The Print Systems Standard Design Document

Version 1.0

MIT Project Athena

April 7, 1989

Copyright ©1988 by the Massachusetts Institute of Technology. All Rights Reserved.

Contents

1 Introduction **1**

1.1 Requirements 2

1.2 Terminology 4

2 Palladium System Overview **9**

2.1 Client-Server Model 9

2.2 Name Service-based Server Advertisements 10

2.3 RPC-based Server Access Protocols 10

2.4 Embedded Tasking Systems 11

2.5 Palladium Print System Model 11

2.6 Job Processing 16

3 Palladium User Interface **19**

3.1 pdpr - Printing A Job 19

3.2 pdpq - List Job Status/Information 25

3.3 pdrm - Cancel Print Job 26

3.4 Administrative Interface 26

3.5 pdstart - Start A Server/Supervisor 27

3.6 pdinit - (Re)Initialize An Object 27

3.7 pdenable - Enable An Object 28

3.8 pddisable - Disable An Object 28

3.9 pdpause - Pause An Object 29

3.10 pdresume - Resume An Object 30

3.11 pdshutdown - Shutdown an Object 30
3.12 pdstop - Stop An Object 31

4 Pdtk - Palladium Client Toolkit **33**

4.1 Print Job 34
4.2 Modify Print Job 35
4.3 Cancel Job 36
4.4 Job Status 37
4.5 List Jobs 38
4.6 Read Object Attributes 39
4.7 Clean Jobs 41
4.8 Promote A Job 42
4.9 Register An Object 43
4.10 Unregister An Object 43
4.11 Modify Attributes 44
4.12 Stop An Object 45
4.13 (Re) Initialize An Object 46
4.14 Shutdown An Object 46
4.15 Enable An Object 47
4.16 Disable An Object 48
4.17 Pause An Object 49
4.18 Resume An Object 50

5 Pdlib — Palladium Server Interface **51**

5.1 Introduction 51
5.2 Pdlib Function Arguments 53
5.3 Pdlib User Functions 55
5.4 Pdlib Administration Functions 59

6 System Management **71**

6.1 Configuration Management 71

6.2 Resource Management 75
6.3 Failure Management 76
6.4 Security Management 78
6.5 Accounting and Quota Management 79
6.6 Management Functions 80

7 Palladium Server Design **83**
7.1 Server Components 83
7.2 User Requests 85
7.3 Administrative Requests 86
7.4 Server Operation and State 87
7.5 Job State 87

8 Palladium Databases **89**
8.1 Print Server Relations 90
8.2 Functional Interface 96

9 Palladium Server Job Scheduler **103**
9.1 Scheduling functions 103
9.2 Miscellaneous functions 106

10 Palladium Printer Supervisor **109**
10.1 Supervisor Status 111
10.2 Supervisor User Functions 111
10.3 Supervisor Administrative Functions 113

11 Conclusion **119**

Chapter 1

Introduction

Recent advances in document publishing products and printer technology have outstripped the support provided by traditional print systems. Users may create documents with complicated formats, fancy fonts and embedded graphics on an inexpensive personal computer. But these documents may need to be printed on an expensive sophisticated printer that must be shared by many users. It is often very difficult for users on PCs to print documents on shared printers using traditional printing systems. Even when the documents can be submitted to the shared print system, the print system may not fully support the capabilities of the new sophisticated printers.

The Unix Print Spoolers used by Athena are examples of traditional print systems. While they have been modified to support access to remote printers, this support requires manual non-centralized configuration management. (i.e. every node must have a local file which contains information about any printers on other nodes which the local users may remotely access.) These systems do not support the printing of documents submitted from non-Unix systems. These systems do not reasonably support printers which use bi-directional asynchronous communications protocols (e.g. high-end laser printers). These systems do not support all of the current and near-term future capabilities of Postscript laser-printers. A new Distributed Print System must be designed to support these new requirements and opportunities.

To solve the problems of distribution and sophistication of print systems, representatives to ANSI, ECMA and ISO have been working together on the definition of an up-to-date Distributed Printing Standard. The resulting standard will support distribution of users of a Print System while providing equitable access to the shared resources offered by the system. This system is intended for use in a heterogeneous environment. ANSI and ECMA have made sufficient progress that we can use their work as a foundation for an implementation of a distributed print system for Athena.

ECMA TC32/TG5 has been working on a standard for Print Services for nearly two years. The Second Draft of the working document was released in November of 1987. We expect a third and final working draft to be available by mid-1988. This work has been favorably reviewed and supported by ANSI X3V1. Recently a new work item for Distributed Printing Applications has been proposed within ISO/IEC JTC1 SC18/WG4. When this work item is approved, we expect JTC1 to use the ECMA work with little or no modification. Athena Distributed Print System

work should follow the current work of ECMA to ensure that future Athena Print Systems can interoperate with other industry print systems.

1.1 Requirements

Athena should solve these remote printing problems by implementing a well defined, reviewed, industry compatible distributed print system. The following requirements provide further detail to guide the design and development of such a system.

1. The print system must be designed for a distributed environment.
2. This print system should give a user equitable access to all shared printers.
3. This print system should provide transparent access to personal resources (e.g. personal non-shared printers attached to workstations).
4. Development of print system clients and servers must be compliant with the developing ECMA/ISO Print Service Standard.
5. These clients and servers should be implemented for BSD 4.3, but be easily ported to System V.
6. Clients, Servers and Supervisors should be implemented such that they do not unnecessarily preclude the use of 8-bit or greater character codes in files to be printed and in the associated job request parameters.
7. The design must not require additional functional support at the workstation (e.g. file transfer queuing, s tagging, local job ids, etc.).
8. The system must be simple to operate both locally and remotely. It is vital that facility operations staff not be required to be "highly computer literate". It is important not to assume that operators know about the underlying system. They most likely will not be able to perform complicated service operations.
9. The system must be simple to manage and maintain from a central location. It is important that the central administration of the service be as simple and automatic as possible. Any work that scales on a per-workstation basis will ultimately be unmanageable. There must be nothing that required central administration to "visit" each server or printer (aside from standard maintenance, paper, etc.).
10. No access to print system operations or information will be allowed without proper authentication and authorization of the user.
11. All necessary printer and print server state information must be available to system managers and users.
12. Printer usage information must be logged and available for system management and accounting purposes.

13. The print system should support quotas for the control of print system usage.

14. It must be possible to stop and start print servers, print supervisors, and printers themselves either remotely or locally, without changing the availability of any other system services.

15. It must be possible to either add or remove servers, supervisors and/or printers without rebooting the entire print system or losing any print jobs.

16. It must be possible to reconfigure the system (by adding or removing servers, supervisors and/or printers) to ensure the successful completion of existing print jobs in the system.

17. It must be possible for the operator to redirect print jobs to different print servers and/or printers.

18. The system should be designed and implemented such that the interfaces between clients and servers are independent of the underlying communications protocol. (e.g. use RPC and other ULA services)

19. The system should be designed to be quickly and easily implemented. It should be easy to maintain and extend.

20. The servers should provide reasonable response time to requests, high accessibility to requestors and the illusion of asynchonous handling of multiple requests.

21. The system must be able to interoperate with existing BSD 4.x and System V print systems.

22. The design must support development of clients and servers that are forward compatible with future revisions of these components.

23. The design must allow different RPC system implementations to be used. The choice may be made either at source-code level, build-level or configuration.

24. The design must allow different Bind Service implementations to be used. The choice may be made either at source-code level, build-level or configuration.

25. The design must allow different Name Service implementations to be used. The choice may be made either at source-code level, build-level or configuration.

26. The design must allow different Authentication Service implementations to be used. The choice may be made either at source-code level, build-level or configuration.

27. The design must allow different Authorization Service implementations to be used. The choice may be made either at source-code level, build-level or configuration.

28. The design must allow different File Service implementations and associated file transfer methods to be used. The choice may be made either at source-code level, build-level or configuration.

29. The design must allow different Accounting Service implementations to be used. The choice may be made either at source-code level, build-level or configuration.

30. The design must allow different Notification Service implementations to be used. The choice may be made either at source-code level, build-level or configuration.

31. The design must allow different Scheduling Algorithm implementations to be used. The choice may be made either at source-code level, build-level or configuration. Scheduling algorithms are outside the scope of these requirements. Scheduling, internal job DBs, printer queues, etc. are implementation-specific concerns.

1.2 Terminology

The following terminology has been extracted directly from the current printing terminology glossary for ANSI X3V1 (Print Services, ODA/ODIF, SPDL, SGML, DSSSL). This glossary is expected to be used by ISO SC18/WG4 (Print Services), ISO SC18/WG8(ODA/ODIF, SPDL, SGML, DSSSL), ECMA TC29 (ODA/ODIF, SPDL, SGML, DSSSL) and ECMA TC32-TG5 (Print Services).

character:

1. An information unit and a member of a set of elements upon which agreement has been reached and that is used for the organization, control, or representation of information; there are two kinds of characters: (i) control characters, whose occurrence in a particular context initiates, modifies or stops the recording, processing, transmission or interpretation of data, and (ii) graphic characters, which are those providing the meaning of the data.
2. A distinguishable, visible element used for the presentation of information in human-readable form.

character collection: a collection of design-independent characters, defined by an enumerated set of structured character names

defined character: a character having a structured name

ISO named character, IR character: a character having an IR ISO-registered name, i.e., character registered with naming authority IR.

named character collection: a character collection having a structured name as provided for in Part 2 of this standard.

ISO named character collection, IR character collection: a character collection having an IR (ISO-registered) name, i.e., a character collection registered with naming authority IR.

creator: the software process that converts a revisable form document into a final form document, or adds final form information to the revisable form information already present in the document.

crop: to cut off or remove the top or extremity of a picture or graphic symbol.

document: when not otherwise qualified, the electronic form of something which may be rendered in visible form so as to present information upon any subject. In this Standard, when not otherwise qualified, a document is assumed to be the electronic form of a document, in a form acceptable to the print server(s) to which the document is (to be) sent. Qualified terms that may be used include:

document description: the electronic representation of a document, in either revisable or final form.

final form document: a document which is in a form suitable for rendering by a print server. As used in this International Standard, this term implies that all decisions regarding the rendering of the document have been made, and generally implies that the document is no longer in a form that is amenable to being edited.

Revisable form and final form are not mutually exclusive concepts; some data formats, including ISO 8613, allow both revisable and final form information to be present in a single document at the same time. Such a document would be simultaneously a revisable form and a final form document.

revisable form document: a document which is in a form suitable for editing and revising by some properly equipped system. This term generally implies that, in addition to its explicit content, the document contains information which identifies its logical structure.

SPDL document: a particular instance of a final form document, encoded in the Standard Page Description Language, and thus in a form suitable for printing by any print server(s) capable of rendering an SPDL document. [Will be changed to SPF if that form of descriptor is retained by ECMA TC29-TGPF].

SPDL document description: a final form document which has been encoded in the Standard Page Description Language defined in this International Standard; an instance of the use of SPDL.

formatted form: (ODA) a form of representation of the document that allows presentation of the document as intended by the originator and that does not support editing or (re)formatting.

formatted processable form: (ODA) a form of representation of the document that allows presentation of the document as intended by the originator and also supports editing and (re)formatting.

processable form: (ODA) a form of representation of the document that allows editing and (re)formatting.

document creator: the software process that converts a revisable form document into a final form document. Unless otherwise qualified the final form document will be assumed to be in SPDL form.

font: a set of characters, each in a particular rendering (usually but not necessarily of consistent design, size, appearance, writing system, etc.), together with a set of attributes associated with individual characters and with the set as a whole; it includes the information required to characterize and identify itself, to assist in the positioning of characters for text composition, and to define the images of the characters.

font reference: the collection of information required to identify and/or characterize a font which would best meet the user's requirements, generally consisting of attributes from a font.

image (n): a unit of visible information.

image (v): to create a visible image (of a character or graphical element) as part of the presentation process. As distinguished from present or render, the term image will be used when the primary concern is that part of the process which creates the visible image.

logical printer: a logical entity that permits a particular set of characteristics and capabilities (e.g., location, resolution, medias available, etc.) to be grouped and referenced as a single entity. The logical printer may represent one or more printing devices having the same capabilities and characteristics; alternatively, a single printing device may be represented by more than one logical printer, where selection of a particular logical printer implies some specific form of processing to be applied to the document, or a particular set of resources, that may be made available, e.g., media weights that may be (manually) loaded if the relevant logical printer is selected.

logical service: a particular set of print service features and/or default values that are predefined on a print server. (Note - a particular print server may offer more than one logical service to its clients).

medium: the physical surface upon which a document is rendered in visible form by a presentation device. Examples include paper and the display screens of video display terminals.

page description language: one of a group of print formats based on the concepts of modern postfix programming languages.

picture: an image which can be considered in isolation from other images, e.g., a page or a diagram on a page.

presentation: the act of rendering electronically encoded information visible to human perception.

print access protocol: the application-level protocol between any print client and any print server.

print client: the software entity that acts on behalf of a user to convey service requests to a print server and to deal appropriately with responses from the print server that arise from the submitted requests.

print format: the generic name for any data syntax with which a final form document may be encoded. Based on the underlying concept of the data syntax the following terms are used.

print job: when a print server accepts a print request from a client, a new print job is created. The print server then generates a print job identifier that is unique within its environment, and returns it to the client to signal acceptance of the request, and to enable the client to interrogate status of the job subsequently.

print request: an operation invoked by a print client to cause a print server to print a document(s). Parameters are sent with the print request to identify the document, and to specify and control the act of printing and producing the document(s), such as number of copies to print, whether

to staple the hardcopy document, etc. The document to be printed may accompany the print request, or the print request may include a reference (pointer) to the document, indicating that the print server is to fetch the document using some other protocol. Actual access and transport of the document to a print server may be accomplished by other protocols that are outside the scope of this document.

service request: within the print access protocol (standard), this term refers to a request for any type of printing operation.

print resource: will be defined to include both logical and physical attributes.

print server: the entity that accepts print requests, together with accompanying documents (or references to them), from print clients and prints the documents in accordance with parameters delivered with the print request and/or incorporated within the document description.

print service: to be defined; a print service is provided by a print server.

print system: to be defined.

printing: unless otherwise qualified or restricted, the act of printing may include any form of output which transforms the electronically encoded and stored form of a document into a visible representation. This includes, but is not limited to, hardcopy printers, plotters, display devices, and microfilm/microfiche generators.

printing device: a physical entity which is capable of rendering documents.

printing instructions: parameters sent to a print server to control or guide and assist the print server in producing the requested document in the form requested by the user. The set of printing instructions are subdivided into groups based on how they may be conveyed to a print server. This classification scheme is depicted graphically in figure 3.1, and the terms are defined in the following subparagraphs.

1. Overlapping instruction subsets
   (a) document instructions: instructions that may be incorporated or carried with the document; these are generally concerned with the appearance of printed pages, or of the structure of the finished document.
   (b) print operation parameters: instructions that may be conveyed to a print server within the print request itself; these are concerned with production and finishing of the document(s) and with management of the print job itself.
2. Disjoint instruction subsets
   (a) document imaging instructions: may be conveyed to the print server only within the document itself.
   (b) print job management instructions: may be communicated only with the print operations parameters.
   (c) document production instructions: may be delivered with the print request or in the document, or both; if a specific instruction appears in both places for a given request and document, the print server will resolve the potential conflict by means of an algorithm that is beyond the scope of this paper.

render: to represent, especially by artistic means; to depict. The term render will be used in this International Standard for the specification of the shapes or attribues of characters or other graphical elements, as by a particular font, especially with respect to a particular instance of presentation; it will be distinguished from presentation by emphasizing the shape to be used rather than the process of imaging the shape.

SPDL fragment: a sequence of SPDL tokens.

structure: an ordered set of elements which can be referenced as a single entity. Elements of a structure may themselves be structures. Examples of structures include pictures, fonts, and parameterized complex objects, e.g., a generic pie chart.

token: an encoded representation of a primitive SPDL object.

The following are several terms that are important to the concepts contained whithin this document, and descriptions of some Athena-specific terms.

Kerberos: The Authentication System at Project Athena which uses a trusted third-party protocol to identify user and services in a network environment.

Hesiod: The name service used at Project Athena. It uses (replicated) servers which provide keyed information through the network. It is built on the Berkeley Internet Name Domain (BIND) library.

Zephyr: A Notification Service in use at Project Athena for sending short, time-critical messages.

SMS: The Service Management System, used at Project Athena to centrally manage and distribute information in the environment.

Chapter 2

Palladium System Overview

As discussed in the introduction, a print system must be developed that satisfies the pressing needs of operations and systems management, as well as supporting the new features of sophisticated printers. The Athena Palladium has been designed to provide equitable access to shared resources in a distributed environment. It has been designed to allow centralized management of the configuration of the distributed print system. It allows distributed access to multiple servers which represent and manage sets of shared resources. Finally, it has been designed to be easily implemented today and readily extended in the future.

These goals have been satisfied by exploiting the distributed systems concepts of:

- Client-Server Model
- Name Service-based Server Advertisement
- RPC-based Server Access Protocols
- Embedded Tasking Systems
- Separation of Functionality

This chapter will describe how these goals will be met by the Palladium Print System.

2.1 Client-Server Model

The client-server model provides a paradigm for the relationship between local user applications and shared resources. The shared resources are said to provide 'services' to the user. The shared resources and the services that they provide are represented and often managed by 'servers'. The user applications are represented as clients of the services provided by the servers. The servers are said to be 'distributed' when their location in a network of computers is independent of the location of their clients.

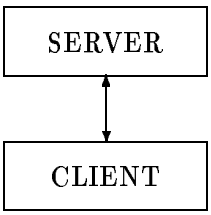

Figure 2.1: The Client-Server Model

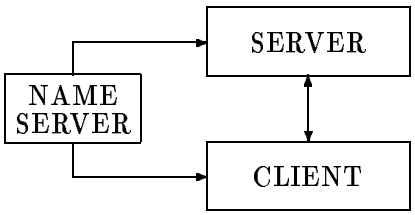

Figure 2.2: The Name Server Model

Thus, distributed servers represent and manage shared resources and the services provided by those resources. Clients represent local user's needs to servers in the distributed environment.

2.2 Name Service-based Server Advertisements

With widely distributed clients and servers comes the problem of how a client finds a server that provides a needed service. We use a name server to help clients identify the server that satisfies it's needs and to help the client find the location (address), of the server.

The name server acts as a logically centralized third-party advertiser of servers and the services and shared resources that they represent.

2.3 RPC-based Server Access Protocols

Traditional distributed applications used custom-designed communications/access protocols. But the proliferation of distributed applications threatens to significantly increase the development and maintenance costs for the distributed environment. The Remote Procedure Call (RPC) paradigm provides a single access protocol for the interfaces to many distributed applications. The RPC

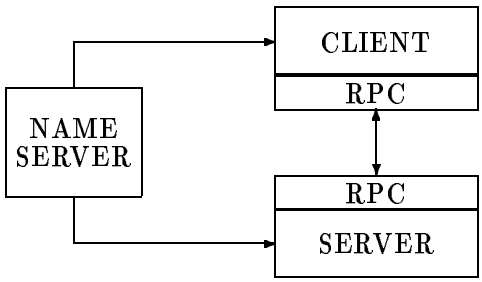

Figure 2.3: RPC-based Server Access Protocol

paradigm is implemented by a set of software development utilities and libraries called the RPC service. Using an RPC service, distributed applications developers need only provide an interface specification to one of the RPC utilities (an RPC Stub Generator) to generate the access protocol code. The RPC paradigm can be exploited to shorten development cycles, thus lowering the development and maintenance costs of distributed applications.

