desired row in the indexed table. The relationship of indexes to tables is shown in Figure 6. Indexes and Tables.

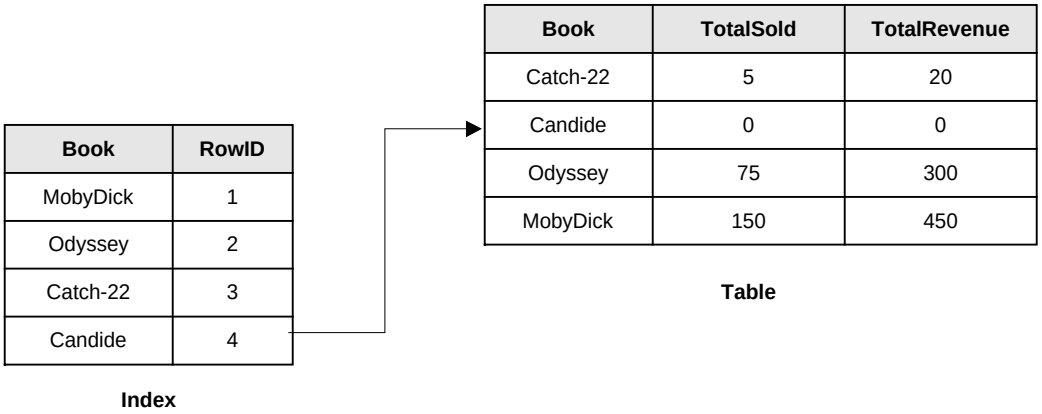


Figure 6. Indexes and Tables

Fragmented Indexes

In the course of directory operations, various creation and deletion processes such as adding and deleting accounts will affect indexes. Over time, an index will develop “holes” from deleted entries and the database will then consist of more and more space formerly occupied by entries for updated or deleted rows. This space is wasted and hurts system performance for three reasons:

- extra space is wasted in the database storing the “holes”
- extra time is used to read and write these “holes” to disk, and
- extra cache space is wasted because the “holes” are stored there also.

For example, if “Odyssey” were updated to “Illiad” in the table, the entry for the “Odyssey” in the index is “dropped” and no longer exists as an entry in the index. However the space that “Odyssey” occupied in the index is not reclaimed and forms a “hole” in the index. This results in fragmentation in the index, because Oracle does not efficiently reuse the space vacated in an index when rows in tables are deleted or updated. On the other hand, tables do not suffer from this problem; the empty space created when a row is deleted will subsequently be used when a new row is inserted.

With a fragmented index, searches take longer. When Oracle needs an index entry from the disk, it doesn't just read in the single index entry, it reads in the entire block of the index containing the desired index entry. For example, if the database needs to look up the “Candide” row in the table above, it reads in the entire block from the index that contains the entry for “Candide”. In our example, this block contains other index entries, as well as a “hole” for the now departed “Odyssey.”

When Oracle reads in an index block from the disk, it stores it in an in-memory cache. If some subsequent query needs an entry, and that entry happens to be in the same block, Oracle avoids having to do a disk read, and can use the index entry straight from the in-memory cache.

The cache stores a fixed number of blocks (with the actual number based on the size of the cache). In order to make room for a new block when the cache is full, Oracle flushes the least recently used block from the cache.

The less fragmented the index, the more index entries per block, which implies more index entries per block stored in the in-memory cache, and a greater chance that a needed entry will be in the cache and not require a disk read to fetch. Therefore the less fragmented the index, the greater the performance.

8.1.2 Reorganizing Database Indexes

Index reorganization is accomplished with the `imdbindexreorg` administrative command. This utility improves performance by eliminating the fragmented free space that builds up in Oracle indexes over time. As mentioned previously, eliminating index fragmentation also increases the performance of the database.

Warning! The `imdbindexreorg` command must be run during a maintenance window when all services that attempt to modify the database have been shut down.

To run `imdbindexreorg`, use the following example as a guide.

```
imdbindexreorg -tablespaces /tmp/table.backup -bloat 70
```

In this example, the command is run and produces an output file to `/tmp/table.backup` (or any other user-defined file) specifying those tablespaces that have been affected by the index reorganization and therefore require a (hot) backup. The command must be run on the host where the Integrated Services Directory runs. The `-bloat` switch allows you to fine tune the sensitivity of an index reorganization. When `-bloat` is specified, if an index is a specified percentage greater than its optimal size (in this case, 70%), a reorganization will be performed on that index. In this way, you can reorganize what is considered a critically bloated index and leave other indexes alone.

