Dienst: Implementation Reference Manual

Carl Lagoze[*] Erin Shaw[†] James R. Davis[‡] Dean B. Krafft[§]

May 5, 1995

Abstract

We describe the architecture and implementation of Dienst: a protocol and server that provides distributed document libraries over the World Wide Web. Dienst is based on a document model that incorporates unique document names, multiple document formats, and multiple document decompositions. Interoperability among Dienst servers provides the user with a single logical document collection, even though the actual collection is distributed across multiple servers. The Dienst protocol uses HTTP (the protocol of the World Wide Web) as a transport layer, making Dienst servers accessible from any WWW client. Dienst is currently used as the infrastructure for a distributed computer science technical report library by a number of U.S. universities. This document is intended as a guide for Dienst site administrators and implementors of other digital library systems. It describes the architecture of an individual server and network of Dienst servers and includes the server installation instructions. Appendices describe copyright issues and retrospective conversion of the Cornell technical report collection.

[*]Computer Science Department, Cornell University, Ithaca, NY 14853, lagoze@cs.cornell.edu
[†]Computer Science Department, Cornell University, Ithaca, NY 14853, shaw@cs.cornell.edu
[‡]Xerox Corporation, Design Research Institute, Cornell University, Ithaca, NY 14853, davis@dri.cornell.edu
[§]Computer Science Department, Cornell Univeristy, Ithaca, NY 14853, dean@cs.cornell.edu

Contents

1 Overview of Dienst

The rapid development of a National Information Infrastructure (NII) promises unprecedented access to global information resources. The current incarnation of the NII, the Internet, provides a preview of future accessibility to information. Every day, millions of people use the Internet to access data and documents that, just a few years ago, were only available at some distant library or by waiting for them to be sent via the mail.

Although the Internet provides access to an enormous amount of information, the current state-of-the-art falls far short of what is commonly viewed as a library service - that is, relatively easy navigation of and access to a set of documents that are part of a collection. The notion of a collection is important in that it implies that the set of documents was not selected haphazardly, but by some trusted intermediary. Current users of the Internet confront an information space where the quality of documents is far from reliable, facilities for locating documents are primitive, and access to a specific document frequently means wading through a Tower of Babel of architecture dependencies and file formats.

Dienst is a protocol and server that provides Internet access to a distributed, decentralized multi-format document collection. From the standpoint of a Dienst user, the collection consists of a unified space of uniquely identified documents, each of which may be available in a variety of formats. Using publicly available World Wide Web clients, users may search the collection, browse and read individual documents in any of their available formats, and download or print a document. The Dienst system also provides site administrators with tools for managing their document collections.

Dienst was developed by researchers at Xerox Corporation and Cornell University as part of the Computer Science Technical Report (CS-TR) project, an ARPA-sponsored effort to create an on-line digital library of technical reports from the nation's top university computer science departments. The architecture has three components: a protocol for searching, browsing, and viewing on-line documents; a server that implements this protocol and interoperates with other Dienst servers; and a set of ancillary utilities for managing a digital collection and maintaining a server. Dienst servers are currently running at a number of computer science departments in the United States, providing access to thousands of computer science technical reports. The technology that underlies Dienst is not specific to the computer science technical report domain and, thus, can be applied to many other document collections. We should note, however, that the current architecture does not address issues unique to copyrighted materials, such as limiting access to authorized individuals.

Dienst can be best understood by viewing it in the context of other generally available technologies for document transfer over the Internet. The traditional baseline technology for document transfer over the Internet is anonymous FTP. While the ubiquity of FTP makes it an attractive technology, it has a number of flaws that limit its usefulness as a medium for a digital library service. There are no standards for organization of FTP directories, file names give little or no information about the contents of files, file formats can only be intuited from suffixes in file names, rapid browsing of files is impossible, and utilities for searching among multiple sites have limited usefulness.

Gopher and the World Wide Web represent significant advancements beyond FTP. Both allow the user to view a unified information space, rather than one of separate addresses or sites. Both permit related information to be linked and automatically traversed by the user. Whereas the only meta information for anonymous FTP documents is a terse file name, Gopher allows more expressive document titles. WWW goes one step further by placing links to a document within the context of another hypertext document. Finally, both Gopher and WWW incorporate the notion of document types (e.g., text, graphics, etc.) and the ability to automatically route a document to an appropriate *helper application*.

The Dienst architecture represents an additional layer of functionality above that provided by the WWW. This layer provides a number of important abstractions - defined collections of searchable and viewable documents, unique document identifiers that provide access to multiple representations of documents, and document decompositions that provide access to individual pages or logical parts of documents. The Dienst protocol uses HTTP, the protocol used over the Web, as a transport layer. This allows the Dienst user interface to exploit all the features of the Web, such as accessibility from such popular browsers of the World Wide Web as Mosaic, Cello, and Netscape.

Each document in a Dienst collection is addressed by its unique identifier, the *docid*. One can think of this docid as analogous to the ISSN and ISBN of the print publishing world. All client requests to a Dienst server for a document are made using the docid as a key; translation from the docid to a server site and file system location is hidden from the user. Geographic distribution of the collection is handled by the fact that each Dienst server is able to map from docid to repository location. The client accesses each document as if it were resident on the server to which the request was made.

The Dienst protocol provides extensible facilities for resource discovery. Clearly, users of the digital library won't always know the docid of a particular document and must use some type of search engine to find the resource. The current Dienst implementation includes a simple search engine that indexes documents using a standard bibliographic format (RFC 1357) and allows clients to submit queries based on the bibliographic fields (e.g., author matches *Gries*). A search request is not limited to the server to which the client is connected. The client may choose to submit the search in parallel to multiple Dienst servers and view the combined results. Search results are presented as a hypertext document, allowing the user to access the distributed documents in a simple point and click manner. The protocol is extensible to more advanced search engines as they become available.

Dienst makes use of the HTTP protocol to provide explicit types for documents, eliminating the need to guess the format. By providing file format meta-information, Dienst allows the Web client to automatically use the correct viewer (HTML, Postscript, TIFF, GIF, etc.) to display the document regardless of format. The ability to handle multiple document formats is important for a number of reasons; in particular, retrospective conversion of a collection that is a mixture of digital and paper forms.

Dienst builds upon HTTP by incorporating the notion of document decompositions. The current implementation addresses physical partitions (pages), but the protocol also supports logical partitions (e.g., chapters, tables, etc.). For example, a user may choose to view a specific page of the document in a specific format, if it is available. One particularly useful facility in the current implementation allows a client to view reduced *thumbnail* images of document pages as a means of searching for significant visual features in the document (e.g. tables, figures, references). Clicking on a *page rectangle* within the thumbnail image allows the user to view the selected page in full.

Since Dienst was developed as a platform for digital library and information retrieval research, both protocol and server are designed to be flexible and extensible. The protocol is specifically designed to support interaction with systems such as software agents, text analyzers, and search engines, and some of these are currently being developed.

2 Organization of this document

The remainder of this document is organized as follows:

- Section 3 describes the document model on which the architecture is based.
- Section 4 describes the components of a Dienst server and the manner in which a network of Dienst servers interoperate.
- Section 5 describes notable features of the Dienst user interface.
- Sections 6, 7, 8, 9, and 10 are the installation instructions for the Dienst software.[1]
- Appendix A describes the protocol for requesting services from Dienst servers.
- Appendix B describes how Cornell and Stanford deal with copyright issues regarding on-line technical reports.
- Appendix C describes the procedures we followed to convert the existing Cornell technical report collection to digital form.

3 Dienst document model

A central component to the Dienst architecture is the concept of a *document model*. FTP, and systems based on it like WATERS[2], are bound by limitations of the file system abstraction. File names have restrictions that limit their usefulness for object identification. Most file systems have no provision for associating meta-data with files (e.g. a description of the object). Directories are the only method for grouping multiple files.

The Dienst document model provides three important abstractions; unique document names, multiple document formats, and document decompositions. Figure 1 illustrates the document model. Appendix A describes the use of this document model in the Dienst protocol.

3.1 Unique document names

Every document in a Dienst collection is identified by a location-independent, unique document identifier, the *docid*. The docid has two parts:

- a *publisher* that is unique within the total Dienst name space. Currently, Rebecca Lasher at Stanford[2] acts as the naming authority who verifies the uniqueness of publisher names.
- a *name* that is unique within the name space for the publisher. The format of this string is left to the publisher.

As described in section 4.2, mapping from a docid to an indexing and repository location is done via a central server registration table that is downloaded by each server. This mapping is transparent to the user and from his/her point of view the collection is a flat set of documents.

[1] These sections are a copy of the on-line installation instructions available at `http://cs-tr.cs.cornell.edu/dienst_install/installation.html`.

[2] `rlasher@forsythe.stanford.edu`

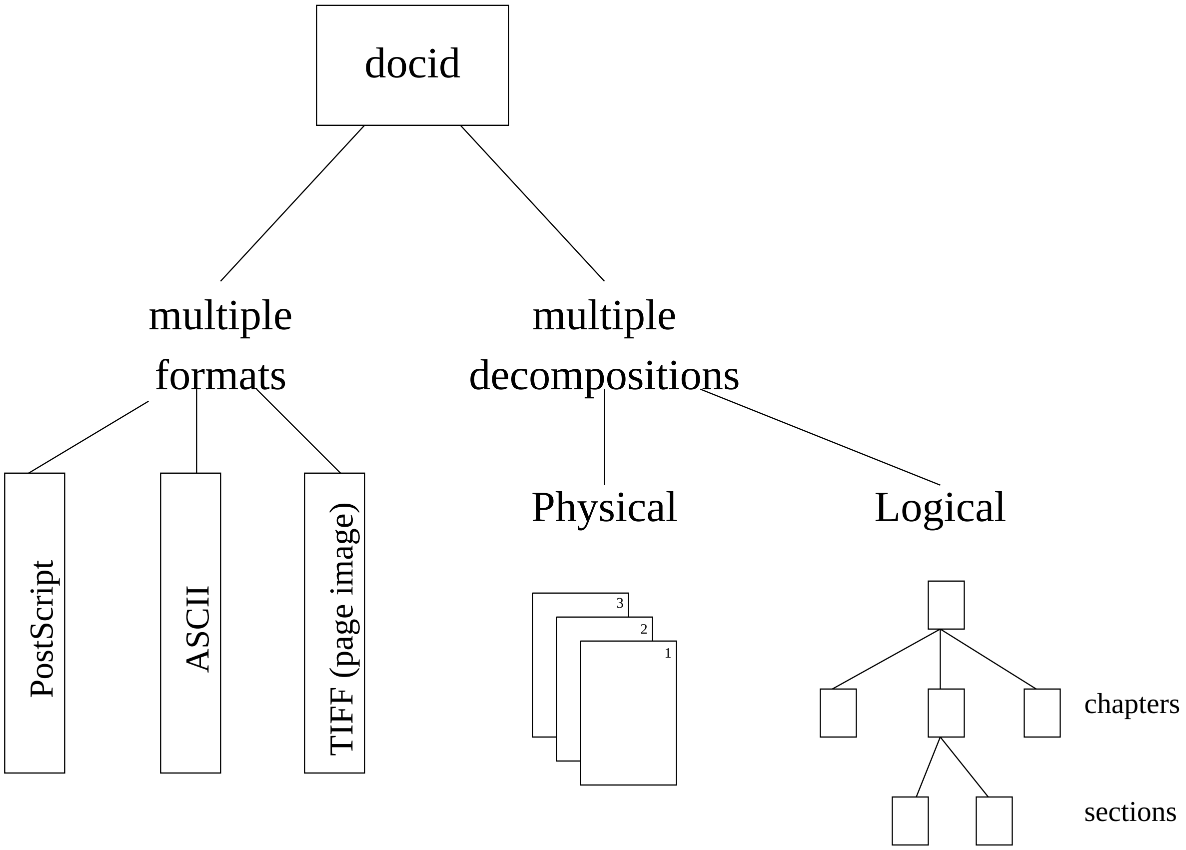


Figure 1: The Dienst document model incorporates notions of unique naming, multiple formats, and multiple decompositions.

3.2 Multiple document formats

The docid provides a means of grouping multiple fixations of a document. While from the user's point of view it would probably be ideal if multiple document formats did not exist, there are several good reasons for supporting this notion. First, documents have multiple origins. For example, most of the early documents in the Cornell technical report collection existed only in paper form. Retrospective conversion to digital form was done via scanning the paper documents to 600dpi TIFF images. On the other hand, new documents are uniformly added in PostScript format. Second, users access the collection from systems with varying display technologies. One user may be accessing the system through a non-bitmapped display that only view can ASCII text, another may have a high-resolution display on a machine with high bandwidth network connections. Finally, the different functions of a digital library are best served by multiple display formats. Medium resolution bitmapped GIF images are best for online viewing on a Web browser, high-resolution TIFF images are best for archival storage, the output of an OCR program acts as the input for logical structure discovery programs, and ASCII text provides the input for full-text search indexing.

The docid allows the user of Dienst to view the multiple formats in which a document is represented as sub-components of a single entity (even though these formats are actually represented by multiple files in multiple directories). The user interface presentation of this is described in section 5.

3.3 Document decompositions

Another important abstraction provided by Dienst is the notion of physical and logical decompositions of a document. Physical decomposition corresponds to pages of a document. Generally, a format that is addressable in separate pages is stored on disk as separate physical files. One example is scanned TIFF images of pages. However, this file-to-page mapping is not required, and the page breakdown may be done programmatically[3] (e.g. separating a PostScript file via the document structuring conventions).

Logical decomposition allows the user to address a document via structural elements; for example, chapters and sections. This structure can be explicit in the document via some type of markup (e.g., SGML) or implicit and discovered via heuristics that use whitespace and other layout characteristics as a guide[4].

4 Dienst server architecture

4.1 Components of a Dienst server

A Dienst server consists of four components; a document database, the server itself, the World Wide Web server, and the Dienst CGI stub. The relationship of these components is illustrated in figure 3.

document database - This is the set of documents provided by an individual Dienst server. Documents are stored as files within the Unix file system. Each document may be stored in multiple formats. At the minimum each document must be represented by an RFC 1357 bibliography file.

[3]The current Dienst implementation only supports physical decomposition for formats that are stored on disk as separate files for each page.

[4]This work has recently been completed and will described fully in a future document.

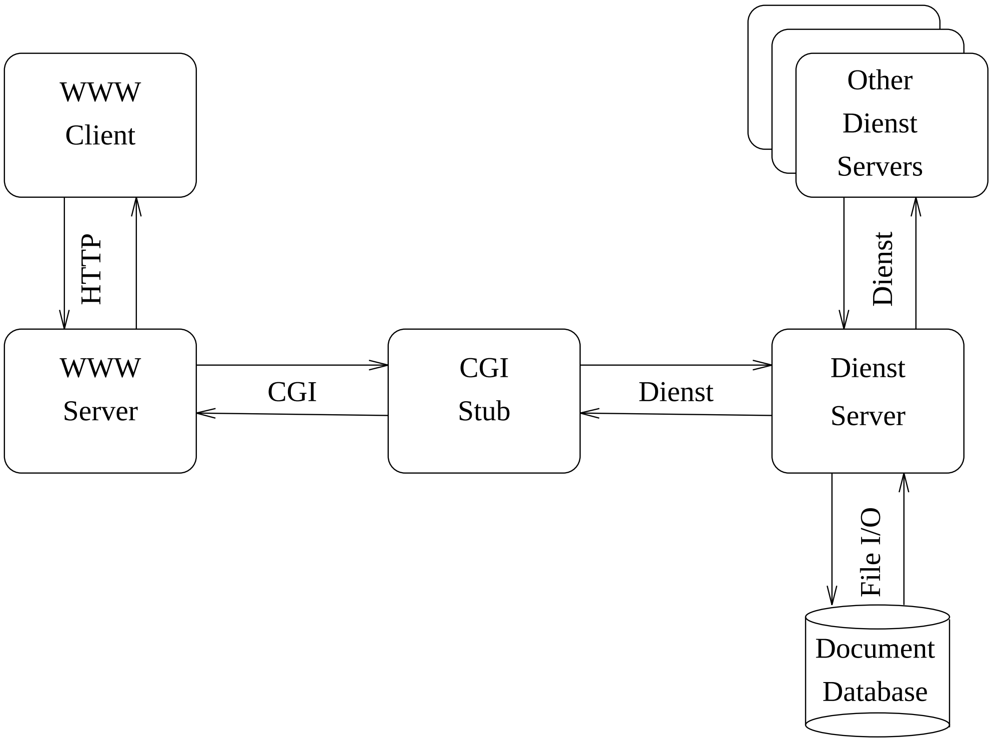


Figure 2: Each Dienst server is accessed through a Web server via the Common Gateway Interface (CGI). Each Dienst server interoperates with other Dienst servers and maps requests for documents to files in the document database.

The server software contains rules for mapping from a tuple consisting of a docid, format, and optional decomposition (e.g., page number) to the actual file in the document database. Rather than imposing a file organization, we allow each site to provide a customized version of this mapping at installation time.

Dienst server - A Dienst server is a standalone multi-threaded process written in *Perl*. The server listens for connections on a port defined in the server configuration file. Valid requests to the server are defined by the Dienst protocol. All responses from Dienst are formatted in the same fashion as HTTP responses. These responses are sent to the requesting client without interpretation by the Web server or CGI stub program. As part of fulfilling a request, an individual Dienst server may make requests to and interpret responses from other Dienst servers. Interoperation among servers is described in section 4.2.

