Web Gateway Design Document

June 25, 2004

Contents

Introduction

The *EOSDIS Version 0 Web Gateway* is the title of the WWW gateway which supports the search, browse, and order capabilities of the EOSDIS Version 0 IMS. The initial prototype (called IMS/WWW) was a product of Hughes STX Corporation and A/WWW Enterprises; Hughes STX Corporation has since taken over the ongoing development of the system.

This is a living document which will attempt to capture the basic design of the Web Gateway as it evolves. Additions, corrections, and requests for clarification should be sent to the author.

Chapter 1

Introduction

This chapter will introduce you to the EOS Data Gateway (EDG). If you've never heard of or used the EDG before, this chapter should give you the basics.

1.1 The big, big picture

There are a number of sites on the Internet (many of them NASA-affiliated) which archive and distribute "earth science data": this data takes many forms, one common kind being satellite pictures of the earth.

Some years ago, it was decided that it would be beneficial to have such sites run servers which all speak the same protocol... servers which would allow a client to...

- **Search** for data, based on high-level criteria (geographic location, time, satellite, etc.)...
- **Browse** the results of search, viewing "thumbnails" of data which satisfied the user's query, and...
- **Order** the actual data, having it delivered to a specified address on some chosen medium (CD-ROM, 9-mm tape, etc.)

The **EOSDIS Version 0 Information Management System** (EOSDIS V0 IMS, or just "V0", for short) was developed to satisfy this goal. From one perspective, it consists of a high-level client-server protocol, and a collection of C libraries which implement the passing of protocol messages between a V0 client and one or more V0 servers.

These V0 server sites are usually called **"Distributed Active Archive Centers"**, or **"DAACs"** for short. Each DAAC may allow access to many individual data items, called **"granules"** (e.g., a single Landsat image): granules are themselves grouped into collections of related granules called **"data sets"** (e.g., the set of all Landsat-5 images).

The first (and most sophisticated) V0 client to be developed was an X-Windows based user interface, commonly called "the GUI". This interface allows a user to send a search to multiple DAACs simultaneously, and then to browse and order the results from the different sites in a seamlessly-integrated fashion.

This system, commonly called the **EOSDIS Version 0 Web Gateway** (or Web Gateway for short), is another, very special kind of V0 client. It is fashioned after the GUI, and allows users to perform many of the same activities... however, it is WWW-based, using CGI scripts to:

- Output (and handle) the search form.
- Display the search results, as dynamically-generated HTML with clickable hyperlinks.
- Fetch and display the browse images.
- Submit the data orders.

1.2 What a "gateway" is

Many user interfaces are built to interact with a single user at a time: the user runs the software on their own machine, or from their own user account on a shared machine.

Web gateways are different. They run on a centralized location, and users come in from all across the Internet. To make matters more complicated, a Web gateway is usually not a simple, dedicated server that is run once per user: rather, each mouse-click on a button or hyperlink initiates a *new HTTP connection* which causes the gateway program to be run again, just to generate the next "screen" (usually an HTML document) that we want the user to see.

This lack of one single, dedicated connection per user is the so-called stateless paradigm of the World Wide Web, and it complicates matters tremendously. Basically, it means that our Web gateway has some basic interface issues to deal with: for example, each time it is run, it must be able to determine, *solely from its input...*

- **Which user it is servicing** (if we are attempting to give each user the illusion of having a dedicated interface).
- **What screen the user wants to go to** (or what action they want to perform).
- **What this user has done previously**, if that information is needed to handle the user's request. *Therefore, it also needs to know...*

The last two items imply the need for some form of "session management": each user has a "session" with the gateway, and within that session they have the illusion of talking to a dedicated service. Therefore...

- It must make use of a database of some sort to **preserve information from screen to screen** for a particular user (e.g., that user's contact address and display preferences). *And therefore...*
- It must **allocate temporary space** for each user's files, and **clean up** those files at some later point (usually after a long period of inactivity).

The EDG, like virtually all web gateways, makes use of the **Common Gateway Interface** (CGI). Thus, at its most basic level, it is a collection of CGI scripts (programs) capable of receiving form input from - and sending HTML output to - any standard WWW browser.

1.3 Components of the Web Gateway

This software consists of the following pieces:

- Perl5 is a fast, powerful, interpreted high-level OO language; it possesses advanced text-processing capabilities which make it ideal for CGI form handling and HTML generation; indeed, Perl has proven itself to be the language of choice for many complex CGI applications.

  A collection of **Perl5 programs/libraries** which perform all the high-level user interface activities: parsing CGI input, generating HTML output, maintaining temporary user areas and databases, keeping a cache of browse images, etc.

- A collection of compiled **C programs** for performing the low-level interaction with the data servers: obtaining "valids" for generating menus, sending queries to DAACs, receiving results and browse images, etc.

  We use C at this level so that we can make use of the existing C libraries.

- Additional **support programs** for performing certain special activities (e.g., searching for documentation on a DAAC's `WAIS` server, generating GIFs for spatial/temporal coverage maps, etc.)

  Some of these are external utilities, while others have been custom-built for the gateway.

- A lot of **support documents**: configuration files, valids files (for generating menus), documentation (HTML and Postscript), GIFs for buttons.

1.4 A quick run-through

What follows is a sort of a "day in the life" of the gateway, showing you how (and when) the various software components are invoked. It is a bit more detailed than the average user will need.

1.4.1 The "welcome" page

The URL that users should place bookmarks to is that of the gateway’s "welcome" page. This is a static HTML page containing a greeting and hyperlinks to the main URLs... among them, the URL of the gateway script.

1.4.2 Starting a session

When a user first visits the gateway script (by following a hyperlink from the "welcome" page), the gateway creates a special temporary “**session directory**” to hold that user’s files. Each such directory is distinguished by a unique “**session id**”; this id is passed between successive invocations of the CGI scripts via the "**sid**" tag: this enables an invoked CGI script to "pick up" where the last script "left off".

When the session is created, a special "session state" database is created inside it: this "session.state" file holds complex information that we want to remember from one invocation to the next, like user preferences.

1.4.3 Moving from screen to screen

Nearly every "screen" (HTML page) output to the user will contain hyperlinks pointing back to the same CGI script. These hyperlinks will always have two special tags embedded in the "query string" portion of the hyperlinks (the part after the "?"):

sid

The session id. This tells the script which user we’re servicing.

mode

The "screen" that this hyperlink leads to... or, the action that it will perform if clicked upon.

These same tags are also embedded in forms, as hidden fields.

1.4.4 Filling out the search form(s)

The first screen the user sees is the "search form". Here, the user builds their high-level query for data, possibly using several "sub-forms" (e.g., one for geographic selection) in the process.

The dynamic generation of GIFs (for example, the map generation during geographic selection) is done via the public-domain “gd” library routines.

1.4.5 Searching

The “real” work (sending inventory search or browse requests to the DAACs and retrieving the results) is done, not by the CGI scripts, but by the special standalone client program, **imsclient**, written in C using the standard IK/IC IMS libraries. The CGI programs invoke this “standalone client” when they need a search or browse function executed, by creating an ODL file which contains the “query”, and then specifying that file in the command-line arguments of **imsclient**.

The **imsclient** program is not run directly by the gateway; instead, an intermediate program, **imsrunclient**, is executed. This program performs any necessary setup, runs **imsclient**, and then performs any necessary post-processing (keep reading...).

1.4.6 Getting the communication status of a search

As **imsclient** executes, it periodically writes its status information to a file, which can be read every few seconds by the gateway. This information is used by the gateway to output a “communication status” document. This document normally auto-updates every 10 to 15 seconds.

1.4.7 Where the search results are placed

When **imsclient** completes a search, the results are placed in GDB files in the session directory: "dataset.gdb" and "granule.gdb". Since **imsclient** is a C program, these results are in a form intended for use by other C programs. However, this creates problems:

- This compact, complex C-based format is not good for the gateway’s Perl scripts.
- We need more information from the messages than the IK libraries currently write to these GDB files (especially to deal with experimental extensions to the ODL).

So, imsclient uses a "hook" into the IK message software to write the ODL messages to a "client.obj" file in a very Perl-parseable format. Then, a "post processing" step is done by **imsrunclient** wherein those contents are turned into Perl objects, and re-saved in "results.gdb".

1.4.8 Accessing/displaying the search results

The user may examine the search results in a number of ways:

- List all data sets (with only basic information shown), or, display the detailed attributes of a single data set.

- List the granules from one or more data sets (with only basic information shown), or, display the detailed attributes of a single granule.
- Display graphics of the spatial/temporal coverage of a set of selected granules. These graphics are generated via the **imscovermap** and **imscovertime** programs.
- Display a "browse" (thumbnail) image of an individual granule, if available.

1.4.9 Browsing a granule

When the user asks to browse a granule, the gateway first checks its local "cache" to see if the files are already available. If they are not, **imsrunclient** is invoked to fetch the "browse product" from the appropriate DAAC.

When a browse image is requested by the user, it is returned by the DAAC as an HDF file. The gateway invokes the special **hdf2gif** utility to extract one or more GIFs from this file; the browse document is created by in-lining the first such GIF, and providing hyperlinks to the other GIFs and to the HDF file. Both the HDF and GIFs are stored in a special shared “cache” area on the gateway machine.

1.4.10 Putting granules in the shopping cart

As users encounter granules they like, they can put them in the special “shopping cart”. This is a GDBM file, `cart.gdb`, which is maintained exclusively by the Perl scripts: it contains the *full* information for all granules which were placed in there... that way, users can perform multiple searches and accumulate granules in their cart, until they are ready to order them.

It is worth noting that `cart.gdb` has the same basic internal structure as `results.gdb`.

1.4.11 Choosing packages for granules in cart

Before a user can submit their "order" (the items in the shopping cart), they must first make sure that packaging information (media type, media format, etc.) is chosen for each granule that needs it. The icon next to the granule in the shopping cart tells if the granule needs to have packages chosen for it.

Granules orderable by package will have a "choose package" button that leads the user to a package-selection form.

Note: *not all granules may be orderable by package.* If a granule has no "packaging" information associated with it, the gateway will (usually) attempt to order it by sending electronic mail to the user-support "contact" for that granule’s data set. The icon next to the granule in the shopping cart tells if the granule will be ordered by e-mail.

1.4.12 Submitting the order

Once packages are chosen for all granules that need them, the conventional orders (if any) are sent out to the DAACs in much the same way that searches are... via **imsrunclient**. E-mail orders (if any) are sent out much as you would except: by using `sendmail` to send the e-mail.

1.4.13 Cleaning up after our users

The session directories and cache files are checked periodically by the **imscleanup** utility, a cron job which usually runs once every 15 minutes. Session directories are deleted if they have not been accessed for several hours; cached browse files are deleted if the cache size exceeds the configured "comfort" level. If the installation is configured to have persistent user sessions (accounts), there is a separate utility: **imscleanaccounts.pl** that will purge old files that accumulate in user sessions. It has checks for time since the account was used and disk space usage. This script is run in a cron job about once an hour.

Chapter 2

System architecture

This section will give you the basics of how the Web Gateway system is organized... it covers the major subsystems, and the design principles behind them.

If, for example, you want to know "how does browse work?", this is the place to be.

2.1 Overview

The system consists of one principle CGI script, written in Perl5. It is called **nph-ims.cgi**. It, and all the related code, documents, support files, configuration files, log files, etc., live under the `~imswww` account of the Web Gateway host.

The script loads in other Perl modules (from the **IU** and **IW** libraries) as needed for the current activity:

- The "IU" (IMS Universal) Perl library contains fairly-generic IMS-related code.
- The "IW" (IMS Web) Perl library contains code which is specific to the CGI portions of the Web Gateway.

The script may also invoke certain "helper" programs (like **imsclient**, **imsgdbdump**, etc.). These programs are most often written in C because they require speed, or because they need to be linked with the existing IMS code base (e.g., the **IK** and **IC** libraries):

- Most of the interaction with the remote DAAC servers – **search, browse, and order** – is done via the V0 IMS protocol. The **imsclient** program, using the IK library, generally does this communication.
- The **guide** searches are done both via the WAIS (Z39.50 V1) protocol using a standard **waisq** program and an XML-based protocol over HTTP. See the **NewGuide** technical note for details on the XML-based protocol.

2.2 Design issues/approaches

This section discusses the rationale behind certain design decisions, when not explicitly covered in the "subsystems" section.

2.2.1 The basic architecture

Variations on the CGI paradigm

As said previously, CGI gateways differ from conventional user interfaces because of the **stateless paradigm** of the Web; that is...

> *Virtually every single mouse click results in the gateway program being run from scratch to generate a single page, after which the gateway program shuts down.*

There are a couple of variations that can be tried to simplify the architecture of such a system:

One-per-session dedicated server

Some have experimented with solutions (like CGI::MiniSvr) that involve the spawning of a dedicated server that listens at a randomly-chosen port number (passed to the client in the first screen): this server maintains persistent information in core like any normal user interface program.

However, such a solution must be immediately rejected if there are a potentially large number of simultaneous users, or if the "inactivity timeout" is fairly long. The reason is that the gateway host could quickly exhaust its process table; the matter is further complicated if each process' core image is large.

Applets

Another approach would have been to employ **applets** for the user interface end, restricting the backend to non-user-interface activities (e.g., the execution of a search). Unfortunately, Java as an open standard predated the initial prototype effort. Plus, the Java Security Model is currently so restrictive that certain activities (e.g., communication with the various DAACs) *still* must be done on the gateway side.

This leaves us with either the CGI or the dedicated-server architecture (now possibly more feasible) for the backend. So although the user interface would be simpler to implement, the general architecture must still deal with many of the same issues.

Netscape plug-ins

Browser-specific solutions were deemed too restrictive, and so were not considered. Plus, solutions which require active user intervention (e.g., software installation at the client side) tend to discourage usage.

For various reasons, we have continued with the conventional CGI gateway architecture, where the gateway program is only running if it is currently satisfying a user request.

Dealing with the stateless paradigm

As stated before, the conventional CGI architecture means that, each time through, the gateway program has to determine, *solely from its input*:

- **What this user wants to do/see.** We get this information from the CGI input tag "**mode**", and use it to load/run the Perl module that handles that mode.
- **Who this user is.** We get this information from the CGI input tag "**sid**".
- **What this user has done before** (remember, we can't just put stuff in global variables!). To do this cleanly, we give each session its own directory, and this is where we put session-specific files and databases.

Note the implications so far:

- We need to **allocate space for the session directory** and all the files it will contain, and we need to reclaim this space later.
- We need some **session state** mechanism which we can use to store information that we'd normally like to put in global variables.
- We need to ensure that we **propagate the session id** to each "screen" generated, so subsequent mouse clicks will "stay within this session".

Choosing the implementation language(s)

Like many systems, the Web Gateway is in a perpetual state of evolution. Such systems fare best when implemented in a **high-level language**, where developers can concentrate on the application rather than on the bits and bytes.

However, we had to make use of the V0 IMS client software... which is written in C, a **low-level language** not too far removed from assembler. It was decided early on that implementing the Web Gateway in C would have resulted in the majority of time spent tracking down obscure memory-allocation bugs. C++ was strongly considered, but it too is a difficult language for rapid prototyping.

The final solution was as follows:

- Use Perl5 for the interface-oriented activities. Perl5 is a fast, high-level, object-oriented language with advanced text-processing capabilities... ideal for CGI handling and HTML generation. It is also interpreted, which makes code-and-test extremely easy. "Python" was another serious contender.

- Bundle the low-level IMS activities into standalone programs, written in C, and using the IMS C libraries.

Note: An attempt was made to dynamically-load the IK libraries directly into the Perl runtime image (allowing Perl modules to call the IK routines). However, this could not be done: the JPL ODL parser code has namespace conflicts with the Perl internals, and the IMS custom GDBM module conflicts with the `libgdbm` code normally loaded by the GDBM_File module.

Speeding things up: the custom HTTPD

Up until a few weeks before the release of the 1.5 version of the Web Gateway, one of the biggest drawbacks of Perl5 was the lack of a true compiler. This meant that the entire source code base was being re-compiled into bytecode for interpretation every time a CGI request came in... resulting in a *very* slow interface.

Note: The famous "undump solution" of Perl compilation can hardly be called that. It is reportedly fraught with problems, and success outside the SunOS 4.x environment is supposedly rare.

To remedy this problem, a custom HTTPD module (**OHTTPD**) was written. This module is used by the **imshttpd** program, also known as the custom HTTPD. It works as follows:

- The custom HTTPD is started at system bootup, whereupon it loads all the Perl code of the Web Gateway, su's to the same non-privileged user that normal CGI scripts run as, and then listens at a non-standard port number (given by **PerlHttpdPort**).
- When a connection comes in, the custom HTTPD forks off a child, and then *immediately* goes back to waiting for the next connection. That's all it does.
- Meanwhile: the child process contains a complete image of the parent... *including all the pre-loaded Perl bytecode*. It handles the socket connection with hardly any startup cost (not even the cost of an exec!).

On a 90MHz Pentium running Linux with 32MB RAM, the transition between many of the interface screens takes no more than second or two... virtually instantaneous, and basically noise when compared to the delays expected from network lag.

Also, since the Web Gateway script generates URLs back to itself by looking at its own URL, the switchover from a standard HTTPD approach to the custom approach required almost no code changes.

For security reasons the capabilities of the custom HTTPD are extremely limited: it can only run the Web Gateway code in the `./lib/perl` directory. This is to prevent the "latro" security hole described at `www.perl.com`.

Note: When the "alpha2" Perl compiler stabilizes, we will evaluate its ability to reliably compile the Web Gateway. A compiled Web Gateway may be slightly slower than the above approach, but maintenance-wise it is likely to be simpler.

Asynchronous operations: the "spawn" module

In our design, IMS client operations (like inventory searches) are done by invoking the **imsrunclient** script and waiting for that script to terminate, checking status files it and its backend (**imsclient**) output every few seconds.

This means that the Web Gateway must be careful to detect when the **imsrunclient** process is no longer running... whether due to a graceful exit, or (as we hope never happens) an ungraceful one.

A general-purpose module, **IU::Spawn**, was developed for this. It allows a parent to spawn a detached child whose health (running, not running) can be queried by any process, at any time.

The scheme (simplified here) is pretty straightforward. The parent [1] creates a lock file with a well-known name, and gets an EXclusive lock on it. It then [2] forks a grandchild which will inherit this descriptor and lock. The parent is then free to close the descriptor. At this point, any process can query whether the child is still running by [3] attempting to get a SHared/Non-Blocking lock on the lock file: success means that the child has exited and released its exclusive lock.

2.2.2 Added security for the IMS/WWW gateway

As you know, anything under a user's session subdirectory "pub" (like generated map images), or elsewhere in the imswww-usr directory, is fair game for being loaded by a WWW browser. All other documents (including user preferences, search criteria, etc.) are off-limits via the browser.

Protecting files against local access

One thing that should be done is to ensure that all sensitive documents, even if they are off-limits to browsers, are chmod 0600. That will mean that generally, only CGI scripts will be able to read them (currently, users logging into an account on the server machine might be able to read them as well).

Using hostname to verify each request

One simple way to make the pirating of sessions (stealing them from another browser if you know the sid) even more difficult is to get the host IP address of the requester (from the REMOTE_ADDR environment variable) and store it in the session state when the session is first created... say, in variable OriginalRemoteAddr.

We could use this in two places:

> On each new CGI request, the requesting host's IP would be compared to the `OriginalRemoteAddr`. A mismatch causes an immediate permission-denied error.

> The **.htaccess** file in the "pub" directory could, instead of providing universal access, **provide access only to that user's host**.
>
> This would likely prevent even the developers from "peeking" into the pub area of another user with a browser... but it would ensure that even a user's "public" files were only accessible from their machine.

Using cookies to verify each request

Another, possibly better way (and now fairly portable) is that *cookies* could be added to the security mechanism. It would work like this:

1. User creates a new session. At this time, a session cookie is generated along with the **sid**. **They are not to be identical in any way, and the cookie should be as random as possible.**
2. The cookie stored in the session state, in the variable DoubleCheck.
3. The cookie is reported back to the user along with the first response, via the HTTP header line:

   ```
   Set-Cookie: IMSWWW_DOUBLECHECK=(cookie);
               path=/;
               expires=(tomorrow GMT);
   ```

   (The IW::ReplyHead module will need to be extended to handle cookies.)
4. Each new request to the gateway host will have that cookie information in it, which will be accessible to CGI scripts via the HTTP_COOKIE environment variable. On each request, when the session is about to be resumed, a check is made: if the user's browser sent back any cookies at all, we make sure that one of them is an

```
IMSWWW_DOUBLECHECK=(cookie)
```

That matched the cookie stored in the session state. If not, we give a "permission denied" error.

2.2.3 Control flow in CGI scripts

Giving each user the illusion of a dedicated interface is extremely difficult in a non-applet environment. Aside from the individual quirks and capabilities of different browsers (and different versions of those browsers), there are features of the CGI mechanism which make it unsuitable for large, complex applications.

Issue: the "back" button

Control flow in a WWW application is to a certain extent determined *invisibly* by the *user*, via the "back" button. The reason this is "invisible" is that the pressing of the "back" button does not result in an event detectable by the CGI script.

Issue: client-side cacheing

Care must be taken when generating URLs that link to other "screens" in the interface, because of the behavior of the page-cacheing mechanism of the user's browser.

There are two conflicting cacheing behaviors to content with:

- We want to be able to hit the "back" button, and have the previous page still be in cache.
- We want to be able to submit a form (or follow a hyperlink to a dynamically-generated page), and bypass the cache, *even if all the CGI parameters are identical to a previous query*.

Cacheing effects lead to two undesirable situations, almost completely out of the script's control:

Bug 1: "Data Missing" You hit the back button, and you see:

```
"Data Missing"
    This document resulted from a POST operation and has
    expired from the cache. If you wish you can repost the
    form data to recreate the document by pressing the
    reload button.
```

In this case, either the browser's page-cache is too small, or else the data has simply expired from cache. Attempts to set an expiration date via `"Expires:"` meet with failure: Netscape 2.0 seems to ignore this header field. Good news: the user can usually just hit reload.