2.4 Embedded Tasking Systems

RPC paradigms enforce a synchronous relationship between client and server. To equitably handle all requests that may be submitted to the server in a short period of time, and to supply reasonable response to each request, the server must have some way of logically doing multiple tasks at the same time. Tasking systems are used to provide this capability by either allocating time slices of a single process to different tasks (for operating systems with 'expensive processes') or by creating new processes for each task that must be accomplished. These techniques will be referred to as "tasking."

2.5 Palladium Print System Model

The Palladium Print System Model will exploit the Client-Server Model, Name Server Advertisements, RPC paradigms and Tasking to improve the managability of distributed print systems and provide equitable access to shared printer resources in the Athena distributed environment.

The Palladium Print System Model has the following specific characteristics:

- The print client represents the user's printing needs to the name server to find an appropriate print server.
- The server represents the capabilities of the shared resources managed by the server to the name server.

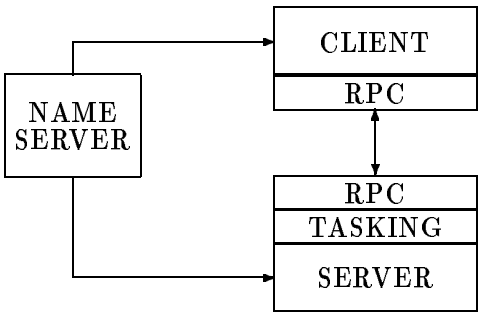

Figure 2.4: Tasking System Model

- The name server tells the client where to find the print server that matches the user's needs.
- The server interface allows access to the services provided by the server.
- The client uses the RPC service to make a service request of the print server.
- The print server tasking system allows multiple clients to make requests at logically the same time.
- Once a request is accepted, the tasking system allows the job scheduler to process the request aysnchronous from the submission of the request.
- The server database acts as a mediator between the request submission routines and the processing routines managed by the job scheduler.
- The scheduler passes responsibility for a job off to a specialized process or server called a 'printer supervisor'.
- The printer supervisor takes responsibilty for printing a single job, using a single print format on a single (type of) printer. Note that a printer may support more than one print format, but for any specific job we presume that only one of those formats is exploited.

The separation of functionality exploited in this design allows for greater flexibility in extensibility and management of the system.

2.5.1 Palladium Client

The Palladium Client selects the server which satisfies the requirements of the user-specified print parameters, file-specific print parameters and any system or user-profile parameters. The client then submits the job to this server. The client may also provide some local job management for the submitted job. Local job management keeps track of jobs for the local user and performs some post-processing such as notification and file deletion. Local job management may also support mapping between global job ids and the name of the server to which the job was submitted.

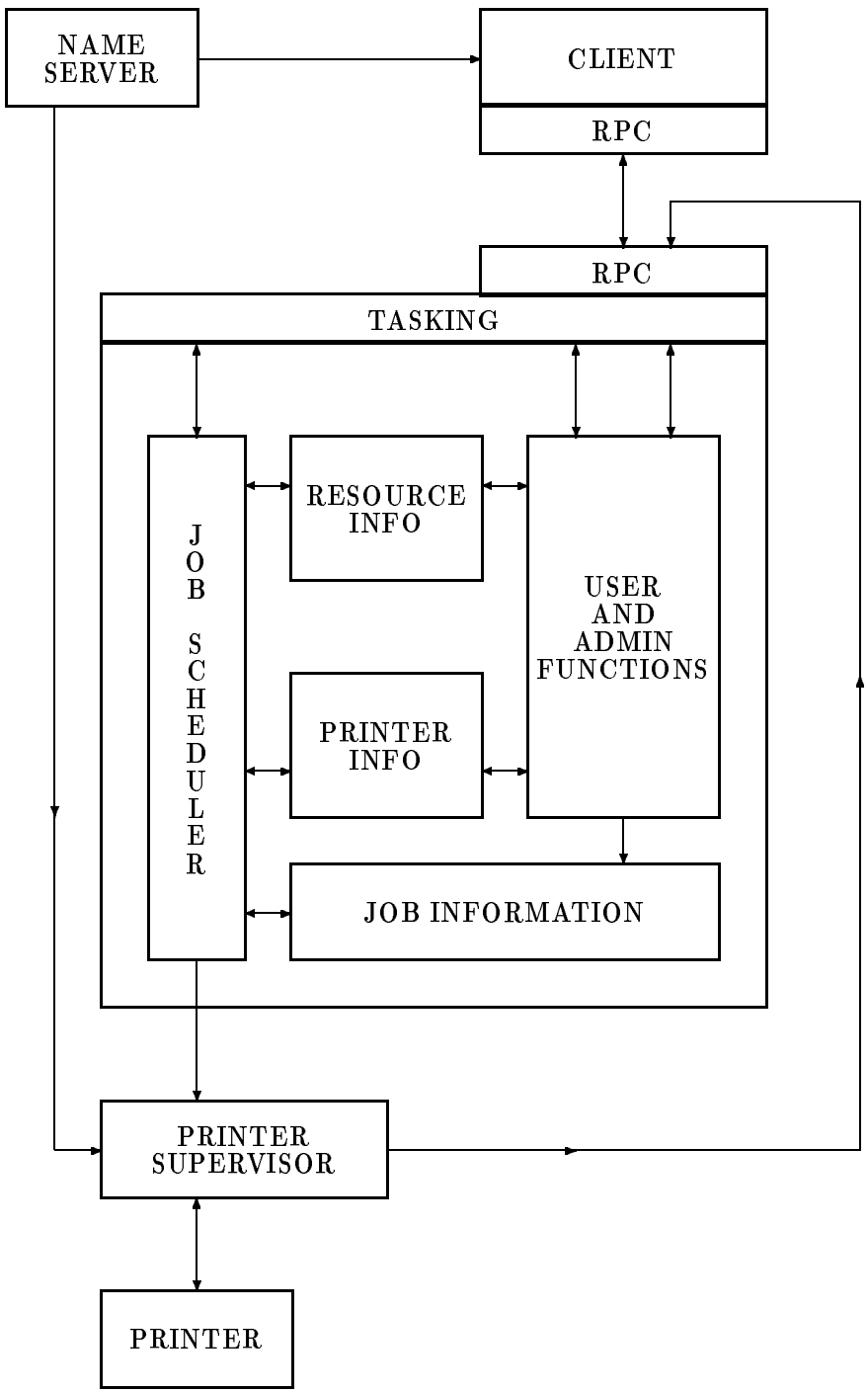

Figure 2.5: Palladium Print System Model

2.5.2 Name Server

The Name Server advertises the capabilities of the shared resources and services provided by different print servers. The name server allows the client to identify the server that satisfies the user's needs and to find that server in the distributed environment.

2.5.3 Palladium Server

The Palladium Server handles requests submitted by the Palladium Client and processes those requests. Most requests are processed as they are submitted, returning the results immediately. Print and ModifyPrintJob requests are validated and, if accepted, added to the Job Info Database. The Job Scheduler locates the right printer which satisfies the job requirements, schedules the job on the printer and tracks the progress of the job. The Job Scheduler records job events in the Job Info Database and reports status and significant events to the client. In addition, the job scheduler maintains all accounting and error log information for the job.

The Server does not understand the syntax or the semantics of the file content. The Job Scheduler merely matches the job requirements with the capabilities of the printers managed by this server. In this way the server may handle a broad range of file types and printer types without major modification to the server software.

The server may be implemented as several processes or tasks within a single process. But it must run off of a single job, resource and printer database and use a single centralized scheduling or queueing mechanism for all the jobs and printers that it manages. Thus equitable shared access is provided, since all clients must go through the same server to get to the printers under the control of that server.

2.5.3.1 Service Interface and RPC

The Palladium Server Interface will be presented as a set of library routines (Palladiumlib), divided into two classes: user interface routines, and administrative routines. These routines will be accessed through an RPC protocol. Palladium Clients make requests of the following library routines to obtain service from a Palladium Server:

1. Print (submit Files to be printed together as a Job)
2. Modify Print Job (that was previously submitted)
3. Read Job Status
4. Cancel Job
5. List Jobs
6. Read Object Attributes

These functions take well-defined arguments and return results, rejections and events. They are further described in the PdLib and PdSupervisor Chapters

2.5.3.2 File Transfer Paradigms

When a print job is submitted to a print server, the contents of the file(s) to be printed must be made available to the server at some point. ECMA recognizes three distinct paradigms:

1. **Server Copy** (submit-with-reference) : The client submits a pointer (such as a system and file name) to the file(s) when it calls the print routine. It is the server's job to retrieve the file.
2. **Contents As Parameter** : The contents of the file are submitted as one of the parameters to the print request.
3. **Copy by Client** : The client puts the file on the server system. The exact method of doing this is:
   (a) Client calls the print function on the server, passing in all parameters except for the contents of the file or pointer.
   (b) Server decides if the job is acceptable. If it is, the server indicates where the client should put the file, and waits for the client to proceed.
   (c) Client copies file to the specified location. It then reports to the server that the file is copied, and the server

Palladium will support all these functions (eventually). Selection of file transfer method is done at function call time. Initially, Copy-By-Client will be used. For a standalone Supervisor, we assume that the spool directory of server is accessible to all the client workstations which have access to this standalone Supervisor so that the client command can copy the file to be printed over to the Palladium server. However when a print call is made to a Server, then the file is passed over the sockets using a protocol very similar to what the BSD *lpd* daemon uses.

2.5.3.3 Palladium Server Database

The Print Server Database contains high-level definitions of logical printer capabilities (e.g. names of media supported, finishing features supported, etc.), definitions of resources (e.g. sizes and colors of media, location of stapling, etc.), and job definitions.

2.5.4 Palladium Printer Supervisor

The Printer Supervisor provides a static process to drive a specific printer (or set of related printers) for the printing of one print job at a time. The Printer Supervisor is entirely responsible for

interpreting the job request. This includes interpreting the content of the file(s) and the reconciling the instructions provided in the print request parameters with the contents of the file. The supervisor has complete control over the printing of the job, including the loading of fonts, forms and other electronic resources into the printer, as well as the transfer of the final form document to the printer. The Printer Supervisor is the equivalent of the printcap setup code of the Berkeley Print Daemon plus the 'child' code of the daemon.

2.5.4.1 Supervisor Interface

The Printer Supervisor provides the same RPC-based interface that the Print Server provides to clients. This allows the Printer Supervisor to be accessed by either a Print Server or by any Print Client. The latter configuration allows a print supervisor to be installed on a workstation to drive a personal printer attached to that workstation. The same interface used by the client to access shared printers through a Print Server can be used by the client to access a local (potentially non-shared) printer. The only difference to the client is the address advertised in the name server.

2.5.5 Palladium Compatibility Gateways

The Athena Palladium implementation will be completely independent of current System V and BSD Unix Print Spooler implementations. Interoperability with existing implementations will be provided through gateways between traditional *lpd* daemons and Palladium servers. This gateway will be accomplished using filters that resubmit the job to the Palladium Server/Supervisor.

2.6 Job Processing

Based on the preceeding model, when a user requests that a file be printed, a very specific and consistent set of steps are performed by the print system:

1. The print client identifies the print format of the contents of the file (e.g. ASCII, Postscript, Interpress, etc.) using the file type or internal print format information as clues. This information will be used by the server to validate the request to print this file on a particular printer.

2. The print client attempts to find the printer (named by the user in the print command) in the name server. This printer name may represent the name of a specific physical printer, a System V printer class, a BSD printer 'queue', or any other logical printer.

3. If the printer name can not be found in the name server, the print command returns an error to the user.

4. If the printer name is found, the returned name service entry supplies the address to which the client submits the request.

5. The print server routine called by the client validates the ability of this server to print the specified file on the specified printer. If the request is invalid, the print server routine returns an error. Otherwise, it is accepted by the server, put in the jobs database for scheduling and a success status is returned to the client.

6. Independent of the submission of the print job request, the job scheduler is scanning the job database and the printer database trying to find jobs that can be printed on available printers. This heap-scheduling paradigm has been chosen over a strict queue scheduling paradigm to support future scheduling algorithms.

7. When the scheduler finds a job whose required printer name matches an available printer name, the scheduler submits the job to the appropriate printer supervisor using a remote procedure call. The use of identical server/supervisor interfaces simplifies system development and allows printer supervisors to transparently act as independent agents to local clients.

8. The printer supervisor prints the job on the specified printer. If intermediate status about the job is appropriate, this status is returned to the print server via the PdReportEvent operation. Accounting information is also returned to the server using the PdReportEvent operation. **Note**: these operations will most likely be implemented using Zephyr.

9. The print server accepts notification of job processing events and accounting records from the printer supervisor. This information is stored in the job database and passed to the user, operators or other services (e.g. accounting services) as specified in the original print request.

10. When processing of the job is complete, the printer supervisor notifies the server of this fact using the PdReportEvent operation. The server in turn must notify the user of this event.

11. Completion of the processing of a job may be independent of the request for additional work by the printer supervisor. The print server provides a separate operation for the the printer supervisor to request additional work. This allows the future implementation of high-end printer supervisors that can process more than one job at a time.

12. Completion of the processing of a job may be independent of the removal of the print job from the jobs database. The user may wish the jobs database entry for a print job to hang around for hours or days after the completion of the job to allow the user to check the completion status of the job. (this is particularly useful in environments where there is no active client process tracking the progress of print jobs, e.g. Athena public workstations) Removal of the job occurs when the scheduler scanning the jobs database notices a job whose 'removal date' has passed.

Chapter 3

Palladium User Interface

The current Palladium User Interface (Pdui) is a command-line interface. Due to time constraints on this prototype, a window-based interface will be postponed until a later date. The current interface provides three user commands and five administrative commands. Future implementations will provide a complete set of user and administrative commands with a verbose set of options. The extent of these options will be determined through system use.

Many of the Pdui commands use the job identifier as one of the command line arguments. The unique identifier assigned to each job is long and complicated, thus to make life a little easier for the user a file is maitained in the user's home directory that contains a mapping between a simple local job identifier and the unique global job identifier. The maintenance of this file occurs through the side effects of each of the user commands. When a job is submitted with the pdpr command, the local and global ids are added to the file. If a job is deleted using pdrm, the local and global ids are removed. Each time the pdpq command is performed the file is updated by removing entries for jobs that have been printed or have been deleted by an administrator. Commands that use the job identifier will work with either the local or global job id.

3.1 pdpr - Printing A Job

The user interface will allow the user to specify arguments related to the following areas:

- Certain Job and Scheduling Parameters
- Printer Identification
- Production Instructions
- Certain Document description parameters
- Job Results Handling
- Event Notification

- File names

As the print operation has a long list of parameters, specification of all parameters at each invocation would prove impractical. In order to keep the interface relatively simple, a profile mechanism will be used. The print request client will scan the profile specified or use the user's default profile. This profile will be used to alter the default values of the job parameters, and will consist of a list of tags with values.

In this way a standard or frequently used job specification can be readily loaded upon invocation without the tedious specification of each relevant parameter. This method also allows a particular invocation to override or augment a portion of the profile without restating it entirely. Printing instructions embedded within the document will not be supported.

3.1.1 Job and Scheduling Parameters

The user will be able to specify explicitly the job-title. If none is specified, the name of the first document file (or stdin for standard input) will be used. The user will also be able to request the inclusion of a cover sheet preceding the print output. The format of the cover sheet will be determined by the site.

A certain degree of freedom is allowed in the specification of scheduling by the user. The user will be allowed to request that the printer wait until a certain time to print the job (job-print-after), and he can specify that the job be retained for a specified time after printing.

3.1.2 Printer Identification

In this implementation, the user must specify the name of a logical printer, through the command line, profile or a system variable. Specifying the logical printer is not the same as specifying a specific printer or device. The logical printer represents a set of advertised attributes and defaults and may be represented by one or more actual printers. When a request for printing is made, a server supporting the specified logical printer is located, through the Hesiod database, and the job is submitted to it. Note that a physical printer may be represented or referenced by more than one logical printer.

3.1.3 Production Instructions

Production instructions that are currently available through the command line options are the number copies to be printed; the number of sides to print, sixplex or duplex; and the layup, the number of logical pages per physical side. The remaining production instructions are specified in the profile.

3.1.4 Job Description Parameters

The command line options allow you to specify orientation, portrait or landscape; the page order, normal or reverse; the document type, Ascii or Postscript; a document title and a document comment. The remaining options must be specified in the profile.

3.1.5 Job Results Handling

The results of a print operation request may result in the successful or unsuccessful production of a document. Acceptance of the print job does not guarantee completion. The user can thus specify whether notification is desired and what method of notification should be used. In the Athena environment, the Zephyr notification Service and electronic mail are examples.

3.1.6 File specification

In this implementation the user can specify only one file at a time. If no filename is given, standard input is presumed, and read up to EOF.

3.1.7 Proposed Shell Command Line

```
pdpr [-a time]
     [-b banner]
     [-c document-comment]
     [-d document-type]
     [-e event-list]
     [-l]
     [-m]
     [-o option[=value] [,option[=value]]]
     [-p [profile]]
     [-r]
     [-s sides]
     [-t document-title]
     [-u pages-up]
     [-v]
     [-z]
     [-B]
     [-C job-comment]
     [-D]
     [-E end-sheet]
     [-H header-sheet]
     [-I separator-sheet]
     [-P printer]
     [-R time]
```

```
     [-S server]
     [-T job-title]
     [-# number-copies]
     [filename]


-a time
        print the job at or after the time specified,
        the format of the time is yymmddhhmm.

-b banner
        Specifies the type of banner page to use,
        banner may be "Full" or "Brief".

-c document-comment
        Specifies the document comment.

-d document-type
        Specifies the document type. Currently the only
        recognized types are: Ascii and Postscript.

-e event-list
        Specifies the events to be notified of if and
        when they occur.

-l
        Specifies printing in landscape, the default
        is portrait.

-m
        Specifies the event notification uses mail,
        (can not be specified with the -z option).

-o option-list
        The option list is a way of overriding values
        defined in the profile. The format of the list
        is :
        [[parametername = value] [, parametername = value]]

-p profile
        Specifies the profile to use, full path name,
        if a profile is not specified then the default
        file ".pdprofile" is assumed to be in the
        user's home directory.
```

```
-r
        Specifies that the pages are to be printed
        in reverse order.

-s sides
        Specifies the number of sides on which
        printing occurs:

        simplex (one sided) or duplex (two sided).

-t document-title
        Specifies the document title.

-u pages-up
        Specifies the number of virtual pages per
        physical side.

-v
        Specifies verbose notification, the user is
        notifies when his job is started, ends or is
        cancelled. The default is to notify the user
        only if his job is cancelled.

-z
        Specifies the event notification system to
        use zephy, (can not be specified the -m option)

-B
        Suppresses the banner page from printing.

-C job-comment
        Specifies the job comment.

-D
        Delete the job after it is spooled.

-E end-sheet
        Specifies the job end sheet. The value of end-sheet
        may be "Full" or "Brief".

-H header-sheet
        Specifiifes the job start or header sheet. The value of
        header-sheet may be "Full" or "Brief".

-I separator-sheet
```

```
        Specifies the sheet in between jobs. The value of
        separator sheet-sheet may be "Full" or "Brief".


 -P printer
        Specifies the name of the logical printer
        to handle the the actual print request.

 -R time
        Specifies the amount of time that the job
        is retained in the database after the job
        has actually printed.

 -S server
        Specifies the name of the server to
        handle the request.

 -# number
        Specifies the  number of copies to print.


ENVIRONMENT:

PDPRINTER
        Is an environmental variable containing the
        default logical printer.

 ~/.pdprofile
        Is the default profile. It contains list of
        items of the form:

 grouping.option: value
        grouping.option.suboption: value

        where the option and value are as specified above.
```

Option specification will first be read from the user's default profile, then from the command line. If there is a conflicting specification of options, the option specificed last will hold.