Next, `imdbindexreorg` reads the value of the `indexReorganizationTimeLimitMinutes` configuration key, which specifies the amount of time (in minutes) that the `imdbindexreorg` command is allowed to run. If `indexReorganizationTimeLimitMinutes` is set too high, index reorganization may take longer than the time defined by your maintenance windows. If this key is set too low and the `imdbindexreorg` command run too infrequently, performance will be hurt and space wasted because the command will not have enough time to defragment the indexes. You can override this key by specifying the `-timelimit` minutes argument.

For more details about `imdbindexreorg`, please see Chapter 12 of the *InterMail Reference Guide*.

8.1.3 Monitoring Oracle Tablespaces

Physical space in a database is partitioned into tablespaces. Each tablespace is composed of one or more files. All the storage for a particular index or table resides in one particular tablespace, although the storage might be split up amongst the several files that compose the tablespace. Figure 7. Tablespaces in a database represents the total space in a database. The space is separated into five tablespaces: SYSTEM, TEMP, RBS, POX01, and POD01.

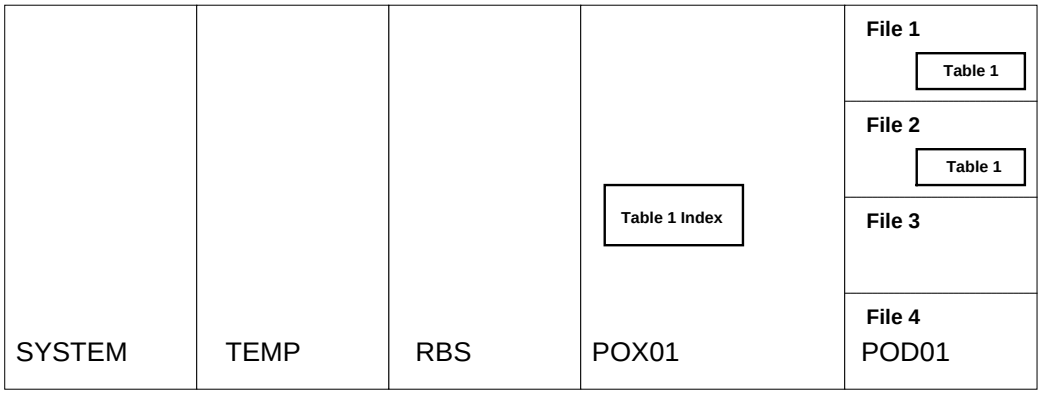


Figure 7. Tablespaces in a database

If we look at a logical partition of the Integrated Services Directory's Oracle database, we can clearly see what sort of difficulties arise in the management of available space. When a table is first created in the database, a certain fixed amount of space is allocated for the storage of its rows. This space is called the table's initial extent. When this space is exhausted, an additional increment of space called an extent is allocated. An extent is a multiple of a block (described earlier), so that if the database is constructed with 10k blocks, table extents (as well as indexes) are created in multiples of 10k. Every table has a particular pre-assigned extent size.

For example, if more space is needed for a given table that has an extent size of 80k, an additional 80k extent will be added. In order to add this additional extent, Oracle will scan the tablespace until it finds a free extent of contiguous blocks (unallocated space in the tablespace) of 80k or greater. The space for the new extent does not need to be contiguous to any of the table's existing extents, and can be added anywhere in the tablespace.

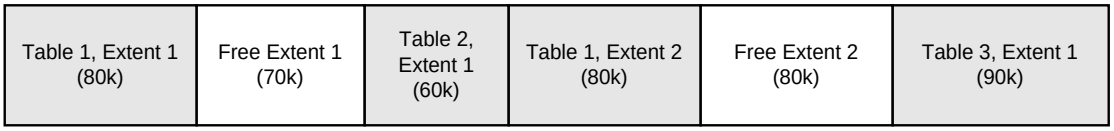


Figure 8. Space Allocation in a Tablespace

To further illustrate the way the Integrated Services Directory database allocates space, refer to Figure 8. Space Allocation in a Tablespace. In this example, a tablespace exists with three tables and two extents of available space. Remember, as a rule, extents are only added in available free extents and these free extents must contain enough blocks to accommodate an additional extent. So, if Table 1 which has two 80k extents (Table 1, Extent 1 and Table 1, Extent 2) needs to add an additional extent, it will search the tablespace for a free extent of sufficient size (80k). Upon finding that Free Extent 2 contains 80k, it will allocate that space, creating a third extent (Table 1, Extent 3).