Bug 2: "nothing happened" Attempting to submit a new search does nothing, except bring up the results of the last search. Nor does the "Erase form and start over" button do anything.

In this case, we have the opposite problem: the search form is being retrieved from page-cache when cacheing is not desired. This is *really* annoying because using `"Pragma: no-cache"` to get around it results in case 1 again.

Our remedy for this is the **uniquifier** tag (q.v.). By generating a unique number which is included in the URI, we "unique-ify" the URL, thus forcing the browser to re-load it.

Form state vs. session state

There are two ways of keeping track of certain kinds of user-state information. Our system uses both:

In a form.

> In this case, state information is embedded in hidden fields in a form. When the form is submitted, the state information is available to the CGI script. The best example of this is the session-id.

In the session-state.

> This involves keeping a file (or files) around in the user's session directory. Inside the files are certain kinds of information that the system has accumulated about the user during the lifetime of the session: for example, their name.

It is tempting to just use the session-state file (in our system, called `session.state`) for everything except the session id. However, a perusal of the pre-1.8 "SRCHFORM" code will reveal that a user's *entire* query was embedded in hidden fields every form returned by the search script: the map-selection form, the parameter-selection form, etc.

The reason for this was the back button:

> ***Warning:*** Many times, users will make a change to their query criteria, decide they don't like it, and as a remedy they will hit their browser's “back” button, mistakenly using it as a form of “undo” button, which it is not: all the back button does is *show you the last page your browser downloaded*.

To avoid confusion, it was imperative therefore that all search-form pages represent exactly the query that would be submitted if the “submit” button on that page were to be pressed. Therefore, instead of storing parts of the query in the session-state, we embedded the *entire* query on every search-form page we generate. That way, no search-form page could ever “lie” to the user.

HOWEVER, as the search criteria became more complex, this became increasingly difficult to implement. In 1.8, we returned to a solution based on storing a "query" object in the session state. The behavior of the system is generally quite intuitive.

2.2.4 Why frames are difficult to implement in CGI

The "simple" scenario

One simple approach that was requested was this:

> Have two frames, with the blue button bar and title in a "title" frame, and the rest of the returned HTML page in a "body" frame. That way, users always see the title and button bar.

```
+-------------------------+
| [] Results              |     Title frame
| [][]  [][][][]  [][][]  |     (fixed)
+-------------------------+
| 4. === === === ===      |
| 5. === === === ===      |     Body frame
| 6. === === === ===      |     (scrollable)
| 7. === === === ===      |
| 8. === === === ===      |
+-------------------------+
```

Sounds nice and simple, right? But consider what happens when you press a button in the blue button bar:

1. You want to generate a new page in the "body" frame, obviously.
2. You *have* to regenerate the "title" frame. Why?

   First: the help button is context-sensitive

   Second: if you don’t, the hyperlinks will suffer cacheing effects; the reason we always generate a new **uniquifier** (u=XXXX) on each page is to *force* browsers to make a new request, regardless of whether or not they think they have the page cached. Remember that bug where you click on the "Results" button, and it comes back immediately with an old Results listing? That’s what we’re talking about.

With me so far? Now consider this, right out of the comp.infosystems.www.authoring.cgi FAQ:

```
--------------------------------------------------------------
Subject: 3.14 Can I write output to several frames at once?
```

```
 A single CGI script can only ever print to one frame.

 However, this limitation may be overcome by using more than one script.
 The first script (the URL of the "submit" button) prints a frameset,
 typically to a "_parent" or "_top" target.   The sources for one or
 more of the frames thus generated may also be CGI scripts, to which
 you can easily pass parameters (eg encoded in URLs with method GET).
* This hack is definitely not recommended.   If you find yourself wanting
 to update several frames from a single user event, it probably means
 you should review the design of your application at a higher level.

 Warnings:
  1. Don't forget to escape your URLs.
*   2. This technique results in your server being hit by multiple
     concurrent CGI requests.   You'll need LOTS of memory, especially
     if you use a memory-hog like Perl.   It can be a good recipe
     for bringing a server to its knees.

 Javascript is often a valid alternative here, but note just how silly
 it can (and often does) look in a different browser.
 ------------------------------------------------------------
```

The issues here are:

- We *can't* have concurrent requests on the same SID. Part of our design, for safety, is to have each new request *kill* any currently-running request with the same SID.

  This design prevents a lot of weird failures that were occurring when simultaneous requests were competing for single-user resources (like the session.state database). Discarding this scheme would very likely invite disaster (it was not implemented on a whim).

  (Consider: what happens when a user presses a button, and then goes "oops" and quickly presses another one? We can't just wait for the first request to just complete: it might have been a search/browse request. Nor do we want the second request to fail.)

- Quite frankly: when a warning like this appears *in the CGI FAQ*, with phrases like "bringing a server to its knees", I listen to it. We'd effectively be doubling the amount of RAM and CPU time used for each request... all to get a simple frame!

- We could have *all requests* handled by a different CGI script, written in C... call it `imsframe.cgi`. It wouldn't use any of the resources, but it could do the following:

  > Return a FRAMESET document, with 2 frames. The first frame points back to `imsframe.cgi`, requesting a blue button bar. The second frame points to the Perl `nph-ims.cgi`, requesting the body.

> I dunno. Sounds like a looooooooooooooot of work for a feature that has no real scientific merit. And we'd still want to handle frameless browsers, wouldn't we? Or would we?

How frames CAN work

If you *really* want frames, here's the best probable way to approach them...

First of all: **make the frames optional**. On some browsers, frames behave quite strangely with regard to the back button. Let users employ non-framed versions of the system.

When a request for a new framed document comes into the Web Gateway, where the last document displayed was not framed (or, where more than one of the frames has changed), the Web Gateway script should do the following:

- *For each frame* you'll want to generate, redirect the $HTML stream to write to a temporary file in the session directory. For example, let's say you want to have the navigator in one frame, the granule list in a second, and the attributes of a particular granule in a third. Write them, respectively, to:

  ```
  $SessionDir/pub/frames/navigator.html
  $SessionDir/pub/frames/granlist.html
  $SessionDir/pub/frames/granattrs.html
  ```

- NOTE that this is not as bad as it might sound:

  The generation of the navigator frame is easy; you could do it automatically in IW::App.

  Any other frame you want to display might *already have been generated* from a previous request, and still be sitting in the ./frames directory>.

- *As a final step,* return a FRAMESET document directly to the browser (i.e., to the STDOUT). The FRAMESET document should contain references to the .html files of the component frames which are sitting under the session directory, *and*, these references should be qualified by the **uniquifier**: e.g.,

  ```
  <FRAME SRC="http://..../812345-678/pub/frames/navigator.htm?987654">
  ```

  The purpose is to force the browser to reload the frame, since the contents of that file may have changed since the last time the browser accessed it.

- If you're clever, you might be able to code the system so that a new uniquifier is output only when the file has actually changed. However, because of back-button effects, it might be safest simply to use it always.

- Notice that if you have presented a document containing two frames, GRANLIST and GRANATTRS, you may want some buttons in the GRANLIST frame to display new granules in the GRANATTRS frame *without* redrawing the entire frameset. This is quite do-able: there is nothing preventing a Web Gateway request from simply returning a document meant to populate a single frame... just as it does now. The above algorithm is just for the cases where you need to draw a frameset for the first time.

Other uses of frames

Another possibility is the use of frame in aiding with valids selection, or browsing lists (e.g., data set list in the top frame, individual data sets in the bottom frame):

```
+-------------------------+
| [] Results              |
| [][]  [][][][]  [][][]  |
|                         |
| 1. === === === ===      |
| 2. === === === ===      |
| 3. === === === ===      |
+-------------------------+
| [] Data set attributes  |
| [][]  [][][][]  [][][]  |
|                         |
| ...                     |
+-------------------------+
```

This *might* work, since each frame need not worry about updating anything more than a single frame. Notice that we sacrifice display area for this effect.

2.2.5 Gateway management

Why require a ~imswww account?

There are several reasons we have you make a special `imswww` user account to house the gateway:

- It makes it much easier (and safer) to manage the system. By establishing this standard, Web Gateway administrators are far less likely to be performing basic administration functions as `root`, so chances of disaster are minimized.
- It makes a nice, consistent email address for the Web Gateway administrator: `imswww@your.host`, which you can `.forward` to `root`, `support`, or whoever.

- You can use the `public_html` directory of `imswww` as a convenient place to put other Web Gateway-related HTML pages.

However, if you are *really* opposed to the idea, you may be able to install everything to a directory like `/usr/local/lib/imswww`, treating this directory as the equivalent to `~imswww`.

> ***Warning:*** As of v.1.5, this is an unsupported method of configuration, since it is both messy and unnecessary. If you encounter problems setting up such a configuration, our support staff might not be able to help you resolve them.

Run-time reading of the setup

In C programs, the setup information is not compiled into the programs, but is instead read in from the file **LibDir/**conf/SETUP.dat at run-time.

This means that all of the SETUP variables can change without requiring a recompile of the system programs.

In Perl (which is interpreted anyway), the setup information is read in from the file **LibDir/**perl/IU/SETUP.pm.

Cleanup of sessions/cache

The Web Gateway works by establishing, for each new user access, a "**session**", complete with a directory for holding that session's files, and a unique "**session id**" which gets passed from page to page so that the system will have a means of keeping each user on their own specific thread of activity. This is a fairly common architecture.

One of the major problems with this approach is that users may never explicitly "quit" the interface; rather, they may simply walk away for hours at a time, or move on to other activities. This could leave these session directories around indefinitely.

One graceful solution is to periodically clean out files and directories which have not been accessed in a predetermined period of time. In this system, there are two kinds of entities requiring such cleanup:

- **Session directories**, which should be removed after a few hours of idle time.
- **Cached browse images**, which should be removed after about a half-hour of idle time, or when the cache grows uncomfortably large.

The gateway does employ this approach, by use of the special **imscleanup** utility. This program is normally run as a **cron job**, making it very stable and minimally-resource-intensive.

Note: Pre-1.5 releases let you run imscleanup as a standalone process. This was judged to be far too resource-intensive (having a Perl image persistently active, for no good reason), and as of v.1.5, this method of usage is not supported.

A slightly more graceful solution would be to define a maximum amount of disk space that the gateway is allowed to take up, and only delete old files/directories when new files/directories do not have sufficient space. This is actually much more difficult to implement, especially in the case of session directories, where the amount of space needed for “new” users is not knowable *a priori*.

As of the 2.0 release, Web Gateway sites have the option of setting up persistent user sessions. A cleanup script is provided, called imscleanaccounts.pl, that will cleanup these permanent sessions. There is a complex series of rules governing when and which files are purged base on how long since the user was signed into their account and the disk space usage. This cleanup script is run in a cron job, usually about once an hour.

2.2.6 Jumping around in a page

Some users have asked why we don’t do this more. Hopefully this section will shed some light.

Terminology

Anchors

These are standard HTML anchor points, of the form:

```
<A NAME="anchor">Anchors aweigh!</A>
```

Scrolling hyperlinks

These are links of the general form:

```
<A HREF="url#anchor">link</A>
```

In other words, this asks the browser to jump to a given URL and then scroll to the ANCHOR point in it.

Internal hyperlinks

These are "scrolling hyperlinks" of the form:

```
<A HREF="#anchor">link</A>
```

Most people use them as ways of jumping about within the "current" page. In actuality, there is no difference between these and other scrolling hyperlinks, *and there is no guarantee that the browser won't attempt to reload the current page.*

Scrolling forms

These are HTML forms which have been crafted to ask the browser to scroll to some anchor point in the page that is returned by the form submission.

How to generate scrolling links/forms

Scrolling links. The way we get a simple hyperlink to "jump" to an anchor inside the resulting CGI-generated page is quite simple: we just make sure the hyperlink looks like this:

```
script#ANCHOR?query
```

Notice that the ANCHOR *precedes* the query. We have a function, **IW::Util::script_url()**, which assembles URLS like this automatically. Any code generating such a link should use that function. Future browsers might possibly change this syntax, and we want to avoid having to search through the source trying to find "hand-made" links.

Scrolling forms. To get a browser to submit a form and then scroll to an anchor inside the resulting page requires a little black magic, since the standard doesn't go into much detail about this. We have determined pragmatically that the following HTML works:

```
<FORM METHOD="POST" ACTION="script/pathinfo#ANCHOR">
...
</FORM>
```

For this case, **IW::Util::script_action()** was created to assemble ACTIONs like this reliably. Again, you should use this function instead of hand-assembling such a link.

Missing destinations

Before you use scrolling links/forms, keep in mind the following

---

Warning: If you create a link/form which points inside a page, *that page must contain the appropriate anchor point.* If not, the browser *might* go to the top of the page (early browsers), or it *might* go to the absolute bottom (recent Netscapes). Odds are, someone will be unhappy and confused.

Our current anchor point strategy is as follows:

1. Suppose we want to create a FORM which points at some search **category**, say "GEOGRAPHIC_REGION", in the main search form. That category may or may not be shown on the search form depending on the type of query! So we create the form, *but we always anchor it to the special anchor name "SCROLL",* giving us a FORM tag like:

   ```
   <FORM METHOD="POST" ACTION="nph-ims.cgi#SCROLL">
   ```

   In addition, we make certain that the form contains the hidden "Scroll to Category" tag:

   ```
   sc = 1
   ```

   And, of course, the category we want to scroll to must be in there somewhere, so we assume that the standard "category" tag will be used:

   ```
   category = GEOGRAPHIC_REGION
   ```

   Note: ModeSLCT does all this for us automatically when we draw a "popup" page.

2. The form is submitted.

3. When the Gateway goes to draw a page, the *default* activity is to place the following at the very top, around the title:

   ```
   <A NAME="SCROLL">...</A>
   ```

   This ensures that if we jump to a page we hadn't predicted (like a dying gasp), there will be an anchor right at the top which "catches" a submit-and-scroll.

   However, if we don't want this, we simply make sure that the **internal_anchor** attribute of the **IW::App** object has been set to some appropriate true value before the page has been started: this suppresses the default anchor up top. And in this case, what better value to use than the name of the category?

   ```
   # If we should scroll-to-category, then suppress the default top anchor:
   if ($REQ->param("sc")) {            # scroll to category
       $APP->internal_anchor($REQ->param('category'));   # true
   }

   # Now, the default anchor won't be output!
   $APP->start_page(Title => 'Search Form');
   ```

   Now that we've suppressed the default anchor, we had better make sure that we *do* output an internal anchor, or we will again be at the mercy of the browser's default (unpredictable) behavior!

4. As we output each category in the main search form, we check to see whether this is the category we want to scroll to. If it is, we output the anchor. Then, to prevent the accidental creation of two anchors, we clear the **internal_anchor** attribute:

   ```
   if ($APP->internal_anchor eq $current_category) {
       $HTML -> A(NAME=>'SCROLL') -> t(' ') -> _A;
       $APP->internal_anchor(undef);      # don't link twice
   }
   ```

   Notice that if we use this code, the anchor will never be output more than once, even if this code is invoked multiple times.

   Warning: **Gotcha!** What about the DELETE action for things like the extended search criteria? We can't link to a category if we plan to delete it!
   We handle this by doing a quick run-through of all the extended criteria we plan to output. If the desired category will be output, it's business as usual. If not, then we place the anchor at the top of the "extended search criteria" section, which is as close as we can get.

Why we don't use internal hyperlinks more

Some older versions of Netscape were observed to have a *very* nasty bug:

Warning: If a user followed an internal hyperlink on an HTML page with a POST-able form in it, and then clicked a SUBMIT button, the first SUBMIT will appear to do nothing more than jump to the top of the page. If the SUBMIT button is then clicked again, the form will be submitted, *but without any of the CGI parameters.*

The way this manifested itself in the Web Gateway was particularly lethal: since one of the form parameters is invariably the session id (**sid**), the user either (1) got a **dying gasp** which said that their session doesn't exist, or (2) the system treated it as a "new request" and quietly placed them into the search form of a *completely new* session.

Since this bug came to light, we began to avoid using any hyperlinks that point "inside" any pages which contain submittable forms.

Note: This may seem like overkill, but given the severity of the manifestation it's a sound policy: bugs with potentially severe results have to be avoided. we can't guarantee that someone with a browser which might trigger them might use the syste, and there's no point in alienating *any* user if it can be avoided.

This bug has only been observed with *internal hyperlinks.* The bug does not seem to manifest itself with anchors of the form:

```
<FORM ACTION="script#SCROLL">
```

Hence, screens like the geoselection popup continue to use submit-and-scroll.

2.2.7 Keeping search results

Search results consist of both *dataset and granule information*, which describe the actual data itself, and *packaging information* which is used solely to put together an order.

The **imsclient** results (in `client.stat` and `client.obj`) are ingested by **imsrunclient**, which stores the information in `results.gdb` like this:

-
-
-
-

This allows us to easily access a granule, given a dataset, and a package, given a granule.

Before each client search activity, we clear out the current results from the session directory.

After each client search activity, we merge the results of the last search with the contents of `results.gdb`.

Do this by copying every `key=value` pair in the **new results** into the corresponding GDB file entry of `results.gdb`. This will overwrite an old record if it's already there, and simply add a new one if it isn't.

Once this is done, the new `results.gdb` is copied back up into the session directory, becoming the **current results**. This is so that we always have a `results.gdb` pair on hand so that **imsgdbdump** may be called at any time.

> ***Note:*** We do this stage in as atomic a fashion as possible; it's vital that we always have a good copy of the merged results! Rememeber, it's possible for a "kill" to be delivered to "imsrunclient" at any time; we don't want the merged results to be in a bad state!

Warning: Probably, any activity that uses **imsgdbdump** should look to copy the **merged results** up to the session directory first.
For example, it's possible that the user could start a search (clearing the session directory), kill it, and then try to do a order options selection. Having **imsgdbdump** (and maybe **imsclient**) be savvy about the merged files will at least ensure that they can "set themselves up" before attempting any operation that requires the GDB files.

2.3 Subsystems

Want to know how a particular part, function, or feature of the Web Gateway works? Here are the major (and minor) subsystems of the Gateway.

2.3.1 Bootstrapping

This section describes what happens when a request the Web Gateway comes in from a browser. It provides an overview of how IW::App, IW::Mode, and IU::Session are used.

Components

Here are the basic players for setup:

```
IW::App          Manage the CGI application setup/shutdown, as well as
                 basic page begin/end, and the dying gasp.
                 One instance: the $APP object.

IW::Mode         Manage the transfer of control to a particular mode.
                 Affiliated with the 'mode' CGI tag.

IU::Session      Link the application to session-specific resources.
                 Affiliated with the 'sid' CGI tag.
                 One instance: the $SESSION object.
```

Input:

```
      IW::CGI         Parse the CGI input tags.
                      One instance: the $REQ object.

And output:
```

```
IW::Reply        Begin/end the HTTP reply which contains the
                 HTML document.  Also manages pagebuffering.
```

	`One instance: the $REPLY object.`
`IW::ReplyHead`	`The header of an HTTP reply, as output by IW::Reply.`
`IW::HTML`	`A facility for outputting HTML into the currently-open HTTP reply stream (usually set up by IW::Reply.)` `Generally one main instance: the $HTML object.`

How it works

1. **WEB SERVER RUNS THE BOOTSTRAP SCRIPT.**

   A CGI request comes into the Web Gateway script, **nph-ims.cgi**. This is a Perl script, which is started up. It loads virtually no other modules at compile time; instead, it does the following:

   (a) **Make the environment variables safe** (like PATH and IFS).

   (b) **Define the bootstrap error handler** called &::dying_gasp(), which will be used until the *real* gasp handler can be loaded. It outputs a simple text document.

   (c) **Load and launch.** Based on the name this script was invoked with (usually `nph-ims.cgi`), the name of a corresponding Perl module is determined (in this case, `nph-ims.pm`). That module is then `require`'d inside an `eval` block; any problems, and we use the bootstrap error handler. Otherwise, control is transferred to that module...

2. **BOOTSTRAP SCRIPT SETS UP THE APPLICATION.**

   Assuming we just loaded `nph-ims.pm`, that means that *minimally*, the basic IW::App and IW::Mode modules (and anything that they in turn load) were brought in successfuly. We continue:

   The `go` class method of IW::App is invoked...

   ```
   go IW::App \&main;
   ```

   ...which tells IW::App to create a new instance of itself (`$APP`), send that instance an `init()` message, and then `eval` the callback subroutine supplied to go() (in this case, \&main).

   Let's see what happens in the init() routine first:

   (a) **Signal handlers** are installed to catch any signals we're interested in. Some are fatal, others (like SIGPIPE) are ignored.

   (b) **A per-invocation log** is opened via a tmpfile. Everything that happens in this request is logged there, for possible inclusion in dying gasps.

   (c) **A new HTML output stream** is established on STDOUT. This is `$HTML`, an instance of IW::HTML.
   (d) **A new CGI interface object** is created. This is `$REQ`, an instance of IW::CGI::Request. It parses the CGI input automatically on creation.
   (e) **A new CGI reply object** is created. This is `$REPLY`, an instance of IW::Reply.
   (f) Debugging (on/off) is decided upon, the HTML stream is told about where it can find the CGI tags, etc.
   (g) Finally, we check to see whether or not the system is currently down.

3. **THE DESIRED MODE IS LOADED AND INITIALIZED.**

   Now that initialization of $APP is done, we invoke the callback routine that was sent into IW::App::go. Assuming this is `nph-ims.cgi`, the callback routine simply does this:

```
sub main {
   my $mode = ($REQ->param('mode') ? $REQ->param('mode') : 'SRCHFORM');
   handle IW::Mode $mode;
}
```

   In other words, figure out the current mode (defaulting it to 'SRCHFORM'), and then send class IW::Mode the message `handle($mode)`.