The sytem administrator can also define a system default profile. If that file is specified then it overrides both the user's profile, and command line options. the system default file is located in

/usr/skel/pd.profile

3.2 pdpq - List Job Status/Information

The request for job information and status can be for a specific job or for a general set of jobs, those on a particular server, or logical printer. The default option is to only display the information on job submitted by the user, however all jobs may be requested.

The file maintained in the user's home directory is used to determine which file are the user's.

3.2.1 Command Line

```
pdpq [-a]
     [-s seconds]
     [-P logical-printer | -S server]
     [jobid]

-a
         Specifies that all jobs should be printed,
         the default is only the user's.

-s seconds
         Specifies that the command should sleep
         the specified number of seconds between
         scans. The command exits when there are
         no jobs or it is interrupted.

-P printer
         Specifies a logical printer. Returns a the
         status of user's jobs submitted to this
         logical printer, all jobs if -a was specified.

-S server
         Specifies a server name. Returns a the status of
         user's jobs submitted to this server, all jobs if
         -a was specified.

jobid
         Specifies a specific jobid, may be a local
         or global jobid.
```

3.3 pdrm - Cancel Print Job

A print job that is pending or printing can be cancelled by specifying the job identifier, either the local or global id. Authentication will occur to establish the identity of the user. Only the owner of a job or a superuser will be allowed to remove a job which was submitted.

If the job being cancelled is actively printing there is no guarantee that the job will actually be removed before printing completes.

3.3.1 Command Line

```
pdrm [-a]
     [-P printer | -S server]
     [jobid]

-a
         Specifies that all jobs are to be removed,
         a logical printer or server must also be
         specified. Only the user's jobs are deleted,
         unless executed by a super user, in which
         case all jobs are removed.

-P printer
         Specifies a logical printer.

-S server
         Specifies a server name.

jobid
         Specifies a specific job to remove.
```

3.4 Administrative Interface

The current Palladium Administrative Interface (pdai) allows the privileged user to start up a server/supervisor, enable and disable, pause and resume, and shut it down, immediately or through an orderly set of steps. Using the Pdai a priviledged user may delete all of the jobs that have been scheduled on a particular server/supervisor. Any operation done by means of this interface will be considered privileged, and will be done without regard for ownership of the job. Therefore, initial authentication as an administrator will be required to invoke the interface.

3.5 pdstart - Start A Server/Supervisor

The pdstart command allows the priviledged user to start a service or supervisor.

```
pdstart [-f configfile]
        [-l logfile]
        [-n]
        [-s]
       -I object-id  -C object-class

-f configfile
         Specifies the name of the configuration file.

-l logfile
         Specifies the name of the logfile.

-n
         Specifies that the scheduler should not be started.

-s
         Specifies that the supervisor is to be started
         in stand alone mode, ie., is is not register with
         any server. Only with a local client.

-I object-id
         Specifies the name of the object; server, supervisor
         or printer.

-C object-class
         Specifies the object class: "server", "supervisor", and
         "printer" and valid object classes.
```

3.6 pdinit - (Re)Initialize An Object

The pdinit command allows the privilegded user to initialize or reinitialize an object. An object can be a server, supervisor or a logical printer. During initialization the object reads its internal information from the database, in the case of the printer its supervisor reads from the database.

```
pdinit -I object-id  -C object-class
```

```
-I object-id
         Specifies the name of the object; server,
         supervisor or printer.

-C object-class
         Specifies the object class: "server",
         "supervisor", and "printer" and valid
         object classes.
```

3.7 pdenable - Enable An Object

The pdenable command allows the priviledged user to enable a server, supervisor or printer. Once enabled that object can begin accepting print requests.

```
pdenable [-d]
        -I object-id
        -C object-class

-d
         Disable the object.

-I object-id
         Specifies the name of the object;
         server, supervisor or printer.

-C object-class
         Specifies the object class: "server",
         "supervisor", and "printer" and valid
         object classes.
```

3.8 pddisable - Disable An Object

The pddisable command allows the priviledged user to disable a server, supervisor or printer. User requests for a disabled server or logical printer are rejected, however, management requests are still accepted. Scheduling continues for spooled jobs. A disabled supervisor will finish printing the current job but will not request or accept additional jobs.

```
pddisable [-d]
```

```
          -I object-id
          -C object-class

-I object-id
          Specifies the name of the object;
          server, supervisor or printer.

-C object-class
          Specifies the object class: "server",
          "supervisor", and "printer" and valid
          object classes.
```

3.9 pdpause - Pause An Object

The pdpause command allows the priviledged user to pause an object. If the object is a server or a logical printer, the scheduling of jobs is stopped for that object, however, management and non-printing user requests are still accepted. If the object is a supervisor the current job is stopped, without losing state so that printing may be resumed later.

```
pdpause [-r]
        [-p page-number]
        [-c copy-number]
        -I object-id
        -C object-class

-r
         Resume scheduling, printing.

-p page-number
         Pause printing at the specified page number.

-c copy-number
         Pause printing at the specified copy numbr.

-I object-id
         Specifies the name of the object; server,
         supervisor or printer.

-C object-class
         Specifies the object class: "server",
         "supervisor", and "printer" and valid
         object classes.
```

3.10 pdresume - Resume An Object

The pdresume command allows the priviledged user to resume an object. If the object is a server or a logical printer, the scheduling of jobs is started for that object. If the object is a supervisor the paused job is resumed from the point that the pause was initiated.

```
pdresume [-p page-number]
         [-c copy-number]
         -I object-id
         -C object-class

-p page-number
         Resume printing at the specified page number.

-c copy-number
         Resume printing at the specified copy numbr.

-I object-id
         Specifies the name of the object; server,
         supervisor or printer.

-C object-class
         Specifies the object class: "server",
         "supervisor", and "printer" and valid
         object classes.
```

3.11 pdshutdown - Shutdown an Object

The pdshutdown command stops an object in an orderly manner. If the object is a server it is first disabled, then paused and finally stopped. A supervisor is disabled and then stopped.

```
pdshutdown [-i]
           -I object-id
           -C object-class

-i
         Immediate shutdown, object is not
         paused or disabled.

-I object-id
         Specifies the name of the object;
```

```
          server, supervisor or printer.

 -C object-class
          Specifies the object class: "server",
          "supervisor", and "printer" and valid
          object classes.
```

3.12 pdstop - Stop An Object

The pdstop command immediately stops an object. If the object is a server it is not paused or disabled first. If the object is a supervisor it i not disabled first, this means the active job will not finish before the stop occurs.

```
pdstop -I object-id  -C object-class

-I object-id
          Specifies the name of the object;
          server, supervisor or printer.

-C object-class
          Specifies the object class: "server",
          "supervisor", and "printer" and valid
          object classes.
```

Chapter 4

Pdtk - Palladium Client Toolkit

The Palladium Client toolkit provides a subroutine level interface to the Palladium Clients which provide the functionality of the shell commands without the necessity of process image overlay.

In addition, the subroutines provide a certain level of input checking and housekeeping related to the clients, and will mask certain unsavory complexity which many implementors will not care to know. Features like Kerberos Authentication, Hesiod resolutions, and jobs log file. The log file contains a mapping of local job ids to global job ids. The less complicated local id may be used to identify the job.

The following Palladium Toolkit routines are available:

User Functions

1. Pdtk_PdPrint - Submits a job for printing
2. Pdtk_PdModify - Modifies an existing job
3. Pdtk_PdReadJobStatus - Reads a job's status
4. Pdtk_PdCancelJob - Cancels an existing job
5. Pdtk_PdListJobs - Lists the specified print jobs
6. Pdtk_PdReadObjectAttributes - List the specified attributes

Admin Functions

1. Pdtk_PdResubmitJobs - Resubmit jobs to a new server
2. Pdtk_PdClean - Removes all Jobs on an object
3. Pdtk_PdPromote - Promote a job
4. Pdtk_PdModifyAttributes - Modify the attributes of an object

5. Pdtk_PdStop - Stop an object
6. Pdtk_PdInitialize - Initialize an object
7. Pdtk_PdShutdown - Shutdown an object
8. Pdtk_PdEnable - Enable an object
9. Pdtk_PdDisable - Disable an object
10. Pdtk_PdPause - Pause an object
11. Pdtk_PdResume - Resume an object

4.1 Print Job

4.1.1 Description

The print tool takes as arguments file desciptor(s) and processing instructions. The client processes the options, checking their validity. It contacts a server able to satisfy the processing requirements or returns an error message. The named file(s) are given in a print request to the server and the initial status of the job is returned. An entry is made in a session file of the successfully submitted job.

The submission of multiple files as separate jobs or as a single job remains open for further study.

4.1.2 Interface

```
int
Pdtk_PdPrint(filename, jobIdIn, jobInformation,
       resultHandling, eventHandling, schedulingInstructions,
       printerId, productionInstructions, documentDescription,
       specialProcessing, security, jobId, spoolFileName,
       jobStatus, pdError)

/* Input Arguments */
String                              filename;
PDif_JobIdentifier                  jobIdIn;
PDif_JobInformation                 jobInformation;
PDif_JobResultsHandling             resultHandling;
PDif_JobEventHandling               eventHandling;
PDif_JobSchedulingInstructions      schedulingInstructions;
PDif_PrinterIdentification          printerId;
PDif_ProductionInstructions         productionInstructions;
```

```
PDif_DocumentDescription              documentDescription;
PDif_SpecialProcessingInstructions    specialProcessing;
PDif_SecurityAccessAccounting         security;

/* Output Arguments */
PDif_JobIdentifier                    *jobId;
String                                *spoolFileName;
PDif_JobStatus                        *jobStatus;
PDif_PdError                          *pdError;


REQUIRES  : Structures the file name and job processing
            instructions.

RETURNS   : An integer status, zero if operation performed
            successfully, a positive number if an error
            occurred in the server/supervisor, or a -1 if
            an error occurred in the RPC call. If sucessful,
            it returns a unique global job id, the name of
            spool file, and its initial job status.

EFFECTS :   If accepted, an entry is made in a job list
            file mapping a local id with the global id.
```

All parameters are structured types according to ECMA TC32-TG5.

4.2 Modify Print Job

4.2.1 Description

The Modify Print Job tool takes as arguments the JobID of the Job the user wishes to modify and the print_job parameters (as above) that the user desires to modify. The job is either modified, if possible, or an error is returned. If an error is returned, the original job remains unchanged in all respects.

4.2.2 Interface

```
int
Pdtk_PdModify(jobId, jobInformation, resultHandling,
              eventHandling, schedulingInstructions, printerId,
              productionInstructions, specialProcessing, security,
```

```
              jobStatus, pdError)

/* Input Arguments */
PDif_JobIdentifier                      *jobId;
PDif_JobInformation                     *jobInformation;
PDif_JobResultsHandling                 *resultHandling;
PDif_JobEventHandling                   *eventHandling;
PDif_JobSchedulingInstructions          *schedulingInstructions;
PDif_PrinterIdentification              *printerId;
PDif_ProductionInstructions             *productionInstructions;
PDif_SpecialProcessingInstructions      *specialProcessing;
PDif_SecurityAccessAccounting           *security;

/* Output Arguments */
PDif_JobStatus                          *jobStatus;
PDif_PdError                            *pdError;

REQUIRES  : A string containing a unique job id of the job
            Structures containing job processing instructions.

RETURNS   : An integer status, zero if operation performed
            successfully, a positive number if an error
            occurred in the server/supervisor, or a -1 if
            an error occurred in the RPC call. If sucessful
            the job status is returned.

EFFECTS   : If accepted, the modifications are made.
```

4.3 Cancel Job

4.3.1 Description

The cancel job client takes as an argument the Job ID for the job which is to be canceled, contacts the appropriate server and tells it to cancel the job. The status of the operation is reported back to the user.

4.3.2 Interface

```
int
Pdtk_PdCancelJob (objectClass, objectId, jobId,
                  security, jobStatus, pdError)
```

```
/* Input Arguments */
String                              objectClass;
String                              objectId;
PDif_JobIdentifier                  *jobId;
PDif_SecurityAccessAccounting       *security;

/* Output Arguments */
PDif_JobStatus                      *jobStatus;
PDif_PdError                        *pdError;

REQUIRES  : A list of  containing unique Job Id's or Names
            to identify job.

RETURNS   : An integer status, zero if operation performed
            successfully, a positive number if an error
            occurred in the server/supervisor, or a -1 if
            an error occurred in the RPC call. If sucessful
            the status of the job when it was cancelled
            is returned.


EFFECTS :   The request to remove the job is made, and the
            entry is removed from the job list file.
```

4.4 Job Status

4.4.1 Description

This tool gets status on specific jobs. The caller specifies the Job Names or JobID's, the name of the service handling the job and a flag indicating the level of information required.

4.4.2 Interface

```
int
Pdtk_PdReadjobStatus(server, jobId, returnJobAttributes,
        security, submissionTime, jobStatus, jobInformation,
        resultHandling, eventHandling, schedulingInstructions,
        printerId, productionInstructions, documentDescription,
        specialProcessing, pdError)
```

```
/* Input Arguments */
char                                *server;
PDif_JobIdentifier                  *jobId;
Boolean                             returnJobAttributes;
PDif_SecurityAccessAccounting       *security;

/* Output Arguments */
LongCardinal                        *submissionTime;
PDif_JobStatus                      *jobStatus;
PDif_JobInformation                 *jobInformation;
PDif_JobResultsHandling             *resultHandling;
PDif_JobEventHandling               *eventHandling;
PDif_JobSchedulingInstructions      *schedulingInstructions;
PDif_PrinterIdentification          *printerId;
PDif_ProductionInstructions         *productionInstructions;
PDif_DocumentDescription            *documentDescription;
PDif_SpecialProcessingInstructions  *specialProcessing;
PDif_PdError                        *pdError;

REQUIRES  : A list of JobID's or Job Names, a boolean indicating
            level of information desired.

RETURNS   : An integer status, zero if operation performed
            successfully, a positive number if an error
            occurred in the server/supervisor, or a -1 if
            an error occurred in the RPC call. If sucessful,
            the requested information.

EFFECTS   : If verbose, reports all information on each job.
            Otherwise, a short summary is reported.
```

4.5 List Jobs

4.5.1 Description

Lists all of the jobs for the specified objectId, the objectClass indicates what kibd of object it is: server, supervisor or a logical printer.

4.5.2 Interface

```
int
```

```
Pdtk_PdListJobs (objectClass, objectId, filter, security,
                 serverLoad, jobHistorys, pdError)

/* Input Arguments */
String                            objectClass;
String                            objectId;
PDif_Filter                       *filter;
PDif_SecurityAccessAccounting     *security;

/* Output Arguments */
String                            *serverLoad;
PDif_T_r90125_32                  *jobHistorys;
PDif_PdError                      *pdError;

REQUIRES  : The name and class of the object. The exact nature
            and use of the filter has not been determined yet.

RETURNS   : An integer status, zero if operation performed
            successfully, a positive number if an error
            occurred in the server/supervisor, or a -1 if
            an error occurred in the RPC call. If sucessful,
            a list of jobs is returned.

EFFECTS   : Updates the job list file by removing entries that
            have already printed.
```

4.6 Read Object Attributes

4.6.1 Description

Reads and returns the attributes for the specified object. In the case of a logical printer this might be the type of printer it is, the type of fonts it supports, the paper currently loaded, etc.

4.6.2 Interface

```
int
Pdtk_PdReadObjectAttributes (objectClass, objectId, filter,
        attributeIds, security, objectAttributes, pdError)

/* Input Arguments */
String                          objectClass;
```

```
 String                          objectId;
 PDif_Filter                     *filter;
 PDif_T_p90125_35                *attributeIds;
 PDif_SecurityAccessAccounting   *security;

 /* Output Arguments */
 PDif_T_r90125_34                *objectAttributes;
 PDif_PdError                    *pdError;


 REQUIRES  : The name and class of the object. The exact nature
             and use of the filter has not been determined yet.

 RETURNS   : An integer status, zero if operation performed
             successfully, a positive number if an error
             occurred in the server/supervisor, or a -1 if
             an error occurred in the RPC call. And the
             attributes of the object.

 EFFECTS   : None

/end{verbatim}


\section{Resubmitting Jobs}

\subsection{Description}

Resubmits the jobs on the object specified to a new server and logical printer,
the user must be a user super or a recognized privilegded user.

\subsection{Interface}

\begin{verbatim}
 int
 Pdtk_PdResubmitJobs(objectClass, objectId, filter, newServer,
         newPrinter, security, jobList, pdError)

 /* Input Arguments */
 String                          objectClass;
 String                          objectId;
 PDif_Filter                     *filter;
 String                          newServer;
 String                          newPrinter;
 PDif_SecurityAccessAccounting   *security;
```

```
/* Output Arguments */
PDif_T_r90125_37                *jobList;
PDif_PdError                    *pdError;

REQUIRES :  The name of the name of the object to move the
            jobs from, and the name of the server and logical
            printer that the jobs are being moved to.

RETURNS  :  An integer status, zero if operation performed
            successfully, a positive number if an error
            occurred in the server/supervisor, or a -1 if
            an error occurred in the RPC call. If sucessful
            a list of the jobs moved is returned.

EFFECTS  :  None
```

4.7 Clean Jobs

4.7.1 Description

Removes all jobs on the specified object. When used, a priviledged user may remove all the jobs on a specified server, supervisor or logical printer.

4.7.2 Interface

```
int
Pdtk_PdClean(objectClass, objectId, notify, security, pdError)

/* Input Arguments */
String                          objectClass;
String                          objectId;
Integer                         notify;
PDif_SecurityAccessAccounting   *security;

/* Output Arguments */
PDif_PdError                    *pdError;

REQUIRES :  The name of the object and its class: server,
            supervisor, or logical printer.

RETURNS  :  An integer status, zero if operation performed
```

```
            successfully, a positive number if an error
            occurred in the server/supervisor, or a -1 if
            an error occurred in the RPC call.

EFFECTS  :  If notify is set then the user is notified that
            his job has been cancelled. Otherwise it is logged
            in syslog.
```

4.8 Promote A Job

4.8.1 Description

A specifed job can be promoted in the queue to print on the next available logical printer. This may only be performed by a priviledged user.

4.8.2 Interface

```
int
Pdtk_PdPromoteJob(objectClass, objectId, jobId, security, pdError)

/* Input Arguments */
String                          objectClass;
String                          objectId;
PDif_JobIdentifier              *jobId;
PDif_SecurityAccessAccounting   *security;

/* Output Arguments */
PDif_PdError                   *pdError;

REQUIRES :  The name of the object that is representing the job,
            server, supervisor or logical printer.

RETURNS  :  An integer status, zero if operation performed
            successfully, a positive number if an error
            occurred in the server/supervisor, or a -1 if
            an error occurred in the RPC call.

EFFECTS  :  The job is promoted.
```

4.9 Register An Object

4.9.1 Description

Allows an object, a supervisor, to be regiestered with a specified server.

