

# **\*:96 Internet application layer protocols and standards**

## **Compendium 5A**

### ***Miscellaneous***

## ***Not allowed during the exam***

**Last revision: 14 Jan 2005**

Bildformat i datorer .....	2-4
CGI Tutorial - Common Gateway Interface.....	5-10
Internet Relay Chat .....	11-12
Other Application Layer Standards .....	13
PICS Internet Access Controls Without Censorship.....	14-18
X.500 - The organisation of information in the Directory.....	19-26

# Bildformat i datorer

När bilder skall hanteras i en dator, måste bilderna, liksom all annan information en dator, omvandlas till en serie ettor och nollor. En sådan etta eller nolla kallas för en bit (se vidare ordet *bit* i NE). Det finns ett flertal olika metoder att lagra bilder i datorer.

Två huvudmetoder är *linjebilder* och s.k. *bitmappade* bilder. Linjebilder beskriver en bild logiskt genom att lagra kommandon av typ ”dra en linje som är 0,1 mm tjock från punkt 12,45 till punkt 26,72”. Linjebilder använder utöver raka linjer också cirklar/ellipser eller delar av cirklar/ellipser (båglinjer). *Bezierkurvor* är ett avancerat format att beskriva en komplex linje som en serie av delsegment av båglinjer och raka linjer. Linjegrafik kombinerast ofta med fyllning av hela ytan inom en slutna linje med en viss färg.

Bitmappade bilder delar upp bilden i ett mönster av bildpunkter, och lagrar färg och ljushet för varje sådan bildpunkt. En sådan bildpunkt kallas för en pixel. För bitmappade bilder gäller att tätare pixlar ger en skarpere bild. Dataskärmar har ofta en pixeltäthet av 72-96 pixels per tum, medan färgbilder i tidningar ofta har en pixeltäthet av 150-300 pixels per tum och tryckt text i tidningar ofta produceras med en pixeltäthet av 1200 pixels/tum eller mer.

Om man bara vill hantera helt svarta och helt vita bildpunkter, räcker det med en bit per pixel. Med 8 bitar per pixel får man 256 olika färgnyanser, och med 24 eller 32 bitar får man miljontals olika färgnyanser.

Bara 8 bitar per pixel kan göra det svårt att bra återge bilder med många färgnyanser. Det finns två varianter av hur man kan återge en bild med bara 8 bitar per pixel. Med den ena metoden använder man en standarduppsättning av höst 256 färger, samma för alla bilder. Den metoden, som används på en del mycket enkla och billiga datorer, ger ibland en kraftig förvrängning. Med den andra metoden väljer man de 256 nyanser som bäst passar för en viss bild. Den metoden ger mycket bättre bild, ofta kan det mänskliga ögat inte skilja en sådan bild från en bild med mycket fler bitar/pixel. Men om man konverterar

en bild med fler bitar/pixel och fler än 256 färgnyanser till en bild med bara 8 bitar/pixel, så förstör man information, man kan inte få tillbaka exakt den ursprungliga bilden igen när man öppnar den lagrade bilden.

När man omvandlar en bild med mer än 256 färgnyanser till 256 färger, använder man sig ofta av s.k. *gittring* (eng. *Dithering*), där ett mönster av bildpunkter, sedd på långt avstånd, kan se ut som en mellanliggande blandfärg.

När man ombildar en linjebild till en bitmappad bild, kan man använda sig av en metod som kallas för *kantutjämning* (eng. *antialiasing*) och som gör att sneda linjer inte blir så taggiga.

Om man vill använda en standarduppsättning av färger som fungerar på nästan alla datorer med färgskärm, använder man den s.k. Netscape eller webpaletten, som består av 216 färger.

Om man förstörar en bitmappad bild, blir pixeltätheten lägre och bilden blir suddigare eller hackigare. En linjebild däremot blir lika skarp även om man förstörar den.

En linjebild måste alltid förr eller senare omvandlas till en bitmappad bild när den skall visas på en skärm eller tryckas på papper. Processen att omvandla en linjebild till en bitmappad bild kallas för *ripping*, från engelska RIP = Raster Image Processor (se ordet *RIP* i NE). Ett närliggande ord är *rendering* (se ordet *datorgrafik* i NE), som handlar om att omvandla en förenklad logisk beskrivning av en ofta tredimensionell bild med belysning och skuggor till en färdig bitmappad tvådimensionell bild.

Bitmappade bilder kräver ofta mycket minne i datorn. En hel A4-sida med 300 pixels per tum och med 24 bitar/pixel kräver 26 megabyte och tar 14 minuter att ladda ner via modem med 32 kbit/sekund. För att spara lagringsutrymme och nedladdningstid använder man sig därför ofta av komprimering. Två vanliga komprimeringsmetoder är varianter av LPV (Lempel-Ziv-Welch) och JPEG. LPV är en icke-förstörande komprimeringsmetod – exakt samma bild kan återskapas när komprimeringen upphävs. LPV ger en kraftig komprimering för

bilder som innehåller återkommande mönster, vilket är vanligt när ursprungligen linjebilder omvandlas till bitmappade bilder.

JPEG bygger på kunskap om det mänskliga ögat. T.ex. sitter de punkter på näthinnan, som kan skilja på olika färger, mycket glesare än de som kan skilja på olika ljushetsgrad. Därför kan man ta bort mycket av färginformationen ur en bild utan att det mänskliga ögat märker något. Bl.a. upptäcker JPEG gradvisa övergångar mellan två färger och ersätter dem med en mera reguljär gradvis övergång, vilket kan lagras med färre bitar. JPEG ger en kraftig komprimering vid bilder som innehåller mjuka övergångar mellan olika färger, vilket är vanligt för t.ex. fotografier. JPEG är inte bra för vissa bilder med stora jämnt färgade ytor. För sådana bilder kan det uppstå s.k. artefakter, alltså onaturliga förvrängningar, när man sparar dem i JPEG-format.

Eftersom JPEG tar bort information ur en bild, får man inte tillbaka samma bild igen när komprimeringen upphävs. Varje gång man öppnar, ändrar i och lagrar en JPEG-bild, blir bilden suddigare och mer förvrängd. Man bör därför vara försiktig med att använda JPEG som mellanformat för en bild som man kan tänkas ändra på i framtiden.

MPEG-formatet för komprimering av video liknar JPEG, men tar också hänsyn till att det ofta är små skillnader mellan två på varandra följande bildrutor i en film.

Några av de vanligaste förekommande bildformaten är:

GIF passar för bilder med mönster och jämnt färgade ytor. Maximalt 256 färgnyanser kan användas, men man kan välja de 256 nyanser som bäst passar för en viss bild. GIF tillåter också delvis genomskinliga bilder, och bilder med *animering*, alltså rörliga bilder bestående av en serie av bildrutor. GIF kan också använda s.k. *interlacing*. Därmed menas att bilden sänds till en dator så att man först kan visa en suddigare bild som sedan blir allt skarpere när mer information har överförts. GIF använder LPV-metoden för komprimering.

PNG liknar GIF, men tillåter mer än 256

nyanser. Även PNG använder en LPV-liknande metod för komprimering, men inte just den ursprungliga LPV-metoden, bl.a. därför att denna metod är patentskyddad till år 2003.

TIFF är en metod som används mycket för att sända bitmappade bilder till tryckerier och för att lagra bilder vid produktion av tryckta skrifter. TIFF kan använda LPV-metoden för komprimering. TIFF är en icke-förstörande lagringsmetod.

BMP är ett på Windowsdatorer vanligt Microsoft-format med egenskaper delvis liknande GIF och PNG.

EPS och Postscript (två varianter av samma format) är det vanligaste formatet för linjebilder, men även bitmappade bilder och text (med angivande av typsnitt och storlek) kan ingå som del i en EPS-bild. Därför är EPS ett vanligt format vid produktion av tryckta skrifter. EPS används för enstaka bilder, Postscript för hela färdiga sidor som skickas till en skrivare. Postscriptfiler kan bli mycket stora jämfört med andra format.



PDF (Portable Document Format) är ett format som komprimerar EPS och Postscript. Det användes ursprungligen av programmet Adobe Acrobat, men stöds numera av flera andra program från olika tillverkare. PDF innehåller också en funktion att typsnitt som inte finns på den mottagande datorn kan ersättas av andra, liknande typsnitt, som även kan ändras för att ännu mer likna det ursprungliga typsnittet. PDF innehåller också funktioner för bl.a. hyperlänkar (klickbara länkar som i World Wide Web) och formulär med ifyllbara fält. PDF används ofta som alternativ till HTML för webbsidor, därför att PDF kan ge en mera exakt återgivning av skaparens intentioner än HTML. En nackdel med PDF, jämfört med HTML, är att sidor inte enkelt kan omvandlas genom att ändra antal tecken/rad för att passa visning i små fönster, t.ex. på mobila datorer och läsare för elektroniska böcker. PDF kan ställas in på mer komprimering (passar för webbdokument) eller mindre komprimering (passar för trycksaksproduktionen). PDF används mycket inom trycksaksproduktion.






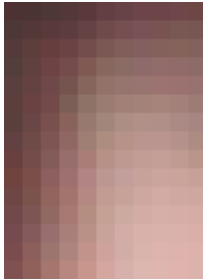
### Kort sammanställning av vanliga format:


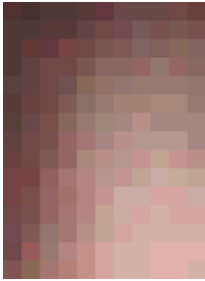

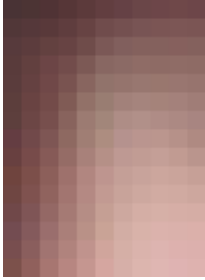

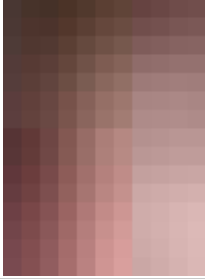


Namn	Vanligt slut på filnamnet	Egenskaper	Användning
GIF	.gif	Max 256 färger, LPV-komprimering, animering och delvis genomskinliga bilder. Om den ursprungliga bilden innehåller mer än 256 färgnyanser förstörs information vid omvandling till GIF-format.	Webbilder, speciellt om de innehåller jämnt färgade ytor.
PNG	.png	Liknar GIF men kan hantera mer än 256 färger och har andra fördelar.	Webbilder, stöds dock inte av äldre webbläsare.
JPEG	.jpeg .jpg	Förstörande komprimering som tar hänsyn till det mänskliga ögats begränsningar. Kan ge mycket kraftig komprimering utan att det mänskliga ögat märker att bilden förstörts.	Fotografier, webbilder.
MPEG	.mpeg .mpg	Format för video, liknar JPEG men utnyttjar även det förhållandet att två på varandra följande bildrutor ofta liknar varandra för att uppnå ännu bättre komprimering.	Video.
EPS	.eps	Linjgrafik, men kan även innehålla text och bitmappad grafik.	Trycksaksproduktion.
Postscript	.ps	Variant av EPS för att återge hela sidor.	Trycksaksproduktion, kommunikation mellan dator och skrivare.
PDF	.pdf	Komprimerad version av Postscript med tilläggfunktioner.	Trycksaksproduktion, trycksaksliknande webb-dokument.

### Bildexempel

Här visas några exempel på bilder som illustrerar effekten av olika bildformat. Alla bilderna är i 72-96 pixels/tum, alltså det normala formatet för bilder i webbsidor (medan tryckta skrifter oftast använder mellan 160 och 1200 pixels/tum). "Förstorad 10 gånger" visar bilden i förstoring, så att man tydligare skall kunna se hur den är uppbyggd. Normalt ser inte människor bilder i sådan förstoring, och bilder som normalt ser bra ut kan se underliga ut vid en så stor förstoring.