Each Dienst server provides three services.

- *repository service* - provides access to documents in the document database. In the current Dienst implementation, a repository may contain documents from several publishers. The documents from a given publisher must all be stored in a single repository.
- *indexing service* - permits searching of documents in the document database. In the current Dienst implementation, a server can only index documents that are stored in the repository at that server. Indexes for the search engine are constructed offline with a utility that reads the RFC 1357 bib files for each document in the database. At startup, the server reads into memory the data from the index files. The search engine allows fielded boolean searches over the indexed bibliographic data. The server also has facilities for full-text search using either the Smart or WAIS search engine. This is described in section 8.1.
- *user interface service* - provides a user friendly, HTML front-end to the index and repository services.

World Wide Web server - The system is designed for access from Web clients through a Web server. Dienst may be accessed through any Web server that supports the Common Gateway Interface (CGI). The current version of Dienst has been tested with the two most popular servers, CERN and NCSA.

Dienst requests are packaged within the *path* portion of the HTTP request to the Web server. A Web server that serves as a front-end for Dienst must be configured to invoke the Dienst CGI stub program when the HTTP request contains a Dienst request. Setting up a Web server as a gateway to a Dienst server is described in section 6.

Dienst CGI stub - As described above, the CGI stub is invoked each time the gateway Web server receives a Dienst request. The stub is a small, simple executable that strips the Dienst request from the HTTP request, packages a few important environment variables that contain information about the request (e.g. the remote host identity, the MIME types the client is willing to accept, etc.), and sends them via socket I/O to the Dienst server.

4.2 Interaction among Dienst servers

A user of a Dienst server perceives a single logical collection, even though the collection is distributed over multiple servers. This is accomplished by interaction between a set of Dienst servers at three functional levels.

server registration - As described earlier, each docid that identifies a document in a Dienst collection consists of a publisher identifier and a name. Each server in a set of interoperating Dienst servers indexes and acts as a repository for documents from one or more publishers. In the current version, all the documents for a publisher must be indexed and reside on the same server. Each server is able to locate the indexing and repository site for a specific publisher using tables that are maintained by a distinguished server, the meta server, and periodically downloaded by the individual server. This allows every server to provide the next two functions.

distributed searching - Each Dienst user interface server provides a front-end for searching the collection provided by all inter-operating Dienst servers. The user specifies in the search form which publishers should be searched. The respective user interface server then dispatches search requests, in parallel, to the Dienst index servers that correspond to the publisher selection(s). The user interface server then parses and combines the results from the requested servers to present a coordinated "hit list" to the user.

distributed document access - Each server, in a set of interoperating Dienst servers, is capable of responding to a request for any document in the distributed collection. The respective server uses the publisher in the docid of the requested document and the mapping tables downloaded from the meta server to route the document request to the server that acts as the repository for the publisher. This routing is transparent to the user.

5 Dienst user interface features

Each Dienst server allows the user to search, browse, and view the collection using any Web client. Several of the features of the user interface are described in this section. Not all Dienst servers provide all the features described in this section, nor are the features available for all documents.

- A number of features are optional (e.g. full-text searching) and may not be installed at the respective server.
- A number of features are dependent on the format(s) in which the respective document is available. For example, in-line page viewing is only available if the document is stored in GIF format.

Forms-based distributed searching - An HTML form allows a user to specify criteria for a search. These criteria are publisher name(s) (at least one must be selected), document name, and bibliographic keywords (title, author, abstract). The search form allows the user to specify *and* or *or* boolean operators between the bibliographic keyword fields. In addition, the user may user boolean operators and precedence symbols (parentheses) within bibliographic fields. An example search form is shown in figure 3.

Unified hypertext hit list - The results of a search (that may have been dispatched to several servers) are presented to the user as a combined list of hyperlinks sorted by publisher and docid. Each *hit* is link to the document summary page for the respective document. If any of the servers needed for a search (as specified in the publisher list in the search form) are not available, this information is printed at the top of the results form. An example hit list is shown in figure 4.

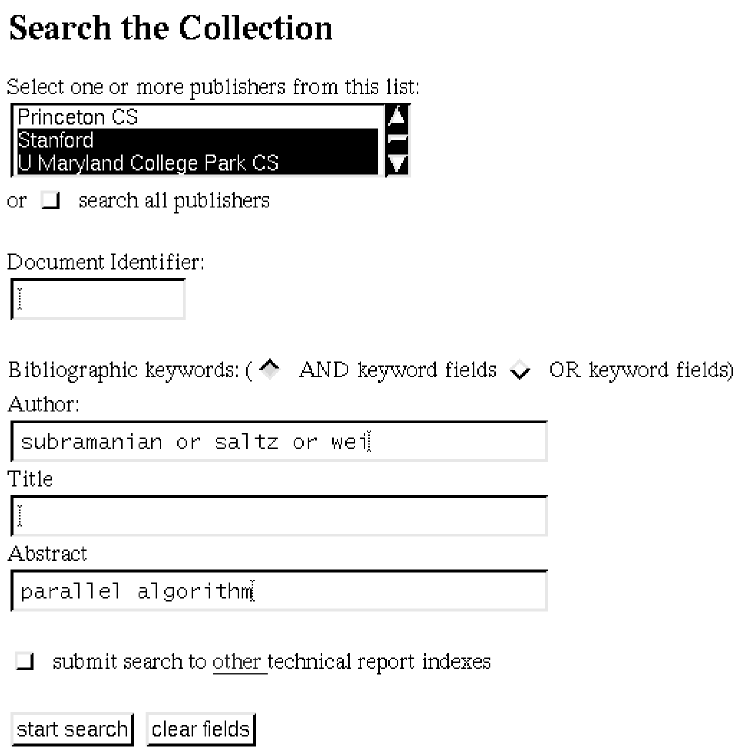


Figure 3: The Dienst search form allows users to enter search criteria for bibliographic fields. Boolean expressions may be used within and/or between fields.

Your search was for

publishers selected:

- Cornell
- Stanford
- U Maryland College Park CS

Bibliographic keywords ("and"ed together)

- author = subramanian or saltz or wei
- abstract = parallel algorithm

Search results

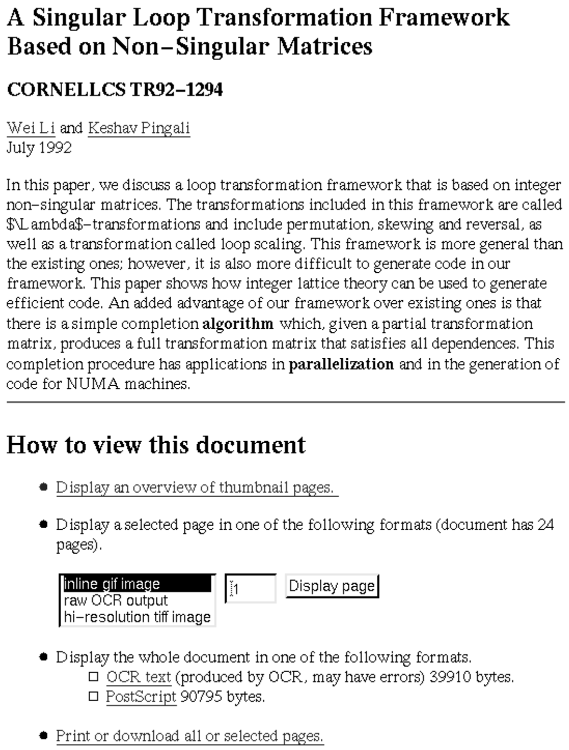
- CORNELLCS:TR92-1294 *A Singular Loop Transformation Framework Based on Non-Singular Matrices.* Wei Li and Keshav Pingali.
- CORNELLCS:TR94-1469 *COMPILING FOR NUMA PARALLEL MACHINES.* Wei Li.
- STAN:CS-TR-89-1275 *A new approach to stable matching problems.* Ashok Subramanian.
- STAN:CS-TR-89-1278 *The complexity of circuit value and network stability.* Ernst W. Mayr and Ashok Subramanian.
- UMCP-CSD:CS-TR-3425 *Parallel Monte Carlo Simulation of Three-Dimensional Flow over a Flat Plate .* Robert P. Nance, H. A. Hassan, Richard G. Wilmoth, Bongki Moon and Joel Saltz.
- UMCP-CSD:CS-TR-3447 *Interprocedural Compilation of Irregular Applications for Distributed Memory Machines .* Gagan Agrawal and Joel Saltz.

Figure 4: Dienst search results are displayed in a hypertext list, sorted by publisher and *docid*. Each *hit* is a link to the summary page for the respective document.

Document summary page - The document summary page serves two purposes. First, it displays the metadata associated with an individual document (e.g. author, title, publication date, abstract). If the search criteria for a document included abstract words, the appropriate abstract words are highlighted in the document summary page. Second, it presents to the user the formats in which the document is available. These are divided into four groups:

- page thumbnails for browsing.
- a structural overview of the document.
- formats for which physical page decompositions are available; the user may choose the appropriate page to display.
- formats for which decompositions are not available; selection of one of these formats will effect a download of the entire document in the respective format.
- a link to the printing and downloading form.

Only those formats available for the specific document will be shown on that document's summary page. An example document summary form is shown in figure 5.



A Singular Loop Transformation Framework Based on Non-Singular Matrices

CORNELLCS TR92-1294

Wei Li and Keshav Pingali
July 1992

In this paper, we discuss a loop transformation framework that is based on integer non-singular matrices. The transformations included in this framework are called $\Lambda$$-transformations and include permutation, skewing and reversal, as well as a transformation called loop scaling. This framework is more general than the existing ones; however, it is also more difficult to generate code in our framework. This paper shows how integer lattice theory can be used to generate efficient code. An added advantage of our framework over existing ones is that there is a simple completion **algorithm** which, given a partial transformation matrix, produces a full transformation matrix that satisfies all dependences. This completion procedure has applications in **parallelization** and in the generation of code for NUMA machines.

How to view this document

- Display an overview of thumbnail pages.
- Display a selected page in one of the following formats (document has 24 pages).

  inline gif image
  raw OCR output
  hi-resolution tiff image
  1
  Display page

- Display the whole document in one of the following formats.
  - OCR text (produced by OCR, may have errors) 39910 bytes.
  - PostScript 90795 bytes.
- Print or download all or selected pages.

Figure 5: The document summary page displays the metadata associated with the respective document and the formats in which the document is available. The user can use the links on this page to display the document in the chosen format.

Structural overview - A structural overview allows the user to view the major structural features of a document, such as chapters, sections, and the like. An example structural overview is shown in figure 6.

Tools for Monitoring and Controlling Distributed Applications.

Keith Marzullo and Mark D. Wood
January 1991

Figure 6: The structural overview allows the user to see the hierarchical structure of the document, revealing features such as chapters and sections. Each structural item is a link to the page on which the item begins.

Page browsing via thumbnail images - Page thumbnails allow users to quickly browse the body of a document and discover and locate structural objects such as tables, graphs, and the like. Each thumbnail page is implemented using a HTML *imagemap*, allowing the user to select on a page and dispatch a URL to view that specific page in readable form. An example thumbnail page is shown in figure 7.

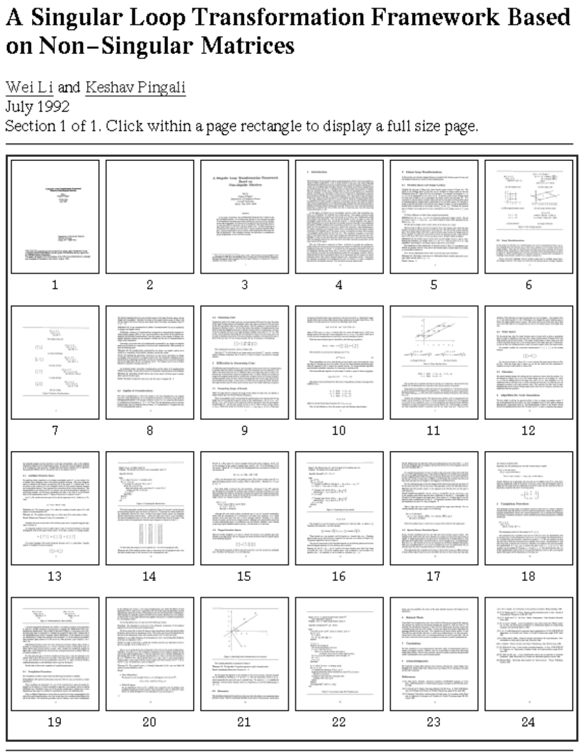


Figure 7: Page thumbnails are a tool for quick browsing of a document for formatting features (e.g., tables). The thumbnails are implemented using the HTML *imagemap* mechanism, which allows a user to select a page and view that page in full.

Inline page image - Inline page images are displayed in a resolution sufficient for on-screen reading of the document. At Cornell we have found that 72 dpi, four-bit deep GIFs have the proper combination of resolution and size for most monitors. The page image display also includes several other user interface objects:

- hyperlinks that allow the user to go the next or previous page of the document, or to the document summary page.
- two radio buttons that control the operation that is performed when the user selects a point within the document page. The two operations are zoom and full-text search, both of which are described below.

An example page-image page is shown in figure 8.

Page level zooming - The user can zoom in on a section of a page image by selecting the *zoom* radio button and clicking within the page image in the page-image display. If the document

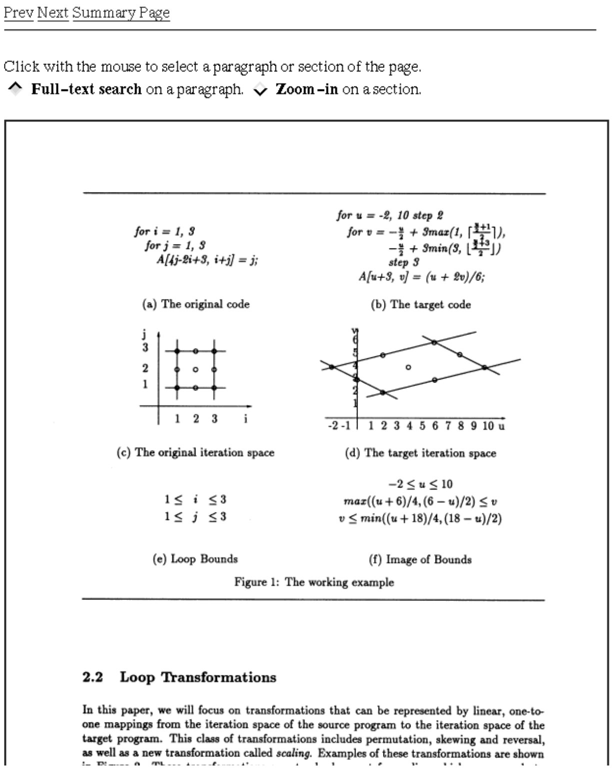
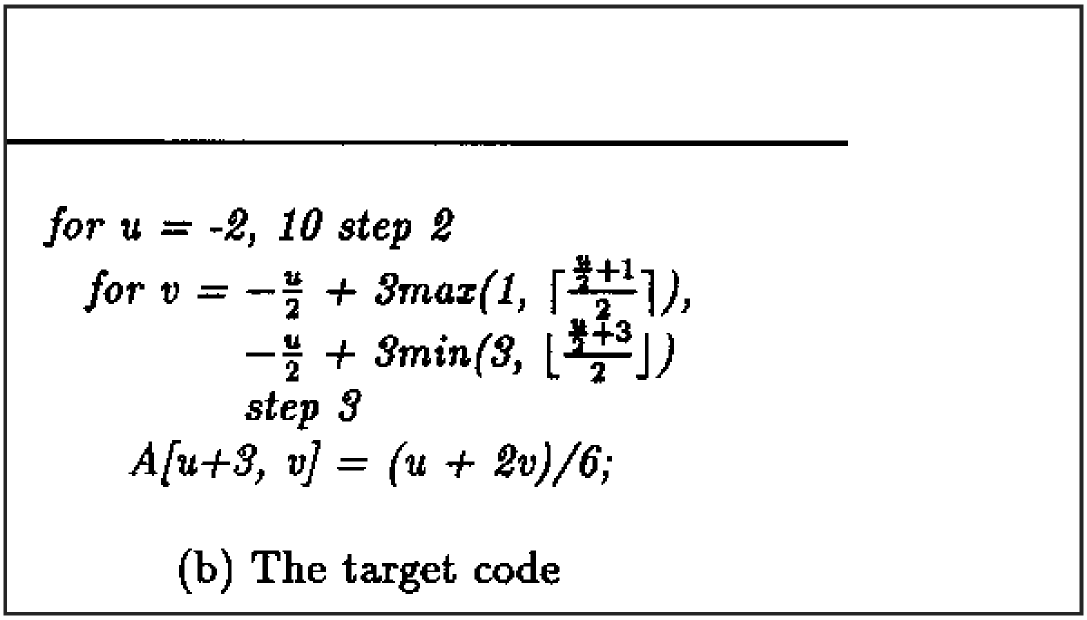


Figure 8: Pages are displayed in 72dpi GIF format with 4-bit deep gray scaling for maximum screen readability. The page display includes links to move back and forth through the document and controls for the zoom and *click-to-search* full-text search feature.

is available in a high-resolution format (e.g. scanned TIFF), a zoomed section of this format will be displayed to the user. An example zoom page is shown in figure 9.