   IW::App::go will run this inside of an `eval`: if anything goes wrong (e.g., the mode or one of the classes it depends on can't be loaded; or the mode runs but throws a fatal exception), IW::App will drop the currently-buffered output and issue a dying gasp.

   Let's see what IW::Mode::handle() does:

   (a) Grab the name of the mode to run, and untaint it.
   (b) Turn it into the name of a Perl module (e.g., `"SRCHFORM"` becomes `"IW::ModeSRCHFORM"`. *This module must be a subclass of IW::Mode!*
   (c) Require that module. This loads in the code that the mode depends on.
   (d) Create a new instance of that mode's class.
   (e) Send that new instance the message `go()`. This is the invocation of the go() method usually found at the bottoms of all IW::ModeXXX modules.

4. **THE MODE RUNS AND EXECUTES THE ACTUAL ACTION.**

   Normally, the mode's go() method will do the following:

   (a) Establish a new session, or resume an existing session given in the **sid** CGI tag. IW::App is entreated to do this; the session, once established, is placed in the $SESSION object.

(b) Send itself the `go_submode()`, which will interpret the **submode** CGI tag as a name of a method to be invoked on the current object. For example, if `submode = "OK"`, then the mode object is sent the message `submode_OK()`. In absence of any submode tag, `submode_DEFAULT()` is assumed.

2.3.2 Browse

Introduction

Browse images are "quick look" images of data granules, obtained on-demand from the DAACs.

To reduce unnecessary network transactions (usually by a single user, or by a group of users with similar agendas), a persistent cache of browse products is kept on the gateway host. The preferred/maximum size of this browse cache may be set in the **SETUP** file.

There are two types of browse requests/products:

Integrated browse

These requests are sent via the V0 IMS protocol to a DAAC to fetch the browse product for a single granule. The browse product is sent back in real time as an HDF file, usually containing several images.

FTP browse

These requests are also sent via the V0 IMS protocol to a DAAC to fetch the browse product for a single granule. However, the DAAC has 24 hours to place the product somewhere in an anonymous FTP area and email the user when it is available.

How it works

1. When a user requests an integrated browse image for a granule, the Web Gateway first takes the **granule key**, and looks up that granule in its "browse file cache".
2. *If an entry in the cache is present under that granule key,* a document is returned to the user which displays the browse product in some sensible fashion.
3. *Otherwise,* a V0 IMS request is made to fetch that browse product via **imsclient**. If it succeeds, a single HDF file (the browse product) will be deposited in the **session directory**.
4. The **hdf2gif** program is run to extract all the GIFs from that HDF file.

5. The HDF and all extracted GIFs are moved into the browse file cache under a single entry key: the granule key (the browse file cache stores a list of one or more file under a single entry key).
6. Now that the files are cached, we go to step 2.

2.3.3 Criterion objects (for query and subsetting)

Introduction

In order to allow the data centers to send back special "subsetting specifications" which would allow the user to enter various types of subsetting criteria during package-selection, **ODL** was created to support it.

The Web Gateway takes this ODL and converts it into objects (which, as of 1.8, are some appropriate subclass of class IU::Criterion). Such objects know how to do the following:

- **Render themselves** as HTML (as part of a form).
- **Read the user input** in from the CGI tags defined when they rendered themselves.
- **Validate** themselves.
- **Output themselves** as ODL, usually for queries or data orders.

For example, the following ODL would become an instance of **IU::Criterion::InputNum**:

```
GROUP = SPECIALIZED_CRITERIA
  CRITERIA_NAME = "MaxGamma"
  CRITERIA_TYPE = "REAL"
  COMMENT = "Brief explanation of MaxGamma"
  CRITERIA_MIN = 0.0
  CRITERIA_MAX = 10.0
  SELECT_NUM = ONE
END_GROUP = SPECIALIZED_CRITERIA
```

And this would become an instance of **IU::Criterion::SelectMany**:

```
GROUP = SPECIALIZED_CRITERIA
  CRITERIA_NAME = "Channel"
  CRITERIA_TYPE = "STRING"
  COMMENT = ("Select one or more of the following instrument channels")
  CRITERIA_VALUE = ("19V", "19H", "36V", "36H")
  SELECT_NUM = MANY
  REQUIRED = Y
END_GROUP = SPECIALIZED_CRITERIA
```

In 1.8, these objects were used to solve the similar problem of extended search criteria.

Overview of classes

Criterion classes

The following classes represent query/subsetting criteria:

```
IU::Criterion            - abstract search criterion
IU::Criterion::Geo       - a geographic region
IU::Criterion::InputNum  - a single number/date in a query
IU::Criterion::InputRange - a range in a query
IU::Criterion::InputText  - a text field in a query
IU::Criterion::SelectMany - a multiple-selection list in a query
IU::Criterion::SelectOne  - single-selection menu in a query
```

Users of criteria

Collections of criteria are used by the following:

```
IU::Query           - a query created through the search form
IU::SubsetSpec      - subsetting criteria in the "choose package" page
```

2.3.4 Definition

Definitions are brief descriptions of the various elements upon which a search can be based.

When a definition page is requested, two parameters are passed to the definition page generation code: the definition type, and the definition item. The type parameter specifies the class of thing for which a definition is to be provided, and currently must be one of:

```
campaign
dataset_id
data_center
parameter
sensor
source
```

The item parameter specifies which element of the previously specified type is to be defined. This varies greatly with the type, but really does not matter to the code. Typical examples would be type=data_center, item=GSFC; or type=parameter, item=CLOUD COVER.

The definitions themselves are maintained in a series of "definition files", which are currently stored on harp.gsfc.nasa.gov, which is referenced through the **ImsDefnUrl** variable. There is a separate file for each definition. Each definition file is a complete HTML file that provides the definition text for the single definition specified by a pair of type and item values.

There are two problems with including the definition files directly in a web page: they contain a <TITLE> tag, which by has no business occurring in the body of a document; and they reference a "thin_blue_line" .gif file, which sounds too much like a cop show, and which isn't in the relative location specified in the definition files.

The serve_defn() routine makes a few simple-minded assumptions about the format of the definition files in order to avoid having to parse these files. The body is assumed to begin with the first <H[1-6]> header tag and everything preceding the first such tag is striped out; and all <IMG> tags are assumed to be horizontal lines that get replaced with a <HR> horizontal rule tag.

To generate a definition page, the associated definition file must be retrieved. To retrieve the definition file, the software attempts an HTTP transfer from **ImsDefnUrl**, the definition file directory on harp, and if found, is then served. If a definition file cannot be obtained, the software serves up a pretty error page that indicates that it's not our fault that the definition file is missing.

Regardless of the outcome an SCRS-format **event log** entry is made that records the type, item, and status of the request.

2.3.5 Dependent valids

This is part of the "search form" subsystem.

Valids

Certain of the search categories – PARAMETER, SOURCE_NAME, SENSOR_NAME – have valids associated with them: these are merely the values defined as being legal. For example, the valids for PARAMETER include:

```
TEMPERATURE
PRECIPITATION
PRECIPITATION RATE
...
```

Specifying a PARAMETER in your query outside this set of valids would be pointless, since (by definition) no data center would have such data archived.

Interdependencies between valids

Since not all combinations are sensible (e.g., CAMPAIGN = "DESERT_MAPPING" and PARAMETER = "SNOW_DEPTH"), choosing some valids in one category will limit the effective choices in another category. The IMS software is aware of many such interdependencies, and the Web Gateway makes them visible to the user to help avoid nonsense queries.

For example, suppose that

```
Data with parameter...        Is only gathered by sensors...
```

```
------------------------------------------------------------
"ADIABATIC TENDENCY"           "ANALYSIS"
"SEA ICE CLASSIFICATIONS"      "SAR", "RADAR", and "AMI-SAR"
```

Step 1: Go to the parameter-selection page and choose the following two parameters:

```
PARAMETER = ("ADIABATIC TENDENCY", "SEA ICE CLASSIFICATIONS")
```

Step 2: Go to the sensor-selection page. You should see a short list:

```
ANALYSIS
AMI-SAR
RADAR
SAR
```

Step 3: Choose "ANALYSIS" for the sensor. This will invalidate parameter "SEA INC CLASSIFICATIONS", so although your query would be:

```
PARAMETER = ("ADIABATIC TENDENCY", "SEA ICE CLASSIFICATIONS")
SENSOR_NAME = ("ANALYSIS")
```

It would effectively be reduced to:

```
PARAMETER = ("ADIABATIC TENDENCY")
SENSOR_NAME = ("ANALYSIS")
```

This invalidation of "PRECIPITATION RATE", by choosing a valid which is incompatible with it, is shown in the Web Gateway by placing a "*" in front of the name of the "invalid valid". So...

Step 4: Go to the parameter-selection page. You should see your current choices as...

```
ADIABATIC TENDENCY
* SEA ICE CLASSIFICATIONS
```

Variants

In the special case of extended search criteria, the interdependent "valids" are not really user criteria at all, but *the names of complex criteria.* These names are called *variants*.

There is a strong parallel between the *valids of core categories* and the *variants of extended categories*. Indeed, just as the dependent-valids mechanism is used to determine which valids of a given core category are compatible with your current query, so it is *also* used to determine which variants of a given extended category are compatible.

For example, suppose that extended category "Cloud Cover" has three different ways of specifying cloud cover:

```
Variant       Description
---------------------------------------------------------------
PERCENT       Enter a single number, 0 to 100.
QUALITY       Select one from a list: "Hazy", "Overcast", etc.
TYPE          Select one or more from: "Cumulus", "Nimbus", etc.
```

Then here is a comparison to show the parallel:

```
                +----------------+------------------
       Category |  SOURCE_NAME   |  CLOUD COVER
                +----------------+------------------
Category type   |  core          |  extended
                +----------------+------------------
Dependencies    |  valids        |  variants
are between...  |                |
                +----------------+------------------
For example...  |  LANDSAT-1     |  PERCENT
                |  LANDSAT-2     |  QUALITY
                |  LANDSAT-3     |  TYPE
                +----------------+------------------
```

It helps if you think of each core valid as being a special kind of variant, one with just a boolean value of true (i.e., "yes, I am part of the query"). For efficiency, they are not implemented in IU::Query this way, but it's a good way to think about it.

Filters

Since there are quite a lot of valids for some categories (e.g., PARAMETER has several hundred), the valids in most categories have been organized into groups called "filters".

> For example, category PARAMETER might have a filter called "SNOW" which acts as a synonym for the valids "SNOW/ICE", "SNOW COVER", "MEAN SNOW DEPTH", etc.

So instead of just selecting valids from a large list, a user can select one or more filters instead, and update the dependent-valids-selection form to see a much-shorter list.

Patterns

Like **filters**, "patterns" are used to aid the user in paring down long lists of valids. In reality, this is nothing more than pattern matching, where "*" is a wildcard.

> For example, if the user types in a pattern "SNOW*" and updates the page, it will act as a synonym for the valids "SNOW/ICE", "SNOW COVER", "SNOWFLAKES", etc.

How it works

The interface. Since the dependent valids mechanism is part of the "IK" C layer of the software, it is necessary to invoke the core functionality through a C program compiled with the IK code. That program is called **imsdepval**. It has one interface class:

```
        IW::DepValid    -- interface to dependent-valids mechanism
```

The algorithm. Basically, you create an instance of this class, "set" the values for all known categories, and then "get" the valids for the category you're interested in. The act of "getting" will invoke **imsdepval**, supplying it with:

- The current categories, and the selected valids (or variants) for each one.
- The filters and patterns in effect, if any.
- The name of the category for which valids – or variants, or filters – are desired.

The results. Internally, the results come back as **chunks**, which are turned into lists of valids, or filters, or variant names, or criterion objects as needed.

Queries. Since the dependent-valids mechanism is most-often used in the context of the current query, the **IU::Query** class has been given a method that creates a dependent-valids interface object representing the current query contraints. This ability to set up an IW::DepValid object with a single method invocation is a real timesaver.

2.3.6 Error handling

In a perfect world, we wouldn't need to consider this at all. However, ours is not a perfect world...

Exception handling

The system uses basic **exception-based error handling**. In the most general terms, if an error is detected, the Perl code detecting simply raises a fatal exception (we provide a function, `puke()`, which takes an error string and an optional error number, and raises this exception). Normally, a raised exception results in a **dying gasp** being sent; however, Perl's "eval" mechanism allows exceptions to be trapped (like a "try-catch" block) and handled in some other way. Currently, we only use strings (in Perl's special **$@** variable) to pass back information; more advanced error-handling can be implemented, but this is material for later development,

The principle benefit is that exception-based error handling is more reliable, since it does not require that code explicitly check all return values. It is common (and preferred) in modern languages, like Java.

Abortable output (page-buffering)

Reasonable methods of error logging/handling continue to be one of the design priorities: this has made the bugs introduced by rapid-pace development easier to track down.

How page-buffering works The basic algorithm - employed for each HTML page to be output - is as follows:

1. The STDOUT filehandle - connected to the HTTP client - is duplicated into a special private filehandle, which we'll call CLIENT.
2. STDOUT itself is redirected to a temporary file [1]. As of v.1.5, this is a `tmpfile()`: it is automatically deleted when the process terminates. This is our page-buffer.
3. All intended STDOUT "output" by the CGI script is now accumulated automatically into the page-buffer file.
4. **If all goes well,** the CGI script invokes a method which rewinds the page-buffer and writes its contents to the "real" STDOUT [2]. However...
5. **If an error is encountered** at any point, the page-buffer's contents are discarded, and an HTML 'error document' (commonly called the "**dying gasp**") is prepared instead. This error document is *also* page-buffered; in the unlikely event of something going wrong while preparing it, an exception is raised and a non-page-buffered, plain-text error document is output.

The benefit of this scheme is that we can prepare all needed output as usual, but still fail gracefully if an error occurs at any point during that preparation.

The drawback is that it may take a little longer for large documents to begin to "appear" at the user's browser, since they are not transmitted until they are completely prepared. A

In the Web Gateway system, the redirection of output is done automatically for all CGI code; the "flushing" of the completed reply is done by the method that outputs the standard page footer, so developers hardly ever need to know that page-buffering is taking place.

The "dying gasp"

This is a special document which is output in the case of a fatal error. It contains

- A description of the error
- A suggestion as to the action to be taken

- Some basic traceback information

More serious dying gasps need to send other debugging information to the developers, but we don't want to be showing a lot of sensitive system information (disk layouts, operating system info, etc.) that can be useful to the developers to just any random user.

If the error is due to something **benign,** like bad user input, the document is already very short. This helps to avoid (1) scaring the user into thinking that the system "crashed", and (2) overwhelming the user with too much information.

However, in the event of a really **serious** error (e.g., an internal logic error, a site misconfiguration, etc.), we need a longer error document, but we don't want to make it into a security risk. What to do?

We devised a way of "marking" the end of the information we want to send to the user bt adding two new methods to **Reply**:

set_truncation_point

This method records the current position that we're at in the pagebuffer.

truncate_output

This method effectively resets the end of the pagebuffer to the truncation point.

We mark the truncation point after the basic informationis output, and then mail the *complete* pagebuffer to the **SiteAdminEmail** address provided in **SETUP**. We then call **truncate_output()** and output a new footer, effectively truncating the page before exposing any sensitive data, but still sending valid HTML to the browser.

2.3.7 Geographic selection

This is part of the "search form" subsystem.

Overview of classes

Support for geoselection screens on search form.

The geoselection screen is complicated enough that it requires its own class trees: **IW::GeoSelect** is the namespace for classes that are used by mode SLCTGEO to perform a particular "thread" of user interaction (e.g., IW::GeoSelect::PolarMap manages polar map selection).

```
IW::GeoSelect::Base            - abstract class for geoselection "widgets"
IW::GeoSelect::PathRowTypein  - do geoselection by typing in paths/rows
IW::GeoSelect::PolarMap       - do geoselection from a polar map
IW::GeoSelect::RangeMap       - do geoselection from a range map
IW::GeoSelect::RangeMapTypein - do geoselection by typing in range
```

Support for geoselection type-in.

Where geoselection is done by a simple type-in form, IW::GeoSelect sits on top of even-more-basic classes, **IW::GeoEntry** (e.g., IW::GeoEntry::Range draws HTML form elements for entering ranges).

```
IW::GeoEntry::Base     - abstract class for "entry" boxes in map selection
IW::GeoEntry::PathRow  - draw "path/row" entry boxes in map selection
IW::GeoEntry::Range    - draw "range" entry boxes in map selection
```

Representing a geographic region.

All these classes sit on top of the actual geocoverage objects, under **IU::Geo**. Each represents a particular kind of geographic region. Notice the new addition of the degenerate **IU::Geo::Global** to represent global coverage.

```
IU::Geo::Base        - an abstract geographic region
IU::Geo::Global      - global granules only
IU::Geo::PathRow     - path-row-based geographic search, like a PATH_ROW_LOC
IU::Geo::Point       - point search region, like a POINT_LOC
IU::Geo::Polygon     - polygonal region, like a POLYGON_LOC
IU::Geo::Range       - rectangular search region, like a RANGE_LOC
```

Conversion between geographic regions.

The managing of conversion from WRS path-row ranges (IU::Geo::PathRow) to lat-lon ranges (IU::Geo::Range) managed by **IU::Geo::WRS**.

```
IU::Geo::WRS         - interface to the World Reference System mechanisms
```

How it works

Everything is done inside of ModeSLCTGEO. That mode uses the standard method, `print_form()`, for rending, which means that the start and end of the page is already taken care of. We just need to concern ourselves with rendering (and interpreting) the stuff in the middle.

1. When you first come to this mode, you will be presented with a list of all possible ways to select a geographic range. Some will involve **map selection**, some will involve simple **type-in**, and a few will involve **no further action.** If you select one that requires no further action, you're done.
2. *The name of the geoselection method you last used* (if any) will be used to create an instance of some subclass of **IW::GeoSelect**. This is the "last geoselector" that was displayed. *Note: currently in the code, we call it a "map" because all geoselection used to be via maps. Now there are type-in boxes and maybe applets, so "geoselector" would probably be a better term.*

   User input (e.g., mouse clicks) on this geoselector are processed by the object's `get_input()` method. The "state" (current geographic region selected, whether 0, 1, or 2 points were selected, etc.) is updated appropriately.

3. *The name of the geoselection method to use now* will be used to create an instance of some subclass of IW::GeoSelect. This is the "next geoselector" to display.

   This "next geoselector" is created in an interesting way: instead of just creating a new instance which is reset to the initial state, the object representing the "last geoselector" is told to `switch_to()` the name of the new geoselector. The switch_to() instance method is a constructor; it returns a new instance of the desired class as follows:

   - If the two geoselectors have the same "basic type" (e.g, range-selection using "Atlas" map, and range-selection using "Coastline" map), then the new instance will have the same "state" as the "last geoselector". That means that switching from an "Altas" to a "Coastline" map after having selected your first point results in a "next geoselector" object *which still has that point selected.*
   - Otherwise, the new instance is simply created empty, at the initial state.

4. The "next geoselector" object is told to `draw()` itself.

   If this "geoselector" is a kind of "type-in" selector (e.g., "Type in Path/Row", then its draw() method will employ some subclass of **IW::GeoEntry**. Instances of **IW::GeoEntry** know how to render themselves as HTML form elements.

5. When the user submits this form, the "next geoselector" we just drew will be known to be the "last geoselector" that was drawn. If OK is chosen, go to step 6; if something that leads us back to more GEOSLCT is chosen, go to step 2.

6. The `ok()` method of SLCTGEO has been triggered. We simply look at a CGI parameter which tells us what kind of CGI tags we should be looking for, and instantiate an **IU::Geo** subclass from the CGI tags. For example: selecting from a range map means that the current region is encoded in the CGI parameters as the tags for an IU::Geo::Range.

   That object is then set as the current Query region.

Drawing maps

The GIFs for the maps (range, polar) are currently created dynamically by the **imsmap** and **imsorthomap** CGI scripts. The map they use, the coordinates displayed, etc., are all sent in via CGI params in the <IMG SRC=""> markup.

If the GIFs get more complicated, another solution (based on writing the GIF to the session's "pub" directory and serving up a simple URL to it) may need to be tried.

2.3.8 Guide

Introduction

Guide searches are a combination of WAIS searches and a special XML-based text search for "guide" documents. These are (somewhat) standardized documents that have been placed on the Web by various data centers.

Guide searches are performed a little differently than normal V0 searches: they involve doing parallel WAIS and XML-based searches. The XML-based searches go to a special URL defined in **SETUP** by Li<SpidexUrl>; the WAIS searches go directly to DAACs.

At present, we are in a transitional stage between all-WAIS searches and all-XML searches. This design document reflects the current status, and includes a section describing the future as it is currently planned.

How it works

If invoked to do a "search" (using GUIDESRCH, the default submode), this **ModeGUIDE** will construct both a WAIS and a XML query based on either...

- *The CGI tags "doctype" and "item".* This is for guide searches done by clicking on the "DI" button next to a listed parameter, sensor, etc., or by asking for Detailed Info from a dependent valids page.
- *An IU::Query object explicitly passed into the mode.* This is for performing guide searches which masquerade as full-fledged searches available on the search form.

Overall design concept **ModeGUIDE** is actually just a skeleton which provides process management and output services; the real work is done by **ModeGUIDEOLD** and **ModeGUIDEXML**.

ModeGUIDE handles a request as follows:

1. Discard any old guide search results.
2. Create results files and process logs for the subprocesses to be invoked.
3. Determine which DAACs are supported by the XML-based search, and save this list for use by the WAIS search (these DAACs will be skipped by WAIS).
4. The processes are forked off using **IU::Spawn**, and are monitored by **ModeGUIDE** thereafter in the standard manner (using **IU::Spawn::attach**()).
5. When the processes complete, the use may choose to view the results either by DAAC or by "relevance".

The WAIS search The query will be executed by **waisq**, and the results turned into HTML. If any documents were "hit", they are displayed in the HTML as hyperlinks which will invoke mode GUIDE with submode GUIDEGET, with an additional CGI tag indicating which "hit" the user selected.