4.9.2 Interface

```
int
Pdtk_PdRegister(server, objectClass, objectId,
                objectAttributes, security, pdError)

/* Input Arguments */
String                          server;
String                          *objectClass;
String                          *objectId;
PDif_T_p90125_41                *objectAttributes;
PDif_SecurityAccessAccounting   *security;

/* Output Arguments */
PDif_PdError                    *pdError;

REQUIRES :  The name of the server to register with and the
            name of the object being registered.

RETURNS  :  An integer status, zero if operation performed
            successfully, a positive number if an error
            occurred in the server/supervisor, or a -1 if
            an error occurred in the RPC call.

EFFECTS  :  Registers that object with the server.
```

4.10 Unregister An Object

4.10.1 Description

Allows an object, a supervisor, to be unregiestered from a specified server.

4.10.2 Interface

```
int
```

```
Pdtk_PdUnregister(objectClass, objectId, security, pdError)

/* Input Arguments */
String                              *objectClass;
String                              *objectId;
PDif_SecurityAccessAccounting       *security;

/* Output Arguments */
PDif_PdError                        *pdError;

REQUIRES :  The name of the server to unregister from and the
            name of the object being unregistered.

RETURNS  :  An integer status, zero if operation performed
            successfully, a positive number if an error
            occurred in the server/supervisor, or a -1 if
            an error occurred in the RPC call.

EFFECTS  :  Unregisters that object from the server.
```

4.11 Modify Attributes

4.11.1 Description

Allows the attributes of an object to be modified, in the case of a logical printer it might be the fonts available or the color of the paper supported.

4.11.2 Interface

```
int
Pdtk_PdModifyAttributes(objectClass, objectId, objectAttributes,
        security, pdError)

/* Input Arguments */
String                          *objectClass;
String                          *objectId;
PDif_T_p90125_43                *objectAttributes;
PDif_SecurityAccessAccounting   *security;

/* Output Arguments */
PDif_PdError                    *pdError;
```

```
REQUIRES :  The name and class of the object and the attributes
            that are being modified.

RETURNS  :  An integer status, zero if operation performed
            successfully, a positive number if an error
            occurred in the server/supervisor, or a -1 if
            an error occurred in the RPC call.

EFFECTS  :  The objects attributes are modified.
```

4.12 Stop An Object

4.12.1 Description

Stop kill a server or supervisor ungracefully, not attempt is made to finishing anything actively printing or to do anything that might be considered nice.

4.12.2 Interface

```
int
Pdtk_PdStop(objectClass, objectId, security, pdError)

/* Input Arguments */
String                          *objectClass;
String                          *objectId;
PDif_SecurityAccessAccounting   *security;

/* Output Arguments */
PDif_PdError                    *pdError;

REQUIRES :  The name and class, server or supervisor, of the
            object be stopped.

RETURNS  :  An integer status, zero if operation performed
            successfully, a positive number if an error
            occurred in the server/supervisor, or a -1 if
            an error occurred in the RPC call.

EFFECTS  :  Stops the object cold.
```

4.13 (Re) Initialize An Object

4.13.1 Description

Tells the object to (re)read its attributes from the database.

4.13.2 Interface

```
int
Pdtk_PdInitialize(objectClass, objectId, security, pdError)

/* Input Arguments */
String                          *objectClass;
String                          *objectId;
PDif_SecurityAccessAccounting   *security;

/* Output Arguments */
PDif_PdError                    *pdError;

REQUIRES :  The name of the object being initialized.

RETURNS  :  An integer status, zero if operation performed
            successfully, a positive number if an error
            occurred in the server/supervisor, or a -1 if
            an error occurred in the RPC call.

EFFECTS  :  The object knows about its attributes.
```

4.14 Shutdown An Object

4.14.1 Description

Gracefully shutsdown an object, server or supervisor. Active jobs are allowed to finish printing, and things are halted in an orderly fashion.

4.14.2 Interface

```
int
Pdtk_PdShutdown(objectClass, objectId, security, pdError)

/* Input Arguments */
```

```
String                              *objectClass;
String                              *objectId;
PDif_SecurityAccessAccounting       *security;

/* Output Arguments */
PDif_PdError                        *pdError;

REQUIRES :  The name and class, server or supervisor, that is
            being shutdown.

RETURNS  :  An integer status, zero if operation performed
            successfully, a positive number if an error
            occurred in the server/supervisor, or a -1 if
            an error occurred in the RPC call.

EFFECTS  :  The object is shutdown.
```

4.15 Enable An Object

4.15.1 Description

Enables an object, an object must be enabled in order for it to accept jobs.

4.15.2 Interface

```
int
Pdtk_PdEnable(objectClass, objectId, security, pdError)

/* Input Arguments */
String                              *objectClass;
String                              *objectId;
PDif_SecurityAccessAccounting       *security;

/* Output Arguments */
PDif_PdError                        *pdError;

REQUIRES :  The name and class, server or supervisor, to be
            enabled.

RETURNS  :  An integer status, zero if operation performed
            successfully, a positive number if an error
            occurred in the server/supervisor, or a -1 if
```

```
            an error occurred in the RPC call.

EFFECTS  :  The object is enabled.
```

4.16 Disable An Object

4.16.1 Description

Disables an object, an object that is disabled will not longer accept jobs. However, in the case of a server, it will continue to process jobs that have already been queued.

4.16.2 Interface

```
int
Pdtk_PdDisable(objectClass, objectId, security, pdError)

/* Input Arguments */
String                              *objectClass;
String                              *objectId;
PDif_SecurityAccessAccounting       *security;

/* Output Arguments */
PDif_PdError                        *pdError;

REQUIRES :  The name and class, server or supervisor, to be
            disabled.

RETURNS  :  An integer status, zero if operation performed
            successfully, a positive number if an error
            occurred in the server/supervisor, or a -1 if
            an error occurred in the RPC call.

EFFECTS  :  The object is enabled.
```

4.17 Pause An Object

4.17.1 Description

A paused object stops printing jobs that are queued, in the case of a server jobs will still be accepted for printing but they will not be printed until the service is resumed. A paused supervisor will stop printing immediately, without losing the job state. The job will resume printing when the supervisor is resumed.

4.17.2 Interface

```
int
Pdtk_PdPause(objectClass, objectId, pauseAt, security,
        pausedJobId, checkpoint, pdError)

/* Input Arguments */
String                              *objectClass;
String                              *objectId;
String                              pauseAt;
PDif_SecurityAccessAccounting       *security;

/* Output Arguments */
PDif_JobIdentifier                  *pausedJobId;
PDif_Checkpoint                     *checkpoint;
PDif_PdError                        *pdError;

REQUIRES :  The name and class, server or supervisor, of
            the object being paused.

RETURNS  :  An integer status, zero if operation performed
            successfully, a positive number if an error
            occurred in the server/supervisor, or a -1 if
            an error occurred in the RPC call.

EFFECTS  :  The object is paused.
```

4.18 Resume An Object

4.18.1 Description

A resumed object starts printing jobs that are queued, in the case of a server jobs that have been accepted will begin printing. A resumed supervisor will start printing immediately if it was printing when paused.

4.18.2 Interface

```
int
Pdtk_PdResume(objectClass, objectId, checkpoint,
        security, pdError)

/* Input Arguments */
String                              *objectClass;
String                              *objectId;
PDif_Checkpoint                     *checkpoint;
PDif_SecurityAccessAccounting       *security;

/* Output Arguments */
PDif_PdError                        *pdError;

REQUIRES :  The name and class, server or supervisor, of
            the object being resumed.

RETURNS  :  An integer status, zero if operation performed
            successfully, a positive number if an error
            occurred in the server/supervisor, or a -1 if
            an error occurred in the RPC call.

EFFECTS  :  The object is resumed.
```

Chapter 5

Pdlib — Palladium Server Interface

5.1 Introduction

The Palladium Library defines the interface to the Palladium Server and Supervisor. (Note: The Supervisor only performs a subset of the functionality, a full discussion on the Supervisor is given in Chapter 10.) The library provides functions for users to submit, track, modify or cancel printjobs, and it provides functions that allow administrators to maintain the printing environment. Access to the libray is done through Remote Procedure Calls (RPC). The use of RPC's provide a simple interface to the distributed services, ie., the supervisors and servers. The RPC paradigm enforces a synchronous relationship between the client and the service. To handle all requests in a reasonable period of time a tasking system will be used. To handle the RPC protocol and Tasking System, the HRPC software from The University of Washington [HRPC88] was chosen. HRPC is a common interface to a number of RPC systems. It provides a generic interface for remote procedure calls, and its use will maintain independence to the underlying protocals and RPC definitions. A detailed discussion on the server design can be found in Chapter 7.

The Pdlib has the following high level functions. These are accessible through the RPC protocol. Each function takes well defined argument(s) and returns results, rejections and/or events. Each function deals with an object where an object can be a server, a supervisor. Actions that are done to printers are done through their supervisor. A discussion on each function follows.

User Functions

1. PdPrint - process requests for printing
2. PdModify - process requests to modify pending jobs
3. PdReadJobStatus - Read Job Status
4. PdCancelJob - Cancel Job
5. PdListJobs - List Jobs
6. PdReadObjectAttributes - Read Object Attributes

Administrative Functions

1. PdResubmitJobs - Resubmits jobs to a different object.
2. PdClean - Removes all of the jobs from a specified object.
3. PdPromoteJob - Move the specified job to the top of the queue for the specified object.
4. PdInitialize - Initialize an object, ie., get all the information needed in order for the object to perform correctly.
5. PdEnable - Enamble the object.
6. PdDisable - Disable the object.
7. PdStop - Abruptly stop the object.
8. PdShutdown - Cleanly shutdown the object.
9. PdResume - Object resumes scheduling for servers, or resumes printing for supervisors.
10. PdInterrupt - Interrupt or suspend and object.
11. PdModifyAttributes - Modify the attributes of an object.
12. PdRegister- Register an object.
13. PdUnregister- Unregister an object.
14. PdRegisterUser - Register a user as being in charge of the object.
15. PdUnregisterUser - Unregister a user as being in charge of the object.
16. PdReadyforJob - Inform an object that a printer is ready for a job.
17. PdJobComplete - Inform an object the a printer has completed a job.
18. PdPause - Object stops scheduling jobs for servers, or it stops printing for supervisors.
19. PdAdvertise - Causes an object to advertise itself.
20. PdReportEvent - An object reports an event.
21. PdFileXfer - File Transfer Call for Copy-By-Client file transfers.

5.1.1 Object Handles

A call to a Server Interface Routine must be **directed** to a particular object. For all the calls in the PdLib interface, the first parameter for the call is a Palladium Object Handle. This is what directs the call to a particular object. Object Handles operate in the same way that RPC handles operate. A handle must be **bound** to a particular object before a PdLib call can be made. Binding is done through the toolkit function **PdBindToName**. The parameter to **PdBindToName** is a string which is the name for a particular object. It returns the address of that object.

Note: While handles are **bound** to a particular object, this does not necessarily mean that the call will be made directly on that object. For example, to shutdown a printer **landscape**, a call is made to **PdShutdown** with the handle bound to the name **landscape**. However, in most cases, foo is not an intelligent printer (eg, an ln03). In this case, **PdBindToName** will bind the handle to the printer's supervisor, and the call will actually be made to the supervisor, which will take care of shutting down the printer.

5.1.2 Subroutine Interface Convention

The subroutines follow the convention of returning a status integer. In all cases, a value of zero will indicate the successful completion of the operation. A non-zero status integer will map to the reason for failure.

5.2 Pdlib Function Arguments

This section briefly discusses the input and output arguments found in the Pdlib functions. The type of each argument is also given. All argument names start with a lowercase letter and are shown in **bold**, all types start with an uppercase letter and are shown in *slanted*. Both argument and type names use uppercase letters to delimit words. A complete definition of each type is provided in Appendix A.

1. **jobId** : *JobIdentifier* - The jobId is a structure that uniquely identifies a print request. The server assigns the id when a print request is accepted. The id is composed of two parts: a name and a time stamp.

2. **jobInformation** : *JobInformation* - The jobInformation structure contains general information about the print job. The structure's fields include a jobTitle, jobOriginator, jobOwner, jobComment, jobStartMessage, jobEndMessage, jobSeperatorSheet, jobEndSheet, etc.

3. **jobStatus** : *JobStatus* - This structure contains the status information for a particular print job. Information contained includes: currentJobState, previousJobState, printerAssigned, printerState, estimatedCompletionTime, copiesCompleted, etc.

4. **jobHistorys** : *SEQUENCE OF JobHistory* - The JobHistory type defines information about a specific job. Specifically the jobId, jobTitle, jobSubmissionTime, printerId, logicalService,

jobOriginator, jobStatus, and documentStyle. jobHistorys is a sequence (array) of these structures.

5. **resultHandling** : *JobResultHandling* - The resultHandling structure contains information on how to deliver the printed document to a recipient.

6. **eventHandling** : *JobEventHandling* - The eventHandling structure information for actions to be taken when sigmificant events occur during the processing of the printjob. For example: How should the submitter be notified if the print job is aborted.

7. **schedulingInstructions** : *JobSchedulingInstructions* - The schedulingInstructions structure contains additional instructions for scheduling the job, ie, jobStartWait, jobEndWait, jobHold, jobPriority, etc.

8. **printerId** : *PrinterIdentification* - The printId structure contains information about a logical printer which helps the scheduler select a printer. A logical printer may represent one or more physical printers. See Chapter 9 for details on the Palladium Server Job Scheduler.

9. **productionInstructions** : *ProductionInstructions* - This structure provides information that affects the rendering and finishing of the document. The structure's fields include: fontSubstitutions, mediaSubstitutions, inkSubstitutions, finishing, output, layup, printQuality, sides, pageSelect, and copyCount.

10. **documentDescription** : *DocumentDescription* - This structure identifies the document and its characteristics and specifies the method by which the document is aquired by the Palladium Server. Its field include: documentProfile, documentUserId, transferMethod, documentFormat, and documentReference.

11. **specialProcessing** : *SpecialProcessingInstructions* - The specialProcessing structure provides special instructions to the Palladium Server to be performed during the processing of the job. The exact uses of this argument have not been defined but are extensible.

12. **security** : *SecurityAccessAccounting* - The security structure contains information about the user which is used for security clearance and accounting.

13. **OBinding** : *ObjectHandle* - The OBinding is a structure that contains the name of an object and the address of the object.

14. **checkpoint** : *Checkpoint* - The checkpoint structure contains information about how much of the document has printed. The fields are pageNumber and copyNumber.

15. **pdError** : *PdError* - This structure contains the error status returned with all Pdlib function calls. The structure's fields are errNum and errMesg. An errNum of zero indicates that the request was successful.

The following two sections define the functional interfaces for the user and administrative functions, respectively.

5.3 Pdlib User Functions

The Pdlib user functions can be called by any authenticated user. The user functions closely follow the specification defined by the ECMA Standard for Print Services [ECMA88]. All arguments and their types are given, the definition of the types are provided in Appendix A. A brief description of each argument is given in the previous section. Each function is of type integer, and error code is returned in one of pdError's fields. The error code is also returned as the value of the function call itself.

5.3.1 PdPrint Function

The PdPrint function handles a client request to the server to print a document on a logical printer serviced by the server. The request contains the information needed by the server to print the document. If the request is accepted a unique jobId is assigned to the job, and the job is added to the database. The jobId and initial jobStatus are returned with pdError indicating that the request was accepted. If the request is not accepted pdError is returned containing an error code and message.

5.3.1.1 PdPrint Definition

```
int PdPrint (

   /* Input  */
   OBinding                 : PDBinding,
   jobInformation           : JobInformation,
   resultHandling           : JobResultHandling,
   eventHandling            : JobEventHandling,
   schedulingInstructions   : JobSchedulingInstructions,
   printerId                : PrinterIdentification,
   productionInstructions   : ProductionInstructions,
   documentDescription      : DocumentDescription,
   specialProcessing        : SpecialProcessingInstructions,
   security                 : SecurityAccessAccounting,

   /* Output */
   jobId                    : JobIdentifier,
   jobStatus                : JobStatus,
   pdError                  : PdError)
```

5.3.2 PdModify Function

The PdModify function handles a client request to modify a print job that has been accepted by the server and is still pending, held or in transit. Only the user who is authorized or the print server administrator can modify a job. Not all information may be modified, specifiacally the documentDescription information. If the request is accepted the effect is the same as if the job had been originally submitted with the modified parameter values, although scheduling of the original job may be impacted. If accepted and updated jobStatus is returned along with an acceptable pdError code. If rejected pdError contains the appropriate error code and message.

5.3.2.1 PdModify Definition

```
int PdModify (

   /* Input  */
   OBinding                  : PDBinding,
   jobId                     : JobIdentifier,
   jobInformation            : JobInformation,
   resultHandling            : JobResultHandling,
   eventHandling             : JobEventHandling,
   schedulingInstructions    : JobSchedulingInstructions,
   printerId                 : PrinterIdentification,
   productionInstructions    : ProductionInstructions,
   specialProcessing         : SpecialProcessingInstructions,
   security                  : SecurityAccessAccounting,

   /* Output */
   jobStatus                 : JobStatus,
   pdError                   : PdError)
```

5.3.3 PdReadJobStatus Function

The PdReadJobStatus function handles a client request inquirying about the status of a print job. If the returnAttributes parameter is false, the jobStatus of jobId is returned, if the returnAttributes parameter is true all the information about the print job is returned. Provided the request is accepted. The pdError parameter is returned with the appropriate value.

5.3.3.1 PdReadJobStatus Definition

```
int PdReadJobStatus (
```

```
   /* Input  */
   OBinding                  : PDBinding,
   jobId                     : JobIdentifier,
   returnJobAttributes       : BOOLEAN,

   /* Output */
   submissionTime            : UTCTime,
   jobStatus                 : JobStatus,
   pdError                   : PdError,

   /* Additional Output if returnJobAttributes is true */
   jobInformation            : JobInformation,
   resultHandling            : JobResultHandling,
   eventHandling             : JobEventHandling,
   schedulingInstructions    : JobSchedulingInstructions,
   printerId                 : PrinterIdentification,
   productionInstructions    : ProductionInstructions,
   documentDescription       : DocumentDescription,
   specialProcessing         : SpecialProcessingInstructions)
```

5.3.4 PdCancelJob Function

The PdCancelJob function handles a client request to cancel a print job. The request is accepted if the user is authorized to cancel the job and the job has not completed printing. If the job is actively printing when the request is made some pages may be printed before the job is successfully terminated. The jobStatus at the time a cancellation is returned. The pdError parameter is returned with the appropriate value.

5.3.4.1 PdCancelJob Definition

```
int PdCancelJob (

   /* Input  */
   OBinding                  : PDBinding,
   jobId                     : JobIdentifier,
   security                  : SecurityAccessAccounting,

   /* Output */
   jobStatus                 : JobStatus,
   pdError                   : PdError)
```

5.3.5 PdListJobs Function

The PdListJobs function permits a summary list of print jobs to be obtained from the print server. The filter argument is used to compile the list of jobs. A history about each job in the list is returned. With pdError containing the appropriate error code.