Left Up Down Zoom Out



for $u = -2, 10$ *step* 2

for $v = -\frac{u}{2} + 3max(1, \lceil\frac{\frac{u}{2}+1}{2}\rceil),$

$-\frac{u}{2} + 3min(3, \lfloor\frac{\frac{u}{2}+3}{2}\rfloor)$

step 3

$A[u+3, v] = (u + 2v)/6;$

(b) The target code

Left Up Down Zoom Out

Figure 9: Page level zooming allows the user to see fine detail on a document page.

Click-to-search full-text search - The user can perform a full-text search based on text in the document by selecting the *full-text* radio button (the default) and clicking within the page image in the page-image display. The server will use the contents of the paragraph in the proximity of the user selection as the source for the search. Mapping from the selection point to the ASCII text is done by an analysis of the OCR output using paragraph delineation heuristics (based on line spacing). A full description of the implementation of full-text searching is in section 8.1. A full-text search returns two sets of results: document level in ranked order and paragraph level in ranked order. The results are presented as hyperlinks so the user may then dispatch a request for the specific document. Note that full-text search in the current release of Dienst only searches the collection on the server to which the search has been submitted. This is because no all servers implement full-text serach, and the ranking of distributed full-text searches is problematic. An example result page from a full-text search is shown in figure 10.

Printing and downloading - Users may choose to print or download all or selected pages from a document. The site administrator decides at server installation times the net domains from which a client request must originate for printing to be available to that client. In addition the administrator decides which printers are available to specific client domains. Clients outside the "printable" domains receive only a form that allows them to download all or selected pages of the document. In the current implementation printing is allowed for documents that are in

Full Text Search Results

Search Text

Figure 1: The working example 2.2 Loop ?ansformations In this paper, we will focus on transformations that can be represented by linear, one–to one mappings from the iteration space of the source program to the iteration space of the t& program. This class of transformations includes permutation, skewing and reversal, as well as a new transformation called scaling. Examples of these transformations are shown in Figure 2. These transformations are standard except for scaling which corresponds to replacing a loop iteration variable by an integer multiple of it.

Search Options Selected

- show document links
- show paragraph links

NOTE: For most TRs, full text is generated by running OCR on the scanned image. Depending on the quality of the original, this may result in some errors in the paragraph text and may also affect the results of the search.

Full documents in relevance order.

0.45 CORNELLCS:TR92–1294
A Singular Loop Transformation Framework Based on Non–Singular Matrices. Wei Li and Keshav Pingali.
0.31 CORNELLCS:TR94–1434
On Program Transformations. Sofoklis G. Efremidis.
0.31 CORNELLCS:TR93–1389
An Algorithm for Processing Program Transformations. Sofoklis G. Efremidis and David Gries.
0.30 CORNELLCS:TR88–922
Compaction–Based Parallelization. Alexander Aiken.
0.28 CORNELLCS:TR92–1278
Access Normalization: Loop Restructuring for NUMA Compilers. Wei Li and Keshav Pingali.
0.26 CORNELLCS:TR85–678
Percolation Scheduling: A Parallel Compilation Technique. Alexandru Nicolau.
0.26 CORNELLCS:TR86–792
A Fine–Grain Parallelizing Compiler. Alexandru Nicolau.

Figure 10: The *click-to-search* feature allows a user to select a paragraph on a page image as the input for a full-text search. Both document-level and paragraph-level results are returned from a full-text search.

PostScript or TIFF format. Documents in TIFF format are translated to ASCII85-encoded images embedded in a level 2 PostScript document.

6 Installing a basic Dienst system

This section[5] of the Dienst installation instructions describes how to install a minimal Dienst server. This is a server that only indexes a collection of bibliographic files, and does not provide access to the body of documents. All directory names, except when noted otherwise, are relative to the `dienst` directory that you create when you download the Dienst server.

6.1 Considering copyright issues

Before you make your documents available on the Internet, we strongly recommend that you think about the copyright issues. Appendix B describes our approach to copyrights and document availability over the Internet.

6.2 Obtaining publisher IDs for your site

Dienst requires that each document in the entire (distributed) collection have a unique location-independent name, known as a document identifier, or *docid.* A docid consists of a *publisher name* followed by a colon and *document name.* For example, the Cornell docid `CORNELLCS:TR94-1418`, consists of a publisher, `CORNELLCS` (the Cornell Computer Science department), and a document name, `TR94-1418`.

The publisher name uniquely identifies a *set* of documents. What defines a set of documents is up to you. You may wish to specify a single publisher for all of your documents and then use some portion of the document name, such as a *series*, to define subsets within that publisher. Or, you may wish to specify multiple publishers. Your server may provide access to the documents of one or more publishers. To propose and secure a publisher name, or names, for your site, please contact Rebecca Lasher, `rlasher@forsythe.stanford.edu`. Your proposal should include a short, one word canonical name (e.g., CORNELLCS) for the publisher(s) and a longer multi-word user friendly name (e.g. Cornell Computer Science) for each canonical name.

A document name uniquely identifies *each* document within its respective publisher. It is your responsibility to ensure that all of your documents have unique names. The format of the document name is up to you. A document name may have additional structure. For example, at Cornell the document name has three logical parts: a *series* identifier (e.g. `TR`), a *year* (e.g. `94`), and a serial *number* (e.g. `1418`).

6.3 Creating your document database

A document database resides on your file system and consists of a directory structure to house your documents in multiple formats. We will describe one format here, the bibliography format. Other formats are described in section 7.

[5]Note: Sections 6, 7, 8, 9, and 10 are copies of the installation instructions that are available on-line at: `http://cs-tr.cs.cornell.edu/dienst_install/installation.html`. Use the on-line copy for the most up-to-date version.

6.3.1 Set up a directory structure

This document assumes that you do not already have your documents organized in a hierarchical directory. If you already have your documents stored online in the fashion described here then you can use your existing organization for Dienst.

You must set up a hierarchical directory structure and house each document in a separate subdirectory. The hierarchy you use should be based on information that can be derived from a document name. For example, at Cornell we group technical reports by year, since the year is contained with document name. You shouldn't base your hierarchy on information that cannot be derived from the document name. For example, since Cornell document names contain no information about the author, we couldn't use that as a component of the database hierarchy.

A document may consist of many files, each one being a different representation of the document. Plain ASCII, PostScript and TIFF are examples of file formats that may be used to represent a document. See section 7 for a discussion of file formats. All formats for a given document **must** be placed in or under a common subdirectory (symbolic links may be used). In this section we will only describe bibliography files for a document.

The directory `examples/db` contains a small subset of the document database that we use at Cornell. It contains the directories for two documents; TR94-1418 and TR95-1474. The structure of this directory and its contained sub-tree is as follows:

- documents are first organized by year; the directory contains two subdirectories, `1994` and `1995`.
- within each year there is a subdirectory for each document; named with the numeric portion of the document name - e.g. `94-1418`.
- each document directory contains the following:
  - a bibliography file, named with the numeric portion of the document name and the `bib` suffix; e.g. `94-1418.bib`.
  - a PostScript file, named with the numeric portion of the document name and the `ps` suffix; e.g. `94-1418.ps`.
  - a subdirectory named `GIF-72-4` containing the inline GIFs; each one named with a left padded page number and the suffix `gif`, e.g. `0001.gif`.
  - a subdirectory named `Composite` containing the thumbnail composites and their associated imagemaps.

You might find it useful to refer back to this example document database throughout this installation. You will probably want to delete the directory containing this sample document database after your installation to save disk space.

6.3.2 Make the bibliography files

A basic server minimally requires only one file for each document in your collection. This must be an RFC 1357-formatted[1] ASCII bibliography file. RFC 1357 is a tagged bibliography format, with over twenty defined fields. Tags always appear at the beginning of lines and are delimited from the values by `::` and a space. The tags that are mandatory for use in Dienst are shown below in the order they should appear in your bibliography files.

- BIB-VERSION - always has the value `CS-TR-v2.0`.

- ID - the identifier of this document in the format `XXX//YYY` where `XXX` is the publisher name and `YYY` is the document name. Note that this **MUST** be the same as a *docid* except that the publisher and document names are delimited by a `//` rather than a `:`.
- ENTRY - the date of creation of the record in the format `month day, year`. The `month` must be alphabetic (spelled out). The `day` is a 1- or 2-digit number. The `year` is a 4-digit number.
- TITLE - the title of the work as assigned by the author.
- AUTHOR - an individual author in last name, first format. Multiple authors are entered by using multiple lines, each with the AUTHOR tag.
- DATE - the publication date in `month year` or `month day, year` format. The `month` must be alphabetic (spelled out). The `day` is a 1- or 2-digit number. The `year` is a 4-digit number.
- ABSTRACT - a free text abstract of the document. This field may consist of several lines of text. Only the first line needs to have the ABSTRACT tag.
- END - this must be the last line and its value must be the same identifier that was used with the ID tag.

An example bibliography file from Cornell is shown in table 1.

```
BIB-VERSION:: CS-TR-v2.0
ID:: CORNELLCS//TR95-1478
ENTRY:: January 19, 1995
TITLE:: Solving Alignment using Elementary Linear Algebra
AUTHOR:: Bau, David
AUTHOR:: Kodukula, Induprakas
AUTHOR:: Kotlyar, Vladimir
AUTHOR:: Pingali, Keshav
AUTHOR:: Stodghill, Paul
DATE:: January 16, 1995
ABSTRACT::
Data and computation alignment is an important part of compiling sequential
programs to architectures with non-uniform memory access times. In this
paper, we show that elementary matrix methods can be used to determine
communication-free alignment of code and data. We also solve the problem of
replicating read-only data to eliminate communication. Our matrix-based
approach leads to algorithms which are simpler and faster than existing
algorithms for the alignment problem.
END:: CORNELLCS//TR95-1478
```

Table 1: The Dienst search engines extracts bibliographic information from RFC 1357 formatted files.

You must adopt a uniform naming scheme for your bibliography files, which you will use when defining storage formats. We recommend that you use the suffix `bib` and use the document name of the docid as the prefix (e.g. the bibliography file for the document `CORNELLCS:TR94-1418` is called `94-1418.bib`). Each bibliography file should be placed in your document database in a subdirectory reserved for the corresponding document.

If you currently store bibliography files in *refer* (a Unix bibliography tool), we provide a tool to convert these files to RFC 1357 format. If you have bibliography files in another format and write a converter, send us a copy and we'll make it available with the Dienst release.

Your bibliography files should be checked for consistency. In particular, you should ensure that each author's name has exactly one spelling in the database. Having a single canonical name ensures that a search by author turns up all relevant documents. The Dienst software contains a number of utilities to check the consistency of your database. These are described later in this document.

6.4 Downloading the Dienst server

The server and associated software is available in a Unix compressed tar file[6]. This tar file is approximately 2.5MB in compressed form and 3.5MB when uncompressed. A large portion of this space is consumed by the `examples/db` directory, which contains a sample document database and which may be deleted at any time. All files in this tar file are rooted under a single directory, `dienst`. This directory contains the following subdirectories. Directories with capitalized names contain code, others contain readable text files.

- `Config` - contains the site-specific server configuration files.
- `Index_Server` - contains the server interface to the search engine.
- `Indexer` - contains the search engine and tools for building it.
- `Kernel` - contains the core Dienst routines.
- `Misc` - contains mail and CGI routines.
- `Optional` - contains the optional server features.
- `Repository_Server` - contains routines related to retrieving documents from the database.
- `Submit` - contains the submission package.
- `UI_Server` - contains all of the user-interface routines.
- `Utilities` - contains *perl*, Unix, and C utility files. The subdirectories `src`, `lib`, `inc`, and `bin` - contain the source, library files, include files, and binaries, respectively, of the C utilities used by Dienst. The subdirectory `smart` contains utilities for the Smart full-text search engine. The modules in the subdirectory `clients` are provided to assist those who want to use Dienst offline or for research purposes.
- `examples` - contains Cornell-specific skeletons to be used as templates for setting up your own site-specific files. It also contains an example document database in the subdirectory `db`.
- `htdocs` - contains HTML documents. The subdirectories `dienst_install`, `dienst_runtime` and `dienst_submit` contain installation, run-time server, and submission documents, respectively.

[6] http://cs-tr.cs.cornell.edu/dienst_install/Dienst.tar.Z

- `logs` - contains all log files produced by Dienst.
- `tmp` - is a temporary work space.

You can place the dienst directory within your Web server directory or in any place on your file system to which you are able to make a symbolic link from your Web server directory. If you are not already running a Web server, you will install it and configure it later. Uncompress and unpack the tar file using the command line:

```
uncompress -c Dienst.tar.Z | tar xpvf -
```

The permissions on the various components of the dienst subtree are set as follows:

- all directories are user and group writable and other readable.
- all files (except those noted below) are user, group, and other readable.
- all files in the Config directory are user and group writable.
- executable files such as `Kernel/dienst.pl` and `Indexer/build-inverted-indexes.pl` are user and group executable.

You should recursively set the user ownership and group ownership of the Dienst sub-tree to identifiers that correspond to the user and group that your Dienst server will run as. For example, if your Dienst server will run as user `dienst_user` and group `dienst_group` then execute the following commands from the directory that contains the Dienst sub-tree:

```
chown -R dienst_user dienst
chgrp -R dienst_group dienst
```

You may remove the `Dienst.tar` file after unpacking it.

6.5 Configuring your server

The Config directory contains all of the configuration files for a Dienst server. These files are:

- `config_constants.pl` - contains definitions of global variables that control the behavior of your server. You **must** change a number of these variable definitions.
- `custom.pl` - contains the subroutines that perform the mappings from a document identifier to individual files in your document database. You **must** complete these subroutines (this will require *perl* programming skills).
- `files.pl` - contains the list of files that are loaded to execute the server. There is no reason for you to modify this file.
- `indexer_config.pl` - contains the definitions of global variables that control the behavior of the indexing engine. There is no reason for you to modify this file.

6.5.1 Modifying config_constants.pl

All the variable definitions that you must change in this file are at the top of the file under the comment `Constants that you MUST CHANGE at the time of installation`. These variables are described below. The configuration file that we use at Cornell is provided as an example in `examples/Config/config_constants.pl`.

- `$permission` - a string that is the copyright message displayed on the summary page for each document. We recommend that you read about copyright considerations before you put your documents online.
- `$localhost` - a string that is the fully qualified domain name of the host on which your Web server is running.
- `$localport` - a string that is the port number of your Web server.
- `%localpubs` - an associative array for specifying the publishers at this site. Each key is the canonical, or official, publisher string (must be one word). Each value is a string that is the corresponding human readable publisher name (can be multiple words).
- `$maintainer` - a string that is the e-mail address of the administrator of your Dienst site. This will appear to the user in error messages.
- `$standalone` - a boolean value (0 or 1) that should be `TRUE` if you are not yet registered or are running Dienst as a standalone server. In this case, only publishers listed in `localpubs` will be accessible from your site. After you are registered, you should set this to `FALSE` or comment out this line. In this case, your list of publishers will be provided by the central meta server.
- `$AF_INET` - an integer specifying the domain for socket calls in the server. You shouldn't need to change this value from the preset value (2), unless the value of `AF_INET` in `/usr/include/sys/socket.h` on your system is different.
- `$SOCK_STREAM` - an integer specifying the type for socket calls in the server. You will need to change this as specified in the comments (from 1 to 2) for Solaris 2.X, and maybe some other non-BSD versions of UNIX. If you are unsure check the value of `SOCK_STREAM` in `/usr/include/sys/socket.h` on your system, and set this variable to the same value.

6.5.2 Writing code in custom.pl

Recall that every document has a unique, location-independent identifier called a *docid* which consists of a publisher and a name, and that the name may include additional information, such as a year or series, that is of local significance. The *perl* code in `custom.pl` is used by Dienst to map from a docid to the locations in your file system of the various representations of a document.

The subroutines in this file that you must write are defined below. Note that when a fatal condition occurs, the respective subroutine should issue a *perl* `die` command with an appropriate error message. The subroutine implementations that we use at Cornell are provided as examples in `examples/Config/custom.pl`.

LDocid_LT

Description: sorts the two input *docids* using local knowledge of the document naming structure. Simply using the *perl* comparison operator, `cmp`, will result in orderings such as `TR68-1`, `TR68-10`, `TR68-2`, ... We recommend that you use the code in your `Parse_document_name` subroutine (defined below) to extract the logical parts of a docid and individually sort these parts.

Arguments: the variables `$a` and `$b` are passed globally, by reference, as defined in the documentation for the *perl* `sort` function.

Returns: an integer less than 0 if `$a` is less than `$b`, 0 if `$a` is equal to `$b`, or an integer greater than 0 if `$a` is greater than `$b`; as defined in the documentation for the *perl* `sort` function.

Fatal Conditions: none.

Traverse_TRs

Description: walks the entire document database; for each directory that maps to a docid (using `directory_to_docid`, documented below) invoke the subroutine supplied as the argument. The arguments to this invoked subroutine should be the docid found and the corresponding full directory path.