If the GUIDEGET submode is used (e.g., if the user has gotten back results, and clicks on one of the "hits"), the guide document (which will be in HTML formant) is fetched from the remote guide server, and displayed to the user.

> ***Note:*** Because WAIS does not use anything like URIs or absolute document identifiers, this necessitates doing the original search over again, telling WAIS which "hit" (e.g., 4th of the 20 hits) we want to "fetch". Thus, we keep every WAIS query around in a file.

The XML search The XML search is based on **Isite**; the XML search code simply creates an Isite query and POSTs it (using **LWP**) to the **SpidexUrl** defined in **SETUP**.

The response is XML which describes the hits found by **Isearch**. These are relatively straightforward, cotaining a description of the hit, the DAAC it came from and a numerical "relevance" of mysterious provenance. Since this relevance is not easily quantifiable to the users, we have chosen simply to categorize the results from "Very High" to "Very Low" relevance; this eliminates arguments about why document A got a score 3.7 points higher than document B (especially since no one seems to be able to explain the scores at all anyway).

The "doctype" and "item"

The "item" is what you normally think of as the search string. For example, "EDC".

The "doctype" is to filter the guide documents so that the user only sees extremely relevant ones. To begin with, each guide document has a particular "type"... for example, although many guide documents might mention data center "EDC", *only one* guide document is the definitive "thing-to-read" to find out about EDC. That "definitive" guide document has been indexed with:

```
doctype=Data_Center
```

Thus, a search for:

```
item=EDC
```

Will turn *all* guide document which mention EDC, but:

```
item=EDC and doctype=Data_Center
```

Should turn up *exactly one* guide document; the definitive one.

Warning: Experience has shown that some data centers do not function quite this way, so it is possible for the latter query to return 0 or >1 hits in certain cases.

The "one-hit" hack

If a GUIDE search hits exactly one document, the "get" and display of that document is done automatically, without necessitating a user click.

This was introduced because certain guide documents are the "definitive" documents for a particular subject, and the indexing scheme has been designed so that particular combinations of "doctype" and "item" should always return a single hit.

2.3.9 Help

"Dynamic" serving of "static" help pages

Throughout the gateway (on every single screen, actually) there are "help" links. Originally, these were linked to the appropriate static HTML page in the User Manual. However, it was observed to be very easy for users to get lost in the User Manual and have a difficult time going "back" to the interface.

The solution was to *serve* the pages of the User Manual *dynamically*, modifying the outgoing HTML so that...

- The link "Help for this page" is located at the top of each page.
- All URLs that would normally point to other "user manual" pages point instead back to the Web Gateway script's HELP mode, with the context set to the appropriate page.
- All URLs that would normally point to things like GIFs, JPEGs, external pages, etc. point to those documents *via the standard HTTPD* at the site.

This all works fairly quickly.

Help context

You will notice that all you need to generate a help URL in the Gateway code is the help context. Basically, this is a short text string that identifies a page in the user manual. For example, the context `"srchform"` refers to the following page (relative to the top-level directory of the Gateway):

```
./docs/manuals/imsuser-srchform.html
```

Adding help pages

To add help pages, you'll have to add IPOD text to the User Manual. Go to `./docs/ipod`, where the IPOD source for the User Manual is located.

2.3.10 Logging

There are two main types of logging that take place:

Trace logs

These are the general activity logs for a program, wherein **debug, informative, warning,** and **error** messages go. They have a compact format, intended primarily for visual scanning.

Event logs

These are the permanent logs used to gather **statistics** on the system, and to debug long-dead sessions. They have a special format, standardized by another task, which we must adhere to.

Trace log

There are two kinds of "trace logging" that take place:

CGI trace (temporary, per-request)

In the CGI script, the activities that occur during execution are logged to a temporary log file, established via the `tmpfile()` mechanism in much the same way as the **page-buffer** file. If all goes well, the file is discarded after output is sent to the user; however, if a problem occurs, the file's contents are spliced into the **dying gasp**.

Run trace (persistent but overwritten)

In some of the programs (like **imsrunclient**), the activities that occur during each run are logged to a disk file in the **session directory**. The file persists after the process exits, but each run of the program by this user overwrites the log file from the previous run.

Writing trace logs **To write trace log entries**, use the **IU::Log** module. This will provide the following subroutines...

```
debug     debug message: generally intended for developer's eyes
inform    informational: something significant happened
warning   warning: an unexpected (but not erroneous) thing happened
error     error: something bad (but possibly not fatal) happened
```

If the first argument appears to be a 3-digit number, it is taken to be the "message number". Normally, this is used for error messages only; see **IU::Err** for the table of error numbers.

Trace log format The format of the trace logs was designed for compactness, and to allow for easy visual scanning. The following are used in the leftmost column to indicate message type:

D	debug message
II	informational
WWW	warning (no, not World Wide Web)
EEEE	error

Here's an example:

```
II   000 1996/09/13 16:33:08 /IW/App.pm 334
     Logging begun
II   000 1996/09/13 16:33:08 /IW/Mode.pm 73
     Loaded IW::ModeSEARCH
D    000 1996/09/13 16:33:08 /IU/Store.pm 163
     obtaining lock 2 on +</home/imswww/dev/docs/imswww-usr/842650348-904
D    000 1996/09/13 16:33:08 /IU/Store.pm 172
     tie to session.state, 2, 384 succeeded
D    000 1996/09/13 16:33:08 /IW/App.pm 596
     setting up HTML output
EEEE 516 1996/09/13 16:33:08 /perl/IU.pm 109
     You need to select a geographic option!... Puking at
     /home/imswww/dev/lib/bin/../perl/IW/Query.pm l.507
EEEE 000 1996/09/13 16:33:08 /IW/Reply.pm 353
     output aborted!
EEEE 000 1996/09/13 16:33:08 /IW/App.pm 1054
     Fatal error: sending dying gasp...
```

Note how the errors stand out visually, and how each message is displayed with its date, time, file and line number.

Event log

Writing/reading event logs **To write event log entries,** use the **IU::Event** module. This module uses advisory locking to prevent accidental collision of entries when multiple processes are running.

Example:

```
# record what the user picked so we can study their behavior.
my $type_in_string;
my @event_type_in;
foreach $type_in_string (@type_in) {
   push(@event_type_in, 'type_in'=>$type_in_string);
}
```

```
my $pick_list_string;   my @event_pick_list;
foreach $pick_list_string (@pick_list) {
   push(@event_pick_list, 'pick_list'=>$pick_list_string);   }

ims_event('pattern',
          'num_matches'=>int(@matches),
          @event_type_in,
          @event_pick_list);
```

To parse event log entries, use the **IU::EventEntry** module.

Event log rollover In some of the more-important utility programs (like **imscleanup**), the events that occur during each run are logged to a disk file which is *appended to* on each run. To prevent such files from getting too large, a very simple strategy is employed: the name of the file that is appended to is qualified by the current date, typically as `YYYY-MM-DD`. That way:

- Each log file covers one day or less of time.
- Logs are shown sorted by time automatically.
- Older/newer logs can easily be detected, moved, or deleted.

Event log format The event logs use a SCRS-standard *parameter=value* format, standardized by another task, and designed for extensibility and expressiveness. A typical log entry looks like this *(* means that it's a standard keyword)*:

```
(*)    daac=v0ieww00                -- id of WG server
(*)    event=inventory              -- event type
(*)    sessid=841943137-2498        -- WG session id
(*)    sid=v0ieww00                 -- id of WG server
(*)    event_date=19960905          -- date of event
(*)    event_time=121105            -- time of event
(*)    addr=fake.name@dunkel.stx.com -- user's e-mail address (fake.name
                                       is used if name is unknown)
(*)    ufrom=69490                  -- uniquifier of refering page
(*)    upage=69559                  -- uniquifier of this page
       maxgrans=10                  -- additional field
       archid=GSFC                  -- additional field
       archid=MSFC                  -- additional field
       archid=EDC                   -- additional field
```

Entries are separated by blank lines. Some events we support *(* means that it's a standard IMS event)* are:

```
badadmin    -- failed login attempt to admin scripts
bad_login   -- failed login to persistent account
browsertest -- user wants to test out their browser
```

```
    comments    -- user comments submitted successfully
    definition  -- definition request
    directory   -- directory search
    dyinggasp   -- an errror occured
(*) guide       -- guide request
    help        -- help request
    intbrowse   -- integrated browse
(*) inventory   -- inventory search
(*) login       -- new session
(*) logout      -- session deleted (by cleanup routine)
    mode        -- a new mode is being handled for a client
    pattern     -- pattern matching was used on the simple search form.
(*) productreq  -- product request (order / download of HDF included!)
    url         -- user followed a data center supplied url.
```

The "mode" event Whenever you submit a form or follow a hyperlink that leads back into the IMS/WWW scripts, the event is logged as an

```
daac=v0ieww01
event=mode
sessid=903067843-9820
sid=v0ieww01
event_date=19980814
event_time=003919
pid=10139
addr=fake.name@greensboro0-090.dyn.interpath.net
mode=GRANLIST
submode=MAP
ufrom=69490
upage=69559
```

Think of it as logging the start of a new CGI request. The CGI tags `mode` and `submode` are logged with this event, to identify what action is about to be done. Also logged is the special `u` tag, also known as the **uniquifier** (sounds like a Schwartzenegger movie, doesn't it?).

2.3.11 Order

Overview of classes

IW::Order

This abstract class represents a collection of items in the shopping cart. The real work is done by the subclasses, IW::Order::Email and IW::Order::Request.

IW::Order::Email

Concrete subclass of IW::Order. Represents an order that will be performed by sending electronic mail to the DAAC contact points; this is done when the user has selected granules for which no packaging information is available (and when the DAAC permits it).

IW::Order::Request

Concrete subclass of IW::Order. Represents an order that can be handled via the standard IMS product-request mechanism.

IW::Order::Item

A "line item", closely corresponding to a LINE_ITEM group in a product request message. Basically, this object is an **IU::PkgCombo** with some additional information. Real order items will have:

```
ArchiveId        [AcctName]
DatasetId        [AddnInfo]
PkgCombo
```

WARNING: under the current limitations of **persistency**, this structure is NOT storable!

IW::MediaOrderLimit

A class used to evaluate the total order size with selected media limit.

How it works

1. It all starts from the shopping cart (mode CART), after the user has reviewed their manifest, asked to order the items in the cart, and supplied their shipping, billing, etc. addresses.
2. Mode ORDER, submode FORM is invoked, which takes all the data sets and granules in the shopping cart and "makes" two new empty objects:
   - An IW::Order::Request, for those items to be ordered via IMS request.
   - An IW::Order::Email, for those items to be ordered via email.

   The objects are then populated as follows...
3. For each granule to be ordered...
   - *If the granule is to be ordered by normal request,* if the granule satisfies the size threshold for the selected media, then for each combo in the granule (see **Order Option (Package) selection**), an **IW::Order::Item** is created and added to the IW::Order::Request object.
     The item will be hashed in the "request" object first by archive id, and then by its **merged-package-key**.
     The first hashing is to organize the items by archive id, since all items from the same archive are placed in the same product request message.
     The second hashing (within each archive) is to prevent the same package from being accidentally ordered multiple times if ordered through granules in two different data sets (in which case the package-keys are different, but the merged-package-keys are identical).

   - *If the granule is to be ordered by email,* then a single "fake" IU::PkgCombo is created for it, and an **IW::Order::Item** is created around that and added to the IW::Order::Email object.

     The item will be added to a list in the "email" object, where the lists are hashed by data set id. The hashing is simply to group the items by data set id, since all items from the same data set are email-ordered together.

4. For each data set to be ordered...

   - *If the data set is to be ordered by normal request,* then virtually the same procedure as was done with granules above happens here. The item representing the data set is added to the IW::Order::Request object.
   - *If the data sets is to be ordered by email...* it currently isn't. Email ordering of data sets is not implemented.

5. Now that all the "line item" objects have been created and squirreled away, all that remains is to do the order:

   (a) A copy of the full manifest is emailed to the user.
   (b) All the email-based orders (if any) are sent. This is done by taking the IW::Order::Email object and ordering the items on a data set-by-data set basis, sending each data set's order to the data set contact person.
   (c) If there were any items in the IW::Order::Requested object, a product request is sent and traced via the comm status page.
   (d) Upon completion of the product request, the user is sent email summarizing how their order went.

2.3.12 Persistency

Introduction

Since CGI is stateless, and we want to give the illusion of a stateful interface, it is necessary for us to have persistent Perl objects. We use them for:

- Session state
- Data sets/granules in the search results
- Data sets/granules in the workbench
- Data sets/granules in the shopping cart

Since persistency is not built into the Perl language, we had to roll our own software.

Overview of classes

GDBM_File

Our solution is based off of the standard Perl GDBM_File module. Basically, our databases consist of GDBM files which contain "flattened" Perl objects (objects which have been turned into scalars).

IU::Storable

This module provides the routines for "flattening" objects into scalars, and "inflating" those scalars back into real objects.

IU::Store

This is the class which acts as a sort of subclass of GDBM_File. It is a tied hash which transparently (1) "flattens" any object you wish to store to the underlying GDBM_File, and (2) "inflates" any object you wish to retrieve. It uses IU::Storable.

A cache mechanism for fetched/stored objects minimizes the amount of inflation.

IU::InvStore

A basic subclass of IU::Store, geared specifically to storing collections of data sets and granules. A tied hash.

Warnings

Warning: You can only store an object in this kind of Perl database if it has *no circular reference chains.* Any attempt to store an object with reference chains will result in an infinite loop.

Warning: DAGs will turn into simple trees on expansion: i.e., if an object A has two reference paths to the same object Z...

```
            .---.
    .------>| B |--.
.---.       `---'   `-->.----.
| A |                   | Z  |
`---'       .---.   .-->`----'
    `------>| C |--'
            `---'
```

...then *two* copies of object Z will be stored:

```
            .---.       .----.
    .------>| B |------>| Z1 |
.---.       `---'       `----'
| A |
`---'       .---.       .----.
    `------>| C | ----->| Z2 |
            `---'       `----'
```

2.3.13 Order Option (Package) selection

Overview of classes

IU::Package

Each instance represents roughly the information in a PACKAGE group. Each possible combination of **media type, media format, and processing option** is represented as an instance of IU::PkgCombo. If subsetting is available, the valid subsetting criteria are represented as an instance of IU::SubsetCrit.

IU::PkgCombo

Each instance represents a single legal combination of media type, media format, and processing option within a given PACKAGE group. Other information, like the approximate cost and the number of granules, is stored here as well.

IU::SubsetCrit

Analogous to the contents of a SUBSET_OPTIONS or SUBSET_SPEC group. Contains one or more IU::Criterion objects, one for each subsetting "criterion".

IU::Criterion

Each one of these represents a single **criterion**... in this case, for subsetting a package.

How it works

1. It all starts from the shopping cart (mode CART), after the user has hit a "order options" button.
2. Mode PKGSLCT is then invoked, with the granule id (or data set id) of the item to which packaging info is to be assigned.
3. The **imsgdbdump** program is used to get the packaging information for the item, as a stream of **chunks** which contains one or more PACKAGE groups. The PACKAGE groups are read in and converted to full-fledged **IU::Package** objects.

   At this point, each IU::Package object has associated with it: (1) a set of **IU::PkgCombo** objects (one for each legal media-type/media-format/processing-option combo), and (2) an optional **IU::SubsetCrit** object, containing a list of IU::Criterion objects (one for each possible subsetting criterion).

```
.---------.2               subset_crit()
| Package |--------------------.
`-+-------'                    |
  | combos()                   |
  |                            V
  |    .----------.          .------------.
  |--->| PkgCombo |2.1       | SubsetCrit |
  |    |----------|          `-+----------'
  |--->| PkgCombo |2.2         |    .-------------.
  |    |----------|            |--->| Criterion   |Band
  `--->| PkgCombo |2.3         |    |-------------|
       `----------'            |--->| Criterion   |Angle
                               |    |-------------|
                               |--->| Criterion   |Exposure
                               |    |-------------|
                               `--->| Criterion   |Resolution
                                    `-------------'
```

   Keep in mind, that at this point, the purpose of these objects is to represent *what the user's possible choices are*.

4. The user selects the packaging options (and, possibly, the subsetting options) they desire. They are also able to specify whether or not they want the selected packaging/subsetting options to apply to *all* items in the cart which have matching package ids. The user submits the form.
5. A list of items in the cart to be modified is constructed. Each item has its current packaging options wiped, and the new ones put in place. The result is that each item to be modified (e.g., granule), has a copy of each *selected* IU::PkgCombo ... and, if subsetting options were chosen for that combo's package, a copy of the subsetting criteria as well, *with the user's selections*.

```
.---------.
| Granule |
`-+-------'
  | package_combos()
  |
  |    .----------.1
  |--->| PkgCombo |
  |    `----------'
  |    .----------.2.2     subset_crit()
  `--->| PkgCombo |------------.
       `----------'            |
                               V
                             .------------.
                             | SubsetCrit |
                             `-+----------'
                               |    .-------------.
                               |--->| Criterion   |Band = "23.4"
                               |    |-------------|
                               `--->| Criterion   |Resolution = "HI-RES"
                                    `-------------'
```

The latter configuration accurately reflects the fact that different packaging combinations for a granule may have completely different user-selections for subsetting criteria.

6. The altered items are stored back into the shopping cart.

2.3.14 Search form

This section discusses the basic search form support.

Overview: how it works

The user's current (or last) query is stored in the session state as an instance of class **IU::Query**.

The parts of the query – parameters, sensors, time range, geographic region, etc. – are added to this object as needed. Each part of the query (like PARAMETER or TIME_RANGE) is called a category.

When a user picks a category of interest and supplies their constraints for it (e.g., when they pick TIME_RANGE and supply their start and end dates), the category will be added to the query object with at least one **criterion** object attached to it. Each criterion object is of some subclass of **IU::Criterion**), and represents a set of constraints.

For example: if a user goes to the "source" selection screen, selects LANDSAT-2, LANDSAT-3, and LANDSAT-4, and hits OK, then the query object will have attached to it the category SOURCE_NAME with a single criterion of class **IU::Criterion::SelectMany** that will contain within it the list of 3 LANDSAT satellites.

Note: Actually, each category has a *list* of one or more criterion objects attached to it, to support the multiple "variants of a category" in the extended search criteria (e.g., "Percent Cloud Cover" and "Type of Cloud Cover" may be two possible variants of the extended category "CLOUD COVER"). In general, though, the standard categories (like PARAMETER and TIME_RANGE) have exactly one criterion object, since (being standardized) they lack variants.

So basically, each screen (i.e., mode) related to the search form is used to edit one particular category of the query.

Piece by piece

Summary of overall search form design ModeABSTRACTSRCH provides a skeleton for the display and manipulation of the search form. The basic page layout such as the title is done by ABSTRACTSRCH with the more detailed layout done in WidgetBox and its component widget classes.

1. Main search form

`IW::ModeABSTRACTSRCH`	`Abstract class that provides basic frame work for search forms.  Creates the widget box and tells the widget box when to update and draw. Also provides legacy support for SLCT popups.`
`IW::ModeMAINSRCH`	`child class of ModeABSTRACTSRCH. This is the main search form used for building Primary, Summary, Detailed, Granule, Local Id, and Browse Searches.`

2. Widget Box

`IW::SrchForm::AbstractWidgetBox`	`abstract container for widgets. Creates the widgets and sticky note, handles updating each widget, and draws the search form, sticky note and widgets.`
`IW::SrchForm::MainWidgetBox`	`layout main search form and handle updates.`

Sticky notes New is the concept of a sticky note widget. Each widget has at its top a sticky note widget, but normally they are not visible. If a widget has an error or message to display to the user, it sets the message in the sticky note widget. Multiple messages from multiple widgets can be set in the sticky note widget. If there is just one message, the sticky note is shown above the widget that issued the message. If there are multiple messages then the sticky note is displayed above the widget the user clicked. A widget can also request that a sticky note message be displayed at the top of the page. This is overridden if any other widget sets a widget specific message. Each message is displayed in the sticky note and there is a link to the widget that issued the message. The page is automatically scrolled down to the sticky note location.

`IW::SrchForm::StickyNote`	track and display errors and messages from the various widgets.

Widgets Widgets generate the display for a section of the search form and handle any updates to the information in that section. Processing of the search parameter CGI tags to update the query is done by the individual widgets. For example: a request to change the search type is handled by SrchTypeWidget, a request to do a dataset lookup by the DepValidWidget, a request to change the geo type by GeoWidget, a request to set max granules by OptionsWidget, etc.

Sometimes a widget needs to request that a new mode be displayed. For example, the Java Mapplet for the Geo Widget. This is done by the widget setting the "next" mode to be displayed via a callback. Many widgets need to ability to "TOGGLE", i.e. change the type of the widget. For example, Standard Date Range to Julian Date Range for the time widget. While not all widgets will use toggle there should be a unified method for toggling a widget.

`IW::SrchForm::AbstractWidget`	abstact class for search form widgets. Outline for interaction of search form widgets with the Search form and query object.
`IW::SrchForm::SrchTypeWidget`	set and track search type. Also handle search context passing. A new feature is search type will be stored in the Query object so the search type is preserved after a session cleanup or upload from a file.
`IW::SrchForm::DepValidWidget`	display and handle dataset lookup, valids selection and granule ID forms. Note: there is no TOGGLE event, doing a GO or SELECT does an implicit toggle to dataset lookup or allvalids respectively.
`IW::SrchForm::GranIdWidget`	display and handle text entry for local ID and granule ID searches.
`IW::SrchForm::GuideTextWidget`	display and handle free text entry for guide searches.
`IW::SrchForm::TimeWidget`	display and handle date and time entries
`IW::SrchForm::GeoWidget`	layout geo widgets and handle updates.
`IW::SrchForm::OptionsWidget`	display and handle options entries
`IW::SrchForm::ButtonsWidget`	display of search form buttons and handle them when clicked.
`IW::SrchForm::LoadWidget`	display and handle upload and download form for query.