Beskrivning	Storlek, bytes (oktetter)	Utseende som man normalt ser bilden	Del av bilden förstörd 10 ggr för att visa hur bilden är uppbyggd.
Linjeritad bild (text i oval), t.ex. eps-format, konverterat till bitmappad bild. Observera taggigheten hos sneda linjer. Anmärkning: Georgia är ett typsnitt gjort för att bli snyggt i låg upplösning utan kantutjämning.	PDF-format: 2 329 GIF-format: 938 TIFF-format: 7 554		

Beskrivning	Storlek, bytes (oktetter)	Utseende som man normalt ser bilden	Del av bilden förstörd 10 ggr för att visa hur bilden är uppbyggd.
Samma bild men kantutjämnad vid konvertering till bitmappad bild. Observera att text blir snyggare i den vänstra bilden, men fulare vid förstoringen till höger.  Observera att taggigheten motverkas genom att sätta in punkter med mellanliggande färger i hacken i den sneda linjen.  Denna bild blir likadan i GIF-format med en anpassad palett, eftersom antalet nyanser är färre än 256.	Tiff-format: 9 518 GIF-format: 2 854		
Samma bild med GIF-format och begränsat till Netscape-palettens 216 färger.  Observera att den blå ytan, som hade en av standardfärgerna, inte kräver gittring, men att triangeln, som hade en färg mitt emellan två standardfärger, orsakar gittring för att färgpunkterna, sedda på långt avstånd, skall smälta ihop till rätt nyans.	2 852		
Fotografi med en teknik som tillåter miljontals färgnyanser (24 eller 32 bitar per pixel).	i TIFF-format 43 094 Helt omprimerat 397 473 i Photoshop-format 53 932 i PICT-format utan komprimering 51 310		

Beskrivning	Storlek, bytes (oktetter)	Utseende som man normalt ser bilden	Del av bilden förstörd 10 ggr för att visa hur bilden är uppbyggd.
Samman bild i GIF-format och med en specialanpassad palett av 256 nyanser som passar för just denna bild. Viss gittering finns, se t.ex. de vita punkterna på näsans högersida.	16 750		
Samma bild i JPEG-format med JPEG inställd på låg komprimering, vilket ger bättre bildkvalitet än högre komprimering.	7 663		
Samma bild i JPEG-format med med JPEG inställd på mycket kraftig komprimering. Här ser man komprimeringen främst i suddighet i de fina linjerna, se t.ex. håret till höger om hakan. Vid förstoring 10 ggr ser man även ett större ruttmönster.	1 371		
Samma bild i GIF-format och begränsat till Netscape-paletten, som inte passar särskilt bra för nyanserna i denna bild. Observera att ett gitter av färgpunkter används för att smälta samman till rätt nyans när man ser det på längre avstånd.	8 687		

# CGI Tutorial - Common Gateway Interface

---

## Table Of Contents

---

[Introduction](#)

[Common Gateway Interface](#)

[Forms - User input to a script](#)

[Getting stuff to the script](#)

[Method=GET](#)

[Method=POST](#)

[Outputting the results](#)

[CGI Reference 1](#)

[CGI Reference 2](#)

[CGI Reference 3](#)

[CGI Reference 4](#)

## CGI Tutorial - Introduction

This tutorial describes the basics of creating scripts that handle requests from a browser. It is based on the way the Apache Unix WWW server interfaces with scripts and programs.

My primary language for creating scripts is PERL, so that's what the examples are in. These could also be written using shell or 'C' programming.

There is a lot of information out there, just go to your favorite search engine and do some searches for CGI for more information.

Text and photographs Copyright (C) 1999-2002 Bob Paauwe & Genny Paauwe  
 Sunday, 28-Apr-2002 10:32:10 PDT

# CGI Tutorial - Common Gateway Interface

## What is CGI?

CGI stands for Common Gateway Interface. It is a specification for interfacing WEB servers and external programs. References to CGI programs are dynamic, that is they can produce different output every time you access it, unlike this document which will always look the same, every time.

## Why use an external program?

The most common example is interfacing to a database engine. If you have a large database and you want people from all over to be able to access (or even modify) the data, you'll need a program to handle the interfacing (SQL queries or whatever). Another common application is all the feed back forms you see. Most of these will send the data to a script which then mails it to the appropriate person.

Text and photographs Copyright (C) 1999-2002 Bob Paauwe & Genny Paauwe  
 Sunday, 28-Apr-2002 10:32:11 PDT

## CGI Tutorial - Forms - User input to a script

### Elements of a Form

Since forms are one of the most common ways to send input to a script, lets look at how a form is put together.

<input type="checkbox"/> Display Query string <input type="checkbox"/> Display Path info <input type="checkbox"/> Display Stdin Rating: 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> <input type="text" value="default"/> Comments: <input type="text" value="Enter your comments here"/> <input type="text" value="Vanilla"/> <input type="text" value="Strawberry"/> <input type="text" value="Rum and Raisin"/> <input type="button" value="Send Data"/> <input type="button" value="Reset"/>	<pre> &lt;form method=POST action="/cgi-bin/example"&gt; &lt;input type=checkbox name="Display" value="query"&gt;Display Query string &lt;br&gt; &lt;input type=checkbox name="Display" value="path"&gt;Display Path info &lt;br&gt; &lt;input type=checkbox name="Display" value="stdin"&gt;Display Stdin &lt;p&gt; Rating: 1 &lt;input type=radio name="Rating" value="one"&gt; 2 &lt;input type=radio name="Rating" value="two"&gt; 3 &lt;input type=radio name="Rating" value="three"&gt; &lt;p&gt; &lt;input type=text name="Field1" value="default" size=20&gt;&lt;p&gt; Comments:&lt;br&gt; &lt;textarea name="Comments" rows=4 cols=19 Wrap=Hard&gt; Enter your comments here &lt;/textarea&gt; &lt;p&gt; &lt;select name="Flavors" multiple size=3&gt; &lt;option&gt;Vanilla &lt;option&gt;Strawberry &lt;option&gt;Rum and Raisin &lt;option&gt;Peach and Orange &lt;/select&gt; &lt;p&gt; &lt;input type=submit value="Send Data"&gt; &lt;input type=reset&gt; &lt;/form&gt; </pre>
---	--

Text and photographs Copyright (C) 1999-2002 Bob Paauwe & Genny Paauwe  
 Sunday, 28-Apr-2002 10:32:12 PDT

## CGI Tutorial - Getting stuff to the script

As we saw, you can use a form to supply input to a script. While this is probably the most common way to pass information to a script, it is not the only way.

You can also pass information to a script by embedding it in the URL. This way you can actually call scripts without having the user do anything more than click on a link. This method is also used by access counter programs.

By combining these different methods of supplying input to a script, you can make the script behave differently, depending on the way it was called. The script used in the previous example is like this. If we call it without providing any input [Example Script](#) it presents the form to the user (and then calls itself again to process the user input).

Text and photographs Copyright (C) 1999-2002 Bob Paauwe & Genny Paauwe  
 Sunday, 28-Apr-2002 10:32:12 PDT

## CGI Tutorial - Method=GET

Method=GET is the default way to pass information into a script. When we called the script from a link with no arguments, it used Method=GET. Any input passed to the script with this method is assigned to the environment variable QUERY\_STRING. It is up to the script to check the value of this variable and process it.

You can specify Method=GET in the <form>. The following form will call the example script with Method=GET.

- Display QUERY\_STRING  
 Display Standard Input

Submit Query

The PERL source code for [example](#), the script used in these examples.

Text and photographs Copyright (C) 1999-2002 Bob Paauwe & Genny Paauwe  
 Sunday, 28-Apr-2002 10:32:13 PDT

## CGI Tutorial - Method=POST

Method=POST will pass the information to the script on standard input. The big advantage of this over the GET method is that there is no limitation on the amount of data that can be sent. However, the POST method can only be used inside the <form> tag.

The POST method can be combined with the QUERY\_STRING and with the PATH\_INFO string to form a number of combinations.

We could set up the form tag like this:

```
<form method=POST action=/cgi-bin/example?Display=stdin>
```

to always set the Display variable regardless even if the user doesn't make a selection. We could pass variables this way that can't even be set in the form.

Any extra path information you add to the end of the script name gets passed to the script in the PATH\_INFO environment variable. Lets call the script with:

```
/cgi-bin/example/my/new/path?Display=path
```

and see what it does.

The PERL source code for [example](#), the script used in these examples.

Text and photographs Copyright (C) 1999-2002 Bob Paauwe & Genny Paauwe  
 Sunday, 28-Apr-2002 10:32:13 PDT

## CGI Tutorial - Outputting the results

Now that you know how to get data into your script for processing, how do you send the results back to the browser? It's pretty easy actually, you just write it to standard out. Ok, so you do have to send a little information back to the server to tell it what you are sending.

Basically, you want to tell the server that everything is fine and you are going to be sending data back. You also tell the server what kind of data you are sending.

You do this by writing server directive headers to standard out as your first output. Then you can follow these headers with your data. The headers and data **must** separated by a blank line. You can send any type of data back (plain text, HTML, a GIF image).

The header lines should look like this:

```
Status: 200
Content-type: text/plain
```

Now following those headers you put your data. In this case I told the server I was sending just plain text.

You can send any valid HTTP header and the server will pass it on the browser.

By naming the script so that it starts with "nph-" you can tell the server that you want to supply all the headers so just pass everything I send you back to the browser.

Script errors and anything written to standard error will be saved in the servers error log file.

The PERL source code for [example](#), the script used in these examples.

Text and photographs Copyright (C) 1999-2002 Bob Paauwe & Genny Paauwe  
Sunday, 28-Apr-2002 10:32:14 PDT

## CGI Tutorial - CGI 1.1 Reference 1

### Environment variables

SERVER_SOFTWARE	Name and version of the server software answering the request
SERVER_NAME	The servers hostname, DNS alias, or IP address
GATEWAY_INTERFACE	The revision of the CGI specification to which this server complies
SERVER_PROTOCOL	The name and revision of the information protocol this request came in with
REQUEST_METHOD	The method with which the request was made (GET, HEAD, POST, PUT)
PATH_INFO	Extra path information as given by the client
PATH_TRANSLATED	Translated version of PATH_INFO, the absolute path
SCRIPT_NAME	The virtual path to the script being executed
QUERY_STRING	Information following the '?' in the URL which called this script
REMOTE_HOST	Hostname of the machine making this request
REMOTE_ADDR	IP address of the machine making this request
AUTH_TYPE	Protocol specific authentication method used to validate the user
REMOTE_USER	Username used to authenticate (for protected scripts)
CONTENT_TYPE	The content type of the data for POST & PUT type queries
CONTENT_LENGTH	Length of the above content
HTTP_ACCEPT	MIME types the client will accept
HTTP_USER_AGENT	The browser being used to send this request

Text and photographs Copyright (C) 1999-2002 Bob Paauwe & Genny Paauwe  
Sunday, 28-Apr-2002 10:32:11 PDT



## CGI Tutorial - CGI 1.1 Reference 2

### Command Line

The command line is only used in the case of an ISINDEX query. It is not used in the case of an HTML form or any as yet undefined query type. The server should search the query information (basically the QUERY\_STRING environment variable) for a non-encoded '=' character. If it finds one, the command line is not to be used. This trusts the clients to encode the '=' sign in ISINDEX queries, a practice which was considered safe at the time of the design of this specification.

For example:

The URL `/cgi-bin/finger?bpaauwe` will run the `finger` command with `bpaauwe` as the command line argument and send back the results generated by the `finger` command.

However, if you use `/cgi-bin/finger?bpaauwe=yes` it will call `finger` without any arguments but set the QUERY\_STRING environment variable to `bpaauwe=yes`.

Text and photographs Copyright (C) 1999-2002 Bob Paauwe & Genny Paauwe  
 Sunday, 28-Apr-2002 10:32:14 PDT

## CGI Tutorial - CGI 1.1 Reference 3

### Standard Input

For requests which have information attached after the header, such as HTTP POST or PUT, the information will be sent to the script on stdin.

The request will contain two headers, CONTENT\_TYPE and CONTENT\_LENGTH which tell the server what type and how much data will come after the headers. For method POST, the content type will always be `application/x-www-form-urlencoded`.

The data will consist of `name=value` pairs separated by the '&' character. Any special characters will be encoded using the %XX representation.

For example:

```
name=bob&name2=joe&name3=%7ESusan
```

Would have a CONTENT\_LENGTH of 33 and the %7E is an encoded '~' character.

This is the way to send large amounts of data to a script, the content length is virtually unlimited.