5.3.5.1 PdListJobs Definition

```
int PdListJobs (

   /* Input  */
   OBinding                  : PDBinding,
   filter                    : Filter,
   security                  : SecurityAccessAccounting,

   /* Output */
   serverLoad                : HierarchicalName,
   jobHistorys               : SEQUENCE OF JobHistory,
   pdError                   : PdError)
```

5.3.6 PdReadObjectAttributes Function

The PdReadObjectAttributes function handles client requests to obtain the attributes of a specific object resource. A filter is passed, the filter specifies an object of list of objects, a list of desired attributes to be returned is also passed. If the request is accepted the values for each specified attribute for each object are returned. The pdError structure contains the appropriate error code and message.

5.3.6.1 PdReadCharacteristics Definition

```
PdReadCharacteristics (

   /* Input  */
   OBinding                  : PDBinding,
   filter                    : Filter,
   attributeIds              : SEQUENCE OF AttributeId,
   security                  : SecurityAccessAccounting,

   /* Output */
   objectAttributes          : SEQUENCE OF Attribute,
```

```
    pdError                     : PdError)
```

5.4 Pdlib Administration Functions

The administrative library offers a rich set of functions that enable a printing systems administrator to maintain the printing environment. Also provided are functions that allow clients, servers and supervisors to notify each other about special events. The maintenance functions include things like starting and stopping objects, registering and unregistering objects and users, etc. Functions of this nature can only be accessed by authorized users. Authorization will be done by instance and ACL lists for each function. Most of the commands effect an object in some way. An object can be a a supervisor, or a server. Actions for printers are done through their supervisors.

Each function is of type integer, and error code is returned in one of pdError's fields. The error code is also returned as the value of the function call itself.

5.4.1 PdResubmitJobs Function

The PdResubmitJobs function allows a system administrator to resubmit print jobs from one object to another. This means print jobs can be moved from one supervisor to another or be resubmitted to a new supervisor on a new server. If accepted the function returns a list the of the jobId's that where resubmitted.

5.4.1.1 PdResubmitJobs Definition

```
int PdResubmitJobs (

    /* Input  */
    OBinding                    : PDBinding,
    filter                      : Filter,
    newServer                   : HierarchicalName,
    newPrinter                  : HierarchicalName,
    security                    : SecurityAccessAccounting,

    /* Output */
    jobList                     : SEQUENCE OF JobIdentifier,
    pdError                     : PdError)
```

5.4.2 PdClean Function

The PdClean function allows the administrator to clean, ie., remove all print jobs. The objectClass and objectId specify the strength of the request. The objectClass and objectId can represent the entire server, a specific supervisor or a logical printer. If accepted, all print jobs with job states: pending, transit, held or active for the specified objectClass are removed. If notify is true, all effected users are notified on the fate of their print requests. The parameter pdError is returned with the appropriate value, 0 if accepted or some error code and message.

5.4.2.1 PdClean Definition

```
int PdClean (

   /* Input  */
   OBinding                  : PDBinding,
   objectClass               : HierarchicalName,
   objectId                  : HierarchicalName,
   notify                    : BOOLEAN,
   security                  : SecurityAccessAccounting,

   /* Output */
   pdError                   : PdError)
```

5.4.3 PdPromoteJob Function

The PdPromoteJob function allows the administrator to place a specified print job at the top of the list of pending print jobs, ie., it would become the next job to be printed. The jobId of the effected job and security information of the requester of provided. If accepted the specified job is promoted to the top of the print queue.

5.4.3.1 PdPromoteJob Definition

```
int PdPromoteJob (

   /* Input  */
   OBinding                  : PDBinding,
   jobId                     : JobIdentifier,
   security                  : SecurityAccessAccounting,

   /* Output */
```

```
   pdError                    : PdError)
```

5.4.4 PdInitialize Function

The PdInitialize function allows the administrator to initialize an object. Initialization allows the object to contact Hesiod to retrieve all the information that it needs. The object may be a server or a supervisor.

5.4.4.1 PdInitialize Definition

```
int PdInitialize (

   /* Input  */
   OBinding                   : PDBinding,
   objectClass                : HierarchicalName,
   objectId                   : HierarchicalName,
   security                   : SecurityAccessAccounting,

   /* Output */
   pdError                    : PdError)
```

5.4.5 PdEnable Function

The PdEnable function allows the administrator to turn an object on, once enabled the object can begin accepting new print requests. The error status is returned in pdError indicating the success or failure of this request.

5.4.5.1 PdEnable Definition

```
int PdEnable (

   /* Input */
   OBinding                   : PDBinding,
   objectClass                : HierarchicalName,
   objectId                   : HierarchicalName,
   security                   : SecurityAccessAccounting,

   /* Output */
```

```
   pdError                    : PdError)
```

5.4.6 PdDisable Function

The PdDisable function allows the administrator to turn an object off, once disabled the object nolonger accepts print requests, however, it will continue to process those requests that were previously accepted. If the object is a supervisor then only the jobs assigned to it are affected, if the object is a server then requests are no longer accepted but all supervisors will print jobs currently assigned to them.

5.4.6.1 PdDisable Definition

```
int PdDisable (

   /* Input */
   OBinding                   : PDBinding,
   objectClass                : HierarchicalName,
   objectId                   : HierarchicalName,
   security                   : SecurityAccessAccounting,

   /* Output */
   pdError                    : PdError)
```

5.4.7 PdStop Function

The PdStop function allows the administrator to abruptly stop any object. Once stopped the object will no accept or process print requests.

5.4.7.1 PdStop Definition

```
int PdStop (

   /* Input */
   OBinding                   : PDBinding,
   objectClass                : HierarchicalName,
   objectId                   : HierarchicalName,
   security                   : SecurityAccessAccounting,
```

```
   /* Output */
   pdError                    : PdError)
```

5.4.8 PdShutDown Function

The PdShutDown function allows the administrator to cleanly stop any object. Once stopped the object will no accept or process print requests. In the process of shutting down the object will finish printing the active job.

5.4.8.1 PdStop Definition

```
int PdShutdown (

   /* Input */
   OBinding                   : PDBinding,
   objectClass                : HierarchicalName,
   objectId                   : HierarchicalName,
   security                   : SecurityAccessAccounting,

   /* Output */
   pdError                    : PdError)
```

5.4.9 PdModifyAttributes Function

The PdModifyAttributes function allows the administrator to modify the attributes for a specified object.

5.4.9.1 PdModifyAttributes Definition

```
int PdModify (

   /* Input */
   OBinding                   : PDBinding,
   objectClass                : HierarchicalName,
   objectId                   : HierarchicalName,
   objectAttributes           : SEQUENCE OF Attribute,
   security                   : SecurityAccessAccounting,
```

```
   /* Output */
   pdError                    : PdError)
```

5.4.10 PdRegister Function

The PdRegister function allows the administrator to register a new object. When registering a new object a list of attributes with this object are also provided.

5.4.10.1 PdRegister Definition

```
int PdRegister(

   /* Input */
   OBinding                   : PDBinding,
   objectClass                : HierarchicalName,
   objectId                   : HierarchicalName,
   objectAttributes           : SEQUENCE OF Attribute,
   security                   : SecurityAccessAccounting,

   /* Output */
   pdError                    : PdError)
```

5.4.11 PdRegisterUser Function

The PdRegisterUser allows the administrator to register a specified user as being a priveledged user of, or administrator for the specified object.

5.4.11.1 PdRegisterUser Definition

```
int PdRegisterUser (

   /* Input */
   OBinding                   : PDBinding,
   objectClass                : HierarchicalName,
   objectId                   : HierarchicalName,
   userName                   : HierarchicalName,
   userId                     : INTEGER,
```

```
   security                  : SecurityAccessAccounting,

   /* Output */
   pdError                   : PdError)
```

5.4.12 PdUnRegisterUser Function

The PdUnRegisterUser allows the administrator to unregister a specified user. Any special priveledges are revoked.

5.4.12.1 PdUnRegisterUser Definition

```
int PdUnRegisterUser (

   /* Input */
   OBinding                  : PDBinding,
   objectClass               : HierarchicalName,
   objectId                  : HierarchicalName,
   userName                  : HierarchicalName,
   userId                    : INTEGER,
   security                  : SecurityAccessAccounting,

   /* Output */
   pdError                   : PdError)
```

5.4.13 PdReadyForJob Function

The PdReadyForJob function tells a supervisor or server that a printer is ready to start another job.

5.4.13.1 PdReadyForJob Definition

```
int PdReadyForJob (

   /* Input */
   OBinding                  : PDBinding,
   objectClass               : HierarchicalName,
   objectId                  : HierarchicalName,
```

```
    security                    : SecurityAccessAccounting,

    /* Output */
    pdError                     : PdError)
```

5.4.14 PdJobComplete Function

The PdJobComplete function tells a supervisor or server that a printer has completed processing a specific job.

5.4.14.1 PdJobComplete Definition

```
int PdJobComplete (

    /* Input */
    OBinding                    : PDBinding,
    objectClass                 : HierarchicalName,
    objectId                    : HierarchicalName,
    jobId                       : JobIdentifier,
    security                    : SecurityAccessAccounting,

    /* Output */
    pdError                     : PdError)
```

5.4.15 PdPause Function

The PdPause function allows the administrator to pause an object, a paused server will continue to accept job but will not schedule, a paused supervisor stops sending data to the printer, and a paused printer stops immediately.

5.4.15.1 PdPause Definition

```
int PdPause (

    /* Input */
    OBinding                    : PDBinding,
    objectClass                 : HierarchicalName,
    objectId                    : HierarchicalName,
```

```
   pauseAt                  : HierarchicalName,,
   security                 : SecurityAccessAccounting,

   /* Output */
   pauseJobId               : JobIdentifier,
   checkpoint               : Checkpoint,
   pdError                  : PdError)
```

5.4.16 PdResume Function

The PdResume function allows the administrator to resume an object, a resumed server will start scheduling, a resumed supervisor starts sending data to the printer, and a resumed printer starts printing.

5.4.16.1 PdResume Definition

```
int PdResume (

   /* Input */
   OBinding                 : PDBinding,
   objectClass              : HierarchicalName,
   objectId                 : HierarchicalName,
   checkpoint               : Checkpoint,
   security                 : SecurityAccessAccounting,

   /* Output */
   pdError                  : PdError)
```

5.4.17 PdAdvertise Function

The PdAdvertise function causes an object to register (or reregister) itself with its superior. Supervisors register themselves with the server, and servers register themselves with Palladium.

5.4.17.1 PdAdvertise Definition

```
int PdAdvertise (

   /* Input */
```

```
   OBinding                  : PDBinding,
   objectClass               : HierarchicalName,
   objectId                  : HierarchicalName,
   security                  : SecurityAccessAccounting,

   /* Output */
   pdError                   : PdError)
```

5.4.18 PdUnregister Function

The PdUnregister function allows the administrator to unregister an object.

5.4.18.1 PdUnregister Definition

```
int PdUnregister (

   /* Input */
   OBinding                  : PDBinding,
   objectClass               : HierarchicalName,
   objectId                  : HierarchicalName,
   security                  : SecurityAccessAccounting,

   /* Output */
   pdError                   : PdError)
```

5.4.19 PdReportEvent Function

The PdReportEvent function reports that some event has occurred. The function accepts the following arguments:

5.4.19.1 PdReportEvent Definition

```
int PdReportEvent (

   /* Input */
   OBinding                  : PDBinding,
   objectClass               : HierarchicalName,
   objectId                  : HierarchicalName,
```

```
   eventLog                  : HierarchicalName,
   eventTime                 : UTCTime,
   eventType                 : HierarchicalName,
   eventInfo                 : IA5String,

   /* Output */
   pdError                   : PdError)
```

5.4.20 PdInterrupt Function

The PdInterrupt function allows the administrator to interrupt or suspend a supervisor.

5.4.20.1 PdInterrupt Definition

```
int PdInterrupt (

   /* Input */
   OBinding                  : PDBinding,
   objectClass               : HierarchicalName,
   objectId                  : HierarchicalName,
   interruptAt               : HierarchicalName,
   interruptingJobId         : JobIdentifier,
   security                  : SecurityAccessAccounting,

   /* Output */
   interruptedJobId          : JobIdentifier,
   checkpoint                : Checkpoint,
   pdError                   : PdError)
```

Chapter 6

System Management

Central management of the entire Palladium Print System is critical to the successful operation of the Athena environment. This chapter discusses the management of the Palladium System. It will introduce the functions provided to perform configuration management, fault management, security management and accounting management.

6.1 Configuration Management

The entities that make up the Athena Print System, print servers, print supervisors, and printers have been described in other chapters. In this section, tools for building and managing distributed printing services constructed from these entities will be described.

Three views of the Palladium configuration will exist and need to be managed:

1. The Hesiod view: This is the view that most users will see of Palladium. Hesiod stores information about the Palladium servers, supervisors and printers (e.g. names, addresses, characteristics, etc.). The Hesiod database is updated from SMS periodically. It can become inconsistent with the information in SMS whenever SMS is changed without an immediate update of the Hesiod database.

2. The SMS view: The SMS database contains a description of a complete Palladium configuration. Because hardware and software can fail, the SMS view of the Palladium configuration may not accurately reflect the real configuration at any point in time.

3. The Real-World view: At any point in time some set of servers, supervisors, and printers will actually be functioning. This represents the actual state of the Palladium system at the point in time.

These views of the Palladium system must be assumed inconsistent due to timing delays caused by the fact that SMS does not track the state of the network in real time. A significant part of managing the Palladium print system will deal with managing these time delays and inconsistencies

as a Palladium system changes over time. The Palladium Print System has been designed to allow the creation of simple tools to help with this management task.

6.1.1 Naming and Addressing

Both Palladium entities (server, and supervisor/printer) need names and addresses that can be used to reference them. All entities will have an **entity name** and one or more **logical names** to reference it.

Because each entity is globally visible in the distributed system (through their entries in the name server), the **entity names** and **logical names** must be globally unique. To make it easy to create, remember and use these globally unique names, they should be constructed as hierarchical names (e.g. nil.athena.mit.edu). This will also allow entities in different clusters or management domains to have the same local name (e.g. nil), while still having globally unique names across cluster or domain boundaries (e.g. nil.athena.mit.edu and nil.lcs.mit.edu).

The names are unique only in their full resolution, i.e. nil.printer.ns.athena.mit.edu, where ns.athena.mit.edu is the domain of the Hesiod nameservers.

Supervisor/printer: Each supervisor drives one and only one printer. The same **entity name** is used to reference both the supervisor and the physical printer. But since no one actually refers to the physical printer with any operation, the name of the physical printer is not very important. A supervisor/printer name is independent of the name of the server that provides scheduling for the printer driven by its supervisor.

Servers: A Server entity name is independent of the of the names of the supervisor/printers that the server represents.

Entities are referenced in all user level print commands (but not system management commands) by one of its **logical names.** All logical printer names are available to users through Hesiod. The specific entity names are for system management use. They are not generally available. Each **logical name** may be associated with one or more supervisor/printer and so it may have multiple entries in its Hesiod domain. Likewise each supervisor/printer may be referenced by more than one logical name. Unless a logical printer is disabled, a server will accept job requests for all logical printers defined for a supervisor/printer that it supports.

Each entity also needs a globally unique address. Servers and supervisor/printers are addressed with an rpc address. This address will be used by any server or client wishing to directly access the entity. Palladium supervisor addresses will not normally be made available to users and their clients, since supervisors do not provide the queuing and job management services available from Palladium servers. This is under system amanagement control, however, as any desired address may be associated with a logical name, whether it be for a server or a supervisor/printer.

There are a number of relationships between printer entities and their names. For a complete description of these relationships see the chapter on the Palladium Database. Also see the Database chapter for information on how the Hesiod name server is loaded from the SMS database.

6.1.2 Configuring and Reconfiguring

The physical configuration of the Palladium Print System is maintained in the SMS database. The configuration is defined by the relationships between the components. Definition of the physical configuration is changed by editing the SMS database entries of the print system components to change the relationships between the components.

When each component of the print system is booted, it obtains its configuration information from the system database. Given the current Athena design, this configuration information will actually come from a local cache created by SMS. Supervisor/printers will register themselves with the server specified in the database for them (passing the server all appropriate printer characteristics and resource definitions).

SMS will periodically cache information used by Palladium in the Hesiod name server and in a local configuration file for each server and supervisor/printer. New supervisor/printers and servers can be added by simply changing the SMS database and booting or rebooting the affected components once local caches have been updated. But, operators and users must normally wait for hours for caches to be updated before they can use a new component. Speedy reconfiguration is often desired (e.g. when reconfiguring to recover from a component failure). So, when new printers and servers are added, they can be more quickly integrated into the Palladium system by manually updating the appropriate configuration files and Hesiod. The official configuration in SMS must also be updated to assure that later updates of cached information do not erase desired manual changes. These manual changes should always be done with care to ensure that no unwelcome side-effects occur.

Methods for displaying and editing Palladium specific information in SMS may be provided through a forms based SMS Palladium editor. This editor may also check the validity of the SMS Palladium information.

Further information about management of the configuration of the Palladium Print System will be provided in a Palladium System Management Guide.

6.1.2.1 System Creation

When a new print service must be created (e.g. during the creation or reconfiguration of a cluster or node) the following steps are take to define and install the components of the system:

1. A management utility allows the system administrator to specify the characteristics and constraints of the printers to be added. This information includes (but is not limited to):
    - supervisor/printer name
    - logical printer name(s)
    - printer device location (node:/dev/lpX)
    - print formats supported
    - finishing features

- paper sizes
- per job defaults

This information is stored and maintained by SMS. The management utility also creates any program images, script entries and local configuration files required for the initialization utilities to start the printer (not required by low-end printers, but may be required of high-end laser-printers).

2. A management utility allows the system administrator to specify the characteristics and constraints of the supervisor/print to be added. This information includes:

   - name of the print server from which this supervisor/printer accepts jobs
   - type of supervisor/printer (independent agent or dedicated)
   - access rights of each user to the operations provided by the supervisor/printer.

   This information is stored and maintained by SMS. The management utility also creates any program images, script entries and local configuration files required for the initialization utilities to start the printer supervisor.

3. A management utility allows the system administrator to specify the characteristics and constraints of the print servers to be added. This information includes:

   - characteristics of the services provided by the print server (e.g. print services, plot services, etc.)
   - access rights of each user to the services provided by the printer server.

   This information is stored and maintained by the name service. The management utility also creates any program images and script entries required for the initialization utilities to start the print server.

Note that this is not a monolithic procedure. Appropriate subprocedures may be used independently to add supervisor/printers to existing printer servers, or print servers to existing print systems.

6.1.3 System Initialization

When an existing print system is started (e.g. server or node boot-time) the following steps are taken to initialize the components of the print system:

1. The shell routine responsible for initializing print systems reads a script or database to learn which components to start. This routine should use a name server as much as possible to learn the details associated with starting individual components. Scripts should contain as little 'hard-coded' information as possible. Ideally a script would contain only the location of the nearest name server and the names of the other components which must be started. All other information about the components to be started (most importantly their addresses) should be accessed through the name server.

2. The print server (the component which provides scheduling across a set of supervisor/printers) is invoked by a routine or script and told it's name and it's address.

3. The print server immediately calls the name server to learn service-specific details about itself. It then performs any necessary internal initialization by rading a server configuration file.

4. Once initialized, the server is ready to start taking management requests. But, since it does not have any printers registered, it can not start taking user requests. The print server is unable to take requests until at least one supervisor/printer registers with it. Note that when a server is in this state it cannot take any user requests, but it will allow management requests, such as 'PdRegister'.

5. The supervisor/printers are invoked by the routine or script independent of the print server. They are told their name and address.

6. Each supervisor/printer immediately uses the printer database to learn service-specific details about itself. It then performs any necessary internal initialization.