Query objects **The $QUERY object.** Basically, when you start a new session, an instance of the IU::Query class is created and stored in the $SESSION state variable `Query`. This is the "current" (or "installed") query object that the various search-form-related screens will be working on. When you do an IU::Query::lookup(), it is stored in the exported global variable **$QUERY**; after you update it, you must do a $QUERY->install to save your changes in the session state.

Categories. Each query object contains a set of zero or more **categories** to which search criteria have been assigned. Categories are things like "PARAMETER", "SOURCE_NAME", "GEOGRAPHIC_REGION", and "CLOUD COVER". The **IU::Category** module is where the information about particular categories resides (e.g., is this category one of the core dependent valids fields?).

Criteria. If a given category is present in the query, then it will have attached to it one or more **criterion** objects. Each criterion object is an instance of some subclass of **IU::Criterion**; each such object represents search criteria that have been supplied by the user.

> For example: if the user adds a time range to their query, then the category TIME_RANGE will be created in their query object, and attached to it will be a single criterion object of class IU::Criterion::SelectRange.

Search type/options. Aside from categories, query objects contain a *search type* (INVENTORY_SEARCH, GRANULE_SEARCH, etc.) and some *query options* (query name, browse-only, etc.)

What's used. Query objects are smart about what categories are relevant to a given search type; the IU::Query::uses() method can be used to determine things like "is TIME_RANGE used in a GUIDE_SEARCH?" There is also an IU::Query::uses_opt() method for determining which search options are used for a given search type.

Categories Each query object contains a set of zero or more **categories** to which search criteria have been assigned. Categories are things like "PARAMETER", "SOURCE_NAME", "GEOGRAPHIC_REGION", and "CLOUD COVER".

The **IU::Category** module is where the information about particular categories resides (e.g., is this category one of the core dependent valids fields?).

Categories consist of the following...

The 7 "core" dependent-valids categories (all of which are lists of strings, and which have the "standard" ODL names):

```
CAMPAIGN
DATASET_ID
DATA_CENTER_ID
PARAMETER
PROCESSING_LEVEL
```

```
SENSOR_NAME
SOURCE_NAME
```

The following **non-dependent categories,** which nonetheless use "standard" ODL names:

`GRANULE_ID_REQ`	`String list; used in granule id searches.`
`TEXT_STRING`	`String; used in guide searches.`

The additional **special categories** (specific to the Web Gateway), with new names:

`GEOGRAPHIC_REGION`	`Geographic constraint: range, polygon, point, path-row, or "global".`
`TIME_RANGE`	`Temporal range: start date and/or stop date.`

Finally, any category name in the extended search criteria is viable:

```
"Cloud Cover"
"Land Cover Classification"
...
```

Such names are not allowed to conflict with any category name listed above.

Criterion objects (as used in query objects) Each **category** in a query has attached to it **one or more criterion** objects. The classes are:

`IU::Criterion::`	`Abstract base class; the following are subclasses...`
`::Geo`	`Select a geographic region`
`::InputNum`	`Input a single number/date`
`::InputRange`	`Input a range of numbers/dates`
`::InputText`	`Input text`
`::SelectMany`	`Select many strings from a list`
`::SelectOne`	`Select a single string from a list`

The reason that each category can have *multiple* criteria attached to it is to support *variants* in the extended search criteria; e.g., for the extended search category "CLOUD COVER", you might be allowed to specify both "Percent Cloud Cover" and "Cloud Cover Type".

Certain categories (the most common ones) will never have more than one criterion object attached to them. Such categories are termed "singleton categories"; when outputting their criteria, there is no need to output the variant names. That is why extended search categories, when displayed in the main search form, have bold "headers" that singleton categories like TIME_RANGE do not.

Note: the IU::SubsetCrit class has been updated to use IU::Criterion objects.

Related subsystems

- **Criterion** objects.
- **Dependent valids** selection.
- **Geographic selection**.

2.3.15 Searching and communication status

Overview

Here is an overview of the players, and their relationships during the search process.

- 1, 2. In SEARCH mode, an IW::Client object is created, and an IW::CommStat object is created to monitor it.
- 3. When the IW::Client object is started, it spawns an **imsrunclient** subprocess.
- 4. The **imsrunclient** program spawns the **imsclient** program, and monitors it. When imsclient is done, it performs some post-processing of the results, and periodically writes the status of these activities (as **chunks**) to the `runclient.stat` file. This file is parsed by the client object, and each activity is represented in it by an IW::PostProcess object.
- 5. The **imsclient** program talks to the DAACs, and periodically writes the status of its connections (as **chunks**) to the `client.stat` file. This file is parsed by the client object, and each connection is represented in it by an IW::Conn object. The C-oriented results are placed in the **dataset.gdb/granule.gdb** files, but they are *also* output as **chunks** to the **client.obj** file.
- 6, 7. As part of its post-processing, **imsrunclient** takes the final **client.obj** file and converts its contents to Perl IU::Dataset and IU::Granule objects, which are written out to the `results.gdb` file (see the **Persistency** subsystem).

The executables and modules

I realize this can get confusing because there are so many players. Let's see if I can summarize the most important ones:

imsclient

A low-level IK executable, written in C.

Queries DAACs, and writes their results in a C-oriented binary format (*dataset.gdb, granule.gdb*). Every 2 seconds, announces the status of each open connection in a single file (*client.stat*).

imsgdbdump (*low-level IK executable*)

The companion low-level IK executable, written in C.

Schooled in the translation of C-oriented binary to textual output. Reads the GDBs, outputting as plain text to the standard output. *Now obsolete for search results.*

imsrunclient

A Perl executable; supervisor of the imsclient/imsgdbdump pair.

Sends *imsclient* off to perform an activity. If the activity was a search, *imsgdbdump* is invoked to interpret what was gleaned from it, and then this script translates the textual output into Perl objects which are stored in a Perl database (*results.gdb*). The status is reported during this post-processing phase, in a separate file (*runclient.stat*).

IW::Conn

A Perl object.

Able to diagnose a connection status in the *client.stat*. Can also render itself as a single row of the comm status page.

IW::PostProcess

A Perl object.

Able to diagnose a post-processing status in the *runclient.stat*. Can also render itself as a single row of the comm status page.

IW::Client

A Perl object, containing one IW::Conn object for each connection, and one IW::PostProcess object for each post-processing activity.

The callable Perl interface to the *imsrunclient* executable. Dispatches *imsrunclient* to the task at hand, and checks in from time to time to see whether or not it's still alive, and also how it's doing (or how it did, since the status files (*client.stat, runclient.stat*) are persistent).

IW::CommStat

A Perl object, containing an IW::Client representing the current/last activity.

After IW::Client checks upon the health and well-being of *imsrunclient*, it is IW::CommStat that displays the statuses to the user.

IW::ModeSEARCH

A Perl "mode" handler.

Orders *IW::Client* to commence a search, and then checks in every 15 seconds... at each iteration, *IW::Client* checks if the search is still running, and then *IW::CommStat* is made to output the current page to the user.

The files

client.obj

ASCII text file of **chunks**, output by *imsclient*, containing the most-recent search results in a quickly-Perl-parseable parameter-value format.

client.stat

ASCII text file of **chunks**, output by *imsclient*, containing the status of its open connections.

dataset.gdb and granule.gdb

GDBM files, output by *imsclient*, and containing the search results as flattened IK_Dataset and IK_Granule structs. We need to keep them around to prevent browse from failing (and, possibly, to look up packaging information).

results.gdb

GDBM file containing the most-recent search results as flattened Perl objects. Generated by converting the *client.obj* file. This is what we use when generating our results screens.

runclient.stat

ASCII text file of **chunks**, output by *imsrunclient*, containing the status of its post-processing activities.

How to start/monitor a client

Starting a client. Basically, you start by creating a new IW::Client object, using the "start" constructor. That constructor will first attempt to "attach" to any existing client, and kill it, so that only one client will be running at any one time.

Attaching to a running client. When the "start" constructor succeeds, a copy of the object is placed in the session state. From then on, anyone that wants to "attach" to it will get a full-fledged copy.

Status of a client. Each IW::Client object has a collection of IW::Conn (and IW::PostProcess) objects, each of which monitors a particular DAAC connection (or phase of post processing). Thus, you can check the status of the client to a detailed level by looking at the statuses of these objects.

How to generate the comm status screen

Creating a comm status interface. Once you start the client (and have an IW::Client object), you can ask IW::CommStatus to create a new comm status interface object for it. The object you get back will be an instance of *some subclass* of the (abstract) IW::CommStatus class; the subclass will have a name identical to the basic type of search:

```
IW::CommStatus::DIRECTORY_SEARCH
IW::CommStatus::FTP_BROWSE
IW::CommStatus::INTEGRATED_BROWSE
IW::CommStatus::INVENTORY_SEARCH
IW::CommStatus::PRODUCT_REQUEST
```

Each subclass is an "expert" on evaluating the status of one particular type of search... in particular, whether or not the search `finished_ok()` or not.

> For example, the `finished_ok()` method of INVENTORY_SEARCH simply asks that at least one granule was returned, by any connection. In contrast, the `finished_ok()` method of PRODUCT_REQUEST works completely differently.

Output of the "animated" comm status screen. We have been using "server push" documents (with a MIME type of `multipart/x-mixed-replace`). Essentially, when the `trace()` method of the IW::CommStat object is invoked, the comm status object goes into a loop wherein it waits for a specified period of time and then outputs another "page".

The `trace()` method can be directed to output the "final comm status page" or not, and even to exit once that page is output.

Issues

Regrettably, in order to know whether a connection *really* succeeded/failed, we can’t just go by the status code: we have to also know what kind of activity was being done (inventory search, data set search, etc), and then we must check the appropriate status value (number of data sets, number of granules).

In 1.5, a lot of this is scattered among IW::CommStat and IW::Client, and this has led to subtle errors in certain situations.

For 1.6, an attempt was made to centralize the logic in IW::Client and IW::Conn. The success/failure condition at higher levels was largely taken directly from the lower levels, with limited regard for the actual type of activity being undertaken.

Miscellany

The IMS/WWW standard abbreviations (as used by imsclient) and terms used to denote "activities" are:

```
Abbrev  Full
-----------------------------------------------------
DS      DIRECTORY_SEARCH
FB      FTP_BROWSE
IB      INTEGRATED_BROWSE
```

```
    IS      INVENTORY_SEARCH
    PR      PRODUCT_REQUEST
```

As of 1.6, the abbreviations are *only* used by imsclient; everyone else uses the full names.

2.3.16 Security

When a **session directory** is first created, a special `.htaccess` file is created at the top level. This file tells the HTTPD not to serve *any* documents from this or any sub-directories.

However, as a simplifying solution for making the dynamically-generated spatial/temporal coverage maps available on the WWW, the `pub` subdirectory of the users' session directories have been made accessible (by overriding that `.htaccess` file with another one). The URL of any such directory is:

```
$DocsUrl/imswww-usr/$SessionId/pub
```

This means that a *determined* user could type in URLs to grab certain scratch files that were generated for another user. Currently, this does not seem to be a great deal of a problem:

- No information of an obviously secure nature is ever put in a user's `pub` area.
- The session directories do not have user names, but cryptic numbers; if this is not sufficient, an empty `index.html` file may be placed in the **DocsDir** to prevent people from poking around and finding other session ids.
- The session directories are only present for a small number of hours before being deleted.

So it seems unlikely that this issue is worth serious consternation at present.

A possible way around this might be to provide a "secure user login" form which will assign that user's directory a `.htaccess` file with a user's specified name and password. This would slow down the initial session-creation process, but would grant security to users who wished it.

2.3.17 Session management

The Web Gateway system needs somewhere to put the search results, user preferences, shopping cart, and other data generated by a user in the course of their interaction with the system. The maintaining of user-private spaces for the duration of a user's interaction with a WWW-based system is often termed “session management.”

Typically, each user's session is distinguished by some form of unique “session id”, which is generated at the beginning of the session, and then embedded invisibly into the forms (or URLs) in all of the HTML pages returned to the client. The net effect is that whenever the user submits a form or presses a button that involves the system's CGI scripts, the session id is passed in as part of the request so the CGI script can “pick up where they left off.”

Note: HTTP cookies could have been used for propagating the session id from screen to screen; however, early releases of this system predate the standardization of cookies, and so they are not used.

In the Web Gateway system, session management is primarily overseen by the **IU::Session** module. This module is not intrinsic to the Web Gateway libraries: one can write a script which does not use it, in which case the script will not have a user-private user session space.

Session id

When a user initiates a session by an initial GET of the **nph-ims.cgi** URL, the CGI script will assign her a unique session id of the form:

```
sprintf("\%ld-\%d", time(), getpid());
```

This id is propagated to subsequent invocations of the Web Gateway CGI scripts via the **sid** tag. The presence of an **sid** tag on invocation means that that session should be *resumed*.

The absence of this tag is not necessarily an error: some "modes" (like SRCHFORM, the search form) will create a new session if needed, whereas others (like HELP, the help pages) simply don't need it.

Session directory

When the session id is first generated, the temporary directory

```
$TopDir/docs/imswww-usr/$SessionId
```

is created to hold any temporary files needed by the gateway to cache user info. For example:

```
$TopDir/docs/imswww-usr/9834933-343
```

Session state

To give us the illusion of having persistent global variables in a CGI environment, we use a "session state" database to retain certain important Perl objects.

The session state is maintained in a secure file in the session directory, `session.state`.

It is an **IU::Store** database which contains flattened Perl objects, inflated on demand.

Cleanup of sessions

The **imscleanup** program, usually run inside a cron job at a minimum of once per hour, will look for any session directory which has not been used (accessed by a WWW user) for a TBD number of hours which will indicate idleness... e.g., 8. If such a session directory is found, it is deleted, and the session is thus "terminated". Attempts to "resume" that session (e.g., by submitting a form with the **sid** filled in) will result in an ‘error’ document being returned.

Inactivity is judged by the kilroy.dat file in the **session directory**, which is "touched" every time a CGI access is made.

The **SessionLife** configuration variable controls the inactivity timeout period.

As of 1.8, the deletion works atomically: the session directory is first renamed from *XXX* to *killed_XXX*. The contents are then recursively deleted.

2.3.18 Uniquifier

What is it?

To force a browser to reload a URL (instead of fetching a copy from cache), it is occasionally necessary to "fool" the browser into thinking that it hasn't fetched the URL yet.

One remedy for this is to qualify hyperlinks with a *unique tag* that effectively forces the browser to think that it has never seen that particular page before. The easiest unique tag to use is the current time in seconds (as given by the `time()` function.

Here are some examples:

1. Here is a way to include a GIF file so that the GIF will always be reloaded:

```
<img src="icons/force.gif?u=829844525">
```

2. And here is a way of creating a hyperlink that will force the re-execution of the CGI script:

```
<a href="/cgi-bin/force?mode=WELCOME&u=829844525">
```

3. And here is a way of creating a hyperlink that will force the re-submission of a form:

   ```
   <input type=hidden name="u" value="829844525">
   ```

4. Here is another, utilizing the PATH_INFO feature of CGI:

   ```
   <form method=POST action="/some/cgi/script/u829844525">
   ```

In all cases, the `"u"` can be replaced by any tag that will be ignored by the CGI script.

Using uniquifier log entries

The uniquifier is now logged with the MODE event. Log entries for MODE events now look like:

```
event=mode
sessid=847928329-24446
sid=v0ieww00
event_date=19961113
event_time=184059
addr=fake.name@batcave.stx.com
mode=DSETLIST
submode=
ufrom=928425
upage=928500
```

mode, submode

The mode and submode in the current CGI request we have received... and therefore, the one we are *about* to handle (since we log all mode events *before* actually handling the mode.

ufrom

The `u=` tag from the current CGI request. It identifies what page the user came **from** (because that page was the one that had that `u=` tag embedded in its hyperlinks... see?).

upage

The `u=` tag that will be placed on THIS PAGE. It identifies the page **that we are about to generate**. We will use `u=UPAGE` in all hyperlinks on this page.

So from the above example, to find out what page the user came from, we look at the `ufrom`... it's `928425`, so we'd just scan upwards until we found an event=mode where `upage=928425`.

We can now determine whether or not (and when) a user hit the back button... basically, if we ever see ufrom "jump backwards" to anything but the upage of the very last mode event, we know that the user "went back" to that page. Here's what it would look like (simplified):

```
event=mode
mode=DSETLIST
ufrom=11
upage=20        # the "id" of this DSETLIST page is 20

event=mode
ufrom=20        # ok... we came from the last generated page
upage=37

event=mode
ufrom=37        # ok... we came from the last generated page
upage=52


event=mode
ufrom=20        # AHA!  They went "back" to the DSETLIST!
upage=66
```

Uniquifiers are basically time %1000000, so unless two screens were generated within the same second (odds?) they can't have the same uniqifier. Note that, for obvious reasons, we don't take the last u= tag we got and add 1. Although we COULD do that with the session state... hmmmmmmmmmmm...

2.3.19 Session Persistency

Introduction

Prior to version 2.0 the Web Gateway used temporary(tmp) session to "create" state in a stateless web. These tmp session are normally cleaned up every (default) 4 hours by imscleanup based on the variable IU::SessionLife.

Granted we could have modified the SessionLife to allow the sessions to LAST longer, but then the issue of SECURITY is raised (and the fact that this method wasn't persistent between releases) So this approach was considered inefficient.

The following requirements were imposed on the 2.0 version of persistency :

- Persistent through future releases of the Gateway.
- Preserves the user's preference file.
- Stop unauthorized users from hijacking persistent sessions.
- Allow users to create new (persistent) sessions.

Design

This is a breakdown of the screens and actions that are accomplished.

Register/Login (submode_SHOW)

Allows the user to Register (creates a new, empty persistent session) or Login (Login to your Persistent Session)

Create a Persistent Profile (submode_CREATE)

Load/intialize object User if originally exists.

Display the appropiate fields for Password, Contract/Billing/Shipping Addresses , V0 Data Access Key and ECS Access Keys.

Persistent Session Created (submode_SETUP)

The following is accomplished :

1. `Reads the User's Address information`
2. `Use the user's lastname + last 4 digits from phone number to create distinct session directory name`
3. `Create the new session directory and fill it in with joy and files`
4. `Display to the user their username and password and ask them to LOGIN.`

LOGIN (submode_RESUME)

The URL to ModeLOGIN (submode_RESUME) isn't created like the standard URL of the Gateway. It calls LOGIN via a SECURED directory (which contains a htaccess file to force authorization of the incoming request. Please see the section on htaccess for complete description) Once the browser/server verify that the user is legitimate then the environment variable REMOTE_USER is set. From there ModeLOGIN is allowed to execute.

1. Reading ENV{REMOTE_USER} to obtain the User's persistent ID using that id to RESUME their session.
2. Loading/installing the saved user prefs from the persistent session directory.
3. Switching to the Searchform.

HTACCESS

This htaccess forces ALL requestes to be verified against a username and password :

```
AuthName Web Gateway username
AuthType Basic
AuthUserFile /directory/where/the/password/file/resides
```

```
<LIMIT GET POST>
   require valid-user
</LIMIT>
```

The first time a request (URL) enters the directory this file resides in, a dialog box is displayed. Once the verification grants access to the file. The server sets the environment REMOTE_USER to the provided username .

Within the Gateway, we use this "username" as the name/"title" of the persistent directory.

Warnings

Warning: The login process restores your session to the state it was in the last time you logged in.
If you've already started searching before logging in, this will discard your current search criteria and user information, replacing it with the persistent session information.

Warning: Once you've logged in a persistent directory, you can't relogin (as a different user), but you CAN create multiple persistent sessions (if you choose to do so).

2.3.20 Log file collection and statistics generation

Log file collection

Each Web Gateway installation sends once per day the log files from the previous days' activities to *imswww-stats@harp.gsfc.nasa.gov*. The log files are extracted from the e-mail and copied to directories to await further processing on harp.

The imsstats routine:

Each public Web Gateway runs the routine *~imswww/pub/bin/imsstats* as a cron job once per day.

- The imsstats routine should only be run on the "public" installation at each site. Testing, development and new installations which aren't public yet should not have their log files collected.
- Be sure the path to the imsstats routine is specified using the *pub* link. This allows for stats collection to be unaffected by Web Gateway upgrades. As the *pub* link is moved the stats will automatically use the new public installation.
- Once per day, usually early in the morning local time the imstats routine is run by the web gateway httpd user account crontab. The crontab entry should look simular to this:

```
# Once a day mail IMS/WWW logs to the imswww-stat account at GSFC:
0 1 * * * /usr/people/imswww/pub/bin/imsstats -cron >/dev/null
```

- The pub/log directory is searched, any log file matching *statlog is processed. The current day's log file, which is still being written to, is skipped.
- A header line is added to the file including its the site statsID (from SETUP) *Filename: v0iewwXX.YYYY-MM-DD.statlog*, this line is used later on to strip off the email header from the data portion of the file.
- The log file with added header line is emailed to *imswww-stats@harp.gsfc.nasa.gov* (this is configured in SETUP).
- Once a file has been e-mailed, it is renamed in the log file directory to *YYYY-MM-DD.statlog.done*.

Log file E-mail handling on harp:

Log file emails are received by the *imswww-stats* account on *harp.gsfc.nasa.gov*. They are stripped of their email headers and copied to directories for further processing by other processes on harp.

- The account *imswww-stats* has a .forward file setup:

```
| /usr/people/imswww/bin/processEmail.ksh
```

- Any e-mail sent to imswww-stats is sent to stdin of the processEmail.ksh script.
- The script processEmail.ksh is run on demand by the mailer any time e-mail arrives for imswww-stats, with contents of the e-mail being sent to stdin of script.