Text and photographs Copyright (C) 1999-2002 Bob Paauwe & Genny Paauwe  
 Sunday, 28-Apr-2002 10:32:14 PDT

# CGI Tutorial - CGI 1.1 Reference 4

## Standard Output

The script is expected to write its data to stdout. This may be a document generated by the script or instructions to the browser.

Normally, the output of the script is interpreted by the server before being sent back to the client. The advantage of this is that the script doesn't need to send the full CGI/1.1 suite of headers for every request.

Some scripts may want to avoid the overhead of the server parsing their output, and talk directly to the browser. CGI requires that the script name start with "nph-" if a script does not want the server to parse its header.

The output of scripts must begin with a small header. The header lines are in the same format as an HTTP header and are terminated by a blank line.

Any headers which are not server directives are sent directly back to the browser. Server directives are:

Content-type The MIME type of the data being sent back

Location A reference to a document (a URL), the server will issue a redirect to the browser

Status Send a status line to the browser

## Status Codes

Status codes have the form `nnn xxxxx` where `nnn` is the 3-digit status code, and `xxxxx` is the reason string.

200	OK	The request was fulfilled
201	CREATED	success, but the textual part of the response line indicates the URL by which the newly created document should be known.
202	Accepted	The request was accepted for processing but processing has not yet been completed
203	Partial Information	
204	No Response	The request was received but there is no information to send back
400	Bad Request	The request had bad syntax
401	Unauthorized	The authorization header is not valid
402	Payment Required	
403	Forbidden	The request is for something forbidden. Authorization will not help
404	Not Found	The server has not found anything matching the URL given
500	Internal Error	The server encountered an unexpected condition, usually incorrect headers
501	Not Implemented	The server does not support the facility requested
502	Service Temporarily Overloaded	
503	Gateway Timeout	

301	Moved	The data requested has been assigned a new URL, the change is permanent
302	Found	The data requested actually resided under a different URL
303	Method	Try a different method
304	Not Modified	The document has not been modified since the date and time specified in the If-Modified-Since field

Text and photographs Copyright (C) 1999-2002 Bob Pauwe & Genny Pauwe  
 Sunday, 28-Apr-2002 10:32:15 PDT

# Internet Relay Chat

## IRC Protocol

Internet Relay Chat (IRC) provides a basic, text-based method that allows multiple users to communicate interactively. TRC has its roots in interactive messaging in the bulletin board system (BBS) community.

Each IRC server hosts multiple channels of conversations, with multiple clients participating on each channel. The channels cover a variety of topics, such as investing, technical subjects, games, sports, regional issues, and language-specific subjects. In fact, you can probably find an IRC channel for just about any subject.

Channels are dynamic in nature, much more so than mailing lists or newsgroups. Given their timeliness, some virtual events are scheduled for a particular date and time so that a user can join the channel when others are connected.

In the simplest case, multiple IRC clients talk to a single IRC server. Typically, multiple IRC servers (connected via an undirected acyclic graph) form an IRC network. An IRC client connects to a single IRC server on the IRC network but can access the channels and users on the other IRC servers. Additionally, there are multiple IRC networks, such as Eris-Free Net (FFnet) and Undernet. Although it is uncommon, it is possible in some cases to connect these separate IRC networks by means of a gateway. Most people use IRC client software instead of a Telnet client to handle conversations and to help with metacommands and macros. An IRC bot, or robot, is a special kind of IRC client software that is computer-generated rather than human operated.

The IRC protocol is described in RFC 1459. This client/server protocol uses Transport Control Protocol/Internet Protocol (TCP/IP) over ports 6660-6669. It is an 8-bit protocol, with most commands using the US-ASCII text character set.

Three different communication methods are used with the IRC protocol: client to client, one to many (one to a list, to a channel, or to a host/server mask), or broadcast (to all IRC servers on that network).

The IRC protocol uses more than one dialect. For example, Undernet uses the same client/server protocol but a different server/server protocol.

Normally, text messages are sent via the IRC protocol. Sometimes, however, a dialect is used to transfer binary files from one client to another. The Direct Client to Client (DCC) dialect is one example of a dialect that is used this way.

## IRC Commands

The IRC protocol contains a variety of commands that cover a variety of categories, mainly connecting a server with another server or with a client, obtaining information about a user or a server, manipulating channels, or sending messages from a client. A few of the commands are optional and are not necessarily supported by all IRC servers and clients. Table 11-1 lists all the IRC commands

**Table 11-1 IRC Commands**

<b>Name</b>	<b>Syntax</b>
Admin	ADMIN [<Server>]
Away (optional)	AWAY [<Message>]
ChannelMode	MODE <<Channel> 1 + - I O p s t l n j b v l <Limit> <User>  [<BanMask>]
Connect	CONNECT <[TargetServer> [<Pon> [<RemoteServer>]]
Error	ERROR <Message>
Info	INFO I<Server>
Invite	INVITE <Name> <<Channel>
Ison (optional)	ISON <Name>  <Name>
Join	JOIN <<Channel> ,<Channel>  [<Key> ,<Key> ]
Kick	KICK <<Channel> ,<Channel>  <User> ,<User>  [<Comment>
Kill	KILL <Name> <<Comment>
Links	LINKS [ <Server> ] <Mask>

List	LIST [<Channel> , <Channel>  [<Server> ]
Names	NAMES [<Channel> , <Channel> ]
NickName	NICK <Name> [<HopCount> ]
Notice	NOTICE <Name> <Text>
Operator	OPER <User> <Password>
OperWall (optional)	WALLOPS <Message>
Pan	PART <Channel> {, <Channel> ]
Password	PASS <Password>
Ping	PING <Server1> [<Server2> ]
Pong	PONG <Server> [<Server2> ]
PrivateMessage	PRIVMSG <Receiver> , <Receiver>  <Message>
Quit	QUIT [<Message> ]
Rehash (optional)	REHASH
Restart (optional)	RESTART
ServerQuit	SQUIT <Server> <Comment>
Server	SERVER <Server> <HopCount> <Information>
Stats	STATS [<Query> [<Server> ]
Summon (optional)	SUMMON <User> [<Server> ]
Time	TIME [<Server> ]
Topic	TOPIC <Channel> [<Topic> ]
Trace	TRACE [<Server> ]
UserMode	MODE <Name> 1 + - i w s o
User	USER <User> <Host> <Server> <RealName>
UserHost (optional)	USERHOST <Name>  [<Name> ]
Users (optional)	USERS [<Server> ]
Version	VERSION [<Server> ]
Who	WHO [<Name> <o> ]
Whols	WHOIS [<Server> ] <NameMask> , <NameMask> , ... ]
WhoWas	WHOWAS <Name> [<Count> [<Server> ]

# 1 Other Application Layer Standards

## 1.1 Introduction

This chapter gives an overview of miscellaneous standards, which are not covered in separate chapters in this book.

## 1.2 Scalable Vector Graphics (SVG)

According to the original World Wide Web standards, documents were sent in HTML format with embedded pictures in either the JPEG or the GIF format. Both these formats are bitmapped formats, they describe a picture by a twodimensional matrix of pixels, with the color given for each pixel.

Bitmapped formats are good for photographs and paintings. But they are not good for pictures whose content can be logically described. For such pictures, they have several disadvantages:

1. Pictures get less and less sharp when you enlarge them. And pictures which look all right on the screen (typical resolution 72-96 pixels/inch) will look blurry when printed on paper (typical resolution 160-2400 pixels/inch).
2. File sizes are often unnecessarily large, which also gives longer download time.

Because of this, the World Wide Web Consortium has developed a standard named Scalable Vector Graphics (SVG). This standard is based on a logical description of the content of a picture, i.e. describing it as consisting of straight and curved lines and polygons, and areas bordered by such lines. Plug ins are available for displaying SVG in web browsers, and future web browsers may have SVG built in.

SVG uses XML to encode the logical description of a picture. For example, the following code defines an ellipse:

```
<ellipse cx="200" cy="300" rx="60" ry="30"
style="fill:none;stroke:#000099;
stroke-width:6;opacity:none" />
```

The full description of SVG can be found in [W3C 2001A] and a tutorial can be found at [Adobe 2002].

## 1.3 XML Schema

An important property of XML is that you can use XML without any syntax specification. The DTD language for syntax specification of XML is not mandatory. This also means that instead of DTD, another syntax specification language can be used. One other such syntax specification language, developed by the World Wide Web consortium, is XML Schema [W3C 2001B].

XML Schema has the following advantages compared to DTD:

1. An XML Schema is itself written in XML.
2. The format is more user friendly, avoids some cryptic encodings in DTD.
3. XML Schema gives more detailed control of the data. It is, for example, possible to define that a value should be an integer, or an integer between 1 and 100.

XML Schema has many similarities to ASN.1.

Here is an example of an XML Schema:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:simpleType name="positiveInteger">
    <xsd:Restriction base="xsd:integer">
      xsd:minInclusive value="1" /
    </xsd:Restriction>
  </xsd:simpleType>
  <xsd:complexType name = "Person" >
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="age" type="positiveInteger" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

This schema defines a data type "Person" which consists of two elements, a name as a string, and an age as a positive integer.

XML Schema defines also some non-XML formatting. For example, the following XML Schema:

```
<xsd:simpleType name="SimpleListOfIntegers">
<xsd:list itemType="myInteger" />
</xsd:simpleType>
```

specifies data which is sent like this in XML:

```
<SimpleListOfIntegers>1 2 3 5 8 13</listOfIntegers>
```

Note that the syntax separating the integers is spaces, not XML encoding. So XML Schema is in fact a standard specifying a somewhat richer standard than XML alone.

A full XML encoding of a list of integers with all syntax separators in XML format, might be:

```
<complexType name="ComplexListOfIntegers">
  <sequence>
    <integer>1</integer>
    <integer>2</integer>
    <integer>3</integer>
    <integer>5</integer>
    <integer>8</integer>
    <integer>13</integer>
  </sequence>
</complexType>
```

## 1.4 References

- [W3C 2001A] Scalable Vector Graphics (SVG) 1.0 Specification, World Wide Web Consortium, September 2001. <http://www.w3.org/SVG/>.
- [Adobe 2002] SVG Zone, Adobe 2001, <http://www.adobe.com/svg/>.
- [W3C 2001B] XML Schema. Part 0 Primer (<http://www.w3.org/TR/xmlschema-0/>), Part 1 Structures (<http://www.w3.org/TR/xmlschema-1/>), Part 2 Datatypes (<http://www.w3.org/TR/xmlschema-2/>), World Wide Web Consortium, September 2001,



# PICS: Internet Access Controls Without Censorship

© Copyright 1996 by ACM, Inc. (note 1) Appeared in *Communications of the ACM*, 1996, vol. 39(10), pp. 87-93.

Paul Resnick  
 AT&T Research  
 600 Mountain Avenue  
 Murray Hill, NJ 07974  
 presnick@research.att.com

James Miller  
 World Wide Web Consortium  
 MIT Laboratory for Computer Science  
 Room NE43-355  
 545 Technology Square  
 Cambridge, MA 02139  
 jmiller@mit.edu

With its recent explosive growth, the Internet now faces a problem inherent in all media that serve diverse audiences: not all materials are appropriate for every audience. Societies have tailored their responses to the characteristics of the media [1, 3]: in most countries, there are more restrictions on broadcasting than on the distribution of printed materials. Any rules about distribution, however, will be too restrictive from some perspectives, yet not restrictive enough from others. We can do better—we can meet diverse needs by controlling reception rather than distribution. In the TV industry, this realization has led to the V-chip, a system for blocking reception based on labels embedded in the broadcast stream.

On the Internet, we can do still better, with richer labels that reflect diverse viewpoints, and more flexible selection criteria. PICS (note 2), the Platform for Internet Content Selection, establishes Internet conventions for label formats and distribution methods, while dictating neither a labeling vocabulary nor who should pay attention to which labels. It is analogous to specifying where on a package a label should appear, and in what font it should be printed, without specifying what it should say.