This works well while there is a free extent to accommodate an additional extent, but when there is no available space, problems can occur. If, after adding an extent for Table 2, Table 3 needs to add an extent, it searches the tablespace for a 90k free extent and finds that only a 70k free extent exists; therefore, the table cannot add an extent. In this case, the tablespace needs to be expanded.

Another potential problem exists. Since database tables grow at different rates, it is possible that Table 2 may grow at a faster rate than Table 1 and need a new extent first. If this happens, Table 2 will search the database and might find that the next available extent is Free Extent 2 and it will allocate 60k of this 80k block. (The database uses a "first fit" algorithm when allocating a new extent for a table.) This creates a problem for Table 1, because now there is only a 70k block (Free Extent 1) and a 20k block (the remainder of Free Extent 2) remaining. Again, the tablespace will need to be expanded.

As you can see, the challenge in managing space in databases is to determine when you will need to expand a tablespace (by growing the size). In order to monitor the growth of Oracle databases, the Integrated Services Directory provides two tools to check on database usage and specifically estimate when the available free space will need to be expanded: `imdbspacecheck` and `imdbspacequickcheck`.

Using imdbspacecheck

The `imdbspacecheck` administrative command monitors the remaining free space in an Oracle database, and estimates when more space will be required. It warns when available space is at the point where it is advisable to add more. It calculates this by checking all possible permutations in the Integrated Services Directory database and also the historical growth of tables, which it will write to a file called:

```
imdbspacecheck.<DB_instance>.hst
```

Each time `imdbspacecheck` is executed, the `imdbspacecheck.<DB_instance>.hst` file will be updated. `imdbspacecheck` can be run while the Integrated Services Directory is in full operation and is intended to be run regularly, but it may take a long time to finish.

When executed, the administrative command will log an "Urgent" message when available space is at the point where more should be added. Two conditions will cause `imdbspacecheck` to log an "Urgent" message:

- when there is an object (table or index) whose last extent is 90% full (or another percentage set in the `extentFullWarningThresholdPercent` configuration key) and there is a possibility that it will not be able to add an extent.
- if there is an object whose last extent is predicted to fill within the next three weeks (set by the `extentGrowthAllowanceDays` configuration key), and there is a possibility that the object will not be able to add an extent.

To run `imdbspacecheck`, use the following example as a guide:

```
imdbspacecheck -thresholdpercent 70 -thresholddays 60 -dir
```

In this example, the command is executed with very conservative parameters. It will report back based on any combination of database growth that will prohibit any extent that is currently at 70% (set by `-thresholdpercent`) to add an additional extent within 60 days (set by `-thresholddays`). Setting these parameters overrides the configuration keys referred to previously.

Using imdbspacequickcheck

Because `imdbspacecheck` is memory-intensive and can take a long time to run, the Integrated Services Directory provides a second administrative command, `imdbspacequickcheck`, to quickly check free space in a database. The `imdbspacecheck` command checks every possible combination of table growth, calculating all permutations and referring to stored historical data, to determine when additional space will be needed. `imdbspacequickcheck` performs a much more cursory check, only checking on current conditions.

The `imdbspacequickcheck` command searches the database to determine the result if any of the existing tables in the database needs an additional extent in isolation. Basically, `imdbspacequickcheck` can warn of an impending space crisis and serves as a last-chance safety net to catch space problems before a crisis erupts.

Like `imdbspacecheck`, `imdbspacequickcheck` logs an "Urgent" message if there is an object (table or index) whose last extent is over a configurable high-water mark (system default setting is 90% full), and the object cannot grow. "Static" objects, that is table and indexes known not to grow, are exempt from this check.

In typical system operation, you should schedule `imdbspacequickcheck` as a frequently run cron job, executed approximately once every thirty minutes. Running `imdbspacequickcheck` imposes an insignificant load on the monitored system. Therefore it can be run much more frequently than `imdbspacecheck`.