Trouble shooting:

In the past there has been an occasional error from processEmail.ksh. It appears to have been caused by the receipt of two emails at the same time, causing two processEmail.ksh to run at the same time. A fixed was implemented for a potential race condition between two scripts running at the same time and the error has not reoccured.

Processing of Log files to create the Statistics

For each V0 Web Gateway installation a web page with statistics for the site is created. A summary page for all sites is also created. The number of Sessions, Searches, Browse, Guides, Comments, Orders, etc. are tracked and shown on a per month basis.

Summary Statistics generation:

Once per day the routine *processSummaryLog_Cron.ksh* is run by the *imswww-stats* account cron job.

- The script picks up new files from the *~imswww/stats/summary/new* area.
- Copies are placed in *~imswww/stats/summary/scrs/*
- Copies are placed in *~imswww/stats/summary/work/*
- Developers are screens out of the log files in *~imswww/stats/summary/work/*
- Next the log files in *~imswww/stats/summary/work.* are converted to httpd log format and placed in the directory *~imswww/stats/summary/web/*
- The summary stats pages is updated with the new log data.

Local Statistics generation:

Once per day the routine *processAllSitesLog_Cron.ksh* is run by the *imswww-stats* account cron job.

- In turn it loops over the directories for each of the Web Gateway sites *~imswww/stats/v0iewwXX/new/* calling the routine *processLocalLog_Cron.ksh*.
- The routine *processLocalLog_Cron.ksh* screens out the developers.
- The log files are converted to httpd log format.
- The stats pages for the site are updated.

Trouble Shooting:

Several times in the past the number of new sessions has jumped to tens of thousands per day over a several day period. The many sessions are created from the same IP address and appear to be a robot or some other automated web indexing routine repeatedly loading the Search Form. When this happened the stats web pages were manually adjusted to remove the "robot" events.

Distribution of Log files to SCRS

Once per week the accumulated log files from all the Web Gateway sites are packaged up and e-mailed to the Statistics Collection and Reporting Service (SCRS) group.

E-mailing log files to SCRS:

The routine *~imswww/pub/bin/emailLog2SCRS_Cron.ksh* is run by the account *imswww-stats* crontab.

- Log files to be mailed to SCRS are accumulated in *~imswww/stats/summary/scrs/*
- Log files for each site (v0iewwXX), usually a week’s worth, are combined together into one file **v0iewwXX.YYYY-MM-DD.week**.

- Each "week" file is mailed to SCRS at *wwwV0@ulabibm.gsfc.nasa.gov*. One file is sent per Web Gateway installation.

Trouble shooting:

Occasionally there will be no log files for the week from a particular Web Gateway site. This can mean:

- The site is down. Check that the site is working, if not contact the Web Gateway administrator for that site.
- The imsstats cron job is not working or email is not getting through. Go to the site, login to admin mode and try to determine if the cronjob is setup properly.
- There may have been no activity at the site for the entire week. Go to the site, login to admin mode and check the log files to see if there was any activity during the previous week. If not, check the *~imswww/stats/summary/scrs/* directory tomorrow to see if a log file is received for the site as a result of your using it.

Storing of Log files for future needs

Sometimes there could be system problems either at our site or at the statistic collections site with the SCRS group. If there is ever a need to rerun the stats or to see the work performed in the past in more detail, the EDG maintains all of the statistics that have been generated from the beginning. They are stored at the following directory in harp.

```
/d6/stats/logs
```

Once per week a cronjob moves stat log files that are older than 60 days old into the store area. The script used is:

```
/usr/people/imswww-stats/mv_stats.sh
```

The script has been set up by the system administrator, if there is ever a problem with it, the system administrator will have to be contacted to solve it.

2.4 Java Design

Want to know what/how a particular part, function, or feature works? Listed are the components that presently makeup our design.

2.4.1 Web Gateway

Introduction

The Web Gateway constantly interacts with the applet. Its roles include:

1. Intializing and displaying the applet
2. Re-establishing the flow of execution after the Applet values have been ingested
3. Handling ModePUKE (errors severe enough to block further execution of the Web Gateway) appropriately

How it works

1. Intializing and displaying the applet

   a.

   The Web Gateway creates a CGI-like parameter string (key=value pairs, separated by ampersands) from the information that the applet needs (e.g., last geoselection, valid pages for the jump menu, etc.).

   b.

   Proper HTML code is generated to display the applet. This includes the HTML APPLET tag and a single PARAM tag to pass in the parameter string.

   c.

   The HTML is passed on to the user's browser, which takes care of downloading and running the applet.

2. Re-establishing the flow of execution after the Applet values have been ingested

   The Web Gateway cannot know the specific state of the applet until the applet itself notifies the Web Gateway that it has finished running. At this point, the Web Gateway has to restart execution of the CGI code to continue.

   The applet accesses a special Mode in the Web Gateway, ModeAPPLET, which picks up the information passed back via the servlet and then vectors off to process this new information.

   It should be noted that the Java classes loaded during execution of the applet are still loaded into the browser's virtual memory. This is an unfortunate and unavoidable side-effect of using an applet.

3. Handling ModePUKE appropriately

   If the applet determines that it has been unable to perform its function successfully (e.g., it is unable to contact the Servlet to write its information to the Web Gateway), it will access the Web Gateway ModePUKE.

The Web Gateway will display either a short/long error message based on the severity of the error. This error may force the Web Gateway to halt execution or simply complain to the user; the severity of the error is determined by the applet.

2.4.2 Applet (Generic)

Introduction

This section provides a description of conponents the applet needs to work with the Web Gateway.

Rules :

1. Separate Frame.

An in-the-page applet can't be destroyed; we found that if a large applet was called multiple times, eventually the browser would run out of memory. We therefore chose to implement the applet in two parts: a very small, embedded-in-the-page applet, using very little memory, and a separate window (a `Frame` in Java terminology). The Frame can be completely discarded between invocations of the applet, thereby greatly reducing the resources needed.

Pros and cons of this approach:

- The applet can easily be used multiple times without causing the user's browser to crash.
- The user gets multiple windows rather than the single-page format found in previous versions of the Web Gateway.

2.

Parameter handling

We chose to embed the applet parameters as a single HTML PARAM field, formatted in a manner similar to a CGI request. This was done because the Web Gateway already has code in place to handle parameters in this manner, and it was decided that it was simpler to have the applet parse this format than it would be to add code to the Web Gateway to handle multiple types of parameterization.

Once the applet is loaded, it access this single parameter string and parses it by first splitting up the line at the ampersand ("&") character, separating it into "key=value" pairs, and then splitting it again at the "=" in each pair to extract the actual keys and values into a hash table. This table is propagated throughout the applet to provide access to the parameters.

3.

Connection to the Servlet

The applet's internal values need to be converted into a "&Parameter= Value" string so that it can be sent to the Servlet. A method is provided that takes a Hash table and converts it back to a string in this format.

4.

Error Handling

The applet must be able to handle both internal errors and errors that may occur from communicating with the Servlet. The StatusArea, which provides in-the-applet user feedback, and the DialogFrame, which provides pop-up dialogs, have been provided to resolve these issues.

Notes: At the time of the initial design and development of the Applet/Servlet/WG interfaces, it was thought (based on all references that we could find) that it was not possible to use an HTTP "POST" request to communicate with the Web Gateway. The Servlet was added to allow arbitrarily-long data to be passed from the applet back to the Web Gateway. New information and experiment have determined that it is possible to do a "POST" within Java; this would mean the applet could directly "POST" its data to the Web Gateway. We are investigating this possibility for future releases to determine if it is dependable and usable for our purposes.

2.4.3 Encoded String

Introduction

At the time of design, it was thought that it wasn't possible for a Java applet to make an HTTP "POST" request. Since we needed to be able to send an arbitrarily-large amount of data to the Web Gateway, we needed to be able to create an alternative that would allow us to write a file where the Web Gateway could see it. Since applets may not write files themselves, we had to come up with a proxy which **could** do so.

The Servlet was created to be such a proxy: it is a non-applet program, running on the Web Gateway server, and can quite happily create files and write to them. The problem now became one of communicating between the applet and the Servlet.

Sockets was the answer.

How it works

1. The applet opens an input/output socket to the **Servlet**.
2. The applet converts its internal parameters into a Parameter=Value pair list with "&" separating each pair (same form as a "GET"/"POST" request) and encodes this to escape characters not acceptable in a CGI request (e.g., embedded ampersands or equals signs).

3. The string is then sent through the socket (using Java's PrintStream) to the Servlet.
4. *IF* there are any errors/messages from the Servlet then they are read back by the applet.
5. All sockets are then closed.

There is, of course, also code in place to handle request/response timeouts, etc. During testing, we determined that Macintoshes using MacTCP were unable to use sockets in Java; however, if they were switched to Open Transport networking protocol, they were able to use them without problems. This appears to be a problem common to all common Macintosh Java-capable browsers. Special code is in place to handle this eventuality as well.

Sample parameter string

```
NorthLat=25%2e3333&SouthLat=%2d1%2e33333&WestLong=29%2e3333&script_name
=%2fims%2dbin%2fdev%2fnph%2dims%2ecgi&MapName=JavaMap&httpd_host=
harp%2egsfc%2enasa%2egov%3a80&sid=871308086%2d13250&EastLong=29%2e3333&
#^#
```

Note that periods, colons, etc., are encoded as %xx (percent sign and two hexadecimal characters.

Notes : As of the design phase for version 1.9 of the Web gateway, this applet/servlet approach was the only way to accomplish the task we had in mind. Since then, sample code has been posted to Usenet demonstrating ways to do an HTTP "POST" with Java. If the browsers support this new solution, we will move to the "POST" approach.

2.4.4 Servlet

Introduction

The Servlet provides a means for Java applets to be able to transmit an arbitrary amount of information back to the session directory, where it can be accessed by CGI programs.

How it works: services A Web Gateway applet sends the servlet a set of keywords and values in standard CGI format. The servlet extracts the "gaeatmp" keyword to determine the session directory corresponding to the request, opens a "Java.txt" file in that session directory, and writes the full original query into this file.

The servlet responds with the string "OK" if the entire operation succeeds, or with an appropriate error message if any part of it fails.

How it works: system interfaces The Servlet listens to a port (ServletPort), defined in the SETUP file. When it gets an incoming connection on this port, it forks off a copy of itself to provide its standard services; the main-line program continues to listen to the socket for further connections.

Forks are done using the standard "fork-twice" idiom: the first process forked off immediately forks again, and then terminates. This causes its still-running child - the "grandchild" - to become a free-running process, which disappears without further ado when it terminates (because its parent has already terminated). The original process waits for the child to terminate, which it does immedately, keeping "zombie" processes from piling up.

Notes :

```
As of version 1.9, the file name(Java.txt) is hardcoded within the
Servlet. This will change in a future release (we will probably use the
key "j.file")
```

2.4.5 Java.txt

Introduction

The Java.txt file resides in the user's session directory. The file contains values obtained from the applet that need to be propagated into the Web Gateway.

How it works

1. When the Servlet receives the string, it parses it to discover which session it "belongs" to, and it opens a "Java.txt" file in that session directory and writes the string to the file.
2. The Servlet sends the applet the error/message that it received from writing the file to the session.

Example :

```
p.1.lat=53.4886&p.0.lat=53.4886&submode=OK&p.geoselected=1&p.projection
=ORTHOGRAPHIC&p.tangent.lon=-105&script_name=/nph-ims.cgi&MapName=Java
enabled Map&j.port=5433&p.tangent.lat=0&p.zoom=1&httpd_host=harp.gsfc.
nasa.gov:8124&jumpmenu=Main Search Form!MAIN_SEARCH_FORM:----------!:
Geographic Region!GEOGRAPHIC_REGION:----------!:Parameter!PARAMETER:
Data Set!DATASET_ID:Sensor!SENSOR_NAME:Data Center!DATA_CENTER_ID:
Source!SOURCE_NAME:Campaign!CAMPAIGN:Processing Level!PROCESSING_LEVEL
:Other criteria...!EXTENDED_SEARCH:----------!:Time Range!TIME_RANGE
:----------!:Search Options!SEARCH_OPTIONS:&host=harp.gsfc.nasa.gov&
p.geotype=POLYGON_LOC&mode=SLCTGEO&sid=879884721-12715&p.pole=X&
p.geoopt=YES&goto=MAIN_SEARCH_FORM&CgiUrl=http://harp.gsfc.nasa.gov/
ims-bin/dev/&p.3.lon=-126.353&p.2.lon=-64.0502&gaeatmp=/usr/people/
```

```
imswww/dev/docs/imswww-usr/879884721-12715&p.3.lat=26.0562&p.1.lon=-
23.2773&p.2.lat=26.0562&p.0.lon=-138.351&
```

The `gaeatmp` key in the string above indicates the session directory to be used. A Java.txt file is opened there and this whole string is written, as-is, into that file.

Notes :

```
As of version 1.9, the file name(Java.txt) is hardcoded within the
Servlet. This will change in a future release (we will probably use the
key "j.file" for this purpose).
```

2.4.6 IW::ModeAPPLET

Introduction

This module is the "bridge" between the applet and the existing Web Gateway. It handles the transfer of data from the applet to the CGI environment.

How it works Once ModeAPPLET has been called (by the applet), the following things occur:

1. The Java.txt file, written by the Servlet under the applet's control, is opened and read - this produces a CGI-like string of characters.
2. The CGI string is then parsed, and a set of "parameter=value" CGI tags are created, including the Web Gateway mode that can handle such a request.
3. ModeAPPLET simply dispatches this mode to handle the request.

Notes : ModeAPPLET is an "internal" mode; it generates no web page and serves only as a generalized dispatcher to other modes.

2.4.7 CGI Parser

Introduction

The Web Gateway originally passed hidden "parameters" from page to page. This "parsing" code already existed to intialize/update "attributes" within the Web Gateway.

How it works Once ModeAPPLET has created the CGI tags, the appropiate parser is called to ingest the CGI values and use them to initialize/update the corresponding attributes.

2.4.8 ModePUKE

Introduction

Java provides us with a mechanism to handle errors either at the time the error occurs or to propagate the error back up the method-call path. This is accomplished by Java's try/catchi mechanism.

In most cases, when an error occurs we can just notify the user, but what about the errors that affect or even stop the Web Gateway?

At present, the Web Gateway's ModePUKE can force an error page in one of two modes:

```
        FATAL - Stops the Web gateway, and sends email to the web gateway
                developers notifying them of the problem.

         CLEAN - Usually used for User Errors (such as the user didn't
                 fill out a required field). This is a way for Web Gateway
                 to complain, but not to halt the Gateway.
```

How it works Since Java code couldn't call Perl code directly, a wrapper (ModePUKE) was needed to handle the communications from Java to Perl.

ModePUKE is called from Java via a HTTP "GET" request and passed 2 parameters (the error number, and the accompanying message) which then is passed to the PERL's puke() function, where the appropiate action is taken (FATAL/CLEAN).

2.5 Support Files

The EOS Data Gateway (EDG) depends on a directory tree of special supporting files. These files come in two sets. The first set is sometimes referred to as the `"distribution files"` and is a static set of files. The second set of files is sometimes called the `"support files"` and sometimes called the `"valids"`, this is a dynamic set of files.

The directory structure for these sets of files should look like this:

```
  ~imswww/
     |
     |--- support_files/
     :       |
             |--- sup_files/ ...
             |--- dist_files/ ...
             :
```

Where imswww is usually the home directory for the user id containing the EDG installation.

Updates to the valids are provided by the EOS Data Gateway Team every two weeks. The latest set of valids is obtained by executing the "run_autoxfer.pm" script, supplied with the web gateway, using the following command:

```
~imswww/imswww-VERSION/lib/perl/run_autoxfer.pm
```

Since the "valids" are updated by the EOS Data Gateway Team every two weeks, to be sure to keep the valids updated, the run_autoxfer.pm script should be run periodically. This script runs very quickly. It can take less than one second if no new valids are available. Therefore, it is recommeded to simply add an entry for the script run_autoxfer.pm to the crontab for the ~imswww account to be run once a day.

The wwwvalids.con file is a server configuration file used by the EDG. The EDG interfaces with many different servers, to allow each server to include especific features for their data, both servers and client use the wwwvalids.conf file for server and client customization. Some of the customization features are the disclaimers, the results list columns, contact information, and many more.

Server Configuration File

The wwwvalids.conf file is distributed as part of the client installation, it is located at the /lib/conf directory. However, since this file contains server information, it gets updated periodically depending on the updates to the servers. For this reason, the wwwvalids.conf file is also distributed as part of the valids files in the support_files/sup_files directory.

During the EDG installation, the installer has to manually copy the file from the sup_files directory to the lib/conf directory, you might want to see **Step 9: Make valids file for e-mail ordering** in **Installing it**.

To keep the EDG installation current with the latest set of valids and the newest wwwvalids.conf file, at installation the installer needs to set up a cronjob for the run_autoxfer.pm program. One of the functions of the run_autoxfer is to get the latest version of wwwvalids.conf into the support_files directory, copy it into the /lib/conf directory, and set up the appropriate changes with it in the lib/conf directory.

Chapter 3

System components

These are the directories, files, libraries, and executables which actually make up the system. Think of this as "one step lower down" than subsystems... here, we're talking more about the individual nuts and bolts than how they fit together.

3.1 Directories and files

This section gives the directory structure of a Web Gateway installation, and highlights some of the more-important files you might see.

3.1.1 Development directories

To facilitate use by both developers and Beta-testers, the development account accommodates multiple installations of the gateway. Of these, two are consistent:

dev

The “developer” install. This is what the developers work with; it is in a constant state of flux. It may or may not be working, or providing accurate and complete results, at any time. Non-developers should not depend on it or any features in it at any time for day-to-day work.

beta

The “beta” install. This is what the Beta-testers use; it is only upgraded at specified times. It will usuallu be fairly operational, but it is a beta-test environment, so bugs may or may not be present.

```
~imswww
    |
    |--- public_html       (accessed by the URL /~imswww/)
```

```
    |      |--- dev  -> ~/dev/docs
    |      |--- beta -> ~/beta/docs
    |      :
    |
    |--- public_cgi       (ScriptAlias for the URL /ims-bin/)
    |      |--- dev  -> ~/dev/cgi-bin
    |      |--- beta -> ~/beta/cgi-bin
    |      :
    |
    |--- (dev, beta)
    :      |--- cgi-bin  ($CgiDir)
           |--- docs     ($DocsDir)
           |--- lib      ($LibDir)
           |--- tmp
           |--- src
           :
```

3.1.2 Installation directories

When the Web Gateway system is unpacked, all the files and directories it needs are actually provided for you in a single directory tree:

```
$TopDir/  (i.e., "~imswww/top")
    |
    |--- bin/             (administrative executables)
    |
    |--- cgi-bin/         (various Web Gateway CGI scripts)
    |
    |--- docs/
    |      |--- btns/           (buttons)
    |      |--- icons/          (icons, logos)
    |      |--- imswww-usr@     (link to ../tmp/imswww-usr)
    |      |--- manuals/        (user manuals)
    |      |--- navigator/      (navigator buttons)
    |      `--- pics/           (pictures, maps, etc.)
    |
    |--- lib/
    |      |--- bin/            (helper executables)
    |      |--- conf/           (gateway configuration files)
    |      `--- perl/          (Perl modules)
    |
    |--- log/             (gateway-wide log files)
    |
    |--- src/             (source code)
    |
    |--- tmp/
    |      `--- imswww-usr/
    |             |--- cache/       (browsefile cache)
    |             `--- SESSIONID/   (session directory)
```

```
   |
   `--- util/             (utilities)
```

3.1.3 Installation files

This is by no means a comprehensive list of the files in the Web Gateway installation; however, it highlights some of the more important ones.

bin

Binaries used for system installation and administration.

configure

Top-level configuration script. *You should not try to run this directly*, as it has a fairly extensive set of required options. Use **myconfig** instead.

cgi-bin

The directory containing the gateway's cgi-bin files. **CgiDir** points here by default. There are a number of different CGI programs here; the one that's most important is **nph-ims.cgi**.

docs

WWW documents (HTML, GIFs, etc.) for the gateway. **DocsDir** points here by default.

docs/btns

GIFs for buttons: small pictures which generally are hyperlinked.

docs/icons

GIFs for icons: small pictures which generally are not hyperlinked.

docs/imswww-usr

Symbolic link to the directory **`TopDir`**`/docs/imswww-usr`, making its contents accessible via URL to browsers. This allows everything in the cache and session directories to become accessible to the end user.

docs/manuals

The Design Manual, User Manual, etc. (HTML and Postscript).

docs/navigator

GIFs for the navigator buttons in the online versions of the user manual, developer's guide, and this design document. These are used by *ipod2html* in building these online versions of the documents.

docs/pics

Larger GIFs used for pictures, map-selection maps, etc.

lib

Files used directly by the gateway. This includes configuration files, Perl modules, and “helper” executables. **LibDir** points here by default.

lib/bin

Binaries used internally by the gateway (i.e., ones which are never invoked directly from the command line).

lib/conf

Configuration files for the gateway.

lib/conf/down.txt

The gateway’s “off” switch. If this file exists, the gateway’s CGI scripts *will not run:* instead, a “system down for maintenance” error will be presented to the user, along with the contents of this file, which should be plain ASCII text (no HTML markup) that explains why the system has been brought down, and when it may be back up.

lib/conf/warn.txt

If this file exists, the gateway’s CGI scripts will output its contents as a warning message at the top of each page. Use this to alert users that the gateway is going down.

lib/conf/params.valid

List of valid parameters, one per line. Generated from the client support files to allow for fast loading. *Don’t edit this file.*

lib/conf/SETUP.dat

Gateway setup, condensed for fast-parsing by C programs. *Don’t edit this file directly:* edit **SETUP**.

lib/perl

Perl code which makes up the gateway.

lib/perl/IU

The Perl library of Web Gateway Utility modules.

lib/perl/IU/SETUP.pm

Gateway setup, turned into a Perl file for fast loading. *Don’t edit this file directly:* edit **SETUP**.

lib/perl/IW

The Perl library of Web Gateway WWW-interface modules.

log

Where system-wide log files should be placed. **LogDir** points here by default.