Arguments:

- `$subroutine` - a string that is the name of the subroutine to be invoked for each docid.

Returns: none

Fatal Conditions: if the call to `$subroutine` returns an error code (`$@` is non-null). Also fatal if the directory traversal fails (e.g., unable to open the directory with the `opendir function`).

directory_to_docid

Description: returns the docid stored in the directory supplied as the input argument.

Arguments:

- `$directory` - a string that is the full path of the directory to map to a docid.
- `$die` - a boolean which if true indicates that the subroutine should abort if the mapping fails.

Returns: a string that is the docid found or the null string if the mapping fails. The docid **must** conform to the format `publisher_name:document_name`.

Fatal Conditions: if the mapping fails and the argument `$die` is true.

docid_to_directory

Description: maps a docid to the directory pathname where the document is stored. It is the inverse of `directory_to_docid`.

Arguments:

- `$docid` - a string that is the docid to map to a directory. The docid will conform to the format `publisher_name:document_name`.

Returns: a string that is the full path of the directory where the docid is stored.

Fatal Conditions: if the mapping fails.

Parse_document_name

Description: extracts any locally defined fields from the document name and returns a list of values. The names of the values it returns must be specified in the list `@local_document_name_variables`. If your document names have no special features you can have this subroutine always return the empty list.

Arguments:

- `$publisher` - a string that is the publisher name of the document.
- `$name` - a string that is the name of the document.

Returns: a list that contains the values of the locally defined fields from the document name. The list must have the same number of elements as the list defined by `@local_document_name_variables` and the values must be in the same order as that defined in `@local_document_name_variables`.

Fatal Conditions: If it is not possible to parse the input arguments (e.g. it is not a legal docid for your site), the code should do one of two things:

- If the variable `$server_name` is defined, call the subroutine `warning` with two arguments: the number 404, and a string indicating the message to be printed to the user (e.g. `illegal publisher name: <publisher>`). `warning` will return an HTML document to the client with the appropriate message.
- If the variable `$server_name` is not defined, die with an appropriate error message.

The lines of code below demonstrate this error handling:

```
if (defined($server_name)) {
    &warning(404, "invalid document name for $publisher: $name");
}
else {
    die "invalid document name for $publisher: $name";
}
```

6.5.3 Declaring storage formats in custom.pl

Storage declarations tell the server what file formats you use and how to recognize them in the file system. Section 7 describes how to add new document formats. This section describes how to add the required bibliography format. You declare a storage format with a call to the subroutine `def_format`, which takes the arguments listed below. The `def_format` calls that we use at Cornell are provided as examples in `examples/Config/custom.pl`

- `format` - a string that is the name for this format, which for bibliography files is always `bib`.
- `type` - a string that is is the MIME content-type for transmitting this kind of information, which for bibliography files is always `text/plain`.
- `internal` - an integer that is set to 1 if this format is for internal use only, or 0 if it should be shown to users. For bibliography files this is always 1.
- `human name` - a string that is a *pretty name* for the format, to be shown to human users, which for bibliography files is always `cataloging information`.

- `description` - a string explaining the purpose or significance of the format, which is always the null string for bibliography files.
- `pattern` - a `sprintf` string, encoding the name of the file which contains this kind of data. For documentation on `sprintf`, see the Unix man page. The string may contain arguments, the value for these arguments come from the next argument to `def_format`. The value produced by `sprintf` is a pathname relative to the directory for the document, which is returned by `docid_to_directory` (described above).
- `variables` - a string containing the names of variables to be used as arguments to the pattern, separated by a colon. These may be chosen from the set: `publisher`, `name`, `page`; in addition, you may add others, if they are defined in `@local_document_name_variables`. See `parse_document_name` above.

6.6 Checking the configuration and database

After you have edited the configuration files run the following programs from the `Utilities/bin` directory.

1. `check.pl` will perform some sanity checks on the customization you have done. In particular, it will check that the functions `docid_to_directory` and `directory_to_docid` are symmetric. It will also print some example pathnames computed from your declared storage formats.
2. `check-bib-file.pl` traverses your entire document database, checking all your bibliography records. It verifies that a bib file exists for each document directory, and that the bib file has valid syntax. When complete, it prints a count of the number of bibliographic records found.

6.7 Indexing your document database

Dienst uses author, title, abstract, and number indexes of your document database for its search and retrieval processing. You create these indexes with a separate command in the `Indexer` directory. To build the indexes `cd` to this directory and run

```
build-inverted-indexes.pl
```

This may take some time, depending on the size of collection, but the complete indexing needs to be performed only once. Each time you add new documents to your collection, you will run this command in its incremental version.

`build-inverted-indexes.pl` appends to the logfile `indexer.log` in the logs directory. You might want to examine this log to view any error messages from the scan of your database.

The utility `collect-bib-names.pl` in the directory `Utilities/bin` can be used as a quick check of your indexing. It loads the database indexes and prints a list of all authors in all bibliography files. You can use this to check that all authors are correctly named in your bibliography files.

6.8 Starting your server

You can start your server by executing the command

```
dienst.pl &
```

in the `Kernel` directory. You may wish to place this command in your system's `/etc/rc.local` so that it automatically starts when your system is booted.

6.9 Testing your server

Immediately after starting up, the Dienst server will load in the database index files, write a startup message to the log, and wait for connections over its assigned port. If all is working correctly you should see no messages on the terminal window from which you issued the command to start Dienst. You can examine the file `logs/dienst.log` to verify that a startup message was recorded in the log.

You can now communicate directly with the server via telnet and verify that it is accepting connections. Issue the command:

```
telnet <machine> <port>
```

where <`machine`> is the name of the local machine running Dienst and <`port`> is the value defined for the variable `$server_port` in `Config/config_constants.pl`. The telnet command should respond with a connection verification and wait for input. You can now type in a Dienst protocol request. Try entering:

```
/Server/Info/Version
```

followed by 6 returns. Dienst should respond with an HTTP response that includes the server version. Another request to try is:

```
/Server/Status/
```

again followed by 6 returns. Dienst should respond with an HTML document giving the status of your server. Finally, to verify that it can find documents in its database, try the request:

```
/TR/<docid>?
```

again followed by 6 returns. In this message `<docid>` is a document identifier of a document in your database. It should be in the format `publisher_name:document_name` (e.g. `CORNELLCS:TR94-1418`). Dienst should respond with an HTML document with information about this document.

Note: each time you contact the Dienst server via telnet, a warning will be placed in the file `logs/dienst.log`. This warning appears since telnet connections to your server might indicate an attempt at a security violation by an outside user.

6.10 Registering your site

When your server and database work, you are ready to register your server with the main server running at Cornell University. (Note that registration at Cornell is only relevant if you are running a computer science technical report server). If you are using Dienst for another purpose, contact `lagoze@cs.cornell.edu` for instructions on how to set up your own central server). Until you register your site, you will only be able to search and view documents in your own database (those from the publishers you have listed in `Config/config_constants.pl`.) Registration will allow you to access documents at other sites and vice-versa. To register, send e-mail to `techrpt@cs.cornell.edu` with the following information:

- Domain name of your server
- Port that it is running on

- Publisher(s) of documents that your server provides (currently, these should be the publisher name(s) you obtained above). For each publisher give a human readable name (e.g. Cornell Computer Science) and a short, single token name in all uppercase (e.g. CORNELLCS).

For example, the above information for Cornell's server is:

Domain Name cs-tr.cs.cornell.edu

Port 80

Publishers CORNELLCS:Cornell University

We will respond to you when your server has been registered. When you receive verification that your server has been registered, you should modify the variables `$meta_server` and `$meta_server_port` in `Config/config_constants.pl` as described above.

6.11 Downloading a Web server

Choose a Web server to download and follow the links to their home sites. There you will find instructions for downloading and installing the server. You should install the Web server on the same machine that you installed your Dienst server.

Dienst has been fully tested with two well-known Web servers; NCSA and CERN. You should choose the server with which you are most familiar. Once you have installed the Web server, you should test it with a sample HTML document to verify that it works independent of Dienst.

6.12 Configuring your Web server to work with Dienst

If you didn't place the `dienst` directory in an existing Web server directory, create a symbolic link to it from your `ServerRoot` directory defined in the `httpd.conf` file. You can do this by `cd`ing to your ServerRoot directory and issuing the command:

```
ln -s <dienst_directory> .
```

where <`dienst_directory`> is the path of your dienst directory.

6.12.1 Creating the home page

You should create an HTML home page for your Dienst server that contains links to some top level Dienst requests (e.g. search, list all TR's, etc). This home page should be placed in the `htdocs` directory and can have any name with a `html` suffix. A minimal home page should have a link to the URL `http:/TR/Search/`, which is the server's fielded search form. The home page used for the Cornell Dienst server is included in the release as `examples/htdocs/dienst.html`. You may want to use this as a template for your server's home page.

You can make the server's home page, as well as it's other HTML documents, accessible in two ways. If you use your Web server solely as a gateway to Dienst, you can configure it to find the documents in the `dienst/htdocs` directory itself. How you do this depends on the http server you use.

- For a `NCSA` http server, set the `DocumentRoot` directive in the httpd configuration, `httpd.conf`, file to the absolute path of the `htdocs` directory. The new path for `DocumentRoot` should also be set in another configuration file, `access.conf`.

- For a CERN http server, set the Pass directive in httpd.conf to the absolute path of your htdocs directory.

If you use your Web server for other purposes, and therefore serve other HTML documents, copy the server's home page in htdocs to the directory from which you normally serve HTML documents. Establish a link from your standard home page, or other appropriate page, to the server's home page, so you can access the server. In that same directory, create a symbolic link to htdocs/dienst_runtime, the directory where the other documents reside.

6.12.2 Configure NCSA httpd

To configure your NCSA http server to operate with Dienst, locate the ScriptAlias directives in the srm.conf file. Table 2 shows the directives to insert at this point, where <path> is the full pathname of your ServerRoot.

```
ScriptAlias /TR/         <path>/dienst/Kernel/nph-dienst_stub.pl/TR/
ScriptAlias /Server/     <path>/dienst/Kernel/nph-dienst_stub.pl/Server/
ScriptAlias /Document/   <path>/dienst/Kernel/nph-dienst_stub.pl/Document/
ScriptAlias /MetaServer/ <path>/dienst/Kernel/nph-dienst_stub.pl/MetaServer/
ScriptAlias /Submit/     <path>/dienst/Submit/
ScriptAlias /Misc/       <path>/dienst/Misc/
```

Table 2: NCSA configuration directives

After you change the srm.conf file you can force a standalone NCSA httpd to reset its configuration with the command:

```
kill -1 <process_number>
```

where <process_number> is the current Unix process number of NCSA httpd on your system.

6.12.3 Configure CERN httpd

To configure your CERN http server to operate with Dienst, locate the Exec directives in your httpd.conf configuration file. Table 3 shows the directives to insert at this point, where <path> is the full pathname of your ServerRoot.

```
Exec /TR/*           <path>/dienst/Kernel/nph-dienst_stub.pl
Exec /Server/*       <path>/dienst/Kernel/nph-dienst_stub.pl
Exec /Document/*     <path>/dienst/Kernel/nph-dienst_stub.pl
Exec /MetaServer/*   <path>/dienst/Kernel/nph-dienst_stub.pl
Exec /Submit/*       <path>/dienst/Submit/submit.pl
Exec /Misc/*         <path>/dienst/Misc/*
```

Table 3: CERN configuration directives

After you change the httpd.conf file you can force CERN httpd to reset its configuration with the command:

```
kill -HUP <process_number>
```

where <process_number> is the current Unix process number of CERN httpd on your system.

6.13 Testing your Web server with Dienst

You have now completed the basic installation. To verify that everything works correctly, start up a Web browser (e.g. Mosaic) and connect to the server's home page. Follow the link to the Dienst fielded search form. If you have not yet registered or received verification of your registration, you will see only the name of the publisher(s) at your site listed at the top of the form. You will not see other Dienst publishers displayed or be able to search for their documents until you have received registration verification and have modified the variables `$meta_server` and `$meta_server_port` in `Config/config_constants.pl`. However, you can search your own collection and should try this now.

7 Adding additional document types to your Dienst server

This section of the Dienst installation instructions describes how to add additional document types to your database and server. The basic installation instructions described how to install a Dienst server with just bibliography files.

7.1 Overview of document types

Dienst supports a multi-formatted document collection. A multi-formatted document is one that consists of many files, each one being a different representation of the document. Plain ASCII, PostScript and TIFF are examples of possible formats for a document. Document formats are either *non-paged* or *paged.*

Non-paged formats: Non-paged document formats are those where there is a single file that represents the entire document. Some common non-paged formats are PostScipt and plain ASCII text. Users will not be able to view specific pages of a document in a non-paged format (without a helper application), but they will be able to download a single file that represents the entire document.

Paged formats: Paged document formats are those where the document is represented by a set of files, each one corresponding to a single page of the document. Two common paged formats are bitmapped images (TIFFs or GIFs) or the output of an OCR process. The Dienst user interface allows users to view specific pages of paged formats of documents. Downloading the individual pages, however, requires multiple Dienst requests.

7.2 Adding new types to custom.pl

You have already defined the bibliography format using a call to `def_format` in the file `Config/custom.pl`. You must now add a new call to `def_format` in `Config/custom.pl` for each document format that will appear in your database. You declare a storage format with a call to the subroutine `def_format`, which takes the following arguments:

- `format` - a string that is the name for this format. `bib` is the reserved format name for bibliography files. There are five other format names that have special meaning to Dienst; you must use these names for these respective formats:

  - `postscript` - a non-paged format that is a PostScript representation of the document (this is the default printable type).
  - `inline` - a paged format that is an image displayed inline on HTML pages; at present the only legal MIME content-type for this format is `image/gif`.
  - `composite` - a paged format that is a composite of *thumbnail* images of pages; at present the only legal MIME content-type for this format is `image/gif`.
  - `composite_imagemap` - a paged format that is the image configuration file for a composite page image.
  - `inline_imagemap` - a description for this is in the instructions for the optional printing feature.

- `type` - a string that is the MIME content-type for transmitting this kind of information.
- `internal` - an integer that is set to 1 if this format is for internal use only, or 0 if it should be shown to users. All the types you add should have this argument set to 0.
- `documentation` - a string that is a *pretty name* for the format, to be shown to human users.
- `explanation` - a string explaining the purpose or significance of the format.
- `pattern` - a `sprintf` string, encoding the name of the file which contains this kind of data. For documentation on `sprintf`, see the Unix man page. The string may contain arguments, the value for these arguments come from the next argument to `def_format`. The value produced by `sprintf` is a pathname relative to the directory for the document, which is returned by `docid_to_directory`.
- `variables` - a string containing the names of variables to be used as arguments to the pattern, separated by a colon. These may be chosen from the set `publisher`, `name`, `page`. In addition, you may add others, if they are defined in `@local_document_name_variables`.

At Cornell, we define the PostScript format as:

```
&def_format ("postscript", "application/postscript", 0,
     "PostScript", "",
     "%s.ps", number);
```

where number is a component of the docid extracted by our `Parse_document_name` subroutine.

So, the PostScript file for document `CORNELLCS:TR94-1418` is stored in the top level directory for the document and is called `94-1418.ps`.

At Cornell, we defined the inline page image format as:

```
&def_format ("inline", "image/gif", 0,
     "inline gif image", "",
     "GIF-72-4/%04d.gif", "page");
```

where `page` is one of the predefined variables that you may use in your `def_format` calls.

So, all inline page image files for a document are stored in a directory `GIF-72-4` in the top level directory for the document. Each file in `GIF-72-4` is a four digit page number with the suffix `gif` (e.g., `0001.gif`). The file `examples/Config/custom.pl` contains the complete list of format declarations that we use at Cornell. Your Dienst server will recognize the added document types when it is restarted, or you can force a code reload with the `reload-code` command in the `Utilities/bin` directory.

7.3 Adding new file formats to documents in your database

The instructions for creating your document database describe how all formats for a given document must be stored in or under a common subdirectory. You can generate additional formats manually or your can use the *format generator utility* (described in section 9) if your documents are available in PostScript format.

For non-paged formats, we recommend that the single file for the representation (e.g., a PostScript file) be stored in the directory with the bibliography file. You may use symbolic links to logically store a format within a document directory, but physically locate it elsewhere on your file system. For paged formats, which by definition consist of a set of files, we recommend that you set up a separate sub-directory within the directory containing the bibliography file and any non-paged types. For example, the non-paged files `94-259.bib`, `94-259.ps`, and the paged files `gif/0001.gif`, `gif/0002.gif`, etc. should be housed under `1994/94-259/`.

The directory `examples/db` contains a small subset of the document database from Cornell, with two documents in multiple formats. You might find this helpful for understanding the details of this configuration.

You do **not** need to re-index your database after adding new formats. Your server will automatically recognize the files that correspond to the formats.

8 Installing Dienst Optional Features

This section of the installation instructions describes how to install the optional features of Dienst. The available optional features are:

- full text searching - allows users to perform a search on the full text of documents in your database.
- page level zooming - allows users to zoom in on sections of page images.
- foreign servers - allows users to submit searches to other technical report document servers as well as participating Dienst sites.
- printing - allows users to download and/or print documents at their site.