To run `imdbspacequickcheck`, use the following example:

```
imdbspacequickcheck -thresholdpercent 70
```

As in the example for `imdbspacecheck`, a conservative strategy is followed where `imdbspacequickcheck` overrides the `extentFullWarningThresholdPercent` configuration key with `-thresholdpercent 70`.

8.2 Expanding the Oracle Tablespace

Once you have determined that the Integrated Services Directory database needs to grow and you have the available hard disk space to do so, you will need to run the `imdbspacegrow` command to expand the available tablespace.

Using imdbspacegrow

The log entries created by `imdbspacecheck` and `imdbspacequickcheck` give warning of an impeding space crisis. When this happens, run `imdbspacegrow` to add space to the identified tablespace.

The following example illustrates how to run `imdbspacecheck`. As this command is interactive, standard output will be shown when it occurs in the process.

```
imdbspacegrow -dryrun
```


When `imdbspacegrow` is executed with the `-dryrun` argument, as in this example, the command does not actually perform the database expansion and instead logs a description of what would happen if you grew the database. It is generally a good idea to perform a dry run of `imdbspacegrow` to see the results before actually adding the space.

After the test `imdbspacegrow` command runs, status information is displayed to standard output:

```
Database name:       IMM1
Database access:     local
ORACLE_HOME:         /disk2/oracle/7.3.3
Ready to proceed? (Y/N) [yes]
```

If the information presented is correct, type `yes` and press Enter; if not, type `no` and press Enter. You will be offered the opportunity to edit this information.

A list of tablespace names is displayed, and you are prompted to enter the name of the tablespace whose size you want to increase. The following is an example of the output that can be displayed; however, your exact display will differ.

```
The database is partitioned into 5 tablespaces.

TABLESPACE LIST

1) SYSTEM
2) TEMP
3) TB1
4) TB2
5) TB3

Please select a tablespace from the list by number: [5]
You selected tablespace TB3.

After you make a selection, InterMail will report the current status of the tablespace
that you chose.

For your information, the tablespace TB3 is currently composed of the following files:

FILE NAME                                   SIZE IN BYTES
/disk2/oracle/admin/D1ALE733/TB3.dbf        32522240 bytes
Ready to continue? (Y/N) [yes]
```

The command prints information about the recommended sizing for the additional file.

Then you will need to specify the size of the new tablespace file:

```
Enter the size of the file to be added to the tablespace: [0] 32522240
------------------------------------------------------------
```

You will then be asked to enter the full pathname for the new file. Typically, the `.dbf` extension is used for Oracle database files.

```
Enter full path name of new file:
/disk2/oracle/admin/D1ALE733/TB3_2.dbf

A file named /disk2/oracle/admin/D1ALE733/TB3_2.dbf of size 32522240 bytes will be
added to tablespace TB3.

Proceed? (Y/N) [yes]
------------------------------------------------------------
```

At this point, the Integrated Services Directory will typically add the new file to the tablespace `TB3`. In this example, since `-dryrun` was specified, the results of the operation are displayed, but the file will not be written to disk.

```
Oracle added a new file named "/disk2/oracle/admin/D1ALE733/TB_3.dbf" to tablespace
"TB3". The actual size of the file is 0 bytes.

NOTE: this differs from the size you requested (32522240).

The tablespace TB3 should be hot backed up now, since its structure has been altered
in a significant way.
```

While `imdbspacegrow` does not interfere with the normal operation of InterMail, it is highly suggested that you perform a hot backup immediately after creating more tablespace. Therefore, it is advisable to run `imdbspacegrow` during a maintenance window or other non-peak time.

8.2.1 Using imdbspacereport

To see an overview of all database information, use the `imdbspacereport` utility, which prints out information about the database, including the exact Oracle version used, the value of all database parameters, and information about space usage in the database.

Note: *In addition to providing an overview of all database information, when* `imdbspacecheck` *or* `imdbspacequickcheck` *warn of low tablespace,* `imdbspacereport` *can be run to obtain a snapshot of space allocation in the database in order to better understand the depth of the impending crisis.*

To run `imdbspacereport`, type the following:

```
imdbspacereport -dir
```

9

Backup and Recovery

This chapter provides an overview of the steps needed to backup and restore the Integrated Services Directory. The topics covered in this chapter include:

- Overview of backup and restore procedures for the Integrated Services Directory — options and strategies available, and recommended hardware required for backup.
- Testing backup and restore procedures.