7. Once initialized, the each supervisor/printer determines whether it is operating as an independent agent or as an agent of a print server. This is determined from information stored in the supervisor/printer's database entry. If the supervisor/printer is operating as an agent for a print server, the name of the print service to which it owes allegiance is specified in the supervisor's database entry.

8. If the supervisor/printer is an independent agent, start-up is complete. The supervisor simply waits for any (appropriately authorized) client to request service.

9. If the supervisor/printer is an agent of a print server it uses the name of the server stored in its database information to look up the address of the server in the name service. The supervisor then registers itself with the server by calling the 'PdRegister' operation. This operation allows the printer supervisor to pass the server necessary static and dynamic information about the printer supervisor and its printers. The static information originated in the supervisor/printer's database entry. The dynamic information reflects the supervisor/printer's current state. The 'PdRegister' operation will be invoked each time important information associated with the supervisor/printer changes.

6.2 Resource Management

In addition to servers, supervisors/printers and jobs, the Palladium Print System must manage definitions of a broad range of print system resources. These resources include media, fonts, forms, printer protocols (printcap) and printer defaults. These resource definitions will be maintained in the SMS database.

Methods for displaying and editing Palladium resources in SMS may be provided through a forms based SMS Palladium editor. This editor may also check the validity of the SMS Palladium resource information.

Further information about management of the Palladium Print System Resources will be provided in a Palladium System Management Guide.

6.3 Failure Management

In a real world environment, there will be failures that must be dealt with quickly. The Palladium print system will allow operations personnel to deal with a failure in the software or hardware and make printing resources available again in a minimal amount of time.

While component software modules increases the number of communication failure points, their separation allows the server to continue functioning even if a supervisor or its related printer hardware fails. Errors reported via the RPC mechanism and timeout heuristics can both be applied in the intelligent automatic handling of many failure modes. These timeout heuristics may be implemented as time-outs from the last successful operation. Periodic status queries will occur when a specified amount of time has passed after the last operation.

The server will maintain status information for itself, and supervisor/printers registered with it. These messages will be updated by the operator and the supervisor/printers via requests and responses to the server. Likewise, each supervisor/printer will maintain status information for itself. The PdReadStatus function will be used to get this status information from both servers and supervisor/printers.

6.3.1 Component Crash Scenarios

For each of the failure modes, there is recovery path that will allow the system to resume printing. While the recovery may not always be transparent to the user, in most cases it will be. Barring a host hardware catastrophe, the operator will be able to maintain the availability of printing resources.

Printer Hardware Fails. In the case of printer hardware failure, the supervisor/printer can inform the server of a hardware failure. The server will then modify its notion of the printer's status, and server notification mechanisms will inform an operator of the failure. The failure of the printer is also available to a user's status queries. In the case of a soft failure, i.e. paper jam, etc. the appropriate manual action can be taken. In the case of a hard failure, the appropriate administrative steps can be taken to repair the hardware, and the operator can instruct the server to block any future print requests for this particular printer.

Supervisor Fails. If the printer supervisor fails, then the server informs the operator of the failure and reports status (or blocks) for all future requests. Since the supervisor/printer cannot be depended on for transmission of the failure mode, the server will be required to detect this failure mode either by a status message from the failing supervisor/printer, or by a timeout heuristic. The operator can take the appropriate action to restart the supervisor/printer.

Server Fails. If a server fails, another server can be started to replace the failed instance. If a server fails because of host hardware failure, then any supervisor/printer (and associated printer

hardware) connected to that hardware becomes unavailable. Any client requests to that server will fail. As long as the supervisor/printer continues to function, it can be instructed to register itself with an operating print server and produce output. If a server's hardware fails and the supervisor/printer lives on a separate host, then the supervisor/printer can be instructed to register itself with another print server. This will allow printers to continue functioning if a hardware or software failure prevents the restoration of the former controlling server.

6.3.2 Job Failure Scenarios

A job failure, that is, a job which completes abnormally, can occur because of a document error, printer hardware failure, server/supervisor failure, or an operator signal to complete or flush the job without regard to status. When a job completes abnormally for any reason, the log message will indicate why the job was terminated. The source of the problem will also be reflected in the job completion message that comes from the supervisor/printer. The server can thus note the abnormality when making an account report. Site policy can determine how to account for the resources used in such a case. If a user requests notification of this event, then any abnormal completion status can be returned to the user. However if the job is successfully completed then the supervisor/printer sends an acknowledgement to the server which will then delete the job from the jobs data base (unless the user requested that the job be held).

If there is a job failure due to critical disk space shortage on the server host it can be classified as a transient job failure. The user might be returned an error message saying "Insufficient disk space on server, try later", and the server won't accept the job. There may be cases where the types of jobs can be classified as large, or computationally intensive, or even exceptional. Such services (printers) should have distinct names in the name space. A user may try to print an ASCII text file on a Postscript printer. If the filter does not have the capability of transforming ASCII to Postscript it should be considered a job failure and the user be notified.

Supervisor/Hardware Failure If job failure or abnormal completion occurs because of supervisor/printer failure, the user can determine from a status query if resubmission is necessary, and if it is possible. Failure may have occurred because of an error in the submitted document, and future resubmissions should not occur until the document has been corrected. If the failure occurred because of a paper jam, hardware failure, etc. the user can examine any partial output and resubmit the job as necessary.

Server Failure If a server crash causes a job failure, then all server information would be lost. To prevent this, the in-core databases will be implemented as write through caches. This will allow for rapid processing of jobs and efficient recovery of state when a server instance fails.

6.3.3 Failure Recovery and Reconfiguration

Administrative utilities will be used to resubmit jobs scheduled for a failed printer. The utility will use a Palladium library call to modify the printer name attribute of the job. If the destination printer is not under control of the local server, then the resubmission utility, PdResubmitJobs, can be used to resubmit the job to the appropriate server. The job's status on the local printer will be

appropriately modified.

Remapping of printer name to print server will occur on both temporary and permanent basis. Reconfiguration on a permanent basis will be done with the Service Management System (SMS).

6.4 Security Management

To maintain the integrity of the system and ensure the credibility of the systems usage controls, security in the Palladium print system will be required to:

- control user access to administrative operations
- control user access to user operations (e.g. PdPrint, PdListJobs)
- ensure the accuracy and credibility of accounting
- implement a credible quota/authorization system

Security is provided by verifying the identity of users and entities ("authentication"), and authorizing users' access to operations.

6.4.1 Authentication

Authentication of the user will be done through Kerberos. A printing service supplied by the hosts running print servers will exist. A user will be required to present tickets for said print service in order to successfully submit a request to the server. Depending on the level of security desired, this requirement can be made for all requests or a selected subset, such as administrative requests.

The supervisor/printer must trust that the server has properly authenticated the user when the job was submitted to the user. In addition, the supervisor/printer must trust that the server has not improperly modified the contents of the job request (e.g. changing the account name to which this job is to be charged). This ensures that the supervisor/printer can credibly authorize usage of supervisor/printer operations. This trust must be created by properly authenticating and authorizing the server to the supervisor/printer. Sometimes jobs may have to be resubmitted from one server to another in case of, say, a printer failure. Kerberos can be used to make the servers trust each other in a secure way.

6.4.2 Authorization and Access Control

Each operation supported by the server will have its own Access Control List (ACL). There is a currently available library, compatible with kerberos, which supports this design.

6.5 Accounting and Quota Management

In order to accurately account for resources used, the Palladium server will log an accounting record for each job completed. The record will have a well-known format, consisting of a timestamp, user ID, the server/supervisor submitting the report, and value/unit pair(s) identifying the resource(s) expended on the user's behalf. These resources may be page printed, printer time used, number of bytes processed, or other units that a supervisor may wish to report.

User quotas can be set to support separate limits for different types of media (should be limited to distinctively different media type but not every possibly different type). Both soft (warning message only) and hard (no servise) limits may be enforced by the server, if the system manager desires to use them.

The accoutning logging will be implemented with a well-accepted system logging mechanism. A utility will be developed that will read and summarize the logs created by Palladium servers. The server will also keep track of the resources used by each user and use this information for the enforcement of hard and soft quotas. In the BSD/Athena environment, *syslog* is the appropriate logging facility. Palladium will use the *syslog* interface for all event logging, until it becomes clear that somthing better is necessary.

Communication between the involved principals must be secure if the logged accounting information is to be considered credible and authoritative. The supervisor reports accounting and logging information to the server using the PdReportEvent PdLib function. Because the server authenticates all comunications with the supervisor before sending it a print job, this communications channel is expected to be secure.

A mechanism such as the one characterized above will provide for accurate record-keeping, but will not necessarily enforce a quota system. Such a system should be centralized. A master list for the principals and quotas will exist. A daemon listening on a well-known port will accept requests to modify the master list or request "authorization" to use a given amount of a resource. Communication with this service would require authentication.

The server should also provide a usage record separate from the accounting record, with measures taken to remove user identity. This information can be used for statistics, preventative maintenance schedules and performance tuning. It is important for privacy and security considerations that this information be kept separate from accounting records.

6.5.1 Quotas

When a supervisor/printer generates a job completion message, the information contained in the message will detail the resources used to complete the job. The server will send an accounting message to the central accounting manager and simultaneously log a local message. The central accounting manager will process the messages and update user status at a specified frequency.

The central accounting host will log any users exceeding their specified usage and maintain a list of these users in a file. When a print request is made, the server will request authorization from the central accounting manager to accept a job. If the user appears on the "flagged" list,

then authorization will be denied. In any other case, authorization will be granted. This quota authorization is completely independent of the authorization to access a server.

6.6 Management Functions

The Palladium functions that modify the state of print entities are described in Table 6.1 in a generic fashion. The first column lists all appropriate generic printer management functions and shows how the function will be implemented for Palladium (c=Shell Command, t=Toolkit function, l=Pdlib function). The rest of the table shows what effect the function will have on a Palladium server, supervisor, or printer. The interfaces for these generic functions can be found in the appropriate chapters on the shell interface, the toolkit interface, or the library interface.

In addition to these functions that modify the printer system state, there exist a collection of functions that retrieve state information. Two types of functions are available: one to retrieve an entity's characteristics and a second to retrieve its status. Characteristics and status information can be retrieved for servers, logical printers, supervisors, and printers.

Function	Server	Supervisor/printer
Start(c)	Server instance is brought up.	Supervisor instance is brought up and printer is started.
Initialize(c,t,l)	Server (re)initializes internal state information by reading configuration information from database.	Supervisor (re)initializes internal state information by reading configuration information from database. Necessary setup information is sent to printer.
Enable(c,t,l)	Accept user requests for the server or a logical printer.	Request new job(s) for printer and accept jobs submitted.
Disable(c,t,l)	Reject user requests for the server or a logical printer. Continue to accept management requests. Schedule jobs unless also paused.	Finish current print job. Don't request or accept additional jobs.
Pause(c,t,l)	Stop scheduling jobs for the server or a logical printer. Continue accepting management requests. Continue accepting user requests, unless also disabled.	Stop printing current job(s) immediately, without losing state so printing may be resumed later.
Resume(c,t,l)	resume scheduling all jobs by the server or for a single logical printer.	Resume printing job(s) that were paused.
Shutdown(c,t,l)	Perform an orderly server shutdown - disable, pause, then stop.	Perform an orderly supervisor (and printer) shutdown - disable, then stop.
Stop(c,t,l)	Kill this server instance quickly, with little, if any, grace if not paused and disabled.	Kill this supervisor and its printer quickly and with little grace if not disabled.

Table 6.1: Palladium System Management Functions

Chapter 7

Palladium Server Design

The Palladium Server is designed to process print and print maintenance requests, printer and server maintenance requests and management requests. The server accepts requests via the RPC mechanism, validates each request, and performs the requested procedure. Each request returns a status integer and possibly data. The returned data is placed in return parameters provided by the client.

Each request performed by the server will read or modify the state of the server databases and indicate its actions to the appropriate subsystems. These subsystems will, in turn, act based on these signals.

7.1 Server Components

The server is composed of several major component subsystems, as shown in Figure 7.1. As the server will be a single process, tasking will be employed to permit pseudo-simultaneous operation of the constituent subsystems.

7.1.1 RPC Interface

The Palladium Server exports a library, PdLib, of remote procedure calls for accessing the server. The library interface is defined in a separate chapter. The library completely defines the possible operations on the server.

7.1.2 Tasking System

The use of an RPC mechanism requires a synchronous relationship between the Palladium Client and Server. In order to handle all server requests in a reasonable time period, the server will require some method of simulating parallel execution of multiple tasks. A Tasking System will be used. When the Palladium Server receives a request from a client, it will use the tasking system

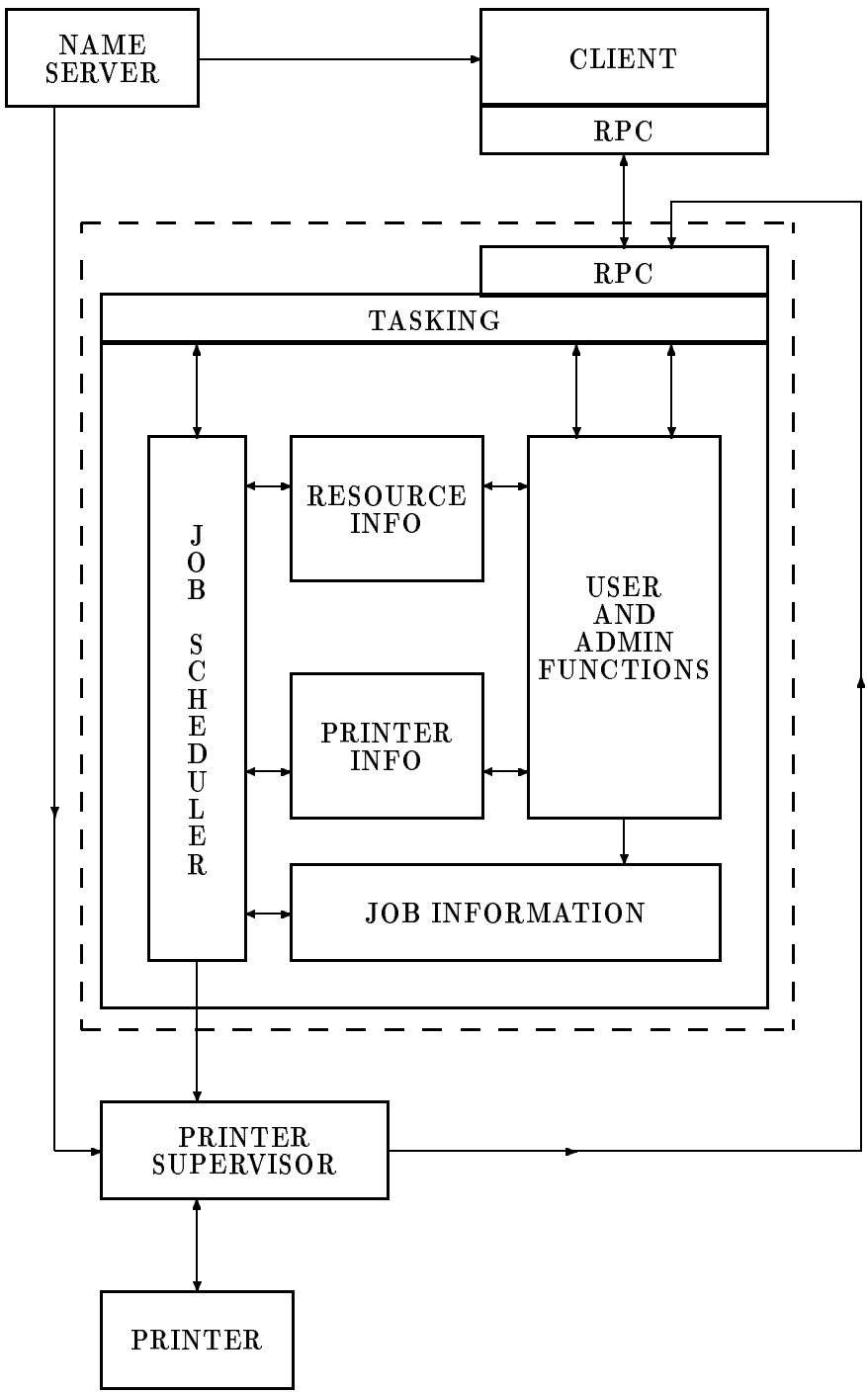

Figure 7.1: Palladium Print System Model

to create a thread to handle the RPC request. Internal calls, like calls to the Job Scheduler, will also make use of the tasking system. For internal calls, the tasking system will support inter-task communication.

7.1.3 Server Databases

The Palladium Server uses and manipulates the Printer Table, Resource Table, and the Print Job Table. The Printer Table is general information and status about each printer represented and managed by the server. The Resource Table defines the resources supported by the printers on the server.

The Print Job Table is information and status on the documents to be printed. Each job will be labeled by a unique job id. The job id is a string that is composed of **server** *and* **printer name**, for identification, and a **time stamp** to provide uniqueness on the server and easy scheduling reconstruction.

The server will maintain a spooling area in the filesystem. The naming scheme for the spooling area will imbed meaning in the filenames. A new server instance will be able to quickly reconstruct the Print Job Information database from artifacts in the spooling area. Each server will have its own job spooling area, whose location it learns during initialization. More details can be found in the Databases chapter.

7.1.4 Job Scheduler

The Job Scheduler performs many functions. It primary purpose is to act on jobs based on signals it receives. It reports status and and significant events to the client; it maintains all accounting and error log information for a job; matches jobs to printers and printers to jobs. A more detailed discussion is in the Job Scheduler chapter.

7.2 User Requests

The PdLib is divided into two sections, User functions and Administrative functions. See the Library chapter for details on their arguments and operation. The following is a list of the available User PdLib functions:

1. PdPrint - process requests for printing
2. PdModify - process requests to modify pending jobs
3. PdReadJobStatus - Read Job Status
4. PdCancelJob - Cancel Job
5. PdListJob - List Jobs

6. PdReadCharacteristics - Read Characteristics of Entity
7. PdReadStatus - Read Status of specified entity

7.3 Administrative Requests

The Administrative functions are used to manage the server. See the Library chapter for more details. The following are requests available from the server:

1. PdSubmitJobs - Redirects jobs to a different enitity.
2. PdClean - Removes all of the jobs from a specified entity.
3. PdPromoteJob - Move the specified job to the top of the queue for the specified entity.
4. PdInitialize - Initialize an entity, ie., get all the information needed in order for the entity to perform correctly.
5. PdEnable - Enamble the entity.
6. PdDisable - Disable the entity.
7. PdStop - Abruptly stop an entity.
8. PdShutdown - Gracefully stop an entity.
9. PdModifyCharacteristics - Modify the characteristics of an entity.
10. PdRegister - Register an entity.
11. PdUnregister - Unregister an entity.
12. PdRegisterUser - Register a user as being in charge of the entity.
13. PdUnregisterUser - Unregisters a user as being in charge of the entity.
14. PdReadyforJob - Inform an entity that a printer is ready for a job.
15. PdJobComplete - Inform an entity the a printer has completed a job.
16. PdPause - Set an entity's state as paused.
17. PdResume - Set an entity's state as resumed.
18. PdAdvertize - Force an entity to advertise itself to it's superior entities.
19. PdReportEvent - Report an event to an entity.
20. PdFileXfer - Used to transfer files for jobs.