8.1 Installing the Dienst Full-Text Search Package

This section of the Dienst installation instructions describes how to install and use the full-text search package. This is an optional feature of Dienst; you can run a Dienst server without this package enabled. Warning!! This feature is still under development and, therefore, installation is not as easy as it should be. Rather than being a definitive "how-to", these installation instructions provide guidelines and examples of how we installed this feature at Cornell. In the future, we hope to improve this process and make it more portable.

In the following instructions, all directory names, except when noted otherwise, are implicitly relative to the `dienst` directory that you created when you downloaded the Dienst server.

8.1.1 Overview of full-text search

The basic Dienst server provides searching of bibliographic data. The full-text package expands the search capabilities by allowing the user to submit searches over the complete text of the documents in your document database. Input to a full-text search is simply a series of words. Results are returned in ranked order - theoretically the documents that "best match" the search words are ranked highest. Generally, the more words in the search request, the more "accurate" the ranking of the results.

Dienst supports two full-text search engines, WAIS and SMART. The software for both of these engines is available free-of-charge for non-commercial use. We, at Cornell, have found that SMART is a generally more reliable engine. In addition, if you are using SMART as your full-text engine, Dienst has support for paragraph level results. This means that the results from a search can include not only the documents that best match the search words, but the paragraphs in documents that best match.

Dienst provides two interfaces to the full-text search engine. The first is a simple form allowing the user to enter the words for the search. The second is a *click-to-search* feature, which allows the user to click on a paragraph in an inline image of a document page. The text of this paragraph is then used as input for the full-text search.

8.1.2 Prerequisites for this installation

In order to install the full-text package, you must have previously installed a basic Dienst system and added document types that contain the full-text of documents. As is the case with the basic installation, you will need a working knowledge of *perl* and Unix to complete this installation.

8.1.3 Overview of this installation

This installation procedure includes the following steps:

1. Downloading and installing the engine software - choose a full-text search engine and install it at your site.
2. Creating the full-text database - compile the text of your documents into a form that can be used by your full-text engine.
3. Creating the paragraph maps - create the files that are necessary for the *click-to-search* interface.
4. Enabling full-text search in Dienst - configure Dienst to allow the user to do a full-text search

8.1.4 Detailed installation instructions

To install the full-text package, please complete the following steps in the sequence.

Downloading and installing the full-text engine software Choose either SMART[7] or WAIS[8] as your full-text engine. You may install WAIS or SMART anywhere on your file system, as long as the executable is available to the Dienst server. Building and installing WAIS or SMART on most UNIX system is quite easy; refer to the `INSTALLATION` and `README` files in the distributions for more information.

[7] ftp://ftp.cs.cornell.edu/pub/smart/smart.11.0.tar.Z

[8] ftp://ftp.cnidr.org/NIDR.tools/freewais/freeWAIS-0.3.tar.Z

Creating the full-text index - In order to search full-text of documents you need to first index that text (note that this is distinct from indexing your bibliography files). The method for creating the full-text index for the specific engine, WAIS or SMART, is described in the documentation for that engine. A more general question, which we will address here, is how to extract full-text from your document database and make it available to the engine's indexer. If your documents are in some ASCII-like format, such as LaTeX or HTML, this task is relatively easy. On the other hand, if your documents are in PostScript or some scanned format, such as TIFF, then this task is more difficult. There are public domain programs for converting PostScript to ASCII. For scanned formats, you must run an OCR program to extract the ASCII.

At Cornell, all of our documents are available in either scanned TIFF or PostScript. We have chosen to use the results of OCR as the input for full-text indexing, because OCR not only extracts text but structure (e.g. paragraph) information about the documents (more on this later under creating the paragraph maps). We convert our PostScript documents to TIFF using the public domain `pbmplus` package. We then scan the TIFF's using Xerox Document Systems ScanWorX software. The result of this scanning process is a document format `xdoc`, stored like other formats in the document database. We then post-process this `xdoc` format to produce a new ASCII format for SMART input, which we call `paragraph_text`. This format contains the ASCII text of the document and markup to partition the text into paragraphs and define the position of the paragraphs on the page (for use by the paragraph link feature described below). We then scan through the document database, find the `paragraph_text` files, and use these as input to the SMART indexer. The following files in the release will help you understand this process:

- `examples/Config/custom.pl` - defines the types `xdoc` and `paragraph_text`.
- `Utilities/smart/build_doc_coll.pl` - scans through the document database, generating `paragraph_text` files from `xdoc` input files.
- `Utilities/bin/find_format.pl` - a utility that allows you scan through your document database and output the pathnames of files of a specific format (useful for creating the input to a full-text indexer).
- `examples/smart` - a directory containing the Cornell specification files for the SMART engine. The files `make_doc` and `make_para` are used to generated the document and paragraph indexes, respectively.

Creating the paragraph maps - The `paragraph_text` files, described in the previous paragraph, include information about the physical position of paragraphs on a document page. At Cornell, we use this information to generate a new format `inline_imagemap`. These are imagemap files for the Web server that define the rectangles enclosing paragraphs on pages. When a user clicks on a paragraph in a page that has one of these imagemaps, Dienst uses the paragraph coordinates to find the ASCII paragraph text in the `paragraph_text` file for the document. This text is then used as input to the full-text search engine. The following files in the release will help you understand this process:

- `examples/Config/custom.pl` - defines the type inline_imagemap
- `Utilities/smart/build_para_maps.pl` - a utility which scans the document database and generates the inline_imagemaps from the `paragraph_text`.

Enabling full-text search in Dienst - The Dienst configuration file `Config/config_constants.pl` contains a number of variables that you must change to enable full-text search in your server. These are listed below:

- `$full_text_engine` - set this to either `wais` or `smart` depending on the full-text search engine that your are using. Leaving this variable undefined means that full-text search is not enabled.
- `paragraph_links_enabled` - set this to 1 (true) if you have have indexed paragraphs as well as full documents (only available if SMART is your search engine).
- `$full_text_warning` - the text of a warning message displayed on the full-text search form and the full-text results form.

The following variables must be set if you use WAIS as a search engine.

- `$waissearch` - the path of the `waissearch` command.
- `$wais_server_dir` - the path of the directory containing the WAIS indexes.
- `$wais_doc_dbname` - the name of your WAIS database; used as the basename for all files in `$wais_server_dir`.

The following variables must be set if you use SMART as a search engine.

- `$smart_spec_dir` - the path of the directory containing SMART specification files.
- `$tr_doc_spec` - the name of the SMART specification file for document level searching.
- `$tr_para_spec` - the name of the SMART specification file for paragraph level searching (must be set if you have paragraph links enabled).
- `$smart_exec` - the full path to the SMART executable.

8.1.5 Additional information

There are three additional files in `examples/smart` that you might find useful if you are implementing SMART full-text search. These are:

- `tr_doc_spec` - the specification file we use at Cornell for document level searching.
- `tr_para_spec` - the specification file we use at Cornell for paragraph level searching.
- `query_skel` - the query skeleton file we use at Cornell.

8.2 Installing the Dienst Page-Level Zoom Package

This section of the Dienst installation instructions describes how to install the optional page-level zoom package. All directory names, except when noted otherwise, are relative to the `dienst` directory that you created when you downloaded the Dienst server.

8.2.1 Overview of zoom

The zoom option provides a point and click interface for zooming in on an area of a viewed page. When the option is enabled, a user can click on a detail of interest. The detail is centered and displayed in an enlarged format. The quality of the enlargement depends on the resolution of the original document. The feature is useful for viewing small type in equations and resolving fine print in diagrams.

8.2.2 Prerequisites for this installation

To install the zoom package, you must have previously installed a basic Dienst system and added document types for images in inline GIF format. To really take advantage of the zoom feature you need high resolution documents in `scanned` TIFF format. Note that the TIFF documents are in addition to the GIF documents. This is because, in general, Web servers can display only GIF images inline. The zoom routines require that you have Jeff Poskanzer's PBM libraries[9] and Sam Leffler's TIFF library[10]. The latter is needed only if you use the TIFF format. As is the case with the basic installation, you will need a working knowledge of *perl* and Unix to complete this installation.

8.2.3 Overview of this installation

If you have only `inline` GIF documents you need only install the zoom software for GIF. For good quality zooming, you need documents of high resolution. If all of your documents also have a `scanned` TIFF representation, you need only install the software for TIFF. If some, but not all, of your documents have a TIFF representation you need to install the software for both GIF and TIFF.

The installation procedure includes the following steps:

1. Installing zoom software for GIF - modify the zoom routines.
2. Installing zoom software for TIFF - modify and compile the zoom routines.
3. Enabling the zoom option - configure Dienst to allow zooming.
4. Configuring your CERN Web server - add a line to enable the zoom option.

8.2.4 Detailed installation instructions

Installing the zoom software for GIF files - The zoom routines automatically crop a rectangular region of a displayed page and scale the resulting bitmap. The bitmap is centered on a user-selected point of the page. The script for the GIF zoom utility resides in the Unix shell script `Utilities/bin/gifzoom`. You must check and, if necessary, change the path names of the PBM utilities for the script. The default pathnames for these utilities as defined in the script are:

```
giftopnm='/usr/local/pbm/giftopnm'
pnmcut='/usr/local/pbm/pnmcut'
pnmscale='/usr/local/pbm/pnmscale'
ppmtogif='/usr/local/pbm/ppmtogif'
```

[9] http://nl.cs.cmu.edu/usr/mlm/ftp/pbmplus.tar.Z

[10] http://nl.cs.cmu.edu/usr/mlm/ftp/tiff.tar.Z

The script assumes GIF files of 72 dpi. Different resolutions can be supported by changing the value of constants `maxwidth` and `maxheight`. The default values for these constants as defined in the script are:

```
maxwidth=612   #8.5 inches at 72 dpi.
maxheight=792  #11 inches at 72 dpi.
```

Installing the zoom software for TIFF files - These zoom routines automatically crop a rectangular region of a TIFF image and convert the result to a GIF image for inline display. The cropped region is centered on a user-selected point of the corresponding GIF page.

The source files for the TIFF zoom utility reside in the directory `Utilities/src/tiffcroptogif`. Compile the executable using the `Makefile`. The `Makefile` uses the `gcc` compiler and assumes that the PBM and TIFF libraries are on the load path for the compiler, or are in the directory `Utilities/lib`. If you need to install these libraries on your system, you can place them in `Utilities/lib`. The binary is automatically installed in `Utilities/bin`. The utility itself does not assume a particular resolution for the input TIFF files or output GIF files; the constants that define the resolutions must be set in the configuration file `Config/config_constants.pl` as described in the next section.

Enabling the zoom option in Dienst - The Dienst configuration file `Config/config_constants.pl` contains a number of variables that you must change to enable printing on you server. You might find it useful to look at the file `examples/Config/config_constants.pl` for the settings we use at Cornell for these variables.

- `$zoom_enabled` - set to 1 (true) to enable the zoom feature.

The following constants must be modified if your document size is other than 8.5 x 11.0 inches.

- `$PageWidth` - the width of the page.
- `$PageHeight` - the height of the page.

The following constants may be modified if you support the GIF type image format and wish to use the GIF zoom. You will probably not need to change them.

- `$Width` - the width of the cropped region of the original page.
- `$Height` - the height of the cropped region of the original page.

The following constants may be modified if you support the TIFF type image format and wish to use the TIFF zoom. `$DpiOut` and `$Scale` must be modified if your TIFF file resolution is other than 600 dots per inch.

- `$DpiIn` - the resolution, in dots per inch, of your GIF files.
- `$DpiOut` - the resolution, in dots per inch, of your TIFF files.
- `$Scale` - the shrink factor for the TIFF file. The resolution of your TIFF file, `$DpiOut`, can be shrunk by this amount to yield the final TIFF file resolution.

The following constants may need adjusting if you changed any of the above constants. You will probably not need to change them.

- `$XOff` - half the width of the cropped region of the original page.
- `$YOff` - half the height of the cropped region of the original page.

Configuring your CERN Web server for the zoom option - If you use an NCSA Web server you don't need to read this. If you use a CERN Web server you must add the following line to the URL translation rules of the httpd configuration file to give Dienst access to the `tmp` directory that the temporary zoom images are stored.

```
Pass    /tmp/*    <path>/dienst/tmp/*
```

where `<path>` is the full pathname of the directory in which you have installed your `dienst` directory.

8.3 Installing the Dienst Foreign Servers Package

This section of the Dienst installation instructions describes how to install the foreign servers feature. This is an optional feature of Dienst; you can run a Dienst server without this feature enabled.

In the following instructions, all directory names, except when noted otherwise, are implicitly relative to the `dienst` directory that you created when you downloaded the Dienst server.

8.3.1 Overview of foreign servers

The basic Dienst server interoperates with a distributed network of other Dienst servers. From the user's point of view, this distribution is not apparent. The user may submit searches to a number of publishers, which are indexed by the distributed sites. A search is processed in parallel by each of these sites and uniform search results are returned to the user.

The foreign server package enhances searching by allowing a Dienst server to interoperate with non-Dienst technical report servers. This feature will not be useful to you if you are using Dienst in a domain other than computer science technical reports. The two servers that are currently supported are:

- UCSTRI - the Unified Computer Science TR Index is an automatically created index of TRs available by anonymous FTP all over the Internet.
- WATERS - the Wide Area Technical Report Service is a WAIS based technical report server from Old Dominion University, SUNY Buffalo, the University of Virginia, and Virginia Tech.

If you install this option, a button will appear on your fielded search page to allow the user to submit the search to these foreign servers. Note that neither of these servers support any type of structured search. Search terms are joined together and submitted as one search string to these servers. Also the documents found by these servers are not available in a structured fashion, as those from Dienst servers are.

8.3.2 Prerequisites for this installation

In order to install the foreign server package, you must have previously installed a basic Dienst system. As is the case with the basic installation, you will need a working knowledge of *perl* to complete this installation.

8.3.3 Overview of this installation

This installation includes only one step - enabling foreign servers in Dienst.

8.3.4 Detailed installation instructions

To install the foreign server feature, please complete the following step.

Enabling foreign server search in Dienst - The Dienst configuration file `Config/config_constants.pl` contains a single variable that you must change to enable foreign server search in your server.

- `$foreign_search_enabled` - set this to 1 (true) to enable foreign server searching.

8.4 Installing the Dienst Printing Package

This section of the Dienst installation instructions describes how to install the document printing and downloading package. This is an optional feature of Dienst; you can run a Dienst server without this package enabled.

In the following instructions, all directory names, except when noted otherwise, are implicitly relative to the `dienst` directory that you created when you downloaded the Dienst server.

8.4.1 Overview of printing and downloading

The basic Dienst server allows users to view documents in multiple formats. The printing package adds two additional features to the basic server.

1. It permits all users of your server to download all or selected parts of their documents to their local machine. The user may then view and/or print the document locally.
2. It allows you to define a set of printers and the originating internet domains of users who are able to print all or part of documents to selected printers.

You will probably not install this feature if your documents are all in a non-paged ASCII-based format such as plain text, HTML, and the like. Most Web browsers allow users to print a page directly. This feature is useful if your documents are in PostScript or are in a format such as TIFF (the result of scanning). The added value of this feature in these instances is:

- PostScript - without this feature, only users who have PostScript viewers (e.g. ghostview) are able to print selected pages of a document. The Dienst print feature allows the user to select pages to print.
- Scanned Image - without this feature, users can only print one page at a time from their browser. The Dienst print code combines scanned pages (those selected by the user) into a single file for printing and/or downloading.

8.4.2 Prerequisites for this installation

In order to install the printing package, you must have previously installed a basic Dienst system. and added document types beyond the basic bibliography files. As is the case with the basic installation, you will need a working knowledge of *perl* and Unix to complete this installation. If you wish to use the included code for translating scanned TIFF files to Postscript, you will need to be familiar with running `make` files to compile software. This compilation requires that you have Sam Leffler's TIFF library on your system.

8.4.3 Overview of this installation

This installation procedure includes the following steps:

1. Using the TIFF translator - you only need to perform this step if you have scanned TIFFs that you wish to make available for printing.

2. Adding new type translators - you only need to perform this step if you have documents in other formats (e.g. LaTeX) that you wish to make available for printing.

3. Enabling printing in Dienst - configure Dienst so it will allow printing by users.

8.4.4 Detailed installation instructions

Using the TIFF translator - The TIFF translator takes the group-4 compressed TIFF files, that are the output from most scanners, and translates them to ASCII85-encoded images embedded in a level 2 PostScript file. Note that these files are quite large, since the pages are actually the embedded TIFF images. Also note that not all PostScript printers support level 2 (but most recent ones do). The code in Dienst that invokes the TIFF translator assumes that the format name of your TIFF files is `scanned`. If you wish to see how we have defined this format at Cornell, refer to the file `examples/Config/custom.pl`.

In order to use the TIFF translator, you must compile it on your machine. The directory `Utilities/src/g4tif2ps` has the source and Makefile for this program. Run make in this directory; the executable will be installed in `Utilities/bin`. This compilation requires that you have Sam Leffler's TIFF library on your system and that it is available to your linker. If you need to install this package, you may put the library in the directory `Utilities/lib`.