The PICS conventions have caught on quickly. In early 1996, Microsoft, Netscape, SurfWatch, CyberPatrol, and other software vendors announced PICS-compatible products. AOL, AT&T WorldNet, CompuServe, and Prodigy provide free blocking software that will be PICS-compliant by the end of 1996. RSACi and SafeSurf are offering their particular labeling vocabularies through on-line servers that produce PICS-formatted labels. In May of 1996, CompuServe announced that it will label all web content it produces using PICS-formatted RSACi labels.

## Flexible Blocking

Not everyone needs to block reception of the same materials. Parents may not wish to expose their children to sexual or violent images. Businesses may want to prevent their employees from visiting recreational sites during hours of peak network usage. Governments may want to restrict reception of materials that are legal in other countries but not in their own. The "off" button (or disconnecting from the entire Net) is too crude: there should be some way to block only the inappropriate material. Appropriateness, however, is neither an objective nor a universal measure. It depends on at least three factors:

1. The supervisor: parenting styles differ, as do philosophies of management and government.
2. The recipient: what's appropriate for one fifteen year old may not be for an eight-year-old, or even all

<http://www.w3.org/PICS/iacwcv2.htm>

fifteen-year-olds.

3. The context: a game or chat room that is appropriate to access at home may be inappropriate at work or school.

Computer software can implement access controls that take into account all these factors. The basic idea, illustrated in Figure 1, is to interpose selection software between the recipient and the on-line documents. The software checks labels to determine whether to permit access to particular materials. It may permit access for some users but not others, or at some times but not others.

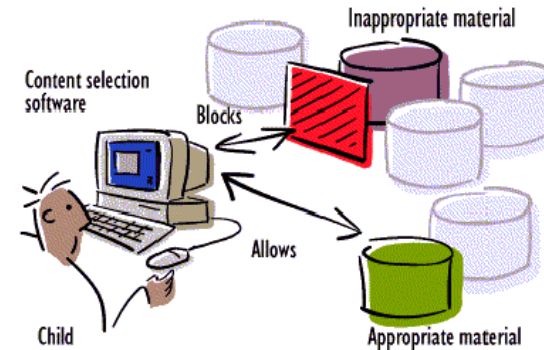


Figure 1: selection software automatically blocks access to some documents, but not others. Acknowledgment (note 3)

Prior to PICS there was no standard format for labels, so companies that wished to provide access control had to both develop the software and provide the labels. PICS provides a common format for labels, so that any PICS-compliant selection software can process any PICS-compliant label. A single site or document may have many labels, provided by different organizations. Consumers choose their selection software and their label sources (called rating services) independently, as illustrated in Figure 2. This separation allows both markets to flourish: companies that prefer to remain value-neutral can offer selection software without providing any labels; values-oriented organizations, without writing software, can create rating services that provide labels.

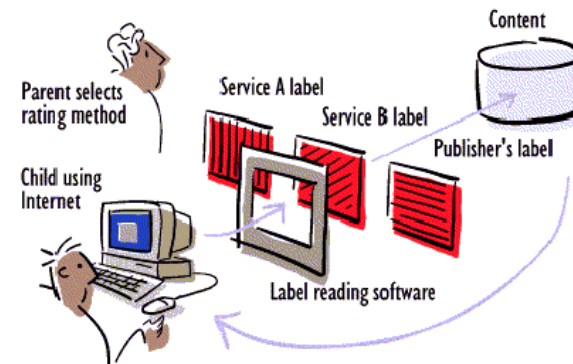


Figure 2: selection software blocks based on labels provided by publishers and third-party labeling services, and on

PICS labels describe content on one or more dimensions. It is the selection software, not the labels themselves, that determine whether access will be permitted or prohibited. For example, if a rating service used the MPAA's movie-rating vocabulary, selection software might be configured to block an eight-year-old's access to PG-labeled documents, but to allow a fifteen-year-old's access to them. Parents can prohibit access to unlabeled documents, confining children to a zone known to be acceptable, or can allow access to any document that is not explicitly prohibited.

Each rating service can choose its own labeling vocabulary. For example, Yahoo labels might include a "coolness" dimension and a subject classification dimension.

Information publishers can self-label, just as manufacturers of children's toys currently label products with text such as, "Fun for ages 5 and up." Provided that publishers agree on a common labeling vocabulary, self-labeling is a simple mechanism well-matched to the distributed nature and high volume of information creation on the Internet.

When publishers are unwilling to participate, or can't be trusted to participate honestly, independent organizations can provide third-party labels. For example, the Simon Wiesenthal Center, which is concerned about Nazi propaganda and other hate speech, could label materials that are historically inaccurate or promote hate. Third-party labeling systems can also express features that are of concern to a limited audience. For example, a teacher might label a set of astronomical photographs and block access to everything else for the duration of a science lesson.

There are two PICS specification documents [6, 8]. The most important components are:

1. A syntax for describing a rating service, so that computer programs can present the service and its labels to users.
2. A syntax for labels, so that computer programs can process them. A label describes either a single document or a group of documents (e.g., a site.) A label may be digitally signed and may include a cryptographic hash of the associated document.
3. An embedding of labels (actually, lists of labels) into the RFC-822 transmission format [2] and the HTML document format.
4. An extension of the HTTP protocol, so clients can request that labels be transmitted with a document.
5. A query-syntax for an on-line database of labels (a label bureau.)

The accompanying [sidebar](#) illustrates these formats and protocols. Four technical features are worth highlighting.

First, the machine-readable service description is a resource that other computer programs can use for automatically generating interfaces that present the service to users. Consider the prototype shown in Figure 3, for configuring selection software. Here the parent is setting rules for what Johnny can visit, based on a rating service which has separate dimensions for language, nudity/sex, and violence. The parent drags the slider to indicate the maximum permitted value on the violence scale, noting the height of the thermometer and the text description (e.g., "Strong, vulgar language...") associated with each level on the scale. The software has taken the thermometer icons and text directly from the service description.

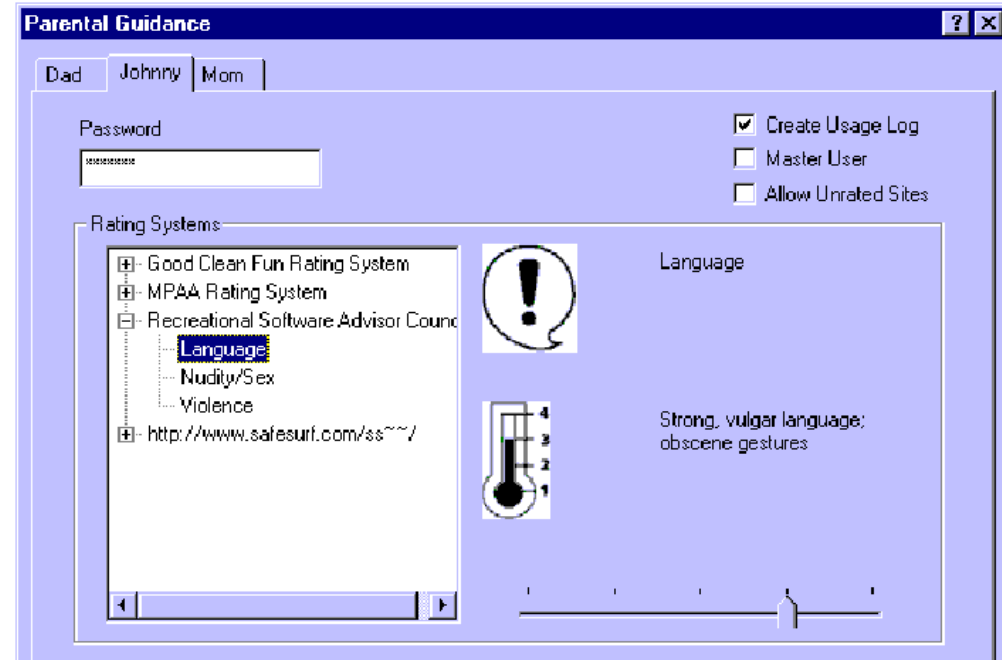


Figure 3: [Prototype software \(note 4\)](#) draws on text and icons in the service description to automatically generate a user interface for configuring selection rules.

Second, a rating service can provide variants of its service description tailored to different languages and cultures. The core elements remain the same, but the text and icons can be different. As a result, the service need not provide multiple versions of labels. The labels rely only on the common, core elements, but, using the variants of the service description, a single label can be displayed to different users in different languages.

Third, we have used URLs wherever universally distinct identifiers are required. For example, the identifier for a rating service is a URL. This has two advantages. First, a URL is a self-describing identifier, because it can be used to retrieve a descriptive document. Second, it leverages the Internet domain name registration system to permit decentralized choice of the identifier, while still guaranteeing distinctness from identifiers chosen by others.

Fourth, we specify that a response to a request for multiple labels must preserve the order of the request. If the server knows several alternative URLs that identify a single document, and the client asks for a label for one of those URLs, the server can send back a label for one of the alternates. The client can still match the label with its original request, from its position in the response, even though the document URLs do not match.

## What PICS Doesn't Specify

In general, PICS specifies only those technical issues that affect interoperability. It does not specify how selection software or rating services work, just how they work together.

browser on each computer, as announced by Microsoft and Netscape. A second method-one used in products such as CyberPatrol and SurfWatch-is to perform this operation as part of each computer's network protocol stack. A third possibility is to perform the operation somewhere in the network, for example at a proxy server used in combination with a firewall. Each alternative affects efficiency, ease of use, and security. For example, a browser could include nice interface features such as graying out blocked links, but it would be fairly easy for a child to install a different browser and bypass the selective blocking. The network implementation may be the most secure, but could create a performance bottleneck if not implemented carefully.

PICS does not specify how parents or other supervisors set configuration rules. One possibility is to provide a configuration tool like that shown in Figure 3. Even that amount of configuration may be too complex, however. Another possibility is for organizations and on-line services to provide preconfigured sets of selection rules. For example, an on-line service might team up with UNICEF to offer "Internet for kids" and "Internet for teens" packages, containing not only preconfigured selection rules, but also a default home page provided by UNICEF.

Labels can be retrieved in various ways. Some clients might choose to request labels each time a user tries to access a document. Others might cache frequently requested labels or download a large set from a label bureau and keep a local database, to minimize delays while labels are retrieved.

PICS specifies very little about how to run a labeling service, beyond the format of the service description and the labels. Rating services must make the following choices:

1. The labeling vocabulary. A common set of dimensions would make publishers' self-labels more useful to consumers but cultural divergence may make it difficult to arrive at a single set of dimensions. Governments may also mandate country-specific vocabularies. Third party labelers are likely to use a wide range of other dimensions.
2. Granularity. Services can label entire sites, or individual documents and images.
3. Who creates the labels. Services can employ professionals, volunteers, or computers to do the labeling. They can also delegate all or part of the label-creation task to content creators or to other rating services.
4. Coverage. Some services may strive for comprehensive coverage of the entire Internet, others for narrower areas such as pornography or educational sites. An interesting intermediate offering may be to label the documents that subscribers ask about: while there are thousands of sites and millions of documents available on the Internet, any particular set of users is likely to ask for access to a much smaller set.
5. Revenue generation. Some organizations that provide labels may choose not to charge anyone, relying on donations or levies on members. Other services can charge subscribers, charge intermediaries such as on-line services for the right to redistribute labels, or charge sites for the privilege of being labeled. We might even see the rise of labeling intermediaries who pay a royalty to values-oriented organizations such as UNICEF for the right to label documents with the UNICEF logo, according to criteria set by UNICEF.

## Other Uses for Labels

New infrastructures are often used in unplanned ways, to meet latent needs. There will be many labeling vocabularies that are unrelated to access controls. The PICS specifications also plan for unplanned uses, by including extension mechanisms for adding new functionality. PICS is a new resource available to anyone who wishes to associate data with documents on the Internet, even documents that others control. Some of the promising applications include:

1. Collaborative labeling services could permit everyone to contribute labels, and use those labels to guide others toward interesting materials [4, 7]. Guidance can be personalized by matching end-users with others who have similar tastes, as reflected in their ratings of documents that both have examined [5, 10, 12].
2. On-line journals could publish all submissions, but attach review labels that each reader could interpret as guides to the best articles [9]. While PICS-compatible labeling services can associate text phrases or icons with values on numeric scales, so that a frequently used annotation such as "seminal article" can be encoded, PICS labels can not

include arbitrary text. A PICS label can, however, include the URL of another document that contains textual annotations, which provides a means of integrating PICS with more general annotation platforms such as ComMentor [11].