7.4 Server Operation and State

The server keeps its own view of its state and the state of the printers registered with it. The server itself is either "enabled" or "disabled," indicating the ability of a client to submit a user request, and "paused" or "resumed", indicating it's ability to output print jobs to Supervisors. Administrative requests are still allowed in either case. The state of the associated printers is obtained via requests to the printer supervisor. As long as calls to the supervisor are successful, the printer is assumed to be up and running. It may be "idle," "active," "enabled," "disabled," "paused," or "resumed," but is nonetheless considered operational. More details can be found in the Supervisor chapter.

Because of this granularity in status checking, the server only discovers the supervisor state after each call. Thus, the server's view of the supervisor may be inconsistent with the supervisor's actual state. To limit these occasions of inconsistency, the administrator will supply the server with a time-out value. If a period of time longer than the time-out value has elapsed since the last successful call to a supervisor, a status query will be made to the supervisor. Should this query fail, the printer can be considered "down."

If a printer is considered non-operational, the appropriate notification of the operator will take place. A job currently active on that printer is considered "aborted." The system operator can intervene at this stage and change the state of the job to either "completed" or "pending," as appropriate. The supervisor must re-register with the server before any further printing can occur.

7.5 Job State

Once successfully submitted, a print job can be in one of several states. A normally scheduled job which is waiting for an available printer is considered "pending." A currently printing job is "active." A successfully submitted job whose associated data (i.e. text) has not been completely transferred is in "transit." A job which is waiting for operator intervention or some other normally requested intervention is on "hold." A job which is finished normally is called "completed," while one that has finished abnormally for any reason is labelled "aborted."

There is a complete specification of transitions to be included which fully describe the succession of states for any job.

Chapter 8

Palladium Databases

This section describes the elements of the Palladium Data Bases, their relationships and their interfaces. The Data Base is presented using a relational database model. Data is manipulated with a single application interface. This consists of a number of functions which forms the **Palladium Database library**. The functions will resemble some of the generic Relational Data Base routines like find, store, join, delete, etc. These may be implemented by using any desired underlying data storage facility (Relational data base, hashed files, text files, Name Service, etc.). The basic idea is to ensure that eventual replacement of the data base package does not cause too many problems.

The following Print Server tables will be used:

J - Print Job Table : contains the documents to be printed and the information needed to print them correctly.

D - Device Table : contains information about the printer and its supervisor. This includes device name, supervisor name, server name, etc. All the **currently chosen** characteristics for a device are stored here. Information that changes quickly, such as device states, is explicitly excluded from this table.

S - Server Table : The table holds all the identification and addressing information for a given server, as well as the location of its spooling directory.

L - Logical Printer Table : contains information about logical printers. A Logical Printer is an alias equivalence for one or more printers. But it is not just an alias because associated with this logical printer are some logical printer characteristics and job defaults.

A - Association Table : handles the many to many mapping from device names to logical printer names. We refer to each device also by their logical name. Therefore for each device name there is at least one logical printer name which will map to this device and can also attribute certain default characteristics.

T - Type Table : contains the type information for a given printer. This contains all the **possible** characteristics and the device table (D) contains only the **current** characteristics.

M - Media Table : The Media relation contains the definitions of media supported by printers.

F - Fonts Table : The Fonts relation contains the definitions of fonts supported by printers.

FO - Forms Table : The Forms relation contains the definitions of forms supported by printers.

C - Character Sets Table : The Character sets relation contains the definitions of character sets supported by printers.

All the tables mentioned above will be loaded into the SMS Management System. The appropriate subsets will be stored as configuration files for Servers and Supervisors.

8.1 Print Server Relations

8.1.1 Print Job Table (J)

This table keeps all the information about the print job. The PrintJob information is passed to the Server when the client does a printjob command. This job information is sufficient for successful processing and printing of the job without requiring further information from the client.

The information stored here is:

- JobId (unique job identifier)
- Job Parameters

Field	Description
- job_title	Job title
- job_originator	Job originator
- job_comment	Job comment
- job_start_message	Job start message
- job_end_message	Job end message
- job_start_sheet	Job start sheet number

- Job result handling

Field	Description
- delivery_name	name of the intended recipient
- delivery_address	address of the intended recipient
- delivery_method	method for printer output delivery
- delivery_information	Further necessary information for the chosen delivery method

- Job event handling

Field	Description
- events	list of possible events
- notification_name	user to which notification is sent
- notification_address	address to which notification is sent
- notification_method	method by which notification is sent
- job_abort_counter_name	name whose values trigger an abort
- job_abort_counter_thresh	value for above name which triggers abort
- job_warning_counter_name	name whose values trigger a warning
- job_warning_counter_thresh	value for above name which triggers warning

- Job Scheduling

Field	Description
- job_start_wait	wait for operator action before commencing
- job_end_wait	wait for operator action before concluding
- job_hold	put job on hold and print after hold release
- job_password	password to be entered for job commencement
- job_print_after	print after a particular time
- job_priority	specifies priority indication for scheduling
- job_deadline_time	specifies time by which user desires the output of the print job
- job_retention_time	time by which job is to be discarded, whether or not the job is printed

- Printer Identification

Field	Description
- printer_name	name of the printer
- printer_location	location of the printer
- printer_model	specifies the make and model of the printer
- printer_type	specifies type of the printer, say laser
- printer_security	specifies required security classification
- printer_speed	specifies speed in pages per minute (ppm)
- printer_resolution	specifies resolution in dots per inch (dpi)

- Production instructions

Field	Description
- font_substitution	list of pair of fonts where each pair has document font and font to be substituted
- media_substitution	list of pair of media where each pair has document media and medium to be substituted
- ink_substitution	list of pair of inks where each pair has document ink and ink to be substituted
- output_specification	specifies the output receptacle

- plex	specifies plex
- number_up	specifies number-up
- orientation	specifies orientation
- layup_specification	specifies lay-up
- copy_count	specifies number of copies

- document description

Field	Description
- document_title	title of the document
- document_creator	author (creator) of the document
- document_comment	specific comments for the document
- transfer_method	specifies transfer method
- document_reference	specifies transfer method specific reference

- Security access and control

Field	Description
- user_name	name of user granting access to "print"
- authenticator	information to enable verification of the identity of the user accessing "print"
- account	specifies an account to be charged

- Job status

Field	Description
- job_status	specifies the status of the job

Various components of the Print Server use different parts of the job data base. For example, the job scheduler is interested only in the job scheduling instructions.

8.1.2 Device Table (D)

This table stores information about each printer and its supervisors. It represents the current capabilities (configured by the local administration , of course) of all the printers which are available. It is this relation which is scanned and used to validate a print request. The following information is stored here:

Field	Description
- DeviceName	Name by which the device is referred to
- DeviceType	Type of device (e.g. LN03R Script Writer)
- DeviceGeographicLoc	Tells about location of the device. Used when a logical printer name is based on geographic location.
- DeviceAddress	Address of the device (say, /dev/tty03 for

	serial printer)
- SupervisorName	The sole Supervisor for this Printer
- ServerName	The Server which serves this Supervisor
- HostName	The machine on which this printer and its supervisor resides
- Info	This field consists of a set of key-value pairs of printer information needed by the Supervisor to control the printer
- Characteristics	This is a key-value pair for all the current characteristics for this printer. This is a subset of all the possible characteristics for this device type as given in the Type Table (T).

8.1.3 Server Table (S)

The server is another process which sits on the network waiting and listening to the printing requests. A supervisor is always contacted by the associated server for all the printing. The server gets the call (mostly) from a PdTk routine over the wire specifying a job to be done. If a job is to be printed or modified the server can receive the job in a number of ways. This job is stored on the server itself. For each server thus the table contains a spool directory where all the data files are kept.

Field	**Description**
- ServerName	name of the Server as the Name Server knows
- ServerHost	name of the host on which the Server resides
- SpoolDir	the spool directory where all the job information for this Server is kept

8.1.4 Logical Printer Table

This contains information about logical printers. A Logical Printer creates some sort of alias equivalence for the printer. But it is not just an alias because associated with this logical printer are some logical printer characteristics and job defaults. It is this property which carries much more weightage than aliases. For example if there is a single printer and it is used for both portrait and landscape regularly, one can define two logical names, Landscape and Portrait, with job defaults of landscape and portrait orientation, respectively. Now the user can refer to the printer names as Landscape or Portrait and the correct orientation comes on the output without explicit user specification.

Field	**Description**
- LogicalPrinterName	Logical name by which the user refers the printer
- LogicalPrinterChars	The default printer characteristics associated with

	the printer when the printer is referred to by this name. This is a subset of all the possible characteristics from the Type table T
- JobDefaults	The job defaults associated with the job when the printer is referred to by this name.
- ACLs	Access Control Lists with permissible names or system defaults.

8.1.5 Association Table(A)

There will be at least one logical printer name for each device name since the only way to access a device (printer) is through the logical printer name.

Field	**Description**
- LogicalPrinterName	The name by which the user refers to the printer either explicitly or from the profile or the environment. Each logical printer name corresponds to at least one device name.
- DeviceName	This is the name of the printer as the supervisor knows. A device name may be associated with multiple logical names but each of these logical names will have different default characteristics.

8.1.6 Type Table (T)

Here we keep all the information related to a particular printer. This is device specific info and is used by the Printer Supervisor and isn't accessed by any other component of the Print System. The relation will contain the following information:

Field	**Description**
- TypeName	primary name of the type of printer
- DataSyntaxes	different data syntaxes supported by this type
- CharacterSets	different character sets supported by this type
- Fonts	different fonts supported by this type
- FontSyntaxes	different syntaxes in which fonts can be accepted by this printer type
- Media	different media supported by this type
- Plex	different plex supported by this type

The Type table contains all the **possible** information for the printer of the given type. This can be thought of as a table for administration purposes and one which is used for filling out the required information in the **current** characteristics field of the device table (D).

8.1.7 Media Table(M)

The Media table contains the definitions for media. These definitions will be used later when a user wants to print to a printer not by name but on the basis of the properties of a resource, say media.

	Field	Description
-	media_name	name of the media
-	media_type	type of the media
-	media_size	size of the media
-	media_weight	weight of media
-	media_grain	grain of media (eg, coarse, smooth, ...)
-	media_tooth	tooth of media (eg, inward, straight, ...)

8.1.8 Fonts Table(F)

The Fonts table contains the definitions for fonts. These definitions will be used later when a user wants to print to a printer not by name but on the basis of the properties of a resource, say fonts.

	Field	Description
-	font_name	name of the font
-	font_type_family	specifies family like courier
-	font_spacing	specifies whether spacing is proportional or fixed
-	font_size	size of the font in points
-	font_scale_factor	the vertical to horizontal proportion
-	font_type_style	specifies the style, say normal, italic
-	font_char_weight	specifies weight for each character, say bold
-	font_char_proportion	specifies proportion for each character, say normal, expanded, condensed

8.1.9 Forms Table(FO)

The Forms table contains the definitions for forms. These definitions will be used later when a user wants to print to a printer not by name but on the basis of the properties of a resource, say forms.

	Field	Description
-	form_name	name of form
-	form_type_name	type of form (electronic or paper)
-	form_relation	relation for the form

8.1.10 Character Set Table(C)

The Character Set table contains the definitions for character sets. These definitions will be used later when a user wants to print to a printer not by name but on the basis of the properties of a resource, say character sets.

	Field	Description
-	char_set_name	name of the character set
-	char_subset_name	name of the character subset
-	char_set_rotation	rotation for the character set

8.2 Functional Interface

8.2.1 Find Operation

Syntax:

```
struct List *
Find(relation_id, capabilities)
int relation_id;
char **capabilities;
```

Description: The **Find** operation takes two arguments. The first argument, *relation_id*, identifies the relation to search. This relation_id is also used as entityType at some places. The second argument, *capabilities*, is an array of strings. Each string is of the form *name=value* pair which are the capabilities we are looking for.

Algorithm: The first argument tells us which relation is to be looked at. A complete search for that relation is done to find an entry which fulfills clients capabilities requirements, and a proper entity for that n-tuple is returned. The *struct List* value is explained below.

Data Structures: The second argument *capabilities* is an array of strings, where each string is of the form *name=value*, for e.g., *printerName=Hermes*. As the name suggests, the primary purpose of the Entity is to be able to do a direct look up in the relations database whenever we want to retrieve information, say by the **RetrieveKey** command. List structure also contains links to other entities which had fulfilled the requirements as passed by the second argument *capabilities*.

Usage is:

```
struct List *
Find(relation_id, capabilities)
int relation_id;
char **capabilities;
```

8.2.2 FindNext Operation

Syntax:

```
struct List *
FindNext(relation_id)
int relation_id;
```

Description: The **FindNext** operation is used to find the next tuple which matches the clients capabilities. These are the same capabilities which were passed to the previous **Find** operation or were used by the previous **FindNext** operation.

Algorithm: The first argument is the relation_id (or entityType).The List structure holds enough information to search for the capabilities which were used by the previous operation.

Data Structures: The main structure used is *struct List*. The structure is a dynamic array of attribute name and attribute value pair. The attribute value can be a string, integer, list or another recursive entity structure depending upon the attribute type.

Usage is:

```
struct List *
FindNext(relation_id);
int relation_id;
```

8.2.3 Add - Delete Operations

Syntax:

```
int
AddEntry(relation_id, capabilities)
int relation_id;
char **capabilities;

int
DeleteEntry(relation_id, entry_ptr);
int relation_id;
struct entry *entry_ptr;
```

Description: The AddEntry operation takes two arguments. The first argument is the name of the relation in which to store the info. The second argument is the set of capabilities as *name=value* pairs for that relation. It is used to add new entries to the database. This function is normally used when someone wants to add a new entry dynamically to the database. Most of the core structures are built from parsing the tables generated by SMS which use other lower level routines rather than this function.

DeleteEntry removes an entry from the database internal structure. If the entry was there only in the core when it was read from the configuration file (while being parsed) it is gone. The only exceptions to these are the JOB and SPVR entries for which the disk entries remain but the core mapping entries are deleted.

Algorithm: The 'AddEntry' operation is used to add new entries to the relation. Based on the first argument we find the relation (or database) in which to store all the information which is passed by the next argument. DeleteEntry gets the entry to be removed and it manipulates the links to eliminate that entry.

Data Structures: The first arg passed is an integer to identify the desired relation and the second arg is an array of strings, where each string is of the form *name=value*.

Usage is:

```
int
AddEntry(relation_id, capabilities)
int relation_id;
char **capabilities;

int
DeleteEntry(relation_id, entry_ptr);
int relation_id;
struct entry *entry_ptr;
```

8.2.4 Insert Key Operation

Syntax:

```
int
InsertKey(entry_ptr, relation_id, name, value, type)
struct entry *entry_ptr;
short relation_id;
char *name, *value;
int type;
```

Description: The InsertKey operation takes five arguments. The first argument is the ptr to structure which is dynamically allocated and re-allocated according to what the user wants. The name and value are character strings for insertion into the struct. The fifth argument specifies the type of the attribute, for e.g., integer, string, etc. **InsertKey** is used by the user repetitively to complete an entry which is then added to the database. Hence if an entry has six attributes, **InsertKey** will be called six times with the same first argument each time so as to fill the entry.

Algorithm: The name of the attribute is checked to see if it is there in the keyword table. If it is then the user supplied type is checked with the table type for semantic checking. If it isn't in the table it is stored in the keyword table. It is also stored in the entry passed by the user.

Usage is:

```
int
InsertKey(entry_ptr, relation_id, name, value, type)
struct entry *entry_ptr;
short relation_id;
char *name, *value;
int type;
```

8.2.5 Set Key Operation

Syntax:

```
int
SetKey(entry_ptr, relation_id, name, value, type)
struct entry *entry_ptr;
short relation_id;
char *name, *value;
int type;
```

Description: The **SetKey** function is used to set the attribute value for a given attribute name. This assumes that a **Find** or **FindNext** call has been made to find the appropriate structure.

Algorithm: The **SetKey** operation is used after the **Find** or **FindNext** operation. The **Find** or **FindNext** operation will return a List structure which will be used by SetKey to set the new value of attribute passed by the user. Some changes may not be possible, for example, to change the job parameters for job being printed.

Usage is:

```
int
SetKey(entry_ptr, relation_id, name, value, type)
struct entry *entry_ptr;
short relation_id;
char *name, *value;
int type;
```

8.2.6 Read (Retrieve) Operation

Syntax:

```
char *
RetrieveKey(entry_ptr, attribute_name, valueType)
struct entry *entry_ptr;
```

```
char *attribute_name;
short *valueType;
```

Description: The **Find** or **FindNext** operation returns an struct List. This List structure was found on the basis of all the capabilities passed to the **Find** or **FindNext** operation. The second argument passed is the attribute name and the value stored in the data base is returned. For data base sanity checking this function also returns the type of value (integer, etc.) in the result variable, valueType.

Data Structures: This routine uses the data structure 'List' returned from a **Find** or **FindNext** call. It searches that to find the value of the attribute passed to this routine.

Usage is:

```
char *
RetrieveKey(entry_ptr, attribute_name, valueType)
struct entry *entry_ptr;
char *attribute_name;
short *valueType;
```

8.2.7 Initialize Operation

Syntax:

```
int
DbInit(relation_id, relation_name, conf_file)
short relation_id;
char *relation_name, *conf_file;
```

Description: This function is used when the administrator wants to (re)initialize the database on the host machine which contains the server and/or the supervisor. Since there are different relations (entity types) using the same function we also pass the entity type and entity name to store some additional information. The argument is the configuration file which was produced by the SMS and shipped to all the machines over the network. There is some time delay over which the actual data as stored by the SMS changes.

Usage:

```
int
DbInit(relation_id, relation_name, conf_file)
short relation_id;
char *relation_name, *conf_file;
```

8.2.8 Locking Operations

Syntax:

```
FILE *
DbLock(relation_id, value, lockType)
short relation_id;
char *value;
int lockType;

int
DbUnlock(file_ptr)
FILE *file_ptr;
```

Description: There are many entities in the data base which may try to access some common information at the same type. One may be trying to write new information in the database whereas other may be reading. To make sure that there is data consistency amongst all the entities accessing the same database, two locking operations are provided. The **DbLock** operation makes sure that the entity named is locked for access according to lockType, which is shared lock for READs and exclusive locks for WRITEs. **DbUnlock** is used to remove the lock for the entity locked by a corresponding DbLock.

Usage:

```
FILE *
DbLock(relation_id, value, lockType)
short relation_id;
char *value;
int lockType;

int
DbUnlock(file_ptr)
FILE *file_ptr;
```

8.2.9 Misc Operations

Syntax:

```
int
GetNameValueType(nameValue, name, value, type)
char *nameValue;
char **name, **value;
short *type;
```

Description: The **GetNameValueType** function takes one value parameter and three result parameters. It tries to parse the *nameValue* argument passed and fills in the remining result parameters from it. The type of value, for e.g., integer or string, is also returned.

Usage:

```
int
GetNameValueType(nameValue, name, value, type)
char *nameValue;
char **name, **value;
short *type;
```

Chapter 9

Palladium Server Job Scheduler

This is the design document for the Job Scheduler. This chapter gives a brief overview of the scheduler. The Job Scheduler is a component of the Palladium Print Server. It has complete access to the jobs database and schedules the jobs to run on the printers managed and represented by the Palladium Server and Palladium Supervisor.

For the Athena implementation, the scheduler will schedule the printing of Server jobs on a First Come First Serve basis. However we should be able to replace this FCFS algorithm with any scheduling algorithm. For this reason the scheduler should contain the code for many different scheduling algorithms and one should be able to compile with the appropriate scheduling algorithm according to the environment. The Scheduler is independent of any other task running on the system.