Adding new type translators - If you have reasonable *perl* programming skills, it shouldn't be hard to make new types printable. A general rule is that the type must be translatable to PostScript. For example, you may have documents in LaTeX that you wish to make available for printing. A typical translator for this would first run LaTeX on the document file and then run `dvips` on the LaTeX output, putting the resulting postscript file in the `tmp` directory, where Dienst could then retrieve it for printing or downloading.

To add new types, you will need to modify the subroutine `make_ps` and in the file `Optional/print_utils.pl`. You should model your new type routine after the subroutine `tiff_to_ps`, which contains the code for invoking the Cornell TIFF translator described above.

Enabling printing in Dienst - The Dienst configuration file `Config/config_constants.pl` contains a number of variables that you must change to enable printing on you server. These are listed below:

- `$printing_enabled` - this variable must be set to 1 (true) to enable the print feature.

- `%printers` - this is an associative array. The keys are the names of printers that the Dienst server can print to. Dienst uses these printer names as the setting for the `-P` option in an `lpr` command; therefore the printer must be listed in your `/etc/printcap` file. The value of each array element is a : separated list of domains of clients who may print to this printer. These domains may be fully or partially qualified (e.g., `martin.cornell.cs.edu`, or `cs.edu`).

- `@printable_types` - this is a list of the formats that can be printed in addition to postscript. Each format in the list should exist in your custom.pl file.

You might find it useful to look at the file `examples/Config/config_constants.pl` for the settings we use at Cornell for these variables.

8.5 Maintaining your Dienst server

This section of the Dienst installation instructions describes tools and techniques for maintaining your Dienst server. There tasks described here are:

- adding new documents to your database
- monitoring log files
- checking the integrity of your database

In the following instructions, all directory names, except when noted otherwise, are implicitly relative to the `dienst` directory that you created when you downloaded the Dienst server.

8.5.1 Adding new documents to your database

This section describes the three steps required to add new documents to your database and force your server to recognize those new documents. The Dienst software release also includes a submission package, which you may use to automate these tasks.

Adding documents to the file system When you originally installed your server, you created your document database. Adding new documents to the file system is done in the same manner. You must add a new directory for each docid that you add, that is accessible in the manner described in your `Config/custom.pl` file. You must, at least, put an RFC 1357 bibliography file for the new document in its respective document directory. In addition, you should put the files representing the available formats for the document in their respective directory. If you add a format not yet used in your Dienst server, you should add this document type to your server.

Updating the inverted indexes To add the new document(s) to your server's inverted indexes `cd` to the directory Indexer and run:

```
build-inverted-indexes.pl <docid1> <docid2> ...
```

where the arguments <`docid1`> <`docid2`> ... are the docids of the documents you are adding. There is no limit to the number of docids supplied as arguments. However, for a large number of docids you may find it easier to create a file where each line is a docid to be added and run

```
build-inverted-indexes.pl -f <docfile>
```

where <`docfile`> is the name of the file.

`build-inverted-indexes.pl` appends to the logfile `indexer.log` in the `logs` directory. You might want to examine this log to view any error messages from the update of your database.

Forcing the server to reload the inverted indexes A Dienst server loads the inverted indexes automatically at startup. You can force a running Dienst server to reload the inverted indexes with the command

```
reload-data
```

in the directory `Utilities/bin`,

8.5.2 Monitoring log files

Dienst maintains a log in the file `logs/dienst.log`. All entries in this file are prefixed by a date/time stamp in the format `dd mm yy hh:mm:ss`. All messages in the log file fall into three categories:

- Errors - these indicate a unexpected fatal error in your server such as a bug in the code or a database or configuration problem. All of these messages have the string `ERROR::` following the date/time stamp.
- Warnings - these indicate a request or client oriented problem. At present, there are three warnings.
  - `Unsupported dienst request` which means that the server received an invalid protocol request; this may indicate a user trying to find a security loophole in your server.
  - `Server contacted via telnet` which means that a telnet connection was made to the server port; this may indicate a user trying to find a security loophole in your server.
  - `Possible malformed tag...` in a bibliography file indicating that an RFC 1357 bibliography file had possibly invalid format.

  All of these messages have the string `WARNING::` following the date/time stamp.
- Informational messages - these are status messages indicating when your server was started, reloaded, etc. These messages have no unique prefix.

We recommend that you monitor your log file for errors and/or warnings. You can do this manually, but a more preferable method is to use an automated monitoring tool such as `swatch`[11].

8.5.3 Maintaining the integrity of your database

The *perl* utility `Utilities/bin/dbcheck.pl` is provided to help you maintain your database. With it, you can check the size, existence, and validity of your documents. The utility is extensible and new routines can be added to support new file formats. The utility consists of five files: `dbcheck.pl`, `dbsize.pl`, `dbcensus.pl`, `dbformat.pl`, and `bibcheck.pl`. The **Configuration** section below describes the site-specific changes that `must` be made to these files; the **Extensions** section describes how to modify the utility to support new file formats; and the **Operation** section describes how to run the utility.

Configuration There are some variables and routines that you *must* change. These are marked `MUST CHANGE`. In `dbcheck.pl`, at the top, you will find four banners marked `MUST CHANGE`.

- The first contains the subroutine `Usage` which describes the operation of the utilities. You should replace all references to Cornell.
- The second contains variables and a routine that are relevant to creating an official document id for one of your documents. This allows the interface of the application to accept shortened id's. For example, at Cornell we allow the document `CORNELLCS:TR99-9999` to be specified as `99-9999` for convenience. You must change all relevant code.

[11] ftp://ee.stanford.edu/pub/sources/swatch-2.1.tar.Z

- The next contains a subroutine that extracts a year from a directory name to allow the utility to check only documents of a particular year. (The name might also be extracted from the docid.) You must change the way the directory is parsed to give the year.

- The last contains the keywords that the utility uses to determine if a document is available. The routine is called when the `NOTES` field of the RFC 1357-formatted bibliographic file is checked. Cornell uses several keywords to indicate that a file is not available. For example, it may have been withdrawn, or replaced by another document. You must change the routine to search for the keywords that your site uses to signal that a document is currently unavailable.

In `dbcensus.pl`, at the top, you will also find a banner marked `MUST CHANGE`. This routine contains site-specific error checks. If you don't have any, just leave the routine empty.

Extensions There are some variables and routines that you may change to extend the functionality of the utility. These are marked `MAY CHANGE`. In `dbformat.pl`, at the top, you will find a banner marked `MAY CHANGE`. This file contains format-specific routines for testing the validity of a particular file format. You can add new routines to support new formats. There are two parts to this process.

- First, add a new variable `<format>_check`, where `<format>` is the unabridged name of the file format you have added support for. Initialize it to the name of the subroutine you'll be adding to check this format. To check a bibliographic file with a format name of `bib`, for example, you might specify `$bib_check = CheckBibFile`. You can choose any name for the subroutine.

- Then, add a subroutine to `dbformat.pl` whose sole function is to check the validity of all documents of the particular type of file format that you specified. If you specified `$bib_check = CheckBibFile` you would add a subroutine named `CheckBibFile` that checks a bibliographic file. `CheckBibFile` might check for the presence and validity of all mandatory fields. To check the validity of an image file, you could exercise it in an image viewing utility.

- Many subroutines have been already written. It is ok to leave them. They will not be invoked unless you support the format they are intended to check.

In `dbcensus.pl`, at the top, you will also find a banner marked `MAY CHANGE`. This file contains format-specific routines for retrieving the length, in pages, of a document from an non-paged file. In an RFC 1357 bibliographic file, for example, there is a field `PAGES` that should contain the length, in pages, of the document. The process of adding to this file is almost identical to adding to `dbformat.pl` outlined above. See the file for instructions.

Operation The database checker has three main operational modes that can be specified on the command line. You can check the size of your files (`-size`), the existence of files (`-census`), and the validity of files of a specific file format (`-format`). Any combination of modes can be specified. If none are specified, the default is to perform the size and census checks. The size check displays one table containing a tally of file formats and their sizes. The census check prints out one table of results for each document checked. The format check produces a verbose error report. The format check takes longer to run.

You can also specify the range of the check on the command line. Specifying nothing will cause all files in your database to be checked. You can narrow the search by specifying a particular publisher (`-pub`), year (`-year`), or single document (`-doc`). You can narrow the search further by specifying a particular file format to be checked (`-type`). The format must be specified exactly the way it is defined in the server's `custom.pl` configuration file. Multiple formats can be checked, but each one must be preceded with a `-type` switch. All checks will print out the `NOTES` field of the bibliographic file if it contains information on why the document is not available.

Following are some examples of usage. To view all of the options, type `dbcheck.pl` with no arguments.

- To check the size totals for all 1999 files, type

  ```
  dbcheck.pl -size -year 1999.
  ```

- To check the validity of all 1999 bibliographic files, type

  ```
  dbcheck.pl -format -year 1999 -type bib.
  ```

- To check the existence of bibliographic and postscript files for document CORNELLCS:TR99-9999, type

  ```
  dbcheck.pl -census -docid 99-9999 -type bib -type postscript.
  ```

- The docids of all checked documents are written to STDERR for logging purposes. To record the error messages in one file and keep a record of the files that were checked in another, direct the output like this if you use the C shell

  ```
  dbcheck.pl -format -year 1999 -type bib -out bib99.chk >&
  bib99.log,
  ```

  or this if you use the Bourne shell

  ```
  dbcheck.pl -format -year 1999 -type bib -out bib99.chk 2> bib99.log.
  ```

  The name of each file checked is directed to the log file `bib99.log`. The actual results of the check utility are directed to `bib99.chk`.

9 The Dienst Library Management Package

This document describes a set of programs that assist in the management of an online collection. There are three components of this package:

- a submit package that automates almost all aspects of entry of new documents in the document database.
- a utility that uses components of the submit package to generate additional formats for documents already in the database.
- additional utilities for other document librarian tasks such as document withdrawal, printing, and modification of bibliographic entries.

This package is currently being used in the Cornell University computer science department, but is still under development. Work still needs to be done to make the package portable to other sites. Use of this package at your site may require some modification of code in the utilities. You may find these instructions sufficient to get the package working at your site. However, if you have any question about installing or using this package, please contact `lagoze@cs.cornell.edu`.

Note that the current version of the library management package assumes that there is only one publisher at your site. All commands in the package that are document specific take a *document name* as an argument. This is the second portion of a full *docid*, which consists of both a publisher and document name. The single publisher at your site is specified by the `$publisher` constant defined below.

9.1 Submission package overview

The four components of the submission package are described below. The submission package assumes that there are two human participants - the author and a document librarian (DL), who acts as the gatekeeper for submissions. The DL does not edit any files in the document database. All files accessible to her/him are in a *working* directory that is explained below.

Submission form for the author - This is an HTML form into which an author enters bibliographic information including document title, author name(s), and abstract. The author must generate a PostScript version of the document and place that file in a file system location that is read accessible by the Dienst server software. At Cornell, our entire file system is NFS mounted, so submitters can place the PostScript file in their home directory.

The submission form is located in `dienst_submit/submit.html` in the release. This form links to a help page at `dienst_submit/submission.html` that has some Cornell specific text; you will need to modify it before using it.

Submission processing script - This a short script that checks the validity of the author submission. Its ensures that all fields of the form have been properly completed and that the PostScript file is readable and is valid PostScript (by running it through `ghostscript`). If the submission is not legal, an error message is sent to the submitter. If the submission is legal, a template RFC 1357-formatted bibliography file is created in the DL's working directory (with a name generated from the date and prefixed by the string `TMP~`), the postscript file is copied to the DL's working directory (with a name generated from the date and prefixed by the string `TMP~`) , and a message concerning the submission is sent to the DL (giving the name of the temporary files that were generated).

The submission processing script is located in `Submit/submit.pl` in the release.

Document clearing utility for the document librarian - No document enters the document database until the DL checks that the submission is from a valid party, verifies that the template bibliography file has correct data, and assigns a unique document name to the document. At Cornell, the DL performs an additional role: copyright clearing. This involves requiring that the author sign a copyright release document. The DL clears the document for entry in the database with the command:

```
install_tr.pl <name> <temp_number>
```

where `<name>` is name that the DL assigns to the document and `<temp_number>` is the temporary name of the template bibliography file (given to the DL in the e-mail message generated by document submission). Note that <`name`> is appended to the publisher defined for this site (defined by the `$publisher` setting described below) to form the full docid for this document. This command completes the template bibliography file and flags it as ready for entry in the database by changing its temporary name to <`name`>`.bib` where <`name`> is the one that was entered as an argument to the command.

The document clearing utility is located in `Submit/install_tr.pl` in the release.

Database builder script - This script runs as a background process that periodically scans the DL's working directory for entries that are ready for inclusion in the database. Upon finding an entry, it creates the database entry for the document, copies the bibliography and postscript file into that directory, generates additional document formats from the source postscript, and sends mail to the maintainer of the server about the addition to the database (reminding that the new document needs to be added to the inverted indexes). The script maintains a log of its activities, called `submit.log`, in the logs directory. The distributed software currently includes format generators for inline GIF page images and composite GIF page images (for quick browsing of page *thumbnails*). We plan to make the package more extensible in the future so sites can install their own format generators.

The database builder script is located in `Submit/build_tr_directory.pl` in the release.

9.2 Format generator utility

The format generator utility allows you to generate inline and composite formats from PostScript files for documents that are already in your database. This utility is really a front end for the database builder script described above. It traverses your document database, using the code in `Traverse_TRs`, and invokes the script `build_tr_directory.pl` for each document. This utility assumes that a PostScript file already exists for each document in your database.

This utility is in the Submit directory and is invoked with the command:

```
build_formats.pl
```

The optional argument `-f`, if present, causes the utility to generate inline and composite formats for documents even if those formats already exist. Without this argument, the utility will generate the new formats only for documents that don't already have them. You can generate inline and composite formats for an individual document with the command:

```
build_tr_directory.pl <name>
```

where <`name`> is the name portion of the docid of the document for which you wish to generate formats. Again the optional argument `-f` determines whether the generator should overwrite existing files. Also, a PostScript format must exist for the document.

In order to use these utilities, you must configure the submission package as defined below; but only need to set those variables used by the database builder.

9.3 Library management tools overview

The software includes a several additional utilities, used for other tasks that are normally performed by the document librarian. All of these utilities are located in the Submit directory of the release. These are described below:

Withdrawing documents Documents are withdrawn from the database for a variety of reasons: for example, the results are proven invalid, the document is replaced by a revised version, or the document has been submitted to a publication that prohibits distribution in other forms. The DL withdraws a document with the command:

```
withdraw_tr.pl <name>
```

where <name> is the name of the document to be withdrawn. The DL is prompted for a string specifying the reason for withdrawal. This command creates a new bibliographic file for the document in the DL's working directory. This bibliographic file contains no bibliographic information other than a `NOTE` field with the value `WITHDRAWN` and the reason (as entered by the DL). The next time the database builder script runs, it will copy this new bibliography file into the database and move all other files for the document into a special user invisible *withdrawn* directory (defined below by the `$withdraw_directory` constant). The Dienst server will display the withdrawn reason to any users who try to view this document.

Producing hardcopy Hardcopy is needed for archival purposes or for document requests from people that don't have electronic access. The DL sends a document to a printer with the command:

```
print_tr.pl <name> <copies>
```

where <name> is the name of the document to be printed and <copies> is the number of copies to be printed. This command generates a cover page image for the document that contains the title, author, and document number of the document and a background image. The title and author are constructed from the bibliographic file for the document. The remaining parts of the cover page are done using a PostScript template. There is a Cornell specific cover page template in `Submit/background.ps`; you will want to change it.

The actual command used for printing the document is defined by the variable `$print_command` as described below. At Cornell we use a package called `EZ_Publish`, that prints to a central campus Xerox DocuTech. This printer handles binding and use of cover stock. The only DocuTech specific code is the actual print command.

Editing bibliographic files Occasionally the DL must edit a bibliographic file to correct misspellings, etc. The DL retrieves a bibliographic file for editing with the command:

```
retrieve_bib.pl <name>
```

where <name> is the name of the document for which the bibliographic file is to be modified. The DL will be prompted for a string that gives the reason for the revision. The command copies the bibliographic file from the document database into the DL's working directory. S/he can then edit the file. The next time the database builder script runs, it will copy this new bibliography file into the database (renaming the old file <name>-<revision> where <name> is the former name of the bibliography file and <revision> is a sequential revision number). The new bibliography file will have a `REVISION` tag included with a sequential revision number and the reason for the revision (as entered by the DL).

9.4 Notes on directory structure and permissions

The Dienst library management package assumes the following directory structure:

- A Dienst document database. This database must be readable by the Dienst server, readable by the document librarian, and writable by the database builder script.
- A working directory for the document librarian. This directory contains two subdirectories; one for bibliographic files and one for postscript files (these are specified by the variables `$ps_directory` and `$bib_directory` documented below). This directory must be read/writable by the document librarian, the database builder script, and the Dienst server. All files placed in here should be writable by the document librarian.

At Cornell we handle this in the following manner:

- The Dienst server runs under a user id that is a member of two groups: `library` and `techrpt`.
- The database builder script runs under a user id that is a member of two groups: `library` and `techrpt`.
- The document librarian is a member of a single group: `library`.
- All files and directories in the document database are world readable, owned by group `techrpt` and group writable.
- The DL's working directory, and its sub-directories, are group owned by `library`, and are group writable. All files written into these directories are set to `library` group ownership and made group writable.