3. Labeling vocabularies may be designed for classification rather than blocking, coupled with indexing engines that search based on labels and with browsers that display them.
4. Intellectual property vocabularies may develop for notifying people about who owns a document and how it may be copied and used. Of course, this is only one piece of the intellectual property protection puzzle, since it offers notification but not enforcement.
5. Privacy vocabularies may develop. End-users could express their privacy preferences and labels would notify them of what information is gathered about their interactions with a web site, and how that information will be used.
6. Reputation vocabularies may develop. The Better Business Bureau could associate labels with commercial sites that had especially good or especially bad business practices. Privacy groups could label sites according to their information practices. There could even be labels for Usenet authors according to the quality of the messages they post; posts from those with poor reputations could be screened out.

## Conclusion

PICS provides a labeling infrastructure for the Internet. It is values-neutral: it can accommodate any set of labeling dimensions, and any criteria for assigning labels. Any PICS-compatible software can interpret labels from any source, because each source provides a machine-readable description of its labeling dimensions.

Around the world, governments are considering restrictions on on-line content. Since children differ, contexts of use differ, and values differ, blanket restrictions on distribution can never meet everyone's needs. Selection software can meet diverse needs, by blocking reception, and labels are the raw materials for implementing context-specific selection criteria. The availability of large quantities of labels will also lead to new sorting, searching, filtering, and organizing tools that help users surf the Internet more efficiently.

## References

- [1] J. Berman and D. Weitzner, "User Control: Renewing the Democratic Heart of the First Amendment in the Age of Interactive Media," *Yale Law Journal*, vol. 104, pp. 1619, 1995.
- [2] D. Crocker, "RFC-822: Standard for the Format of ARPA Internet Text Messages," <http://ds.internic.net/rfc/rfc822.txt>, August 1982.
- [3] I. de Sola Poole, *Technologies of Freedom*. Cambridge: MIT Press, 1983.
- [4] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry, "Using Collaborative Filtering to Weave an Information Tapestry," *Communications of the ACM*, vol. 35, pp. 61-70, 1992.
- [5] W. Hill, L. Stead, and M. Rosenstein, "Recommending and Evaluating Choices in a Virtual Community of Use," Proceedings of CHI 95 Conference on Human Factors in Computing Systems, Denver: ACM. 194-201.
- [6] T. Krauskopf, J. Miller, P. Resnick, and G. W. Treese, "Label Syntax and Communication Protocols," World Wide Web Consortium <http://w3.org/PICS/labels.html>, May 5 1996.
- [7] D. Maltz and K. Ehrlich, "Pointing the Way: Active Collaborative Filtering," Proceedings of CHI 95 Conference on Human Factors in Computing Systems, Denver: ACM. 202-209.



[8] J. Miller, P. Resnick, and D. Singer, "Rating Services and Rating Systems (and Their Machine Readable Descriptions)," World Wide Web Consortium <http://w3.org/PICS/services.html>, May 5 1996.

[9] A. M. Odlyzko, "Tragic Loss or Good Riddance? The Impending Demise of Traditional Scholarly Journals," *International Journal of Human-Computer Studies*, vol. 42, pp. 71-122, 1995.

[10] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "GroupLens: An Open Architecture for Collaborative Filtering of Netnews," Proceedings of CSCW 94 Conference on Computer Supported Cooperative Work, New York: ACM. 175-186.

[11] M. Roscheisen, C. Mogensen, and T. Winograd, "A Platform for Third-Party Value-Added Information Providers: Architecture, Protocols, and Usage Examples," Stanford University CSDTR/DLTR (<http://www-diglib.stanford.edu/diglib/pub/reports/commentor.html>), November 1994, updated April 1995 1995.

[12] U. Shardanand and P. Maes, "Social Information Filtering: Algorithms for Automating "Word of Mouth"," Proceedings of CHI 95 Conference of Human Factors in Computing Systems, Denver: ACM. 210-217.

## A Tour of the PICS Specifications

Figure 4 shows the description of a sample rating service, based on the MPAA's movie-rating scheme. The initial section includes general information about the service. The second section describes each of the dimensions, or categories, and the scales used for each. In this case, there is just a single category, with five possible values: G through NC-17. In actual labels, these values would be represented by the integers 0-4; the service description allows a software program to determine that a value of 1 corresponds to the PG rating and even to display the PG.gif icon to a user.

```
((PICS-version 1.1)
(rating-system "http://MPAAAscale.org/Ratings/Description/")
(rating-service "http://MPAAAscale.org/v1.0")
(icon "icons/MPAAAscale.gif")
(name "The MPAA's Movie-rating Service")
(description "A rating service based on the MPAA's movie-rating scale")

(category
(transmit-as "r")
(name "Rating")
(label (name "G") (value 0) (icon "icons/G.gif"))
(label (name "PG") (value 1) (icon "icons/PG.gif"))
(label (name "PG-13") (value 2) (icon "icons/PG-13.gif"))
(label (name "R") (value 3) (icon "icons/R.gif"))
(label (name "NC-17") (value 4) (icon "icons/NC-17.gif"))))
```

Figure 4: A PICS-compatible description of a service that is based on the MPAA movie rating scheme.

Figure 5 shows a sample PICS label (actually a label list containing just one label.) The URL on the first line, which identifies the labeling service, makes it possible to redistribute labels yet still identify their original sources. The label can also include information about itself, such as the date on which it was created, the date it will expire, that the label is associated with a certain document (in this case, "<http://www.gcf.org/stuff.html>"), and the label's author. The last line shows the attributes that describe the document: a "language" value of 3; "sex" 2; and "violence" 0.

```
(PICS-1.1 "http://old.rsac.org/v1.0/" labels
on "1994.11.05T08:15-0500"
until "1995.12.31T23:59-0000"
for "http://www.gcf.org/stuff.html"
```

```
by "John Doe"
ratings (1 3 s 2 v 0))
```

Figure 5: A sample label list from the service described in Figure 4.

Anything that can be named by a URL can be labeled, including documents that are accessed via FTP, gopher, or Netnews, as well as HTTP. PICS proposes a URL naming system for IRC, so that chat rooms with stable topics can be labeled.

Labels can include two optional security features (not shown in the example.) The first is a cryptographic hash of the labeled document, in the form of an MD5 message digest. This enables software to detect whether changes have been made to the document after the label was created. The second is a digital signature on the contents of the label itself, which allows software to verify that a label really was created by the service mentioned in it and that the label has not been altered.

PICS specifies three ways to distribute labels. The first is to embed labels in HTML documents, using the META element in the document header. The general format is `<META http-equiv="PICS-Label" content='labelist'>`. Other document formats could be similarly extended.

The second distribution method is for a client to ask an HTTP server to send labels along with the documents it requests. Figure 6 shows a sample interaction: the HTTP GET request includes an extra header line asking for labels and saying which service's labels should be sent back. The server includes two extra header lines in the response, one of which contains the labels.

**Client sends to HTTP server [www.greatdocs.com](http://www.greatdocs.com):**

```
GET foo.html HTTP/1.1
Accept-Protocol: {PICS-1.0 {params full {services "http://www.gcf.org/1.0/"}}}
```

**Server responds to client:**

```
HTTP/1.1 200 OKDate: Thursday, 30-Jun-95 17:51:47 GMT
MIME-version: 1.0
Last-modified: Thursday, 29-Jun-95 17:51:47 GMT
Protocol: {PICS-1.0 {headers PICS-Label}}
PICS-Label: ...label here...
Content-type: text/html
...contents of foo.html...
```

Figure 6: Method 2- requesting a document and associated label from an http server.

The third way to distribute labels is through a label bureau that dispenses only labels. A bureau can distribute labels created by one or more services. This separation of labels from content allows third-party labeling even when the publishers do not wish to distribute the labels: for example, the Simon Wiesenthal Center can label hate speech without the cooperation of neo-Nazi groups.

A label bureau is implemented as an HTTP server that accepts URL query strings in a special format. Suppose a label bureau is available at <http://www.labels.org/Ratings>. A client interested in a label for the document <http://www.questionable.org/images> would send the request shown in Figure 7 to the server at [www.labels.org](http://www.labels.org).

```
GET /Ratings?opt=generic&
u="http%3A%2F%2Fwww.questionable.org%2Fimages"&
s="http%3A%2F%2Fwww.gcf.org%2Fv2.5"
HTTP/1.0
```

Figure 7: Method 3- requesting a label from a label bureau, separately from the document the label refer to. Note that inside a URL query string it is necessary to encode : as %3A and / as %2F.

## Notes

1. This article has been accepted for publication in Communications of the ACM. Copyright c 1996 by the Association for Computing Machinery, Inc. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Publications Dept, ACM Inc., fax +1 (212) 869-0481, or [permissions@acm.org](mailto:permissions@acm.org).

2. PICS is an effort of the World Wide Web Consortium at MIT's Laboratory for Computer Science, drawing on the resources of a broad cross-section of the industry. Project history, a long list of supporting organizations, and details of the specifications may be found at <http://w3.org/PICS>.

3. Thanks to Netscape for providing Figures 1 and 2.

4. Figure 3 was generated in November of 1995, using prototype software written by MIT student Jason Thomas. Source code for this prototype, and other reference software are available from the [PICS home page](#). Since then, RSAC has created a new rating system for the Internet (RSACi), which separates nudity and sex as separate dimensions. Several companies have announced products that, like this prototype, can read any PICS service description and generate a user interface with similar features to the one shown in Figure 3.

## Chapter 2

# The organisation of information in the Directory

### 2.1 INTRODUCTION

This chapter looks at the models of the information that is held in the Directory. The Directory as a whole is quite complex, and difficult to understand. As an aid to understanding what the information looks like, and how it is distributed and managed, various models are described in the Standard. Each model presents a simplified view of just one aspect of the Directory information. This makes it easier to understand, since we only need to grapple with the concepts of one model at a time. We can then try to link the models together in our mind, in order to get a clearer picture of the Directory as a whole.

One model gives a view of the Directory information, as it is seen by a typical user. (This was, in fact, the only information model described in the '88 edition of the Standard.) This model, the Directory User Information Model, simply called the Directory Information Model in the '88 Standard, does not recognise that the Directory is distributed. From its perspective, there is a large amount of information held in the Directory, and users can access all of it, providing that they have the appropriate access rights. To enable the correct information to be accessed, the users need a way of identifying the items they want. Consequently, chunks of information are given globally unambiguous names. The way in which Directory information is identified, or named, is described in this model. The model is sufficiently general to allow any sort of information to be held in the Directory, and the Standard does not mandate what should be held there. However, if implementations of X.500 are to be globally, or simply widely, accessible, then it is necessary to have a subset of this information internationally standardised, so that implementations from Mongolia to Massachusetts can understand it. The Standard does therefore define a subset of the types of user information that can be held in the Directory, in order to aid the international use of X.500.

The user information needs to be managed and administered by local administrators. Human administrators will need to store management information in the Directory, that controls aspects of the user information. Examples of these controls are: the sort of user information that can be stored in this part of the Directory, and who can access it. The Directory Operational and Administrative Information Model provides the 'administrator's view' of the information stored in the Directory. Administrators 'see' that there is more information stored in the Directory, than do typical users. This model still presents the Directory as a global information base, and the distribution of the information between computer systems is not visible to the model. A certain amount of the administrative information is defined in the Standard. This is regarded as the minimum needed for administrators to manage their portion of the Directory information, in an open environment. Specific implementations may provide administrators with additional non-standard control information. This is not restricted by the Standard.

The Directory User Information Model, and the Directory Operational and Administrative Information Model, collectively make up the Directory Information Models. Directory Information, refers to the Directory from a (single) global perspective, and distribution of the information between computer systems is not relevant. In other words, the Directory information is viewed as if it were held in a single centralised database.