The primary function of this material is to familiarize you with the important concepts behind backup and restore. You can then apply these concepts in your specific backup and restore environment. This is not a prescriptive, total step-by-step method. Each site will have specific issues concerning their provisioning system, the size of their database, available hardware, and available man hours, that must be addressed on a case-by-case basis.

9.1 Overview

As the sole source of directory information for InterMail, InterRADIUS, InterManager, and InterLDAP, the Integrated Services Directory is an important component to back up on a regular basis.

The frequency of backups is dependent on the rate of change of the data in the Integrated Services Directory. If your site creates or modifies a large number of accounts, you should back up this data more often. Because the speed with which the system can be recovered depends on the number of changes that have occurred since the previous backup, you should use recovery time as a factor in creating your backup schedule.

Environment backups should include regular incremental backups of non-InterMail files and occasional backups of InterMail binaries and configuration files. The procedures discussed here do not backup non-message data as a part of a continuous backup strategy. Examples of InterMail components that may change include the Oracle version, InterMail patches and upgrades, and configuration files.

9.2 Types of Backup

There are two types of backups of the Integrated Services Directory:

- Occasional complete image backups of the host system software, Oracle software, configuration files, and other resident software.
- Regularly scheduled hot backups of the Integrated Services Directory database.

Complete Image Backups

Complete image backups of the Integrated Services Directory (copying the entire file system, or at least the Oracle portion) can only be carried out when the service is down, and should be performed during regularly scheduled maintenance windows and/or as opportunity allows when servers are down for other reasons.

Complete image backups should be made any time you change the software configuration to the Integrated Services Directory host, including the first occasion you install the Oracle software on this host.

Hot Backups

Scheduled hot backups of the Integrated Services Directory are implemented by a cron job (at least daily), while journaling and journal snapshots are ongoing.

9.3 Making a Hot Backup

The following items of data are crucial to fully recover Oracle, and should be part of all hot backups:

- Critical Data Files
- Archived Redo Logs
- On-line Redo Logs

If any one of them is missing, the results of some, if not all, committed transactions will be lost. In Integrated Services Directory terms, this translates into new accounts being lost.

Critical Data Files

The critical data files contain the database tables, the *rollback segments*, and the *system tablespace*. The database tables contain data associated with all Integrated Services Directory objects. The rollback segments are vital to being able to successfully complete recovery. The system tablespace basically contains the “map” of the database that describes the schema and what tables are in which blocks in what files. All these are critical pieces of information.

Not all data files are critical, in particular the index data files and the temporary tablespace data files. Indexes can readily be recreated from the tables. Temporary tablespaces are merely “scratch” areas that can be quickly recreated.

Archived Redo Log

An archived redo log is a copy of a former *on-line redo log* that contains all the changes to the database during a particular period of time. This period is typically about five minutes. Archived redo logs are critical to restoration when a critical data file has been lost. The archived redo logs are needed to "roll forward" a backup copy of the data file. If one of the needed archived redo logs is missing, it is analogous to a link missing from a chain. A backup copy can only be rolled forward to the point in time where the missing log started recording changes.

Archived redo logs should be protected by copying them to the network appliance shortly after they are created. This ensures two copies always exist: one in Oracle's `archive_dest` directory and one on the network appliance, or one in Oracle's archive_dest and one among the on-line redo logs.

On-line Redo Logs

The on-line redo logs contain the most recent set of changes to the database. One particular on-line redo log will be the "current" log where Oracle is recording all new changes. If the on-line redo logs are lost, in particular the "current" log, the most recent set of changes to the database will be lost.

The on-line redo logs are protected by Oracle's software mirroring feature.

9.3.1 Backup of Critical Data Files

You need to run the `imdbhotbackup` administrative command as a cron job to make backup copies of critical data files.

The syntax for this script is:

```
imdbhotbackup [-dryrun] [-cronjob]  [-sleeptime N] [-backupindexes]
      [-othertemporarytablespaces Temp1 Temp2. . .] [-backuptemp]
      [-backupdestdir directory_name] [-totalbackupskept N]
      [-backupcopydestdir directory] [-totalcopybackupskept N]
      [ [ -backupdirexception file_name directory_name] ...]
      [-logdir directory] [-backupcommand command] -dbname SID
      -id USER/PASS[@SID] [-dir]
```

For a detailed explanation of the arguments for this command, see Chapter 12 of the *InterMail Reference Guide*.