Makefile

Top-level system makefile. The default rule prints out a list of all options.

myconnfig

A simple script that properly parameterizes **configure**. You should edit *this* file if you want to chang your build configuration.

setup

Top-level set-up script.

src

Source code directories, used to build the gateway's binaries.

src/*

An individual package

src/*/Makefile

Standardized makefile for this package.

tmp

Default temporary directory for the gateway.

tmp/imswww-usr/cache

Where browse images from remote data centers are cached.

tmp/imswww-usr/SessionId

A session directory.

tmp/imswww-usr/SessionId/.htaccess

An NCSA HTTPD file which prevents user agents from examining the files in this directory. This is to prevent users from spying on the files of other users.

tmp/imswww-usr/SessionId/browse.odl

The last "browse" request, in ODL form. This is used as input by **imsclient**.

tmp/imswww-usr/SessionId/client.stat

The connection statuses of the **imsclient** program, as a stream of RFC822-style "chunks", one per connection. Updated every few seconds by the program as it executes.

tmp/imswww-usr/SessionId/depvalidq.odl

The ODL for the last query to **imsdepval**.

tmp/imswww-usr/SessionId/hdfnames.gdb

The filename for the returned browse image. It uses the granule key and the image id as the key to the dbf.

tmp/imswww-usr/SessionId/imsclient.pid

A small file containing the PID of the last-run **imsclient** program. This allows a **imsclient** running in the background to be identified and stopped by a CGI script.

tmp/imswww-usr/SessionId/.kilroy

This file lets the cleanup script know when the last access of this session was. It is "touched" every time the user does something involving an Web Gateway CGI script.

tmp/imswww-usr/SessionId/order.odl

The last "order" request, in ODL form. This is used as input by **imsclient**.

tmp/imswww-usr/SessionId/results.gdb

The GDBM file containing all the results from ODL-based searches. Contains "frozen" Perl objects describing datasets, granules, and packages.

tmp/imswww-usr/SessionId/runclient.log

The log file for the last run of the **imsrunclient** program.

tmp/imswww-usr/SessionId/runclient.stat

The progress of the **imsrunclient** program, as a stream of RFC822-style "chunks", one per connection. Updated every few seconds by the program as it executes.

tmp/imswww-usr/SessionId/search.html

The last "search" query, as a short HTML file. This is interpolated into the bottoms of dynamically-generated pages as a short reminder of the last search.

tmp/imswww-usr/SessionId/search.odl

The last "search" query, in its ODL form. This is used as input by **imsclient**.

tmp/imswww-usr/SessionId/session.state

The current session state, as an RFC822-style "chunk". This file is read every time a gateway CGI script starts up, and is written back out every time it shuts down.

tmp/imswww-usr/SessionId/syslog

The syslog from the last **imsclient** run.

util

Installation utilities for the gateway.

3.2 Documentation (IPOD)

Both the User Manual (where "help" pages come from) and the Design Manual are written, not in HTML, but in an even simpler language called IPOD, a derivative of the Perl POD format for documentation. IPOD takes POD a little step further without adding much nonsense.

The IPOD documentation is converted to HTML for the on-line documentation using **ipod2html**. It is converted to LaTeX (and thence to Postscript) for the off-line documentation using **ipod2latex**.

The IPOD pages may be found under `$TopDir/docs/manuals/ipod`.

3.2.1 Introduction

IPOD extends the syntax of POD:

- It provides an `=include` command (actually, this is handled by the IPOD preprocessor, **ipodpp**, which allows you to include other IPOD, POD, and flat-text files right into your documentation...

```
Here's B<my> signature file:

=include verbatim="/home/eryq/.signature"

Here's B<my> home directory, I<right now>:

=include verbatim="ls -l |"

I<Neat, hunh?>
```

- It allows comments in your documentation, through the new == command (this is also handled by the IPOD preprocessor, **ipodpp**.
- It provides a consistent `=begin`/`=end` syntax, for defining (and parsing or ignoring) environments.
- It defines several standard `=begin`/`=end` environments which explicitly type lists, rather than relying on interpretation of the first `=item` command. These are: `list`, `desc`, `enum`, and `quote`:

```
Like this....

=begin quote

This is a quotation paragraph.

=end quote
```

- It defines several standard `=begin`/`=end` environments for admonishments: `note`, `warn`, etc:

```
Like this....

=begin warn

Objects in mirror may be closer
then they appear.

=end warn
```

- It provides a standard syntax for command arguments which are not free text. This syntax parallels the SGML tag syntax, so it is easily grasped...

```
Like this...

=include module=yes pod="foobar.pod"
```

- It formally extends `=head` to at least 6 levels, and adds the special `=head0` for defining the title of a document.

3.2.2 The IPOD specification

The most important point is this:

> **Any syntax that is legal in the *perlpod* specification is legal in IPOD**.

Angle-markup expansion

Formally, **markup may be nested** (this was never explicitly stated in POD). Since the Ipod:: library handles all the hard parts for you, there's no excuse for a translator *not* to support it.

For you curious types, the Ipod:: parsing algorithm expands text left-to-right and outside-to-inside, so the following text would be encountered by the translator exactly as it says:

```
First B<second> third B<fourth S<fifth I<sixth> also-fifth> also-fourth>
```

For translator authors, this means that your code would never be presented with the job of translating "sixth" before it had been given a crack at "fifth"... and the same for "fifth", "fourth", etc. This makes it *much* easier to write reliable translators.

Standard POD angle-markup

The following are all supported, with their usual semantics:

```
MARKUP     MEANING
-----------------------------------------------------------
B<bold>    Typeset argument in boldface
C<code>    Typeset as code (typewriter face)
E<esc>     Escape a character... only E<lt> and E<gt> are guaranteed
F<file>    Typeset a filename
I<ital>    Italicize

L<link>    Create a link, to either a section or an indexed entry:

           L<index entry>      explicit index entry
           L<NAME>             the NAME section of this page
           L<"NAME">           ditto
           L<OTHER/NAME>       the NAME section of the OTHER page
           L<OTHER/"NAME">     ditto

           If an INDEX-ENTRY, it will be typeset as-is.
           If a SECTION, the last element of the path will be typeset.

X<entry>   Create an index entry
S<nbsp>    Typeset with non-breaking spaces
Z<>        A zero-width entity
```

New IPOD angle-markup

The following are added:

```
MARKUP     MEANING
-----------------------------------------------------------
A<arg>     Hyperlink-like things: URLs and email addresses
R<>        A hard return
```

Standard POD commands

The following are all supported, with their usual semantics:

```
COMMAND       MEANING
-----------------------------------------------------------
=back         end a list
=cut          begin cut (terminated by any =command)
=head[1-2]    headings
=item         list item
=over         begin a list
=pod          a no-op
```

New IPOD commands

The following commands are handled by **ipodpp**, the IPOD preprocessor:

```
COMMAND       MEANING
------------------------------------------------------------
==            a comment (handled
=include      include (and parse) another file
```

The remainder are handled by the IPOD parser (and, hence, the translators):

```
COMMAND       MEANING
------------------------------------------------------------
=begin ENV    begin environment ENV
=end ENV      end environment ENV
=head0        main title of document
=head[1-6]    chapter, section, subsection, etc. headings
```

3.2.3 IPOD syntax

The "=head0" command

Exactly one of these should appear, at the beginning of a document. It gives the document title.

The "=head[1-6]" commands

These are analogous to `<H1>...<H6>` in HTML, and to `\chapter`, `\section`, etc. in LaTeX.

Note that **ipodpp** automagically adjusts the heading level when you include files... so if file `outer` contains:

```
------------------------------------

=head3 OUTER HEADING

=include ipod="inner.ipod"

------------------------------------
```

and file `inner` contains:

```
------------------------------------

=head1 INNER HEADING

=head3 Inner subheading

------------------------------------
```

Then preprocessing will yield:

```
------------------------------------

=head3 OUTER HEADING

=head4 INNER HEADING

=head5 Inner subheading

------------------------------------
```

The "base" of the headings in the embedded file is adjusted to fit "below" the heading level in the outer file; all headings in the embedded file are changed to maintain the proper relative numbering.

The "over/back" construct

The `=over`/`=back` construct still works, of course:

```
   STUFF BEFORE...

   =over 4

   =item *

   Rhyme, the first sister.

B
   =item *

   Reason, the second sister.

   =back

   STUFF AFTER...
```

However, to make things a bit more explicit (both in the markup and in the converters), I wanted some syntactic distinctions. So, three new list-making environments are introduced:

The "list" environment

The `list` environment is intended for simple bulleted lists. It parallels `itemize` in LaTeX, and `ul` in HTML. The argument to `=item` is optional and is currently ignored; however, a "*" as the first argument will always be tolerated:

```
STUFF BEFORE...

=begin list

=item *

Rhyme, the first sister.

=item *

Reason, the second sister.

=end list

STUFF AFTER...
```

This would typeset something like:

```
STUFF BEFORE...

     * Rhyme, the first sister

     * Reason, the second sister

STUFF AFTER...
```

The "desc" environment

The `desc` environment is intended for description (definition) lists. It parallels `description` in LaTeX, and `dl` in HTML. The argument to `=item` gives the term being defined:

```
STUFF BEFORE...

=begin desc

=item Dictionopolis

The kingdom of the south, ruled by Azaz The
Unabridged.

=item Digitopolis

The realm of the north, presided over by
the Mathemagician.
```

```
    =end desc

    STUFF AFTER...
```

This would typeset something like:

```
    STUFF BEFORE...

    Dictionopolis
        The kingdom of the south, ruled by Azaz The
        Unabridged.

    Digitopolis
        The realm of the north, presided over by
        the Mathemagician.

    STUFF AFTER...
```

The "enum" environment

The `enum` environment is intended for enumerated lists, *when just want items numbered sequentially as 1, 2, 3, etc.* It parallels `enumerate` in LaTeX, and `ol` in HTML. The argument to `=item` is optional, and will very likely ignored:

```
    THIS PACKAGE CONTAINS THE FOLLOWING ITEMS:

    =begin enum

    =item

    One (1) genuine turnpike toolbooth...

    =item

    Three (3) precautionary signs...

    =item

    Assorted coins for use in paying tolls.

    =end enum
```

This would typeset something like:

```
    THIS PACKAGE CONTAINS THE FOLLOWING ITEMS:

        1. One (1) genuine turnpike toolbooth...

        2. Three (3) precautionary signs...

        3. Assorted coins for use in paying tolls.
```

The "quote" environment

The `quote` environment is intended for indented, formatted text: It parallels `quotation` in LaTeX, and `blockquote` in HTML.

```
and recited in a loud clear voice:

=begin quote

My angles are many R<>
My sides are not few R<>
I'm the Dodecahedron R<>
Who are you?

=end quote

"What's a Dodecahedron?" inquired Milo, who was...
```

This would typeset something like:

```
and recited in a loud clear voice:

     My angles are many
     My sides are not few
     I'm the Dodecahedron
     Who are you?

"What's a Dodecahedron?" inquired Milo, who
 was...
```

The "note" and "warn" environments:

The `note` and `warn` environments are similar, but intended for displaying admonishments:

```
STUFF BEFORE...

=begin note
```

```
You can swim all day in the Sea of Knowledge and still
come out completely dry.  Most people do.

=end note

=begin warn

You can lose too much time jumping to Conclusions.

=end warn

STUFF AFTER...
```

This would typeset something like:

```
STUFF BEFORE...

    -----------------------------------------------------------
    NOTE:    You can swim all day in the Sea of Knowledge
             and still come out completely dry.  Most people
             do.

    -----------------------------------------------------------
    WARNING: You can lose too much time jumping to
             Conclusions.

STUFF AFTER...
```

The =include command

Use it like this:

```
STUFF BEFORE...

=include ipod=/some/file.pod

=include module pod="other.pod"

=include verbatim=/home/myacct/.signature

=include verbatim="ls -l |"

=include
     verbatim="docs/mail.msg"
     prefix="   > "
```

The following parameters are recognized:

ipod=EXPR

Open the given EXPR, and parse it as IPOD source.

pod=EXPR

Open the given EXPR, and parse it as vanilla POD source. This distinction is meant to handle any future incompatibilities.

verbatim=EXPR

Open the given EXPR, and insert its contents verbatim into the current output as preformatted text (shifted over 4 spaces). No interpretation of "="-commands or angle-escapes will take place. You can change the indentation with the `prefix` tag.

module

A quick-and-dirty way to include the documentation for a module which has been "done up" in standard POD module style, where the file begins with `"=head1 NAME"`:

- The module name is automatically extracted from the paragraph after the NAME field.
- A `=head` at the appropriate level is inserted, with the module name as the title, wrapped in a X<>.
- A `=begin module` and `=end module` are wrapped around the file's contents.

3.3 Executables

The following programs are used by the Web Gateway.

3.3.1 CGI scripts

drawglobe.cgi

Generates the orthographic globe views used by the Java geoselector.
Language: Perl.

grab-cal.cgi

Used by an SSI directive on the home page to embed the **Uptime Calendar**.
Language: Perl.

imsmap.cgi

Generates the rectangular maps (GIFs) for the search form's geoselection mode.
Language: C.

imsorthomap.cgi

Generates the polar maps (GIFs) for the search form's geoselection mode. *Language: C*.

imswdbtile.cgi

nph-ims.cgi

This is the main CGI script for the system. It is an NPH script because it needs to be able to return special MIME headers and documents (like `multipart/x-mixed-replace`). When you're talking about the Web Gateway, you're talking about this script. *Language: Perl.*

3.3.2 Helper programs

These are generally invoked directly by the system software, or by cron jobs. They should live in the `$TopDir/lib/bin` directory.

hdf2gif

After an integrated browse product is fetched, this extracts the images embedded in it, and writes them out as GIF files. Uses the HDF libraries. *Language: C.*

imsdepval

Generates the dependent valids lists (and the lists of valid categories and extended search criteria) for the search form. *Language: C. Interface: IW::DepValid.*

imscleanup

Perform the cleanup of old session directories and old browse files. Normally run in a cron job. *Language: Perl.*

imscleanaccounts.pl

(Optional) Perform the cleanup of persistent user accounts. This script is only run if your installation of the IMS Web Gateway is setup to have persistent user accounts. It is normally run as a cron job. *Language: Perl.*

imsclient

Performs the action of sending an ODL query to the remote DAAC servers, writes out the status every few seconds to a status file, and outputs the results to files in the session directory. Generally invoked by **imsrunclient**. *Language: C. Interface: imsrunclient.*

imscovermap

Create the spatial coverage map GIF (and the postage stamps) for one or more granules, displayed in the "map" results screens. *Language: C.*

imscovertime

Create the temporal coverage graph GIF for one or more granules, displayed in the "time" results screens. *Language: C.*

imsgdbdump

Old program for extracting objects from the now-unused GDB files; still used for returning media/subsetting options for a given data set/granule. *Language: C.*

imsrunclient

Runs the **imsclient** program, performing some additional processing afterwards to convert results to Perl objects. *Language: Perl. Interface: IW::Client.*

waisq

Standard WAIS query program. Used in performing "guide" searches.

3.3.3 Administrator utilities

These are generally run directly by administrators of a Gateway. They should live in the `$TopDir/bin` directory.

configure

Configure the system before a full compile, during installation. See the discussion of the **configure** program. *Language: Bourne shell.*

myconfig

A very small shell script which simply invokes **configure** with the appropriate required parameters. *Language: Bourne shell.*

setup

Update the system after a change to the **SETUP** file. See the discussion of the **setup** program. *Language: Perl.*

3.3.4 Developer utilities

These are generally run only by developers. They should live in the `$TopDir/bin` directory. The ones of notaqble significance are:

fixscript

Fixes up the first line of Perl scripts to use the Perl path defined un **myconfig**. *Language: Bourne shell.*

imsstats

Mails a copy of the event log to the **imswww-stat** account as defined by the **StatsEmail SETUP** variable. *Language: Perl. Not part of the Web Gateway development.*

ipod2html

Converts an IPOD document to an HTML tree. Used to build the user and design manuals. See the **Documentation** chapter. *Language: Perl. Not part of the Web Gateway development.*

ipod2latex

Converts an IPOD document to a LaTeX document, for conversion to Postscript. Used to build the user and design manuals. See the **Documentation** chapter. *Language: Perl. Not part of the Web Gateway development.*

mkvalids

Compiles the valids file into a quickly-loadable, pre-parsed format. *Language: Perl.*

3.4 Libraries

3.4.1 The Perl "IU" library

Location: $TopDir/lib/perl/IU.

This library contains the lowest-level Perl modules. Basically, if your modules does *not* require that it be running under a CGI script, then it really should go here for maximum reusability.

Base Configuration:

```
IU::           IU library intrinsics, like puke().
IU::Err        Low-level error handling and error values.
IU::IW         Makes IU modules savvy about the IW environment,
               so methods like from_CGI(), etc. will work when used
               in a CGI script.
IU::SETUP      Auto-generated Perl version of the SETUP file.
IU::Util       Misc. utilities.
IU::Version    The system version.
IU::SystemName The system's official name.
```

External file/data formats:

```
IU::Chunk         Parse/generate the internal format for structured
                  information exchange.
IU::ClientChunks  Chunk compositing for partial results
IU::Coverage      Granule/dataset geographic coverage.
IU::Diag          Scan the diagnostics file (for dying gasps).
```

```
IU::Escape        Escape string literals for shell scripts, Perl, etc.
IU::ODL           Cheap ODL parser.
IU::SetupFile     Parse the SETUP file.
IU::Storable      A storable object.
IU::Store         A place to put IU::Storable objects... derived from
                  GDBM_File.
```

External agents/processes:

```
IU::HTTP       Cheap way to do HTTP GETs (deprecated).
IU::Mail       Send email.  Obsolete: should be replaced with MIMEIU::Lite.
IU::Spawn      Spawn an asynchronous process that we can keep track of.
```

Logging (high-level):

```
IU::Event      Log to the SCRS standard log file.
IU::EventEntry Describes an event log entry.
IU::Log        Provides debug(), warning(), error(), etc.
```

Logging (low-level):

```
IU::Logger       Actual logging mechanism.

IU::ExclFileLogger  Log to an exclusive log file.
IU::FileLogger   Log to a file.
IU::SubLogger    Log to a subroutine.
IU::SysLogger    Log to the syslog.
IU::Syslog       Low-level syslog interface (less broken than SyslogIU::).
IU::TermLogger   Log to the terminal.
```

Gateway entities:

```
IU::FileCache   Manage a cache of files, 1 or more files per cache entry.
IU::InvStore    Storage for data sets/granules/etc. from inventory search.
IU::Session     A session directory in th Web Gateway.
IU::SharedStore Cacheless shared access to IU::Storable objects
IU::WWWValids   ODL configuration file for the system
```

IMS entities:

```
IU::Account                A user account at a DAAC.
IU::AccountStatus          Results from an account status request
IU::AccountStatusRequest   A DAAC account status request
IU::Addr                   A contact address.
IU::ArchiveODL             ODL used to describe the DAAC connections.
IU::BrowseFile             A browse (preprocessed data with description) file
IU::Dataset                A data set returned from a search.
IU::FancyURL               A URL with an attached comment.  Intended for the
                           various "URL" groups.
IU::FileReader             (Misnamed?) Read state/country names into
                           structures.
IU::Granule                A granule returned from a search.
```

```
  IU::Interface::Orderable  Class describing access methods that orderable
                            objects must define.
  IU::OrderStatus           ODL for an order status request.
  IU::OrderStatusInfo       ODL for an order status result.
  IU::Package               ODL describing an available package
  IU::PkgCombo              A tuple of package, processing option, media type,
                            and media format, possibly with other info.
  IU::PSA                   ODL describing PSAs in wwwvalids.conf
  IU::SpecializedCriterion  ODL extension for arbitrary search request data
  IU::SpecializedResult     ODL extension for arbitrary search result data
  IU::TrackedOrder          Orders descriptions from DAACs
  IU::TrackedOrderManager   Creates/manages tracked order objects
```

ECS interfacing entities:

```
  IU::ECSProfileRequest     Create an ECS profile request
  IU::HandlingFee           ODL support for DAAC one-time service charges
  IU::LineItemResult        ODL support for price estimate line items
  IU::MediahandlingFee      ODL support for per-medium charges by DAACs
  IU::MediaMaximumCapacity  ODL support for media sizes
  IU::PriceEstimate         ODL support for price estimate requests.
  IU::RequestResult         ODL support for subsetting requests.
  IU::ResultGroup           ODL support for subsetting results.
```

Geographic search regions:

```
  IU::Coord             - degree-minute-second parsing
```

```
  IU::Geo::Base        - an abstract geographic region
  IU::Geo::Global      - global granules only
  IU::Geo::PathRow     - path-row-based geographic search, like a PATH_ROW_LOC
  IU::Geo::Point       - point search region, like a POINT_LOC
  IU::Geo::Polygon     - polygonal region, like a POLYGON_LOC
  IU::Geo::Range       - rectangular search region, like a RANGE_LOC
  IU::Geo::WRS         - interface to the World Reference System mechanisms
```

```
  IU::Vec               - XYZ vectors for map operations
```

Search criteria:

```
  IU::Category                 - a category (e.g., PARAMETER) in a query
  IU::Options                  - a collection of criteria for subsetting
  IU::Query                    - a query object

  IU::Criterion                - abstract search criterion
  IU::Criterion::FixedField    -
  IU::Criterion::FloatingScene -
  IU::Criterion::Geo           - a geographic region
  IU::Criterion::InputDate     -
  IU::Criterion::InputNum      - a single number/date in a query
  IU::Criterion::InputRange    - a range in a query
  IU::Criterion::InputText     - a text field in a query
  IU::Criterion::SelectMany    - a multiple-selection list in a query
  IU::Criterion::SelectOne     - single-selection menu in a query
  IU::Criterion::Time          -
```

Utilities:

```
IU::Ask                   - yes-or-no prompt
IU::DAACExpansions        - short-name to lomg-name mapping for DAACs
IU::Errors                - object used to collect error strings,
IU::Filehandle            - Added first-class filehandle support to
                            Perl 5.005 (later versions implement this
                            themselves). Possibly should be deprecated.
IU::HTAccess              - create a .htaccess file\
IU::Interface::CGI        - CGI data manipulations
IU::LockFile              - lock file management
IU::Setup                 - internal format of SETUP
IU::StatusCodes           - Convert ODL status codes to text messages
IU::URL                   - Resolve relative URLs using SETUP info
IU::UserAdmin             - generic password management
```

3.4.2 The Perl "IW" library

Location: $TopDir/lib/perl/IW.