9.5 Prerequisites for this installation

To install the library management package, you must have previously installed a basic Dienst system and added document types for images in inline and composite GIF format. The database building utility requires that you have Jeff Poskanzer's `pbmplus` package. The location of these utilities are defined in the script `pstogifdir` as:

```
PPMTOGIF='/usr/local/pbm/ppmtogif'
PNMSCALE='/usr/local/pbm/pnmscale'
PPMQUANT='/usr/local/pbm/ppmquant'
```

You will need to change these paths as required on your system. In addition, the database building utility and the submission processing script require that you have `ghostscript`. The location of this utility is defined in both `Submit/pstogifdir` and `Submit/submit_config.pl` as:

```
GS='/usr/local/gnu/bin/gs'
```

You will need to change this path as required on your system.

As is the case with the basic installation, you will need a working knowledge of *perl* and Unix to complete this installation.

9.6 Configuring the submission package

The file `submit_config.pl` contains configuration constants for the library management package. The constants that must be changed are all at the top of the file under the comment `MUST CHANGE`. Some constants are used by the building utility, others are specific to the other library utilities - these are labeled as such below.

The following are constants used by the database builder:

- `$gif_directory` - a string that specifies the pathname of the directory in which the database builder script should generate inline gifs from the postscript. Note that this directory is linked to from the document database directory for the document.
- `$build_directory` - a string that specifies a directory to use as a format building area. Make sure this directory has sufficient room to generate many temporary files (for a large document this may consume 50-100 MB).
- `$withdraw_directory` - a string that specifies where the files related to a document should be placed upon document withdrawal. Users will not be able to view files in this directory.
- `$publisher` - a string that is the local publisher. The package at this point only handles a single publisher. This string is used to build the docid in the bibliography file (prepended to the document name supplied by the DL).
- `$organization` - a string defining the `ORGANIZATION` field for the bibliography entries.
- `$tr_librarian_addr` - e-mail address of the DL.
- `$tr_librarian_group` - group identity of the DL (for Unix file permission).
- `$db_files_group` - group and owner of files in document database.
- `$gs` - a string that is the `ghostscript` command used to generate formats from PostScript; just change the path to correspond to your system.

The following are constants used by other utilities. If you plan to use only the database builder, you don't need to edit these.

- `$ps_directory` - a string that specifies the directory into which the submission processing script should copy PostScript files for processing by the DL.
- `$bib_directory` - a string that specifies the directory into which the submission processing script should generate template bibliography files for processing by the DL.
- `$ftp_directory` - comment this out, it is Cornell specific.
- `$latex_directory` - comment this out, it is Cornell specific.
- `$print_command` - a string that is the command used to print documents in the `print_tr.pl` script. The shipped setting is specific to the Cornell EZ-Publish system.

9.7 Configuring your Web server to enable Submissions

If you use an NCSA Web server you don't need to read this. If you use a CERN Web server you must add the following line to the httpd configuration file:

```
Exec   /Submit/*   <path>/dienst/Submit/submit.pl
```

10 Advanced server customization

10.1 Replacing the Dienst built-in search engine

The Dienst server contains a simple, but adequate, search engine for bibliographic data. Data for the search engine is in the form of inverted indexes created by `build-inverted-index.pl`. The indexes are stored on disk in the directory `Indexer/Indexes`. At server startup, or on receipt of a `USR2` signal, these indexes are read into memory into associate arrays. All database searches use these associative arrays.

The system assumes that bibliographic data is available in RFC 1357-formatted ASCII bibliography files. However, the only RFC 1357 specific code in the server is in the file `Indexer/parse_bib_file.pl`. Section 10.2 describes how to use other bibliographic formats in Dienst.

All communication between the server and the search engine flows through the subroutines in the file `Indexer/indexer_interface.pl`. The components of the interface are described below. Using a new search engine requires replacing the calls to the Dienst database engine within these subroutines with semantically equivalent calls to the new search engine.

Tagged_Search

Description: Perform a fielded search on the database. The input is an associative array whose keys are the fields to search on and the values are the search criteria for the respective field. The supported fields are:

- `publisher` - a string that specifies the publisher of matching documents. A match is successful if the input value matches a leading sub-string of the documents publisher (e.g. `CO` matches `COLUMBIA` and `CORNELL`).
- `number` - a string that specifies the document name of matching documents (use of `number` for the field name is the result of a historical inconsistency). A match is successful if the input value matches any sub-string of the document's name (e.g. `94` matches `94-1418` and `68-194`).
- `author` - a string that specifies authors' first or last name or names of a matching document (see the rules for bibliographic keyword matching below).
- `title` - a string that specifies words in the title of a matching document (see the rules for bibliographic keyword matching below).
- `abstract` - a string that specifies words in the abstract of a matching document (see the rules for bibliographic keyword matching below).
- `any` - a string that specifies words in any of the bibliographic keyword fields (i.e. title, abstract, and author). Thus `any=foo` is equivalent to `author=foo`, `title=foo`, `abstract=foo`.

Rules for bibliographic keyword matching - Words in the three bibliographic keyword fields (author, title, abstract) are matched to bibliographic entries according to the following rules:

- Each word matches any word in the respective field that begins with the respective word. For example, the word `comp` matches `computer`, `computation`, `comprehensive`, etc.
- The value for a keyword field may contain the logical connectors `and` and `or`. For example, `robotics or vision` in the abstract field, will return documents that have the word `robotics` or `vision` in their abstracts. `robotics and vision` in

the abstract field, will return documents that have both the word `robotics` and `vision` in their abstracts. Multiple words that are not separated by `and` are assumed to be `and` separated. For example, `computer vision` in the abstract field, will return documents that have both the words `computer` and `vision` in their abstracts. Finally, parentheses may be used to group words. For example, `Gries or (Teitelbaum and Field)` in the author field, will return documents authored by `Gries` or by `Teitelbaum` and `Field`.

- Finally, the input to the subroutine may specify either the logical connector `and` or `or` between the bibliographic keyword fields (the default is `and`). For example, `or`ing `robot` in the Title field and `robotics` in the abstract field will return documents that have either `robot` in their titles or `robotics` in their abstracts. `and`ing these fields will return only those documents that have `robot` in their titles and `robotics` in their abstracts.

Arguments:

- `list` - a reference to an array that will be filled with a **sorted** list of the docids that match the search criteria.
- `terms` - an associative array where keys are the fields to be searched and values are the search criteria for the respective fields (as described above).
- `and` - a boolean which is true if the criteria in terms should be `and`ed together and false if the criteria should be `or`ed together.

Returns: a string that is a status message if errors occurred in the search.

Get_Bib_Data

Description: Get bibliographic data for a document.

Arguments:

- `fields` - a reference to an associative array whose keys correspond to the bibliographic fields to be returned. The supported bibliographic fields are:
  - `TITLE` - the title of the document.
  - `AUTHOR` - the author(s) of the document (multiple authors are separated by a colon (:).
  - `ABSTRACT` - the abstract of the document.
  - `DATE` - the entry data of the bibliography record.
  - `NOTES` - any descriptive notes about the document.

  Each element of the associate array is set to the corresponding untranslated entry in the bibliographic record.
- *docid* - a string which is the document identifier of the document for which data is to be retrieved.

Returns: a string that is a status message if errors occurred in the retrieval.

Get_All_IDs

Description: Get a list of all the docids in the database sorted in lexigraphic order.

Arguments

- `list` - a reference to an array that will be filled with a **sorted** list of the docids.

Returns: a string that is a status message if errors occurred.

Get_All_Authors

Description: Get a list of all author names in the database, order is undefined.

Arguments

- `list` - a reference to an array that will be filled with the list of author names.

Returns: a string that is a status message if errors occurred.

Get_Indexer_Status

Description: Return an HTML document describing the status of the indexer (number of records, etc.). The contents and format of this status document are undefined, it is simply displayed.

Arguments

- `string` - a reference to a string that will be filled in with the HTML status document.

Returns: none

10.2 Using other bibliography formats

Dienst is constructed to use RFC 1357-formatted ASCII bibliography files. However, the only code in the system that is RFC 1357 specific is in `/Indexer/parse_bib_file.pl`. Using foreign format bibliography files with Dienst requires reimplementing the subroutine `parse_bib_file`. The interface to this subroutine is described below:

parse_bib_file

Description: Reads a disk-resident bibliographic entry for a docid and returns bibliographic data.

Arguments

- `path` - a string that is the full pathname of the bibliography file.
- `fields` - a reference to associative array whose keys specify the bibliographic data to be retrieved from the file. The supported data keys are:
  - `title` - the title of the document.
  - `author` - the author(s) of the document in last, first format (multiple authors are separated by a colon (:).
  - `date` - the publication date in `month year` or `month, day, year` format. The month must be alphabetic (spelled out). The `day` is a 1- or 2-digit number. The `year` is a 4-digit number.
  - `notes` - a free format string with any descriptive notes about the document.
  - `entry` - the date of creation of the bibliographic record in the format `Month Day, Year`. The month must be alphabetic (spelled out). The `Day` is a 1- or 2-digit number. The `Year` is a 4-digit number.
  - `abstract` - a free text string that is the abstract of the document.

  The values of each element are filled in with the data for the respective key that is stored in the bibliographic file. The array element `docid` is always returned with the docid of the document.

- `requested_docid` - the docid for which data is requested (a bibliography file may contain multiple bibliographic records but this is discouraged).
- `loghandle` - a file handle which should be used as the argument to the subroutine `Log_Error` if errors occur in processing. The arguments to `Log_Error` are:
  - `loghandle` - the same as described above.
  - `string` - a string describing the error.

Returns: `true` if no errors occurred, `false` if errors.

A Dienst protocol

All communication with Dienst servers is done via the Dienst protocol. Since Dienst servers are accessed through gateways from Web servers, the Dienst protocol uses HTTP (the protocol of the World-Wide Web) as a transport layer for requests. A Dienst request occupies the *path* component of a URL that is used by a Web client to construct an HTTP request for a Web server. Thus, Dienst protocol requests conform to the rules for the path component of Uniform Resource Locators (URL's). For example, all space characters in a URL must be written as `%20`. Other special characters are:

- `/` - separates components in the URL.
- `?` - separates optional arguments from the rest of the URL.
- `#` - indicates reference to a named anchor within a document.
- `=` - separates name from value in an argument list.
- `&` - separates multiple arguments after a ? .

If these characters are used in any way other than the intended meaning they must be escaped as hexidecimal equivalents preceded by a `%` character..

All responses from a Dienst server are framed as full HTTP responses including such items as `http version`, `status`, and MIME-typed `content header` of the response.

This appendix contains examples of the Dienst protocol that is used by the current implementation of Dienst. We present this information with two *caveats.*

1. This description is neither complete nor precise. Some aspects of the protocol are not described and those that are described are done so informally.
2. The impreciseness and incompleteness are on purpose since the protocol is still being developed. The format and meaning of requests will probably change as the architecture is further developed.

Fixed portions of protocol requests are shown in `typewriter` font. Variables within protocol requests are shown in *italics.* Many protocol requests include the variable *docid.* This is a combination of publisher and name, separated by a colon (e.g., `CORNELLCS:TR92-1321`).

A.1 Repository requests

A.1.1 Requests about the entire collection

`/Server/List-Contents`
Return a list of the docids available from this server, one per line. The type of the returned document is `text/plain`.

`/Server/Bibliography`
Return the bibliographic information (in RFC 1357 format) for documents stored in the repository. This is a `text/plain` document with each bibliographic record beginning on a new line.

These requests can be modified to restrict the information by appending a question mark, followed by a modifier which may be:

`file-after =` *UT*
Where UT is a date and time expressed in RFC 1036 format (the standard by HTTP), all records whose file was modified on or after the date specified. Note that this is distinct from any dates encoded internal to the file, eg. the entry date. An example of data in such a format is `19 Jul 93 06:00:00 GMT`. (Remember to use `%20` for the space characters.)

A.1.2 Requests about an individual document

The following requests return information about a single document. Requests are provided for accessing the document as whole, or indexing it by page number, or for returning meta-information such as the bibliographic data or the formats in which the document is available. It is anticipated that other forms of document addressing might be added in the future, e.g. access to the table of contents or list of figures.

It will often be the case that a document is available in more than one form. The client may explicitly select a data format using the `Accept` request field in the header. (See the HTTP specification for details.)

In addition, the client may encode further restrictions on the data format in the request. (The reason for this feature is to allow inline images in HTML documents to specify an image format. The HTML syntax for inline images does not provide any means of specifying the desired data type, so it must be encoded into the URL.) The encoding is expressed by following the request with `?type=`*type*, where *type* is one of a set of format names defined for documents in the repository (e.g., `inline`, `scanned`).

`/Server/TR/`*docid*`/Body`
Return the body of the document. Possible returned data types include, but are not restricted to `text/plain` and `application/postscript`.

`/Server/TR/`*docid*`/Page/`*nnnn*
Return a single page, where the document is available in discrete pages. The returned data type depends on the paged format (e.g. `image/tiff` or `image/gif`).

`/Server/TR/`*docid*`/NPages`
Return the number of pages for this document, when it is available in discrete pages. Return type is `text/plain` and consists of the number alone.

`/Server/TR/`*docid*`/Bibliography`
Return the bibliographic information in RFC 1357 format. The return type is `text/plain`.

`/Server/TR/`*docid*`/Formats`
A list of the formats the server is prepared to deliver the document in, one per line. The format of these lines is: `content type size` where `content type` is the MIME content type, and `size` is in bytes, if it can be determined. (In general, the size can only be determined if the data is stored in a single file.) There is no guarantee that, if the data is retrieved in this form, that this is the number of bytes that will actually be transmitted, as it is possible that the file might be stored compressed, but be transmitted uncompressed, or vice versa.

A.2 Indexer requests

This section describes the Dienst protocol for searching a bibliographic database. This feature allows one to query a Dienst server that has a bibliographic database and obtain a list of all documents matching a search query. Although the search facility is less powerful and general than that of Z39.50 or SQL it is also much easier to implement and use, and arguably it is sufficient for meeting the bulk of actual user needs. Note that the Dienst server implementation includes a simple bibliographic database for RFC 1357 bibliographic records, but the protocol does not depend upon the format of the database or the search engine. The protocol will work equally well with other, more powerful, back-end search systems.

The indexer protocol consists of a single request that returns a `text/plain` response. This response is mainly intended to be processed by programs, not people. In fact, this indexer request is used by a Dienst user interface server to dispatch searches to distributed indexers (as described in section 4.2).

> `/Server/Index/?`*terms*
> Where *terms* are the specifications for the search, in the form *term[&terms]*. A single term has the form *name=value*. Each *name* is chosen from the following:
>
> - `publisher` - documents for this publisher should be searched (e.g., `publisher=CORNELLCS`). There may be multiple instances of `publisher=`*value* in the *terms* list. Note that the indexer that receives this protocol request does **not** route it to another indexer that contains the specified publisher. Distributed searching takes place at the user interface level as described in section A.3.
> - `number` - documents whose document name (the second component of the *docid*) contains the string specified by *value* should be candidates for the search. For example, `number=259` matches `TR94-2590` and `TR89-259`.
> - `abstract` - *value* specifies words to match in the bibliography abstract; rules for bibliographic keyword matching are given below.
> - `title` - *value* specifies words to match in the bibliography title; rules for bibliographic keyword matching are given below.
> - `author` - *value* specifies words to match in the bibliography authors; rules for bibliographic keyword matching are given below.
> - `boolean` - The boolean connector between fields (see below).

The absence of a specification for one of the *name* fields (e.g., there is no `abstract` specification in the *terms*) or a *name* without a *value* (e.g., `abstract=`) are semantically equivalent. They both mean that there are no search specifications for this field.

Rules for bibliographic keyword matching - Entries in the three bibliographic keyword fields (`Author`, `Title`, `Abstract`) are matched to bibliographic entries according to the following rules:

- All matching is case insensitive.
- Each word matches any word in the respective field that begins with respective word. For example, the word `comp` matches `computer`, `computation`, `comprehensive`, etc.
- The value for a keyword field may contain the logical connectors `and` and `or`. For example, `robotics or vision` in the `abstract` field, will return documents that have the word `robotics` or `vision` in their abstracts. `robotics and vision` in the `abstract` field, will return documents that have both the word `robotics` and `vision` in their abstracts. Multiple words that are not separated by `and` are assumed to be `and` separated. For example, `computer vision` in the `abstract` field, will return documents that have both the words `computer` and

`vision` in their abstracts. Finally, parentheses may be used to group words. For example, `Gries or (Teitelbaum and Field)` in the `author` field, will return documents authored by `Gries` or by `Teitelbaum` and `Field`.

- Finally, the search string may specify either the logical connector `and` or `or` between the bibliographic keyword fields (the default is `and`). This is specified using the `boolean` specifier in the *terms* list. For example, `boolean=or` with `robot` in the `Title` field and `robotics` in the `abstract` field will return documents that have either `robot` in their titles or `robotics` in their abstracts. `boolean=and` with these fields will return only those documents that have `robot` in their titles and `robotics` in their abstracts.