Another model is concerned with how the information is distributed between the different computer systems (DSAs § 4.1.2) that provide the Directory service. This is the DSA Information Model. The DSA Information Model describes a model of the information that needs to be held by a single computer system, in order for it to co-operate with others in providing a service to users of the Directory. It is a 'DSA's view' of the information stored in the Directory. Some of this information will be User Information, some of it will be Administrative Information, and some of it will be its own operational information. As an example of the latter, a system will need to store information about where other parts of the User Information are stored. In this way a DSA can relate its bit of information to the global Directory information.

A global database obviously is not controlled by one central authority, but rather by lots of independent authorities. The way in which the management and administration of the global database is distributed between these independent authorities is explained in the Directory Administrative Authority Model. Each independent authority will be responsible for the information stored in its area of the Directory. The way in which this authority may be delegated to sub-authorities is explained. So is the way in which a clean transition is made between two independent authorities. The Administrative Authority Model is described in § 2.11.

### 2.2 OBJECTS AND ENTRIES

The complete set of all information held in the Directory is known as the **Directory Information Base (DIB)**. As described above, not all of this information is visible to normal users of the Directory. Also, when we talk about Directory information, we are not concerned with the distribution of the information between different computer systems.

The DIB consists of entries, and these entries are related hierarchically to each other. The precise form of this hierarchy is described in § 2.4 and § 2.5. Entries are the basic building blocks of the database.

In the Directory User Information Model, an entry holds information about an object of interest to users of the Directory. These (Directory) objects might typically be associated with, or be some facet of, real world things such as information processing systems or telecommunications equipment or people. So there can be a Directory entry for an X.400 Message Transfer Agent (§ 10.2), and another one for the manager. However, it is very important to note that Directory objects do not necessarily have a one-to-one correspondence to real world things. This has typically caused a lot of confusion to non-experts, many of whom assume that every entry in the Directory contains all the relevant information about one real world thing. This is not necessarily so. Directory objects, and hence entries, can have a one-to-one correspondence with real world things, or can have a many-to-one or one-to-many relationship with real world things. For example, a Directory object/entry may be a mailing list containing the names of many real people (one-to-many correspondence). Alternatively, a real person may be

represented in the Directory as both a residential person object/entry and an organisational person object/entry (many-to-one correspondence). In the latter case, the organisational person Directory entry would hold information that is relevant to describing the person in their working environment, holding their office room number, internal telephone extension number, electronic mail address, and the department etc., the residential person Directory entry would describe the person in their residential capacity, holding their home postal address and home telephone number etc.

Objects that have similar characteristics are identified by their **object class**. Every object entry in the Directory is a member of at least one object class. So, for example, there is an 'organisational person' object class for organisational person entries, and a 'residential person' object class for residential person entries. Object classes are described more fully in Chapter 3.

#### Fig. 2.1 The Directory (user) information model

## 2.3 ATTRIBUTES

Each piece of information that describes some aspect of an entry is called an **attribute**. An attribute comprises an **attribute type** and one or more **attribute values**. An example of an attribute type might be 'telephone number' and an example of a telephone number attribute value might be '+44 61 745 5000'. This is the extent of the information model described in the '88 edition of the Standard, and this is shown pictorially in Fig. 2.1. (The model also describes how entries are named, and the distinguished attribute value will be described in § 2.4 and § 2.5.)

The Standard recognises that users (or their administrators, to be more precise) of the Directory will want to define their own attribute types, and that the number of different attribute types held in the Directory will be almost boundless. However, there will be a set of attribute types that are commonly usable by most, if not all, users of the Directory. It would be sensible to standardise this set, so that different users in different parts of the DIB are not referring to exactly the same type of information but using different identifiers. Part 6 of the Standard defines a commonly usable set of user attribute types, and a selection of these are shown in Table 3.2.

## 2.4 THE STRUCTURE OF THE DIB

Entries held in the DIB are structured in a hierarchical manner, using a tree structure, which is similar to a structure chart used by most hierarchical organisations. The DIB can therefore be represented as a **Directory Information Tree (DIT)**, in which each node in the tree represents a Directory entry (Fig. 2.2). Many entries have superior and subordinate entries, and these are called non-leaf nodes or non-leaf entries. Entries which only have superior entries, and no subordinate entries, are called leaf nodes or leaf entries. At the apex of the DIT is the root entry, and this is the only entry without a superior. (In an organisational chart it might be the Managing Director or President who has no superior line manager.) Below the root of the DIT there will typically be entries representing countries or multinational organisations, and below these, entries representing organisations and organisational units (divisions) respectively. (Below the President or MD in an organisational structure chart might be vice-presidents or divisional managers of functional areas such as sales and production, and below these various

line managers and other employees.) Minor differences between an organisational structure chart and the DIT structure are in how layers of the structure are constructed and named. It is recognised, for example, that the root DIT entry will not actually exist in the real world Directory (who will manage it?, who owns it?), whereas the Head of an Organisation always exists! Consequently, users are not allowed to read the root DIT entry, and it will not contain any user attributes. It only occurs in the DIT model for completeness. Within an organisational structure chart, each box is usually named with the role of the occupant, e.g. Chairman, secretary, Northern Sales Manager, Requisition Clerk, and often contains the name of the person currently occupying that position. The layer below identifies the staff reporting to this person. In the DIT, each layer represents objects contained in or associated with the superior node (e.g. organisations are based in countries, and organisational units are contained within organisations). The naming of the entries in the DIT takes a different form from that of organisational structure charts, and this is described below.

#### Fig. 2.2 The structure of Directory entries

It will often be a relatively simple task to map an organisation's structure into part of the DIT. However, organisations do not have to rigidly follow their organisational structure when designing their DIT structure. They may opt for a flatter structure in the DIT, so that browsers cannot determine the structure of the organisation. Also, as described in § 8.6, access controls can be used to prohibit browsers from discovering the structure of an organisation's DIT subtree.

## 2.5 NAMING ENTRIES

Each object entry known to the Directory is distinguished from all other objects by its name. Thus each entry is said to have a **distinguished name (DN)**. Each entry takes the name of its **parent** (or superior) entry in the DIT, and has appended to this a **relative distinguished name (RDN)**. An entry's RDN uniquely identifies it from all of its peers (Fig. 2.3). Since each RDN is guaranteed to be unique below any particular non-leaf node, each entry is guaranteed to have a unique DN. There could thus be an organisation called ACME Tools in the US, and a completely unrelated company bearing the same name in the UK. Both would still have unique distinguished names, since each is prefixed by a different country name.

A RDN is syntactically a set of attribute type and value pairs, enabling an entry to have a multi-component RDN (as in Fig. 2.3 with the Sales unit in Swindon). However, in the vast majority of real DIT entries to date, RDNs usually only contain one attribute type and value pair. This is very likely to continue to be the case. Only in exceptional circumstances will administrators prefer or need to use more than one attribute value in a RDN.

#### Fig. 2.3 A hypothetical DIT, showing the relationship of RDNs to distinguished names

An entry's RDN is held in the entry along with all the other attributes. Those attribute values which form part of the RDN are flagged as being **distinguished** by the DSA software. This explains why Fig. 2.1 contains a distinguished value box. However, users do not have to be told which are the distinguished values, unless they have permission to read them (Chapters 5 and

8). So it is not possible for a user to surreptitiously determine the RDN of an entry from its attributes.

**Note.** Conforming '88 DSAs do have to return the distinguished name of an entry, in reply to Read operations (and also to Compare, List and Search if an alias name was used by the user). This restriction has been dropped from the '93 version of the Standard.

The only entry allowed to have a null distinguished name and RDN is the root entry. However, as stated before, it only occurs in the DIT model for completeness.

## 2.6 ALIASES

It is often convenient for an object to be known by more than one name, depending upon its role or referenced context. To allow for this, the concept of aliases has been introduced into the Directory model of names. Each Directory object only has one distinguished name, and therefore one entry in the DIB, but it may have one or more alternate alias names (with corresponding alias entries in the DIB). The name of an entry therefore only has to be **unambiguous**, i.e. given the name you will get to precisely one entry in the DIT, but it does not have to be **unique**, i.e. the only name that will identify the entry.

### Fig. 2.4 An alias entry

Each alias name has a corresponding alias entry in the DIT. This contains a pointer to the object entry, as shown in Fig. 2.4. (The pointer is actually the name - in the '88 edition it has to be a distinguished name - of the object entry which it points to. The pointer, like all other information, is held as an attribute - the **aliased object name** attribute - of the alias entry.) Alias entries have a special object class of alias (§ 3.5), to distinguish them from object entries. All alias entries are leaf entries in the DIT.

As a real life example, Salford University owns a business known as Salford University Business Services (SUBS), which is an organisation registered at Companies House in the UK. It has the distinguished name {Country = GB, Organisation = Salford University Business Services Ltd}. However, people may think that the business is a department within the University, since it is located next to the university campus. Therefore, an alias has been created below the University of Salford entry, with the (distinguished) name {Country = GB, Organisation = Salford University, Organisational Unit = Salford University Business Services}. This alias points to the real Business Services entry in the DIT. Anyone referencing SUBS via the alias name in the University will be automatically re-routed to the real Business Services entry by the Directory, through a process termed **alias dereferencing** (§ 2.7).

In the '88 edition of the Standard, an alias may not point to another alias (w/w 2.2) but in the '93 edition of the Standard this restriction has been dropped. The feature may be useful in the following situation. A person moves between divisions within an organisation. The original distinguished name is replaced by an alias which points to the new distinguished name in the new division. The person is thus still contactable by either their old or their new name. Suppose the person moves again within a very short timescale (some organisations re-organise frequently!). A second alias could be inserted into the DIT, replacing their previous 'new' name, and pointing to their new 'new' name. The person can now be identified by all three names. This second alias insertion is not allowed by the '88 edition of the Standard.

## 2.7 PURPORTED NAMES, NAME RESOLUTION AND ALIAS DEREFERENCING

A directory user will generally give a 'purported' name as input to a directory query. A purported name is syntactically a name (i.e. a sequence of RDNs) but it may or may not name an actual entry in the DIT. Name Resolution is a procedure carried out by the Directory, which determines if the purported name is valid, i.e. if it unambiguously identifies a single entry in the DIT. Logically, name resolution is the process of sequentially matching each RDN in a purported name to an arc in the DIT, beginning at the root and working downwards in the DIT. If an alias is encountered, this is dereferenced and name resolution recommences again from the root, with the name pointed to in the alias entry.

For example, by reference to Fig. 2.4, suppose a user quoted the purported name {C=GB, O=University of Salford, OU=Salford University Business Services, OU=Software Services, CN=Dave Morgan}. Name resolution would start at the root, and would eventually reach the alias entry {C=GB, O=University of Salford, OU=Salford University Business Services}. Alias dereferencing would replace that part of the purported name that had already been used, by the 'aliased object name' value in the alias. The purported name would thus become {C=GB, O=Salford University Business Service Ltd, OU=Software Services, CN=Dave Morgan}. Name resolution would now start again from the root, with this new name.

A description of how name resolution and alias dereferencing work in the distributed directory, is given later in § 4.5, § 9.13.6 and Appendix B.

## 2.8 COLLECTIVE ATTRIBUTES

It was soon realised after 1988 that the above information model was too simplistic. There need to be different types of information (attributes) stored in the Directory. For example, some user attributes could apply to a whole series of entries, e.g. all the employees of an organisation may be contactable on the same telephone number, with a switchboard operator locally routing the call to an internal extension number. It would be very inefficient to have to repeat the organisation's telephone number attribute in the entries of all the employees. Ideally, the telephone number should be stored once in the Directory, along with an indication of the entries to which it applies. Then when a user reads an employee's entry, to which this shared attribute applies, it would be returned along with the other user attributes of the entry. This new type of information is allowed in the '93 edition of the Standard, and is called a **collective** attribute.

Collective attributes are a special type of user attribute, which have the above properties. They are part of the '93 Directory User Information Model. They are read from entries like other user attributes, but they cannot be updated like them. This is because they are not actually stored in the entry. They are stored in a special entry, called a **subentry** (§ 2.12). Subentries are not visible in the User Information Model. They are normally only visible to administrators. A subentry contains a description of the DIT subtree of entries to which its collective attributes apply. Using the example above, there could be a subentry associated with the organisation's entry, which describes the complete subtree below the organisation's entry. This subentry would hold the organisation's collective 'telephone number' attribute. This collective attribute would then apply to all the entries of the employees of the organisation.