`imdbhotbackup` can, by default, only handle backups to disk. To handle tape, write a customized command to backup a file to tape, and pass that command into `imdbhotbackup` by using the `–backupcommand` flag.

Here is a sample crontab entry for the "oracle" Unix user for backup to tape. The crontab entry belongs to Oracle so that there will be no permission problems accessing data files.

```
30 3 * * * ORACLE_HOME=/volX/oracle/7.3.2 ORACLE_SID=IMDX
LD_LIBRARY_PATH=/volX/imail/lib:/volX/oracle/7.3.2/lib /volX/oracle/imdbhotbackup –
cronjob –sleeptime 120        –othertemporarytablespaces TEMP –backupdestdir
/volY/backup –totalbackupskept 1 –logdir /volY/backup/LOGS –backupcommand
/usr/local/bin/backuptotape –dbname IMD1 –id imail/imail
```


By default, the oracle user is not a member of the InterMail group and will not be able to access the Configuration Database. Therefore the `–logdir` flag is used to prevent the command from trying to lookup `logDir` in the Configuration Database and failing.

Example: Backing Up to Tape

Even though all the data files are being backed up to tape, the `–backupdestdir` is still needed. This directory stores the following:

- Scripts to recreate indexes and temporary tablespaces
- A backup history file maintained by `imdbhotbackup`
- A backup copy of the database control file

An example of what the customized `/usr/local/bin/backuptotape` might look like:

```
#! /bin/sh
if echo "$1" | cpio –ocBv > /dev/tapexas
   then
     exit 0
   else
     exit 1
   fi
```

Example: Backing Up to Disk

In this example, a disk array is dedicated to holding backups. It is desired that two backup copies be kept on the array: the latest backup and the previous one. The disk array is mounted at `/volB`.

Here is a sample crontab entry for the "oracle" Unix user. The current backup will be stored in `/volB/ ORACLE_HOT_BACKUP_IMD1/CURRENT`, and the previous backup will be stored in `/volB/ORACLE_HOT_BACKUP_IMD1/PREVIOUS.1`. The crontab entry belongs to Oracle so that there will be no permission problems accessing data files.

```
30 3 * * * ORACLE_HOME=/volX/oracle/7.3.2 ORACLE_SID=IMD1
LD_LIBRARY_PATH=/volX/imail/lib:/volX/oracle/7.3.2/lib /volX/imail/bin/imdbhotbackup –
cronjob –sleeptime 120 –othertemporarytablespaces TEMP –backupdestdir /volB –
totalbackupskept 2 –logdir /volB/BACKUP_LOGS –dbname IMD1 –id imail/imail
```

9.3.2 Copying Archived Redo Logs

Oracle records all changes to the database in the current on-line redo log. At some point this log, which is a fixed size becomes full. Oracle then switches to a different redo to record changes in. The previous "current" log is then copied by Oracle to the `archive_dest` directory. The copy in `archive_dest` is known as an *archived* redo log.

The `imdbcopyarchredo` utility addresses two problems with archived redo logs. Number one, archived redo logs are a single point of failure that can prevent full recovery. Losing one archived redo log is enough to make full recovery impossible. `imdbcopyarchredo` addresses this problem by copying archived redo logs to the network appliance soon after they are created so that two copies exist on independent devices.

Another problem with archived redo logs is that, left unchecked, they eventually fill up the `archive_dest` device. When this occurs, the database literally freezes up. `imdbcopyarchredo` addresses this problem by regularly pruning old archived redo logs from the `archive_dest` directory.

`imdbcopyarchredo` is intended to be run as a cron job at an interval that is a small multiple of the average time between log switches. The idea is to make sure `imdbcopyarchredo` runs frequently enough so that an archived redo log is copied before its on-line redo log original gets overwritten.

9.3.3 Crashing during a Hot Backup

If a crash occurs while `imdbhotbackup` is running, there is a good chance the database will not come back up cleanly when the machine is rebooted. If this does happen, all the Oracle background processes will be successfully launched, so doing a "`ps`" will reveal that database processes are running, but the database will not be accepting any connections.