9.1 Scheduling functions

The scheduling strategy is chosen by the administrator at the scheduler compile time. The following are the basic functions which are used by the system to schedule jobs:

```
int
PdScheduleNewJob(jobId)
char *jobId;

int
PdScheduleFreeSpvr(spvrName)
char *spvrName;

int
PdKickScheduler(spvrName)
char *spvrName;
```

```
void
PdPauseScheduler()

void
PdResumeScheduler()

void
PdRecvJob()
```

Function Overview:

The basic **scheduler** task is to schedule the jobs on the basis of the scheduling algorithm chosen by the administrator at the **make** time. This function can be as simple as comparing the time of the jobs (to do a FCFS scheduling) or it may be complex enough to incorporate the time of job submission, priority of the job, deadline of the job, etc. All the jobs are actually scheduled by the first two functions. Both these functions are called by the server.

The first function *PdScheduleNewJob* is called when a job is submitted by the user. Actually it is called by the server when it receives notification from the client program that the transfer of the files to the server is over. If there is no printer available then the job is left in the jobs data base itself. Remember that this will be picked up by the scheduler later when it get calls to schedule a job on a free supervisor now.

The other function **PdScheduleFreeSpvr** is called so as to schedule a free supervisor with a job from the jobs data base. When a job is completed on a printer, the supervisor makes an RPC call to the Palladium Server notifying it of job completion on the printer "PrinterName" and the Server can then call the scheduler function **PdScheduleFreePrinter** passing it the name of the supervisor.

PdKickScheduler is used for synchronizing information about a particular supervisor and it's state and logical printers between the Server and the Scheduler. Even though the Scheduler is a part of the Server, because of the lack of shared memory in 4.3BSD we have to use disk files for passing information between server and scheduler. When the Supervisor (re)registers with the server for a set of new or updated logical printer list, the scheduler also has to be informed. The server calls *PdKickScheduler* function as a hint to Scheduler to go re-read the file for the passed supervisor and update it's internal information.

PdPauseScheduler is used by the server to inform the scheduler that it has been paused. This is needed because even though the server won't call the scheduler using *PdScheduleNewJob* when it is paused, the scheduler is/maybe called when one of the supervisors under this server finish their currentjob, via *PdScheduleFreeSpvr*. In this case the scheduler should know that its' server is paused and not scheduler any more jobs. **PdResumeScheduler** is used by the server to inform the scheduler that it has been resumed now and that the scheduler can restart scheduling and sending jobs for printing. There is a skulker which runs periodically and removes any files which were left for retention.

PdRecvJob is actually used by the Palladium pdpr(1pd) client to pass a file from client to the server. It uses socets to pass the file over the wire rather than using the file system as a

file transport mechanism. In near future, this will be replaced by a general purpose file transfer paradigm.

Algorithms:

The main part of the schedule is done by the administration chosen scheduling algorithm. Some of the different functions possible are: First Come First Serve, Assigning priorities to jobs, assigning deadline time to jobs, computing priority with aging in account, etc.

For scheduling a **new job**, it works as follows:

```
  - Find the logical printer for this jobId
  - Find all the supervisors corresponding to this logical printer
    This is done from the SUPERVISOR TABLE as that has the
name of logical printers registered rather than the
ASSOCIATION TABLE which is static.
  - Find the first SPVR (for present) which is free.
  If yes, make a call to the supervisor
If all supervisors are busy (or not free) quit as PdScheduleFreeSpvr()
will pick up the job later.
```

For scheduling a job for a **free supervisor**, the following algorithm is used:

```
 - From the supervisor table, get all the Printers registered for this
   supervisor
 - Find the queues corresponding to these logical printers, and
   remove duplicates (at present each queue in QUEUE table is
   equivalent to one logical printer)
 - Get the next printable job from each queue, at present we are
   using the FCFS algorithm
 - From all the first printable jobs for each queue, get the
   first one which will go to the supervisor
 - See if the job characteristics of the printer match with
   the supervisor characteristics (at present we assume they do)
   If yes,
       Stuff job parameters in structure and make a RPC call
       to the supervisor
   If no,
       Get next job from this second prioritized queue and do
       the same.
```

Data Structures:

This task does not export any major data structures. It uses all the information from the jobs data base.

Subroutine Interface:

The **PdScheduleNewJob** function uses the JobId of the new job it just received from the server. This JobId uniquely identifies the job and helps finding the entry pointer which serves as an index into the jobs data base for the corresponding job. The **PdScheduleFreeSpvr** function uses the supervisor name which just got free to see if there is a job in the data base which can be sent to a free printer managed by this supervisor. The server knows the supervisor from which the call came and this happens when the supervisor informs the Server about the completion of a job. There are other routines for fetching this job information from the job database.

9.2 Miscellaneous functions

The following functions will be implemented to work on the jobs data base. These do a miscellaneous things of comparing for jobs and printers on the basis of printer and jobs, respectively. These will not be there in the present Athena implementation.

```
i)   compare_jobs(J1,J2,P)

     This function selects one of the two jobs (J1 or J2)  to be scheduled
     on the PRINTER P. In this case the user requested both the jobs to be
     printed on PRINTER P and the scheduler decides which one to send first on
     the basis of scheduling algorithm used.

ii)  compare_printers(P1,P2,J)

     When a user chooses a printer on basis of capabilities a number of
     printers may be capable of printing the job.  The above function
     chooses one of the two printers P1 or P2 to send the job J to.

iii) list_available_jobs(list)

     returns all the available jobs in the parameter 'list'.

iv) list_available_printers(list)

     returns all the available printers in the parameter 'list'.

v)  select_job(P,list_of_jobs)

    returns a job to be printed on the printer P from the list of jobs.

vi) select_printer(J,list_of_printers)

    returns a printer from the list of printers which is capable of
    handling job J.
```

vii) match_jobs(P,list_of_jobs)

returns in the second argument the list of jobs which can be printed on the printer P.

viii)match_printers(J,list_of_printers)

returns in the second argument the list of printers on which the job J can be printed.

Chapter 10

Palladium Printer Supervisor

This chapter will describe the interface and design of the Palladium Printer Supervisor. The Printer Supervisor is a deamon that accepts a job and prints it on a printer. For the initial implementation we are assuming one job will be processed at a time. In addition, we are assuming that one and only one type of printer will be driven by any instance of a printer supervisor. Different supervisors may support different types of printers. For the purposes of the initial implementation, the printers described in the current printcap file and driven by the current *lpd* may be considered to be of the same printer type.

The initial design of the Palladium Printer Supervisor is based on the existing BSD *lpd*. The code may be completely rewritten (at the implementors discretion) but the design of this initial supervisor will continue to use the printcap-style information to control the current set of Unix-supported printers using the existing *lpd* child code.

The Palladium Printer Supervisor exports the standard Palladium Server Interface. It can be used as "dedicated" to a server, as in a standard printer configuration, or in "free agent" mode, so that a the Palladium system can be used to drive personal printers. It will export same interface to print clients and print servers that the Palladium Print Server exports. This interface includes the following operations:

User Functions

1. PdPrint - process requests for printing
2. PdModify - process requests to modify pending jobs
3. PdReadJobStatus - Read Job Status
4. PdCancelJob - Cancel Job
5. PdListJobs - List Jobs
6. PdReadObjectAttributes - Read Object Attributes

Administrative Functions

1. PdResubmitJobs - Resubmits jobs to a different entity.
2. PdClean - Removes all of the jobs from a specified entity.
3. PdPromoteJob - Move the specified job to the top of the queue for the specified entity.
4. PdInitialize - Initialize an entity, ie., get all the information needed in order for the entity to perform correctly.
5. PdEnable - Enable the entity.
6. PdDisable - Disable the entity.
7. PdStop - Abruptly stop the entity.
8. PdShutdown - Cleanly shutdown the entity.
9. PdModifyAttributes - Modify the attributes of an entity.
10. PdRegister- Register an entity.
11. PdUnregister- Unregister an entity.
12. PdRegisterUser - Register a user as being in charge of the entity.
13. PdUnregisterUser - Unregister a user as being in charge of the entity.
14. PdReadyforJob - Inform an entity that a printer is ready for a job.
15. PdJobComplete - Inform an entity the a printer has completed a job.
16. PdPause - Entity stops scheduling jobs (server) or it stops printing (printer).
17. PdResume - Entity resumes scheduling (server) or resumes printing (printer).
18. PdAdvertise - Causes an entity to advertises itself.
19. PdReportEvent - An entity reports an event.
20. PdFileXfer - Used to transfer files for printing.

This interface is described for the Print Server in the PdLib and Server Design chapters. It's meaning in terms of a Print Supervisor is different, and is described in a later section.

The same interface was chosen for both the Print Server and Print Supervisor to simplify development of the multiple components of the Palladium system. The current Pdlib interface is rich enough to support access to a wide range of value-added printing functionality. Use of the same interface reduces overhead of designing and implementing multiple interfaces, reduces the potential number of design and implementation bugs and forces a broad and thorough exercise of the design of the single interface.

While the same interface is presented by both the Palladium Print Server and the Palladium Print Supervisor, they each provide different levels and types of value-added functionality. Thus,

calls in Pdlib have different effects and meanings depending on the type of entity to which the call is directed. These differences are reflected, not through different interfaces for the server and the supervisor, but through somewhat different functional designs. Where the Palladium Print Server supports asynchronous access and request handling for clients, the initial Palladium Printer Supervisor will support only one client request and processing at a time. The initial Palladium Printer Supervisor will not support the queuing of requests. Therefore, it will not be able to support multiple job printing on multiple printers without the support of a Print Server. The design of the initial Palladium Printer Supervisor should not be construed as constraining the design and implementation of future supervisors to a one-job-one-printer model.

10.1 Supervisor Status

The overall state of a supervisor comprises following states within it: the state of the input (accepting print jobs),the state of the output (printing accepted jobs), if a job is currently active and printing, if the supervisor needs attention and if the supervisor is shutdown.

The *input* status is either **enabled** and accepting new print jobs, or **disabled** and not accepting new print jobs. The *output* status is either **paused** and not printing (like pressing the off-line button on a printer; the output processes are suspended), or **resumed** and printing normally. The *printing* status is either **active**, implying that a job has been accepted and not yet completed, or **idle**, meaning that there is no job printing. The *attention* status is set if an operator intervention is needed to print the active job (i.e. out of paper). The *shutdown* status indicates that the supervisor will stop after the current job has finished.

The input and output states are actually stop-action flags; ie, the **paused** state is what actually stops printing of output, and is not simply a reflection of the fact that nothing is printing. It means that nothing *can* print. **resumed** simply means not paused; it does not imply that something is actually printing.

The status of a supervisor can be obtained with a call to the **PdReadObjectAttributes** function. What is returned is a bitmask which reflects the different states. The changing of these values is done through the PdLib Interface. **PdPause** and **PdResume** change the output state; **PdDisable** and **PdEnable** change the input state; **PdPrint** and **PdCancel** or **PdClean** change the printing state; **PdShutdown** changes the shutdown state. Figure 10.1 shows how these commands change states using the bits to represent each of the input, output, and printing states.

10.2 Supervisor User Functions

The PdLib chapter describes each of the PdLib functions in the Palladium Server Interface in terms of a Print Server. The following is the specification of the meanings of the user PdLib calls in terms of a Print Supervisor, and a description of how they differ from the calls to a Print Server. For each, the arguments are the same as described in the PdLib chapter, but their actual effects and operation (and even appropriateness) may be radically different.

operation	starting state							
	00000	00001	00010	00011	00100	00101	00110	00111
PdPrint	00100	00101	E	E	E	E	E	E
PdEnable	00000	00001	00000	00001	00100	00101	00100	00101
PdDisable	00010	00011	00010	00011	00110	00111	00110	00111
PdPause	00001	00001	00011	00011	00101	00101	00111	00111
PdResume	00000	00000	00010	00010	00100	00100	00110	00110
Done	00000	00001	00010	00011	00000	00101	00110	00111

The five bits **abcde** represent:
a: attention status (1 = needs attention)
b: shutdown status (1 = shutdown pending)
c: printing status (1 = active, 0 = idle)
d: input status (1 = disabled, 0 = enabled)
e: output status (1 = paused, 0 = resumed)
E implies that the call will result in an error.

Figure 10.1: Supervisor State Transition Table

10.2.1 PdPrint Function

The PdPrint function handles a request to print a document. If a specific printer on the Supervisor is not requested, the Supervisor will select one based on availability and capability.

For the first implementation, only one job will be handled, so the PdPrint call may be rejected by the Supervisor if it is already busy printing. An appropriate error code will be returned in this case.

10.2.2 PdModify Function

The PdModify function handles a request to change the specifications of a print job that has been accepted. Only the system administrator or the job originator may modify a job. A job may be modified while it is waiting to be printed; once printing is started, the job cannot be modified.

Because the first implementation of the Supervisor will handle only one job at a time, this job will be active immediately; once a job is accepted, no others will be accepted until it is finished. This means that no jobs will be 'inactive' or waiting, thus **PdModify** will always return an error.

10.2.3 PdReadJobStatus Function

The PdReadJobStatus function handles requests to return the specified parameters for a print job. It returns information about the printer being printed on, the status of the job, etc. An error code

is returned if the specified job is not currently handled.

This function returns information about a single print job, so it will operate the same in the first-cut Supervisor implementation as it would for the full-blown implementation. The status will always be active, since the supervisor does not queue jobs.

10.2.4 PdCancelJob Function

The PdCancelJob function removes a job that has been previously accepted by the Supervisor. Only a system administrator or the user originating the job can cancel a job. If a job is currently printing, it can be flushed by stopping the printing at it's current point and ending the job. This can be done if the print job has soft failed, and the administrator wishes to cancel the rest of the job before putting the printer back on line.

Since the first-cut Supervisor will only hold the currently active job, all PdCancelJob requests will be flushes of the currently active job.

10.2.5 PdListJobs Function

The PdListJobs function returns a summary list of all jobs on the Supervisor. Specific jobs or classes of jobs can be selected by passing acceptance criteria to filter out the jobs desired.

Because the first-cut implementation of the Supervisor will handle only the currently active job, this function will return at the most a summary list of one job.

10.2.6 PdReadObjectAttributes Function

The PdReadObjectAttributes function handles requests to obtain the attributes of the Supervisor objects (**jobs**, **logical printers**, **physical printer** and **supervisor**). This function acts the same for both Supervisors and Servers, as well as for the first-cut Supervisor implementation.

10.3 Supervisor Administrative Functions

10.3.1 PdResubmitJobs Function

This function takes waiting jobs scheduled for an entity and requeues them elsewhere. Because a supervisor will only handle one job at a time and that job will be active, rescheduling will be done by the server (cancel the active job and submit the job to new printer). So, calling **PdResubmitJobs** on the supervisor will result in an error.

10.3.2 PdClean Function

This function will operate the same on a supervisor as for a server. Since there is only one job on a supervisor at a time, this would be the same as calling **PdCancelJob** for that job.

10.3.3 PdPromoteJob Function

Since Supervisors will hold only one job, and that job will be active, the **PdPromoteJob** function has no meaning on a Supervisor. Promoting an active job makes no sense, and the only addressable job on a supervisor is active; this function will always result in an error.

10.3.4 PdInitialize Function

The **PdInitialize** function will operate the same on a supervisor as on a server. It causes the supervisor to reread it's startup information from the nameserver and config files.

10.3.5 PdDisable Function

The **PdDisable** function causes a supervisor to cease accepting and asking for print jobs and sets the job input status to "disabled". It will finish any print jobs currently printing, and can be called at any time.

10.3.6 PdEnable Function

PdEnable causes the printer to resume asking for and accepting print jobs. It sets the job input status to "enabled". It does not affect any jobs currently printing and can be called at any time.

10.3.7 PdStop Function

PdStop kills the supervisor process. It interrupts any currently active print jobs or other actions and stops the supervisor. The supervisor's status after a **PdStop** is "down". Calling **PdStop** without having shut down the supervisor could potentially cause lost print jobs, inconsistent accounting, etc.

10.3.8 PdShutdown Function

PdShutdown is a graceful killing of a supervisor. It first disables the printer (as a **PdDisable** call would) and deregisters the supervisor from it's superiors. It then waits for any currently active jobs to finish printing and kills itself (**PdStop**).

Function	Objects		
	Job	Logical printer	Supervisor
PdReadObjectAttributes	Returns job attributes	Returns logical printer attributes	Returns Supervisor/Printer attributes
PdClean	Error	Error	Cancel active job
PdEnable	Error	Accept job for logical printer	Accept jobs
PdDisable	Error	Cease accepting jobs for logical printer	Cease accepting jobs
PdStop	Error	Error	Kill supervisor
PdShutdown	Error	Error	Kill supervisor gracefully
PdModifyAttributes	Error. See PdModify	Change logical printer attributes	Change supervisor/printer attributes
PdRegister	Error	Error	Error
PdUnregister	Error	Error	Error
PdRegisterUser	Error	Error	User is added to administrator list
PdUnregisterUser	Error	Error	User is removed from administrator list
PdPause	Pause active job. The Supervisor will still accept new jobs	Error	Stop printing on supervisor
PdResume	Resume paused job	Error	Resume printing of jobs

Figure 10.2: Supervisor Object Management Functions

10.3.9 PdModifyAttributes Function

PdModifyAttributes operates the same on a supervisor as it does on a server. The attributes of the supervisor objects (**jobs**,**logical printers**, **physical printer** and **supervisor**) are changed to reflect the parameters of the call.

10.3.10 PdRegister Function

This call registers a new resource with an entity. In the initial implementation of a supervisor, only one printer is registered per supervisor, and registration/unregistration of resources will not be supported.

10.3.11 PdUnregister Function

This call unregisters an subservient entity, eg unregisters one printer of a farm from the supervisor. The initial implementation of a supervisor will only support one printer, and the registration/unregistration of resources will not be supported.

10.3.12 PdRegisterUser Function

PdRegisterUser adds the specified user to the administration list of the supervisor.

10.3.13 PdUnregisterUser Function

This call removes a user from the administration list of the supervisor.

10.3.14 PdReadyForJob Function

This call is used to tell an entity when a subservient entity is ready for a print job. For supervisors of non-intelligent printers, this call is meaningless and will return an error.

10.3.15 PdJobComplete Function

This call is used to inform a superior entity that a print job has been completed. This is meaningless to supervisors of non-intelligent printers.

10.3.16 PdPause Function

PdPause causes a supervisor to stop printing any jobs. It will stop a job that is currently printing. The supervisor will still accept new jobs, if possible (eg, if paused while waiting for a job). The job

output status will be set to "paused".

10.3.17 PdResume Function

PdResume causes a supervisor to resume printing output to it's printer(s) from the point at which it was paused. The job output status will be set to **resumed**.

10.3.18 PdAdvertise Function

This function causes a supervisor to reregister with it's superior entity. It will reregister it's resources and status, updating them on its server.

10.3.19 PdReportEvent Function

This function informs the supervisor of events that occur, and is used as a generic signaling resource between entities.

Chapter 11

Conclusion

This specification reflects the design of print clients and print servers for a distributed environment. This design has been heavily influenced by ongoing work on Print Service Standards in ISO, ECMA and ANSI. We hope that this design can be a living document (within the realistic bounds of backwards compatibility and available resources), growing as new functionality is developed by Athena and changing to continue to track the progressing standards work.