## 2.9 THE DIRECTORY OPERATIONAL AND ADMINISTRATIVE INFORMATION MODEL

There are also other types of information (attributes) that need to be stored in the Directory. Whilst user attributes are typically set and used by users of the Directory, they are transparent to the operation of the Directory system - telephone number is a good example of this. Their presence or not in the DIB does not affect the actual operation of the Directory. The system will still function perfectly well without them (although their absence may cause considerable frustration to one or more users of the Directory). In the '93 edition of the Standard, these types of attribute are called **user** attributes. It was decided that the attributes described in the '88 edition of the Standard most nearly match this definition, and so all '88 defined attributes have been re-classified in the '93 Standard as user attributes (but see w/w 2.1).

Other attributes however, are related to the operation of the Directory itself. Some might be set by the software, in order to monitor or control the system, e.g. the date-and-time an entry was last modified. Others might be set by administrators or users of the Directory, and be used by the software in order to correctly process users' requests. Access Control attributes are examples of this type. These attributes are called **Directory operational** attributes. They are part of the Directory Administrative and Operational Information Model.

A major property of Directory operational attributes is that they are not normally visible to ordinary users of the Directory. This means that if a user wishes to read all the attributes in an entry (assuming they have the required permissions - see Chapter 8), a simple Read request (§ 5.6) asking for all attributes will not return any operational attributes in the result. (Only user attributes are returned.) In order to be accessed, operational attributes have to either be specifically named in the Read request, or selected via a '93 extension parameter **Extra Attributes**. Table 3.5 lists the Directory operational attributes that have been standardised.

Besides these 'hidden' attributes - hidden to normal users, that is - there are also some 'hidden' entries, that are usually only visible to administrators. These hidden entries are called **subentries**. Subentries mainly hold operational and administrative information. A fuller description of subentries is given in § 2.12.

Whether an attribute is user/collective or operational is specified at the time that it is defined. It cannot change definitions during its lifetime.

## 2.10 ATTRIBUTE HIERARCHIES

Another new concept introduced in the '93 standard is that of attribute hierarchies. This acknowledges the fact that some attribute types may be very similar to each other, and are in fact refinements of a more general type. For example, telephone number is more generic than office telephone number, which is more generic than direct dial office telephone number. A more general attribute type is said to be a **supertype** of a more specific attribute type. Conversely, a more specific attribute type is said to be a **subtype** of a more generic attribute type. When accessing the Directory for a particular attribute type, the Directory will return any values of that type, and of any of its subtypes, that it finds. Thus if an entry had all three types of telephone number attribute present, and a user requested the values of the telephone number attribute, the values of all three attribute types would be returned. If a user requested values of

the office telephone number attribute, then these values, and those of the direct dial office telephone number would be returned. The user is always able to tell which type of attribute is returned, e.g. which is the direct dial number, since the actual attribute (sub)type is returned in the answer along with the values.

## 2.11 DIRECTORY ADMINISTRATIVE AUTHORITY MODEL

A significant enhancement to the '88 standard is the addition of an Administrative Authority Model. This recognises that different parts of the DIT will be administered, or managed, by different organisations. At some point in the DIT, there will be a complete break between one administration and another, whilst at other points in the DIT there will be a devolving of some power onto sub-authorities. An example of the former might be between the administrator of a country entry, and the administrator of an immediately subordinate organisational entry. (A country's entry (name) may be administered by the national PTT or the national standards body, since these are the official representatives of the country at the ITU-T and ISO. An organisation's entry may be administered by the organisation.) An example of the latter might be the delegation of some authority within an organisation to the administrator of an organisational unit. Both of these are catered for in the administrative model, by allowing the DIT to be divided into subtrees, that are controlled, or administered, by different authorities. Two types of subtree are defined: autonomous administrative areas (AAA), and inner administrative areas. Autonomous administrative areas start at an entry in the DIT, called an **autonomous administrative point** (AAP), and continue downwards until either leaves or other autonomous administrative points are encountered (Fig. 2.5). Applying Fig. 2.5 to the USA, the first AAP under the DIT root could be the {C=US} entry. Below this could be entries for each of the 50 States. These might be administered jointly by the national carriers such as AT&T and MCI, since they represent the US at ITU-T. Below these would be entries for American organisations. Each organisation would be completely responsible for administering their part of the DIT, i.e. their organisation's entry and all the entries below it (although they could always sub-contract this to a service provider).

### Fig. 2.5 Autonomous administrative areas

Inner administrative areas start at an entry in the DIT - an inner administrative point, or IAP - that is within an autonomous administrative area. (IAPs are always subordinate to AAPs.) Inner administrative areas continue downwards until the end of the autonomous administrative area is reached. Entries within an inner administrative area are still under the overall control of the autonomous administrative authority, but some degree of control is also exercised over them by the administrator of the inner administrative area. Inner administrative areas may be nested (Fig. 2.6), but the bottom of all inner areas is always the bottom of the enclosing AAA (actually the specific area, § 2.11.1). The fact that inner areas always go to the bottom of the enclosing specific area has caused confusion at the Standards meetings, even amongst the experts. This is because areas are always implicitly defined - they only stop when a leaf or a new specific administrative point (administrative entry) is reached. However, collections of entries within an area may be explicitly defined (§ 2.12.1), and these can always stop before the bottom of the area is reached.

## **Fig. 2.6 Inner and autonomous administrative areas**

### **2.11.1 Specific Administrative Areas**

According to the Standard, administrative areas control three specific aspects of administration. In other words, the administrative authority for an administrative area can operate in three different roles.

1. **Subschema administration** is concerned with administering the Directory subschema that is in operation in this part of the DIT (Chapter 3 gives a fuller description of the schema).
2. **Access control administration** is concerned with administering the security policy that is in force in this part of the DIT (Chapters 7 and 8 give a fuller description of security).
3. **Collective attribute administration** is concerned with administering the collective attributes in this part of the DIT (§ 3.11.1).

An AAA can therefore be viewed from each of these three aspects, the view revealing either a subschema specific administrative area, an access control specific administrative area or a collective attribute specific administrative area. In order to visualise this, try to imagine that an AAA subtree has been sliced horizontally into three layers, i.e. an access control specific administrative area sandwiched between a subschema specific administrative area and a collective attribute specific administrative area, see Fig. 2.7.

#### **Fig. 2.7 Autonomous administrative area**

#### **Fig 2.7a Subschema administrative area**

#### **Fig 2.7b Access control administrative areas**

#### **Fig 2.7c Collective attribute inner administrative**

**areas**<http://www.isi.salford.ac.uk/staff/dwc/Version.Web/Chapter.2/fig2-7.gif>

For each specific aspect of administration, the AAA can be partitioned into one or more specific administrative areas. In order to visualise this, imagine that any one layer of the sandwich can be sliced vertically into pieces.

In order to understand the complexity of the model, consider, for example, the organisation {C=GB,O=XYZ} shown in Fig. 2.7. The organisation's entry represents an autonomous administrative point, with the AAA extending down to the leaves. The organisation consists of three operating divisions: sales, production and R&D. The subschema needs to be consistent across the whole organisation, so that the divisions may freely understand each other's information. Thus the subschema specific administrative area is congruent with the AAA. (Note that the Standard forbids subschema inner administrative areas.) Any division which wishes to define a new data type, must first register it at the organisational level, via the administrator responsible for the subschema. Security policy, on the other hand, does not need to be consistent across the whole organisation. This is because the R&D division undertakes a lot of collaborative research with outside institutions, and needs to impose its own (more liberal) security policy. Thus the AAA is partitioned into two specific security administrative areas, one covering the R&D division, and one covering the rest of the organisation. The two administrators can thus act independently with respect to security administration. With respect

to collective attributes, it is decided that this specific administrative area should be congruent with the AAA, so that, for example, the main switchboard telephone number can be attributed to every entry. However, each division is delegated authority to define its own collective attribute inner administrative areas, to which their own specific collective attributes may be attached as shown in the bottom diagram of Fig. 2.7.

Operational and collective attributes can be attached to an administrative area. They then appear to be shared by, and present in, all of the entries enclosed within the area. It is also possible to explicitly define subsets of the entries within an administrative area, i.e. entry collections (§ 2.12.1), and to attach operational or collective attributes just to them. Access control information, collective attributes and subschema management make extensive use of the administrative model.

Remember that the Directory Administrative Authority Model does not depend upon the distribution of the DIT between different systems (DSAs). When the DIT is distributed between many DSAs, and an administrative area spans one or more DSAs, the Directory has in-built mechanisms to ensure that the administrative information is copied to the right systems. The administrative information is propagated from one DSA to another via Hierarchical Operational Bindings (§ 4.8 and § 9.19).

### **2.11.2 The 'Administrative Role' Directory Operational Attribute**

The model defines an operational attribute, called **administrative role**, that is used to identify the start of autonomous, specific and inner administrative areas. The attribute is positioned at administrative entries, and is multi-valued. Each value indicates one of the roles of the administrative point. Six values are defined: one value signifies autonomous area, there is one value for each of the specific aspects of administration, i.e. subschema, access controls and collective attributes, and one value for each of the allowable types of inner area (subschema cannot have an inner area). For example, an AAP entry must hold this attribute with four values stored in it. (An AAP is always the start of a specific area for each of the three aspects of administration, plus it is autonomous.) A specific administrative point below this, that identifies the start of another partition of a specific administrative area, will contain the administrative role attribute with a single value that identifies the specific aspect. An inner administrative point below this will also hold the attribute, only this time its values will be either or both of the allowed inner areas (access control and collective attributes).

## **2.12 SUBENTRIES**

So how and where is this operational and collective attribute information stored in the DIT, so that it can be attached to a set of entries within an administrative area. The original idea, as might be deduced from Fig. 2.7, was to store the shared information in the administrative points. This information needs to comprise:

- a description of the DIT subtree, or set of entries, to which the 'shared' attributes apply (this is stored in a 'subtree specification' attribute); and
- a list of the 'shared' attributes that should be applied to a particular DIT subtree or entry collection.

However, an entry consists of a set of unrelated attributes. Here we have relationships between several sets of attributes. This could be solved by creating one complex attribute, that contained both a subtree specification, and the shared attributes. Then each value of the complex attribute would contain the information describing one set of shared attributes. Alternatively, and this is the method that has been chosen, the administrative entry could be regarded as having several subentries contained within it. Each subentry would consist of:

- the subtree specification attribute;
- the list of attributes that are to be applied to the subtree/collection of entries; and
- the RDN of the subentry (so that it can be uniquely identified).

This is shown in Fig. 2.8. But we have forgotten one thing. Each entry, as we shall see in the Chapter 3, has an **object class** attribute that controls which attributes can be stored in it. So why should not a subentry have one too? It should, so that accounts for the final component of a subentry as shown in Fig. 2.8.

### **Fig. 2.8 The structure of a subentry**

Thus each administrative point has a set of subentries beneath it, which are conceptually part of the administrative entry. Subentries are not visible to normal users of the Directory. Each subentry is uniquely identified by its own RDN. Each subentry contains a description (subtree specification) of the DIT subtree or collection of entries that it governs. It also contains a list of operational and collective attributes that are to be applied to each entry within the scope of the subtree specification. Finally, this list is controlled by an object class attribute. Subentries are used to describe the subschema that is to control a DIT subtree, to apply collective attributes to a set of entries, and to enforce access controls on a part of the DIT.

#### **2.12.1 The Subtree Specification Attribute**

Subtree specifications are very comprehensive. Not only can DIT subtrees be precisely defined, but specific types of entry within a subtree can also be identified. The latter are sometimes known as entry collections. Entry collections will probably not be a true subtree of the DIT, and so the term **subtree refinement** is also used to describe them.

The subtree specification consists of three optional components. If all of them are absent, then the subtree starts at the administrative point, and contains all of the entries in the administrative area. This is the default.