The problem is that a crash during a hot backup of a data file leaves its file header in an inconsistent state. Oracle detects this at startup, issues an error saying "ORA-01113: file X needs media recovery," and refuses to fully open the database.

To get the database fully operational, log in as the "`oracle`" Unix user, and run Oracle's `svrmgrl` utility as follows. Make sure the `ORACLE_HOME` and `ORACLE_SID` environment variables are set properly first.

```
% echo $ORACLE_HOME $ORACLE_SID
% $ORACLE_HOME/bin/svrmgrl
>connect internal
>startup
ORA-01113: file N needs media recovery
ORA-01110: datafile N: '/foo/bar/snafu.dbf'
>recover datafile '/foo/bar/snafu.dbf'
>alter database open;
```

If another ORA-01113 error is issued, just repeat the `recover datafile` and `alter database` steps until finally the database stops complaining. Use an `exit` command to terminate `svrmgrl`.

9.4 Restoring from Backup

The backup strategy should ensure that the database can be fully recovered if all the files on any one device are lost. The procedures here assume nightly hot backups, and that a day's worth of archived redo logs are available in `archive_dest`

Loss of the Oracle Home Directory

In this section we discuss what to do if the "`oracle`" Unix user's home directory is lost. If installation defaults were taken, all the Oracle binaries and libraries, plus database parameter files, have been lost.

The Oracle binaries, located under the `$ORACLE_HOME` directory, are not backed up nightly. Recovering in this situation will be painful if another copy doesn't exist within easy reach. Try copying Oracle binaries from another machine. If that isn't a possibility, the Oracle binaries --- absolutely everything under `$ORACLE_HOME` should be committed to tape.

Recreate the "oracle" home directory, including `.cshrc` and `.profile` files with `TNS_ADMIN`, `ORACLE_SID`, and `ORACLE_HOME` set properly. `LD_LIBRARY_PATH` must include `$ORACLE_HOME/lib`.

In the following instructions, assume that HOME is set to the `oracle` user's home directory path name.

1. Restore `$ORACLE_HOME` from a copy.
2. Make directories `$HOME/admin/$ORACLE_SID/dump_dest/bdump`, `…/cdump`, and `…/udump`. Make `$HOME/admin/$ORACLE_SID/pfile`.
3. Copy the hot backup file `BAK.init$ORACLE_SID.ora` to `$HOME/admin/$ORACLE_SID/pfile/init$ORACLE_SID.ora`.
4. Create a symbolic link:

   ```
   ln –s $HOME/admin/$ORACLE_SID/pfile/init$ORACLE_SID.ora
   $ORACLE_HOME/dbs/init$ORACLE_SID.ora
   ```

5. Copy the hot backup file `BAK.configIMDB.ora` to `$HOME/admin/$ORACLE_SID/pfile/configIMDB.ora`.
6. Copy the hot backup files `BAK.tnsnames.ora` and `BAK.listener.ora` into `$ORACLE_HOME/network/admin/tnsnames.ora` and `…/listener.ora`.

Loss of System Tablespace

The system tablespace file is typically named `SystemTablespace.dbf`. The system tablespace records the schema and structure of the database, and Oracle cannot start the database without it.

Use the following procedure to recover from Oracle system tablespace file failures. Use this procedure only if you have a thorough understanding of the process and if you are certain that only the data files you are restoring are the ones affected and that you are operating on the correct tablespace. This type of restoration is only effective when a single tablespace has been damaged due to media failure. The advantage of restoring only data files is that restoration time is reduced compared to a full oracle restore.

It is possible to run the data file recovery over a live system recovering only the specific tablespace. However, if any InterMail tablespaces (including TEMPORARY, SYSTEM, or RBS) are involved, the system should not be mounted and open for InterMail operation since the database objects required will be affected. It is important to note that ONLY complete recovery may be used for data file recovery. If you do not have a complete set of redo logs available for Oracle complete recovery, you are required to perform a full Oracle recovery.

1. Copy `BAK.SystemTablespace.dbf` from your last hot backup to `SystemTablespace.dbf` in the proper directory.
2. Run Oracle Server Manager and start the Oracle instance in restricted mount mode.