Examples of *terms* are:

> `publisher=CORNELLCS&author=Hopcroft`
> Find all documents with publisher `CORNELLCS` and author `Hopcroft`.
>
> `publisher=CORNELLCS&publisher=XEROX&title=robot&abstract=robot&boolean=or`
> Find all documents with publisher `CORNELLCS` or `XEROX` and the substring `robot` in the abstract or title.

The returned document is of type `text/plain`, and contains two parts; a header and a body. The header contains no blank lines, and is separated from the body by a single blank line. The body consists of 0 or more records, each describing a document that matches the search criteria.

The header lines are, in order:

```
Version: 1.0
Count: N
```

where N is the number of records to follow.

A record consists of the following lines, in order. Each record, except the last, is followed by a blank line.

```
DOCID
TITLE
AUTHOR(s)
DATE
```

Note: If there is more than one author name, a colon separates the names. Table 4 shows the document returned by the Cornell server for the request `/TR/Search/?author=Donald and Xavier`.

A.3 User interface requests

The server returns a `text/html` document in response to all user interface requests.

> `/TR/`*docid*
> A page summarizing information about the document. This includes the title, author, date, and abstract and links to, or information about, various formats the document is available in. It may include other information as well.

```
Version: 1.0
Count: 3

CORNELLCS:TR88-929
On the Complexity of Kinodynamic Planning
Canny, John:Donald, Bruce Randall:Reif, John:Xavier, Patrick
August 1988

CORNELLCS:TR88-947
A Provably Good Approximation Algorithm for Optimal-Time  Trajectory Planning
Donald, Bruce Randall:Xavier, Patrick
November 1988

CORNELLCS:TR90-1095
Provably Good Approximation Algorithms
Donald, Bruce Randall:Xavier, Patrick
February 1990
```

Table 4: Dienst reply to an indexer request

`/TR/`*docid*`/ShowPage?format=`*type*`&page=`*n*
Displays page *n* of the document in format *type* where *type* is a repository-defined name for a format (as in repository requests). Includes navigational links to get to next or previous page and to the summary page.

`/TR/Search/`
Returns a search form for searching the collection.

`/TR/Search/?`*terms*
Make a search request which may result in indexer requests to multiple sites depending on the publishers listed in *terms*. *terms* are formatted exactly as in indexer requests. The user interface server will dispatch the search to the index servers that correspond to the *publishers* specified in the search form. This request returns a set of "hits" formatted as hyperlinks. Each hyperlink is to the human readable page summarizing information about the respective document (as described above).

A.4 Miscellaneous Requests

Each of these requests returns a `text/plain` document.

`/Server/Info/Version`
returns the version of the server, e.g. `Dienst 3.5.1`.

`/Server/Info/Time`
returns the local time, e.g., `Wed, 19 Apr 95 14:05:09`.

B Copyright considerations

IMPORTANT: This appendix contains a description of the approaches that Cornell and Stanford have taken regarding copyright of reports made available via Dienst. This should NOT be interpreted as legal advice. Every institution must rely on the advice of its own counsel.

At most U.S. academic institutions, the author owns the copyright to any books, articles, or technical reports produced. Until 1989, works published without a copyright notice fell into the public domain (unless steps were taken within 5 years to correct the error). In 1989 the United States joined the Berne Convention. Works published after March 1, 1989 are copyrighted even if they don't have a copyright notice on them.

In most cases reports done with government money are not in the public domain. The government can make copies but at most institutions, the author owns the rights.

Most CS-TR institutions ask authors to sign a form granting the institution a non-exclusive, revocable, royalty-free license to publish, perform, display, and distribute the works. One author's signature is binding even though all authors are usually asked to sign.

If an author has already signed or plans to sign an exclusive agreement with a publisher for a particular work (or for substantially the same work) in a particular format, that author cannot then sign a non-exclusive agreement with the institution for the same work in the same format.

If an author signs a non-exclusive agreement with an institution for a technical report and then decides to publish the same work elsewhere, the author should inform the publisher of this previous agreement.

It is important for a server manager to get the author to sign a form granting the Server Management non-exclusive rights to publish, perform, and display the works before any works are loaded. If the author indicates s/he has already signed an exclusive agreement with a publisher, the technical report should probably not be mounted on the server without permission of that publisher.

At some institutions the authors do not own the rights to their works. Each institution should be clear about copyright ownership before mounting technical reports on servers.

There are several ways to handle articles that are submitted to publishers.

1. Ask the authors not to sign the exclusive agreements with publishers. Ask them to modify the publisher's standard agreement to allow the institution to keep the work up on a server.

2. Work with publishers by making special arrangements that will allow the technical reports to stay on the servers even if an article is published.

3. If the author requests, remove the technical report from the server and point to the printed article.

B.1 Copyright handling at Cornell

Another consideration on copyright is both the notice that you include with the technical reports to inform those accessing them of their rights and what those rights should be. Potential rights include: viewing on-line and transmitting over the network (this may be legally considered a "performance" of the work); making printed copies; distributing copies to others; and selling copies to others.

One approach that is taken fairly frequently is to just dump responsibility on the user to: 1) figure out what the restrictions on each technical report are; and 2) make whatever use they want in light of those restrictions. Unfortunately, most reports include no copyright notice at all, which, since 1989, has meant all rights are reserved by the authors.

The user could assume fair use applies, and make a personal copy, but to the extent that transmission over the network is considered a performance rather than a copy, the explicit statutory fair use exceptions may not even be applicable. The user can also assume that if the author has put the work up on the server, then they at least have the right to look at it there (and make a printed copy?), but they will still be in the dark on whether they could send a copy along to someone else.

Perhaps a better approach is the explicit sub-licensing of rights to the user. This makes very clear what rights they have to make copies, quote, or redistribute the technical report. However, you do need to grant these rights in a way that allows the author to withdraw the technical report from further distribution.

To accomplish this, Cornell includes a sub-license notice on their server for those accessing Cornell technical reports.

> You are granted permission for the non-commercial reproduction, distribution, display, and performance of this technical report in any format, BUT this permission is only for a period of 45 (forty-five) days from the most recent time that you verified that this technical report is still available from the Computer Science Department of Cornell University under terms that include this permission. All other rights are reserved by the author(s).

This notice does require that the agreement signed by the authors have a "lead time" associated with the revocation of copyright (or withdrawal). Each author must sign a copyright agreement form before his/her report is placed on our server.

B.1.1 Cornell copyright agreement form

```
I, the undersigned, represent that I am (the)/(an) author of the paper:

which I have submitted for publication as a Computer Science
Department Technical report. I hereby grant to the Computer Science
Department of Cornell University the non-exclusive right to reproduce,
distribute, display, and perform this work in any format, including
electronic formats, and to sub-license these rights to others. I
represent that I have the right to make these grants, and I retain the
right to withdraw this technical report from further reproduction,
distribution, display, and performance on sixty (60) days written
notice to the Computer Science Department. In the case of withdrawal,
the Computer Science Department's sole obligation with regard to
sub-licensees will be to notify them by electronic or written means of
the withdrawal.

Please be aware that the following Notice Concerning Copyright will be
included with the electronic distribution copies of technical reports:

You are granted permission for the non-commercial reproduction,
distribution, display, and performance of this technical report in any
format, BUT this permission is only for a period of 45 (forty-five)
days from the most recent time that you verified that this technical
report is still available from the Computer Science Department of
```

```
Cornell University under terms that include this permission.  All
other rights are reserved by the author(s).
```

B.2 Copyright handling at Stanford

Stanford University[12] has developed two types of permission forms for their technical reports. One is for a particular report and the other gives blanket permission for any technical reports published in the Computer Science Department's technical reports series.

B.2.1 Stanford single report copyright permission form

```
Please use this form to grant Stanford University a license as
specified below for a single Technical Report or Technical Note. If
you have already signed a Non-exclusive Right To Publish Blanket
Agreement with Stanford, you do not have to sign this form.


YOUR INTELLECTUAL PROPERTY RIGHTS AS AUTHOR

Signing the attached "Stanford Technical Report or Note
Non-exclusive-Right-To-Publish Single Agreement" gives Stanford
non-exclusive revocable rights to all forms of the technical report or
note including print and electronic. You can still publish this report
or note as an article or book but you will need to tell the publisher
that you have given Stanford non-exclusive rights to the work.

Below you will find examples of a form you can attach to the
publisher's copyright form.

Coauthors should be aware that any of the authors can sign agreements
with publishers and that this becomes legally binding.

Authors of theses that are technical reports should know that in the
UMI agreement the author grants exclusive rights to the reproduction
and distribution in microform and from microform to paper.

Stanford CSD Technical Report or Note
Non-exclusive-Right-To-Publish Single Agreement

THIS AGREEMENT, made this ______ day of _________, 19__, between
____________________("Author") and Stanford University ("Stanford")
concerns a technical report/note by the Author now entitled
________________________________________________ (the "work").

(1) Author grants and assigns to Stanford a non-exclusive, revocable,
royalty-free license to publish, perform, display, and distribute the
```

[12] The text for this section was contributed by Rebecca Lasher(rlasher@forsythe.stanford.edu).

work in print, microform, and electronic form, directly or through sub licenses or both, at Stanford's option.

(2) Author agrees that Stanford may make charges for copies of or access to the work in print, microform, or electronic form.

(3) Stanford agrees to make the work available electronically for at least one year after the date of submission and as long thereafter as, in Stanford's sole discretion, conditions justify; Stanford may, at its option, also make the work available in print or microform in the manner that other Stanford University technical reports and notes are distributed.

(4) STUDENTS: NOTE THIS SPECIAL PROVISION FOR DISSERTATIONS: Author and Stanford agree that if Author submits a dissertation, a) Author shall identify the work as such on the submission form; b) PROVIDED THAT AUTHOR SO IDENTIFIES THE WORK, this license shall cover electronic and print form but not microform for that specific work; c) Stanford shall make that work available in electronic form and print form as provided above but not in microform.

Author

ATTACHMENT FOR PUBLISHER AGREEMENTS

Note: this form should be attached to the publisher copyright form when the material you are submitting for publication is in whole, or in substantial part, available through the Stanford University Electronic Library.

I, _________________________, have granted Stanford University non-exclusive rights to perform, display, and distribute all/ a substantial part/ of an earlier version of (name of article)____ to be published in (name of publication)____ Stanford has the right to make this information available through electronic distribution and through its printed technical reports series.

B.2.2 Stanford blanket copyright permission form

Please use this form to grant Stanford University a license as specified below for all of your Technical Reports and Technical Notes. Once you have signed and submitted this form to your department's Technical Reports Coordinator, you will not have to sign any Single Permission Forms.

YOUR INTELLECTUAL PROPERTY RIGHTS AS AUTHOR

Signing the attached "Stanford Technical Reports and Notes Nonexclusive-Right-To-Publish Blanket Agreement" gives Stanford nonexclusive revocable rights to all forms of the technical reports and notes including print and electronic. You can still publish the reports and notes as articles or books but you will need to tell the publisher(s) that you have given Stanford nonexclusive rights to the works.

Below you will find examples of a form you can attach to the publisher's copyright form.

Coauthors should be aware that any of the authors can sign agreements with publishers and that this becomes legally binding.

Authors of theses that are technical reports should know that in the UMI agreement the author grants exclusive rights to the reproduction and distribution in microform and from microform to paper.

Stanford Technical Reports and Technical Notes
Nonexclusive-Right-To-Publish Blanket Agreement

THIS AGREEMENT, made this ______ day of _________, 19__, between ___________________("Author") and Stanford University ("Stanford") concerns all technical reports and all technical notes submitted by the Author to Stanford University after the above date for inclusion in the STAN-CS, CSL, KSL, NA, Logic, CS-TR or CS-TN series.

Author agrees to submit only works to which Author has the right and authority to make the grants and assignments set out below. For every report the Author elects to submit for publication in any of the above-named series:

(1) Author grants and assigns to Stanford a nonexclusive, revocable, royalty-free license to publish, perform, display, and distribute the works in print, microform, and electronic form, directly or through sub licenses or both, at Stanford's option.

(2) Author agrees that Stanford may make charges for copies of or access to the technical reports in print, microform, or electronic form.

(3) Stanford agrees to make the works available electronically for at least one year after the date of submission and as long thereafter as,

in Stanford's sole discretion, conditions justify; Stanford may, at its option, also make the works available in print or microform in the manner that other Stanford Computer Science technical reports and notes are distributed.

(4) STUDENTS: NOTE THIS SPECIAL PROVISION FOR DISSERTATIONS: Author and Stanford agree that if Author submits a dissertation, a) Author shall identify the work as such on the submission form; b) PROVIDED THAT AUTHOR SO IDENTIFIES THE WORK, this license shall cover electronic and print form but not microform for that specific work; c) Stanford shall make that work available in electronic and print form as provided above but not in microform.

Author

Stanford

ATTACHMENT FOR PUBLISHER AGREEMENTS

Note: this form should be attached to the publisher copyright form when the material you are submitting for publication is in whole, or in substantial part, available through the Stanford University Electronic Library.

I,____________________, have granted Stanford University nonexclusive rights to perform, display, and distribute all/ a substantial part/ of an earlier version of (name of article)____ to be published in (name of publication)____

Stanford has the right to make this information available through electronic distribution and through its printed technical reports series.

C Building the Cornell collection

The Cornell technical report collection consists of over 1500 documents. Technical reports range in length from seven pages to over 200 pages (large technical reports are usually dissertations). The average technical report is thirty-six pages. The first technical report was published in the department in 1968. In the latest full year of record, 1994, sixty-eight technical reports were published.

Retrospective conversion to digital form of the existing collection began in 1993, early in the

CS-TR project. At this time the department had paper records of all technical reports. The quality of archive copy of earlier reports varied widely; many were on faded typewritten paper, some even on 11x14 inch tractor feed paper. Some reports, especially the later ones, were available in electronic form. Many electronic forms existed including `troff`, LaTeX, TeX, `PostScript`, and other proprietary and non-proprietary formats (some not available anymore!).

Rather than trying to recover existing electronic forms, we decided to adopt the uniform approach of scanning the paper archival copy of all existing documents. This process was undertaken over a nine month period by a production scanning facility run at Cornell by Cornell Information Technologies (CIT). Using Xerox scanning equipment, CIT converted the technical reports to 600 dpi group 4-compressed TIFF images. Each TIFF file ranges in size from around one kilobyte for a blank page to almost two megabytes for a high-populated complex page (e.g. a high-quality photographic image). The size of the total scanned collection exceeds 3.6 gigabytes.

It should be noted that scanning a collection, even as modest as the Cornell CS TR's, is time consuming, labor intensive, and not without problems. Even the most careful scanning technician occasionally misses pages, skews pages, or misses part of a page due to a unnoticed fold when the page is put on the scanner bed. These problems are difficult, if not impossible, to detect automatically. In addition, any problems that are detected are computationally intensive to correct. For example, a simple ninety-degree rotation of a 600 dpi TIFF image (due to incorrect scanning orientation) can take up to thirty minutes on a reasonably equipped SPARCstation 10.

An example illustrates the difficulty of correcting scanning problems. We discovered after all scanning was complete that many of our older TR's were scanned from pages that were oriented in landscape mode - two pages side-by-side. The result was a TIFF file containing two page images, which made correct page mapping impossible in the document server. While it was easy to find files with this problem (by reading the height and width from the TIFF header with a publically available TIFF package), reasonably quick correction required handcrafting c-code to split the files. Even with the handcrafted code, the location and correction process took over a week of compute time on a powerful workstation.

In addition to the manual scanning of documents, we also had to manually enter the RFC 1357 bibliographic files. While it would have been easy to write translators between RFC 1357 and other common bibliographic formats such as BibTeX, `refer`, etc, a consistent electronic bibliographic format was not available for all the TR's.

Although the scanned images are an excellent archival form, they are far from ideal for electronic distribution. They are large and slow to transmit over existing networks, few printers can directly print TIFF images, and Web browsers can only display them via a separate *helper application*. To provide increased functionality to users of the collection; we use the TIFF files as the input for a number of format conversion processes:

- using the public-domain `pbmplus` package we produce both the 72 dpi 4-bit deep GIF images for inline page images and the 10 dpi per page thumbnails for document browsing.
- using the Xerox ScanWorX software, we produce `xdoc` files that contain data on both the text and structure of each page.
- using in-house software we analyze the `xdoc` files to produce ASCII output and paragraph marked-up ASCII output, which are input for the full-text search engine.
- using in-house software we (at each download or print request) produce ASCII85-encoded level 2 PostScript files that can be sent to most PostScript printers.

We implemented the Dienst library management package in late 1994 to automate the collection building process. Rather than using scanned documents, this package takes as input PostScript files. RFC 1357 bib files are automatically generated from information provided by the user in an HTML submission form. This package is fully described in section 9.

Acknowledgements

This work was supported in part by the Advanced Research Projects Agency under Grant No. MDA972-92-J-1029 with the Corporation for National Research Initiatives (CNRI). Its content does not necessarily reflect the position or the policy of the Government or CNRI, and no official endorsement should be inferred. This work was done at the Design Research Institute, a collaboration of Xerox Corporation and Cornell University, and at the Computer Science Department at Cornell University.

References

[1] Danny Cohen. A format for e-mailing bibliographic records. RFC-1357.

[2] Kurt J. Maly, Edward A. Fox, James C. French, and Alan L. Selman. Wide Area Technical Report Server. Technical Report TR-92-44, Old Dominion University, 1992.