- The first component (**base**) specifies the name of an entry below the administrative point, at which the subtree is to start.
- The second component (**chop**) specifies the precise shape of the DIT chunk below the starting point (base). This might or might not be a subtree. Chop optionally specifies where to start including entries, counting down from the base, and where to stop including them. This is expressed as two integers, a minimum and a maximum number of RDNs. Chop also allows particular smaller subtrees to be excluded from the main one (using a chop before or chop after construct). The only restriction placed on chop is that the lowest entries must be at or above the bottom of the administrative area.

- The third component (**specification filter**) selects only those entries from within the subtree, that match the given object class (§ 3.5) criteria. For example, to select people's entries from an organisation's subtree, the filter would be object class = organisational person. To select people who use X.400 Email, the filter would be object class = organisational person and object class = MHS-User. After reading Chapter 5, you will see that this filter is similar to the one used in the Search operation, except that only object class criteria can be used.

Each of the components may be individually selected for inclusion in the subtree specification. Fig. 2.9 shows the sorts of subtree (or subtree refinements/entry collections) that can be specified by different combinations of the above components. By way of a further example, referring to Fig. 5.3, suppose a subtree specification is to be placed in a subentry below the {O=XYZ Plc} administrative point. The subtree refinement is to include only the people in the R&D unit, but it is also to exclude the Communications Team. The subtree specification would be as follows:

**Base** is OU=R&D,  
**Chop Before** OU=Communications Team,  
**Filter** on Object Class = organisational person

### **Fig. 2.9 Examples of subtree specifications**

Once the precise shape of the subtree or collection of entries has been defined, the administrator can then attach access controls, collective attributes and subschema operational attributes, to the entries described by the subtree specification.

## **2.13 THE DSA INFORMATION MODEL**

Up to this point in time, all our deliberations have been about models of the global Directory information, viewed as if it was held in a single centralised database. Distribution of the data between computer systems (DSAs) was not important. Neither was replication of the information between systems. Once we start to look at the subset of the global information that is held on just one system (DSA), we have left the world of the global Directory database behind, and have entered a world where information is only seen from a system's eye view. This system's eye view is called the DSA Information Model. Anything that is not held within a DSA's local database is not known about, and consequently appears not to exist. Thus the DSA Information Model needs to consider what information an individual DSA needs to hold in its local database, in order to successfully co-operate with other DSAs to provide a distribution and replication transparent service to the user. Unless the DSA holds the entire global database (this is not too unreasonable from an organisation's initial perspective - see § 9.9), the DSA will need to know where and how its bit of the DIT fits into the bigger picture. A typical DSA that only holds some of the DIT, will need to have the following sorts of information:

- the complete set of user attributes for the entries that it masters, i.e. information about that part of the DIT that it controls. **Master** means that this DSA is the holder of the master copy of the entry, and if other copies exist elsewhere, they must have originated from here. Each entry is only mastered in one DSA, see Chapter 6;



- knowledge of where this set of entries fits into the global DIT, i.e. distribution information;
- if it holds copies of other entries, knowledge of where they came from, and how and when they will be updated, i.e. replication (consumer) information; and
- if it has supplied copies of its entries to other DSAs, details about these arrangements, i.e. replication (supplier) information.

A lot of this information is not visible in either of the Directory Information Models. We thus need to define some additional types of entries and operational attributes, that can hold this extra DSA information. These attributes are described in § 2.13.1, the entries are described in § 2.13.2.

### 2.13.1 DSA Operational Attributes

Operational attributes that are needed by a DSA, but which are not needed by either of the Directory Information Models, are called DSA operational attributes. By reviewing the list in § 2.13, we can see that there are actually two different sorts of DSA operational attribute.

DSA shared attributes are operational attributes that are dependent on the way that the DIT is distributed and replicated between computer systems (DSAs), but are independent of the computer systems (DSAs) in which they are held. In other words, they always have the same value whichever system they are stored in. The specific knowledge attribute, which contains the names and addresses of the DSAs which hold the master and shadow copies of a Directory entry, is an example of a DSA shared attribute. The value of the specific knowledge attribute will be the same for a particular entry, whichever computer system it is retrieved from.

**Note.** This was originally correct at the time of its definition. It is now not strictly true, because of security or administrative reasons, a particular administrator is allowed to remove some of the DSA Access Point values from the attribute before copying it to another system. In a completely open world, the shared value would be the same in all DSAs.

**DSA specific attributes** are operational attributes whose values depend both on the way that the DIT is distributed and replicated between computer systems (DSAs), and also on the specific computer systems in which they are held. An example of a DSA specific attribute is the supplier knowledge attribute. This is the name and address of the DSA that is providing a copy of part of the DIT to a particular DSA. Assuming that several computer systems are holding copies of the same entries from the DIT, and that each system is getting the information from a different source (Chapter 6), then the value of the supplier knowledge attribute stored in each system will be different.

### 2.13.2 DSA Specific Entries

The DIT consists of Directory entries. Clearly DSAs must hold Directory entries. A DSA will need to know the name of each entry that it holds, as well as the entry information associated with each entry, i.e. Directory operational and user attributes, as shown in [Figs 2.1 and 2.2](#). However, DSAs also need to hold other information, i.e. DSA operational attributes, along with the names of Directory entries. Sometimes the name of an entry will be held, without any of its associated entry information. Only DSA operational attributes will be held with the name.

DSAs need to do this, in order to link their Directory entries to other ones in the DIT. For example, the name of a remote entry may be held, along with a pointer saying in which DSA it actually resides. Therefore different DSAs may hold the same DIT name, but with different contents, the contents often being specific to the holding DSA. When writing the Standard, it was thought to be more precise, and less confusing, if it was said that DSAs do not hold Directory entries as such (since their contents are fixed), but rather, that they hold **DSA Specific Entries** (or DSEs). In this way different DSAs could hold the same name, but with different contents. In other words, the contents of an entry, as held in any given DSA, are DSA specific.

### Fig. 2.10 Part of a DSA Information Tree and its DSEs

The portion of the DIT held by a particular DSA, is called the **DSA Information Tree**. It is a subtree of the DIT, but it always starts at the root. A DSA must hold all the DSEs from the root of the DIT to the Directory entries that it holds. So for example, a DSA holding entries for the XYZ organisation, must also hold DSEs for the root and for the country in which XYZ is located. Fig. 2.10 shows part of a typical DSA Information Tree. In the example, the DSA holds all the Directory entries for the XYZ organisation, except for the entries within the credit control division. These are held in another DSA, and are pointed to by a subordinate reference. The DSA also holds an indirect pointer (or superior reference) to the root of the DIT, in order to connect this DSA Information Tree into the global DIT. (See Chapters 4 and 9 for more details.)

### Fig. 2.11 Various views of a DSE's attributes

Figure 2.11 shows the three different views of the information held in a particular DSE. Figure 2.12 shows the three different views of a DSA Information Tree. The three views, Directory User, Directory Administrator and DSA Administrator, correspond to the information that is defined in the Directory User Information Model, the Directory Operational and Administrative Information Model and the DSA Information Model respectively. (Real Directory users are not restricted to one view. A normal Directory user might be able to see administrative information and DSEs without entry information, and an administrative user will certainly be able to see all DSEs and DSA operational attributes. A 'real' user's view will typically depend upon their access rights.)

### Fig. 2.12 Various views of a DSA Information Tree

Each DSA must hold, as a minimum, one or more DSA specific attributes for each DSE that it holds. The other types of attribute are optional, as shown in Fig. 2.10. The DSA specific attribute that must be held in each DSE, is the **DSE Type** attribute. This attribute signifies the type of a DSE, and it governs the other attributes that can be held in the DSE. The attribute can only have one value, and this is encoded as an ASN.1 bit string. More than one bit may be set for any given DSE. Each bit has the meaning described in Table 2.1.

So, for an entry that has been copied from another DSA, bits 3 and 11 would be set (w/w 2.3). It will be necessary to read Chapter 9 before a full understanding of all of the meanings can be gained.

A DSA is modelled as always holding DSA specific attributes in its root entry. One example of such an attribute, is the **My Access Point** attribute. This is a DSA's own name and address. Another example is a superior reference, which points to a DSA holding entries higher up in the DIT.

**Table 2.1 The meanings of the bits in the DSE Type attribute**

<i>BIT</i>	<i>Name</i>	<i>Meaning for this DSE</i>
0	root	The root of the DSA Information Tree
1	glue	Represents knowledge of a name only. No other information is held with it
2	cp	The context prefix of a naming context
3	entry	Holds an object entry
4	alias	Holds an alias entry
5	subr	Holds a subordinate reference
6	nssr	Holds a non-specific subordinate reference
7	supr	Holds a superior reference
8	xr	Holds a cross reference
9	admPoint	An administrative point
10	subentry	A subentry
11	shadow	Holds a shadow copy of an entry
13	immSupr	Holds an immediate superior reference
14	rhob	Holds information about a superior administrative point or subentry, passed by a RHOB
15	sa	Subordinate reference DSE points to an alias entry

### WEIRD AND WONDERFUL

2.1 '88 systems make no distinction between user, collective and operational attributes. They all have the same semantics, i.e. they only apply to the entry in which they reside, they may be set by the user, and they are all returned with a Read All Attributes operation (w/w 5.6). All of them except two do not affect the operation of the Directory. They are thus most similar to '93 user attributes. The decision was made that All Attributes defined in the '88 standard would become user attributes in the '93 standard. Under this '93 classification of attributes, the two '88 attributes that cause a problem are the **object class** attribute and the **aliased object name**

attribute. (The former attribute describes the type, or class of an entry, and this controls which attributes can be stored in it (§ 3.5), and the latter is the pointer held in an alias entry, and used in alias dereferencing.) Although they may be set and read by users, they are clearly bound up with the operation of the Directory, and therefore are technically operational attributes. But operational attributes were defined as not being returned in a Read All Attributes request. It was suggested that these two attributes should indeed be classified as operational attributes, and that the restriction on Read All Attributes be slackened, so that specifically named operational attributes could be returned by this operation. However the international decision did not favour this approach, and instead chose to classify them as user attributes, but to add a paragraph in the Standard saying that they are really semantically operational attributes but must remain as user attributes to facilitate interworking with '88 systems.

2.2 The original reason for not allowing aliases to point to other aliases was not documented. However, the author believes that it was either to avoid looping, or to prevent aliases from having both backwards and forwards pointers (in the original model entries used to have back pointers to their aliases). Whichever reason it was, neither now exists in the '88 Standard. Alias back pointers were removed at the Draft Proposal stage. Looping was avoided by the introduction of trace information. However the restriction on aliases was never removed, and this resulted in an extra parameter being added to Chaining Arguments (aliased RDNs), and extra procedures being written for Part 4 of the Standard, to enforce the restriction (§ 9.13.6). After all that, the '93 edition of the Standard removed the restriction, which caused even more procedures to be written to describe how '93 DSAs should deal with the aliased RDNs argument!

The original model of aliases, initially proposed during the development of the '88 Standard, had forward pointers from the alias entry to the object entry, and back pointers from the object entry to its aliases. But the back pointers were dropped when it was realised that modifying the relative distinguished name of an entry, or removing an entry from the DIT, would be too complicated to define in a distributed environment. A two-phase commit protocol would almost certainly be needed, and many experts did not want to introduce transaction processing into the Directory. Consequently we can now be left with 'dangling' aliases. These are aliases that point to a non-existent object entry, because it has been either deleted or renamed. It is up to administrators or implementors to clean up these old aliases.

2.3 Observant readers will notice that Table 2.1 does not define bit 12. Bit 12 was originally used, during development of the '93 Standard, to signal an entry that had been deleted from the DIT, but that had a 'name re-use' timestamp attached to it. This indicated the time during which the name of the entry could not be re-used to create a new entry with the same name. The entry was thus 'dead' to users, but still 'alive' in the DIT. It was called a Zombie DSE (the living dead!). However, the 'name re-use' timestamp was subsequently replaced by allowing entries to have a unique id assigned to them (§ 8.4.2). Names can then be re-used immediately, but each re-use will involve the allocation of a different id. This is the feature that actually appears in the '93 Standard. Whilst the Zombie DSE is no longer needed, it still lives on in spirit, by keeping bit 12 from being re-assigned. Who says that Standard's writers do not have a sense of humour!

010731