   ```
   SVRMGR>  CONNECT INTERNAL
   SVRMGR>  STARTUP RESTRICT MOUNT;
   ```

Note: *If you need to restore the database to a different path or file system you will need to issue an ALTER DATABASE command with the CREATE DATAFILE option on a mounted (but closed) database to redirect the file paths for the tablespace data files. A detailed discussion of this process and when and how to use it can be found in the Oracle Backup and Recovery Documentation.*

---

3. You must have all redo log files (including the online redo logs) without any gaps in the directory specified in the `init$SID.ora` file. You may run the following in server manager (where datafile1… are the file system names for the data files including the path as listed in SYS.DBA_DATA_FILES). This step will take some time.

```
SVRMGR>  RECOVER AUTOMATIC DATAFILE 'datafile1', 'datafile2',… ;
SVRMGR>  ALTER DATABASE OPEN ;
```

Note: *If your redo log files are not in the original path/file system, you must also specify the path using the FROM path clause for the RECOVER command. Also, other recover options are available if the above commands cannot recover the database. A detailed discussion of the RECOVER command can be found in the Oracle Backup and Recovery documentation.*

---

4. When the process completes and it is successful, you can then shutdown the instance using Server Manager.

```
SVRMGR>  SHUTDOWN IMMEDIATE
```

It is advisable to restart the Machine to ensure consistency and stability.

5. Start the Oracle database.

Loss of Un-Backed Up Temporary Tablespace

By default, `imdbhotbackup` does not backup temporary tablespaces. It does include scripts with the nightly hot backup that can be used to quickly and easily recreate each one. For every temporary tablespace that isn't backed up, there will be scripts `FilesOffline`*`TEMPTBS`*`.sql` and `RecreateTempTbs`*`TEMPTBS`*`.sql` included in the hot backup.

Create all necessary directories.

```
Svrmgrl
>connect internal
>startup mount
>@/archive/store1/HOT_BACKUP_COPIES_IMD1/CURRENT/FilesOfflineTEMPTBS.sql
>recover automatic database
>alter database open
>@/archive/store1/HOT_BACKUP_COPIES_IMD1/CURRENT/RecreateTempTbsTEMPTBS.sql
```

Loss of a Data Table Tablespace

Files making up data table tablespaces are included in the hot backup. For each corrupted data table tablespace file, copy a hot backup copy to where the on-line copy should be.

```
Svrmgrl
> connect internal
> startup mount
> recover  automatic database
> alter database open
```


Loss of Rollback Segment Tablespace

By default, there is one rollback segment tablespace in an InterMail database containing all the database's rollback segments. Rollback segments are vital to the Oracle database recovery algorithm. Files making up rollback segment tablespaces are included in the hot backup.

For each corrupted rollback segment tablespace file (typically there will be just one, `RollbackTablespace.dbf),` copy a hot backup copy to where the on-line copy should be.

```
Svrmgrl
> connect internal
> startup mount
> recover  automatic database
> alter database open
```

Loss of Index Tablespaces

The index tablespaces are not included in the hot backup by default. The hot backup does include scripts for recreating all indexes from scratch, `IndexFilesOffline.sql` and `RecreateIndexTablespaces.sql`.

Make absolutely certain that `immssgc`, the garbage collector, is not and can not run until all the indexes are restored. Comment out the InterMail user's crontab entry for `immssgc`, and do a "`/bin/ps –ef | grep immssgc`" to make sure it is not running.

```
Svrmgrl
> connect internal
> startup mount
> @IndexFilesOffline.sql
> alter database open
> @RecreateIndexTablespaces.sql
```

Loss of an On-Line Redo Log

The on-line redo logs should be mirrored in hardware or software. The goal is to make it unlikely both copies of an on-line redo log will be lost. If that happens, the database cannot be fully recovered.

When one mirror is lost, it can be restored by just copying the other mirror. By default, redo log files have names like `RedoGrpNMemX.dbf`. Files that have equal values of N in their redo log names are mirrors of one another. The X is the mirror number, 1 to the number of mirrors.

Loss of the archive_dest Director

All the archived redo logs have been lost. Take a hot backup as soon as possible.

Loss of the Above in Combination

If several files are lost belonging to several of the categories, combine the instructions from each section as follows:

1. Do all the steps from each section up to the point `svrmgrl` is run.
2. Run `svrmgrl`
3. Do all the subsequent steps from each section up to the point the "`recover automatic database`" statement is issued.
4. Run "`recover automatic database`".

Index