This library contains the higher-level Perl modules that are specifically geared towards the WWW interface. If your module does a "use" of any "IW" module, or if it expects to be running in a CGI environment, then it probably needs to go here.

Application core interface:

```
IW::Agent            - attributes/capabilities of the browser
IW::App              - routines for an EOSDIS V0 CGI application
IW::Java             - interfacing Java applets to the system
IW::Mode             - dispatch activity based on the "mode" CGI tag.
IW::Navigator        - centralization of intra-application links
IW::Popup            - a subform invoked as part of another form
```

Base configuration:

```
IW::Babel            - centralize all conversion to/from printable names
IW::Util             - utility functions for the IMS/WWW CGI scripts
```

CGI input and output:

```
IW::CGI              - CGI interface for the EOSDIS V0 scripts
IW::Reply            - CGI reply interface for the Web Gateway scripts
IW::ReplyHead        - lightweight replacement for broken CGI::Response
```

HTML output:

```
IW::HTML             - HTML stream subclass, for Web Gateway applications
IW::Note             - an admonishment
```

HTML "widgets": Anything which needs rendering, but which is fairly complex, is likely to have been given its own class under IW::Widget.

```
IW::Widget::DepValids    - manage dependent valids selection (mode SLCTDEPVAL)
IW::Widget::QueryEdit    - manage the "grey navigator" for editting queries
IW::Widget::QueryUpload  - widget for query upload/download
IW::Widget::SearchType   - widget for showing/selecting the search type
IW::Widget::ShowCategory - widget on search form for displaying category
IW::Widget::ShowDepval   - widget on search form for  dep. valids category
IW::Widget::URLTable     -
IW::Widget::ViewBrowse   -
```

Search form "widgets": Like general-purpose widgets, but designed specifically for search forms.

```
IW::SrchForm::AbstrctWidget          - Base widget class
IW::SrchForm::AbstrctWidgetBox       - Base widget layout class
IW::SrchForm::AccountStatusWidget    - account status for & updates
IW::SrchForm::AccountStatusWidgetBox - Layout widgets for account
                                       atatus searches & handle updates
IW::SrchForm::AirsButtonsWidget      - Search button for AIRS browse
IW::SrchForm::AirsGeoWidget          - Geo entry for AIRS browse
IW::SrchForm::AirsLinksWidget        - links specific to AIRS browse
IW::SrchForm::ButtonsWidget          - "Start search" and related buttons
IW::SrchForm::DepValidWidget         - Dependent valids display and
                                       interface.
IW::SrchForm::GeoWidget              - Geoselection (general) interface
IW::SrchForm::GranIdWidget           - Granule ID input
IW::SrchForm::GranuleOptionsWidget   - Granule limit
IW::SrchForm::GuideTextWidget        - Guide search text
IW::SrchForm::LoadWidget             - Upload query widget
IW::SrchForm::LocalIdWidget          - local granule ID input
IW::SrchForm::MainWidgetBox          - Main search form layout & management
IW::SrchForm::NodesWidget            - Ascending/descending node input
IW::SrchForm::OptionsWidget          - Search options input
IW::SrchForm::OrderStatusWidget      - Order status info input
IW::SrchForm::SrchTypeWidget         - Search type change widget
IW::SrchForm::StickyNote             - Diagnostic/messaging utility widget
IW::SrchForm::TimeWidget             - Time input
```

User preferences/info:

```
IW::Address          - gerenalized address info
IW::DaacAddress      - DAAC address info
IW::Disclaimers      - dataset/granule disclaimers
IW::User             - user preferences/profile information
IW::UserAddr         - represents a user address
IW::UserAffil        - represents a user affiliation
IW::UserDevel        - represents a developer's special user preferences
IW::UserDisplay      - represents user's presentation preferences
IW::UserKeys         - represents a user data access key(s).
IW::Query            - an IU::Query.
```

Support for search form:

```
IW::Date             - date/time objects
IW::DepValid         - interface to the dependent valids mechanism
```

```
IW::CommStat::DIRECTORY_SEARCH - monitor a directory search's progress
IW::CommStat::INVENTORY_SEARCH - monitor an inventory search's progress
```

```
IW::ModeABSTRACTSRCH - base search for architecture
```

Support for geoselection on search form. IW::GeoWidget is the namespace for classes that are used to perform a particular type of geoselection:

```
IW::GeoWidget::Mapfile        - a GIF file for a map

IW::GeoWidget::Base           - abstract class for geoselection "widgets"
IW::GeoWidget::GlobalGranOnly - select only global granules
IW::GeoWidget::GlobalMap      - do geoselection from a global map
IW::GeoWidget::JavalMap       - do geoselection via Java applet
IW::GeoWidget::PathRowTypein  - do geoselection by typing in paths/rows
IW::GeoWidget::PointRowTypein - do geoselection by typing in a point
IW::GeoWidget::PolarMap       - do geoselection from a polar map
IW::GeoWidget::RangeMap       - do geoselection from a range map
IW::GeoWidget::RangeMapTypein - do geoselection by typing in range
```

Support for communication status screen:

```
IW::Client           - spawn/poll the standalone client
IW::Comm::Manager    - Comm status interface mixin class
IW::CommStat         - output the communication status screens
IW::Conn             - a connection status object (one per DAAC)
IW::PostProcess      - a post-processing status object (one per activity)
IW::StatusBar        - a status object with an LED display
```

Support for displaying results:

```
IW::DatasetList      - utilities for outputting a data set list
IW::GranList         - utilities for outputting a granule list
IW::GranMap          - granule spatial coverage.
IW::GranTime         - granule listing
IW::Minitable        - tabulated information in data set/granule list
IW::ObjectList       - abstract class for outputting a data set/granule list
IW::Sort             - create multi-key sort menus, and do the sorting
IW::SummaryList      - object list for presenting tabulated information
```

```
          ObjectList
               |
          SummaryList
           /       \
   DatasetList    GranList
```

Support for browse:

```
IW::Comm::BrowseReq        - integrated browse request
IW::Comm::FTPBrowseResult  - ftp browse results
```

```
IW::CommStat::FTP_BROWSE          - show an FTP browse request progress
IW::CommStat::INTEGRATED_BROWSE   - show an FTP browse request progress
```

Support for order:

```
IW::CartList         - shopping cart contents
IW::Order            - a user's order
```

Support for price estimates:

```
IW::Comm::PriceEstimateResult

IW::CommStat::PRICE_ESTIMATE_REQUEST
```

Support for ECS accounts:

```
IW::Comm::AccountStatusResult
IW::Comm::ProductResult
IW::Comm::ProductStatusInfo
IW::Comm::ProfileRetrievalResult
IW::Comm::ProfileSubmitResult
IW::Comm::ProfileUpdateResult
IW::Comm::UserPasswordChangeResult
```

```
IW::CommStat::ACCOUNT_STATUS_REQUEST       - show account status progress
IW::CommStat::PRODUCT_REQUEST              - product request progress
IW::CommStat::PRODUCT_STATUS_REQUEST       - product status request progress
IW::CommStat::PROFILE_RETRIEVAL_REQUEST    - show profile retrieve progress
IW::CommStat::PROFILE_SUBMIT_REQUEST       - show profile retrieve progress
IW::CommStat::PROFILE_UPDATE_REQUEST       - show profile retrieve progress
IW::CommStat::USER_PASSWORD_CHANGE_REQUEST - show password change progress
```

Support for ADMIN tools:

```
IW::SessionList      - utilities for outputting a session list
```

3.4.3 Additional Perl modules

Location: $TopDir/lib/perl.

The following are modules so generic in capability that they were developed outside the IU/IW namespace.

```
HTML::Stream   - Easy output of HTML to a filehandle.
OHTTPD         - Custom Perl HTTPD infrastructure (deprecated).
```

```
imsrunclient   - Core of the imsclient/EDG interface.
makePersistent - Sets the EDG up for use with persistent users.
makewelcome    - Builds static pages and blocks inappropriate access.
nph-ims        - Core CGI program
run_autoxfer   - Automatic valids transfer.
setup          - Applies the SETUP file to the EDG.
```

3.4.4 CPAN modules

Location: $TopDir/lib/perl/CPAN.

These are modules which come from CPAN (the Comprehensive Perl Archive Network) and which can easily be installed outside of Perl itself.

```
Digest::MD5    - Support module for libwww-perl
HTML::Parser   - Support module for libwww-perl
MIME::Base64   - Support module for libwww-perl
Text::Template - Used to create XML and HTML files from templates
URI            - Support module for libwww-perl
libwww-perl    - HTTP protocol support (ModeGUIDEXML, others)
```

3.4.5 The C "GCMD" library

Location: $TopDir/src/GCMD. Not developed by the Web Gateway team.

Library of GCMD (Global Change Master Directory) routines which probably is/was used by IK_lib for performing GCMD (DIF) searches.

3.4.6 The C "IK" library

Location: $TopDir/src/IK_lib. Not developed by the Web Gateway team.

This is the library which forms the foundation of all the V0 IMS executables.

3.4.7 The C "IW" library

Location: $TopDir/src/IW_lib.

This library sits on top of the IK library. It adds functionality to IK, and also contains modules for web-related activities in C (e.g., for **imsmap**).

There's the extremely-generic stuff:

```
IW          - top-level generic routines
IW_Chunk    - output of structured information
IW_Cgi      - CGI interface code
IW_Log      - functions for logging events
IW_ODL      - things to make some ODL processing easier
```

Perl enthusiasts will be glad of the following:

```
IW_Array    - variable-sized arrays
IW_Hasht    - variable-sized hashtables
IW_Str      - variable-sized strings, and some string utils
```

And people building IMS-related applications will want:

```
IW_Conn     - connection status (used by imsclient)
IW_Key      - data set and granule keys, in ASCII form
IW_Stage    - stage of processing (used by imsclient)
```

Chapter 4

System data dictionary

These are the configuration variables, environment variables, etc., currently in use throughout the system. They are the medium by which information is represented and exchanged.

4.1 Configuration variables

The following configuration variables are the ones set up by the `configure` and `setup` scripts.

4.1.1 Hardcoded/derived variables

The following variables are either hardcoded into the system, derived from the `configure` script, or derived from each other.

CgiDir

The same as "**TopDir**/cgi-bin".

This is where the Web Gateway CGI scripts are, as a directory.

DocsDir

The same as "**TopDir**/docs".

This is where the Web Gateway HTML docs are, as a directory.

LibDir

The same as "**TopDir**/lib".

This is where various configuration files, binaries, and Perl packages are kept.

LogDir

The same as "**TopDir**/log".

Where log files related to system-wide activity – statistics, cleanup history, usage history, etc. – are to be placed.

Customization: if you want to have all logs placed in a different directory (for example, `/var/log/imswww`), you can delete the **`TopDir`**`/log` directory and replace it with a link to the directory of your choice, making certain that this new log directory is writable by world. *Be careful!* If you run multiple versions of the gateway at any one time, make certain that the logs are not all routed to the same directory, since their filenames are identical, and they will overwrite each other.

TopDir

The top directory of the distribution; that is, where the `configure` script, the `cgi-bin` directory, the `docs` directory, etc. are located.

This value is set up by `configure`. The following variables are derived from it:

- **CgiDir**, which is "**TopDir**/cgi-bin".
- **DocsDir**, which is "**TopDir**/docs".
- **LibDir**, which is "**TopDir**/lib".
- **LogDir**, which is "**TopDir**/log".

These variables are holdovers from the earlier days of the system, when all these values were independently controllable (a feature which is unnecessary, and which needlessly complicated the setup process). However, since they act as convenient abbreviations for the more popular directories, they have been retained.

Version

This version of the Web Gateway. Typically, this will be a number like ‘1.2’ or ’1.2.3’, with a possible trailing ’b’ and number to indicate a Beta-test package: ’1.2.3b7’. The scheme is pretty standard:

- The first number is the *major release*. New major releases are output whenever the system undergoes extensive modification. Major releases often differ significantly in their architecture or features.

- The second number is the *minor release*. New minor releases are output when new features have been added, but when the overall architecture remains the same.
- The third number (if present) is the *patch level*. New patch releases are output when bugs have been detected and fixed, but when no new features have been added.
- If a 'b' is present, the number following it denotes the beta level of this release.

The version number is hardcoded into the release; you cannot change it, nor should you attempt to try.

4.1.2 Critical SETUP variables

The following variables in the **SETUP** file are critical towards getting your site to run.

AccountMax

How much disk space should be allowed for each user account. This maximum limit should be enough to handle the results of a large search. If a user exceeds this size and they are not signed-in, their results will be purged. It they are signed-in they will be warned. Specify a size with units of Mega Bytes. It should typically be:

```
"20mb"
```

AllAccountsMax

How much disk space should be allowed for ALL user accounts. This maximum limit should be quite large, since each account can be typically 1-5 MB (although could be smaller or bigger). Specify a size with units of Mega Bytes or Giga Bytes. It should typically be:

```
"1gb"
```

AdminCrypt

The crypt'ed password for the administrators to use, exactly as you'd see it in an /etc/passwd file... but PLEASE, DO NOT USE AN ACTUAL SYSTEM PASSWORD FOR THIS! Note: the special value "NO PASSWORD" allows ANYONE in!!!

CalendarEmbed

PATH to the CGI script which embeds calendar info. (You'll need to update this to reflect where your cgi-bin is relative to your docs/imswelcome. It has to be a relative path! The sample value is probably right if you're following our install directions.)

Recommended: whatever works, but "../../cgi-bin/dev/grab-cal.cgi" should be it.

CalendarTimeout

The calendar-embed script will wait this many seconds before deciding the calendar is unavailable.

Recommended: 10 (minimum).

CustomizeMetadata

Default for whether or not we allow the user to customize metadata coming back as results.

Recommended: YES.

CgiUrl

Where the Web Gateway CGI scripts are, as a URL with a trailing '/'.

You should set this so that it "points" to the top-level `cgi-bin` directory of the current installation.

CriticalAllAccountsMax

How much disk space should be allowed for ALL user accounts before more drastic measures are invoked to clear space. This maximum limit should be quite large, since each account can be typically 1-5 MB (although could be smaller or bigger). It should generally be larger than AllAccountsMax. Specify a size with units of Mega Bytes or Giga Bytes: e.g., "500mb" or "2gb". It should typically be:

```
"2.5gb"
```

DisplayBrowseSearch

Should we display the AIR Browse search stuff? Recommended: "YES".

DocsUrl

Where the Web Gateway HTML docs are, as a URL with a trailing '/'.

You should set this so that it “points” to the top-level `docs` directory of the current installation.

EcsProfilegateway

Which daac in archive.odl is for ECS profile submit requests. If multiple ECS profile gateways are available, they should all be listed here and separated by semi-colons(;). Recommended: *empty* for now.

EcsProfileSmc

This is the name of the User Profile Gateway to ECS that all account messages must go. Recommended: "SMC-UPG"

EcsRegistration

Where is the ECS User Registration tool? Recommended: Undetermined as yet. Leave blank for now.

GaeaDataDir

Path to the directory that holds the IMS valids. It will also contain links to the **GaeaDistribDir** directory. It should typically be:

```
"/home/imswww/support_files"
```

GaeaDistribDir

Where the Web Gateway distributed support files are located. This should be the full path to the files obtained in the **Preflight checklist**, Step 5.

GranColumnsMax

Maximum number of columns for a Data granule listing. This is only an advisement, not a hard limit. Recmmended: 20.

HavePersistentSessions

Allow the users to have Persistent directories on the server. The default is "NO", change it to "YES" to enable this option. Recommended: "YES".

HttpdHost

This is the hostname that will be used in URLs pointing back into the system, and also for e-mail sent to/from this site; leave it empty ("") for the default hostname.

ListLength

Default number of rows per page for dataset/granule/cart list. Recommend: 20.

ListMax

Default maximum number of items total which can be in a granule/cart list. Recommend: 1000.

MapCgiScript

Map server script used by the Geo selection java map.

-
-

Default: "imswdbtile.cgi". Recommended: "drawglobe.cgi".

MapResolution

Used when MapCgiScript cwis set to "drawglobe.cgi", You must install the correct map database in support_files/support_dist/map. See **Installation instructions** for more details.

-
-
-

Recommended: "ULTRA".

MorePaths

This is added to the $PATH variable, to pick up...

```
compress     (for unpacking hdf.Z files)
gunzip       (for receiving compressed trees, and HDF.gz)
gzip         (ditto)
waisq        (for guide searches)
zcat         (for hdf2gif)
```

PerlHttpdPort

Port number for the Web Gateway *custom* Perl HTTPD... NOT the normal one! UNDER NO CIRCUMSTANCES SHOULD YOU PUT THE NUMBER ’80’ HERE! Since this is a deprecated feature, you should leave this value set to an empty string.

PersistentDir

Base Directory for the Persistent Sessions. It should typically be:

```
"/imsdev/persistentUser/"
```

PolicyAccess

Determines if "US Government", "local", or no ("") access policy link is displayed on the welcome page.

PolicyPrivacy

Determines if "US Government", "local" or no "" privacy policy link is displayed on the welcome page.

PolicySecurity

Determines if "US Government", "local" or no "" security policy link is displayed on the welcome page.

PrimaryPort

Port where the primary HTTPD on your system listens. Should only be changed if you run your primary httpd on a port other than 80.

Recommended "80".

PSAMax

Because of DAAC database limitations, searches with more than a certain number of PSAs will fail. Until this is changed, set this to "2" (the maximum possible).

ReleaseType

Which type of release: Public, Hidden, Preops, or Hidden-Preops ? Leave this as "Public" unless you know why you’d want to use one of the others.

SecuredDir

The name of the password-protected subdirectory in **CgiBin** used to support persistent users. The installation instructions specify that this is "secured", so leave it that way unless you're sure of what you're doing.

SendmailCmd

How to invoke sendmail on this system, so that (1) the message is scanned for recipients, and (2) a dot on a line by itself doesn't end input. E.g.: "/usr/lib/sendmail -t -i".

SiteName

Set the name you wish to use for your installation. This must be a simple string. For example: "NASA GSFC". This string will appear on the Welcome page in the welcome line: "Welcome to the NASA GSFC installation of the EOSDIS V0 Web Gateway" and in the start search line: "Start Search on NASA GSFC"

SiteSpecDir

Specify the directory you will use to hold site specific info and documents for the gateway, such as welcome_header.html and weclcome_footer.html. We recommend "/usr/people/imswww/site_specific".

StatsId

Our local signature, for stats. DO NOT FILL THIS IS WITH A RANDOM NUMBER! CONTACT THE Web Gateway DEVELOPERS FOR A LEGITIMATE NUMBER FOR YOUR SITE!!!

WwwUser

The user and group that HTTP scripts should run as. This must be identical to your httpd's configuration! UNDER **NO** CIRCUMSTANCES SHOULD THIS BE ROOT!!!

4.1.3 Site-specific cache/session options

The following variables in the **SETUP** file affect how the various global resources are managed. Adjust these based on your site's capabilities.

CacheComfort

What’s a *comfortable* size for the browse cache? Cleanups continue until the cache shrinks below this size. Specify a number with units: e.g., "500kb", or "1mb", or "4mb".

CacheMax

What’s the *maximum* size of the browse cache? E.g., "5mb". Cleanups are forced when the cache exceeds this size. CURRENTLY, the cache MAY exceed this size BETWEEN cleanups.

CacheShelfLife

How long can browse products remain unrequested in cache until they’re "stale" and ready for deletion? Specify a time with units: e.g., "40s", or "20m", or "1h". Recommend: "30m".

CacheLinger

How long after the logical deletion of an item from cache until the item can be physically deleted? Recommend: "2m".

SessionLife

How long can sessions remain unused until they’re deemed "stale" and are deleted? Specify a time with units: e.g., "40m", "2h", etc. Recommend: "4h".

LogoutTime

How long should a logged in account be inactive before we log the user out? Used by imscleanaccounts.pl. Specify integer time, optional blanks, and units. Recommend: "1 d" for one day.

ResultsLifetime

How long can results be kept in a user account which has not been accessed (to free up disk space and keep info from becoming out-of-date)? Used by **imscleanaccounts.pl**. Specify integer time, optional blanks, and units: e.g., "10d" for 10 days. Recommend: "14 d" for fourteen days, the valids cycle.

GuideKillTime

Under CRITICAL disk space conditions (account usage exceedingCriticalAllAccountsMax), if guide search results are not accessed for this period of time, then they will be deleted - even from accounts where the user is signed in. Specify a time, optional spaces, and units: e.g. "5m" for 5 minutes. Recommend: "15 m".

4.1.4 Site-specific miscellaneous options

You can change these if you want, but you probably don't need to.

DictFile

The dictionary file used as a corpus for generating random passwords. Assumes a path realtive to the "dict" directory in your "docs" directory unless you specify a fully-qualified pathname. We supply a relatively-reasonable English dictionary as the default, Recommended: "knuth_words".

EmailGasp

Should copies of serious "dying gasps" be e-mailed to the SiteAdmin address (below)? YES or NO. Recommended: YES.

ForceDebug

Turn debugging on/off for entire site (usually for debugging or development only!). YES = force debugging on, NO = don't force debugging on. This also enables full debugging output on dying gasps; not recommended unless your test system is inside your firewall. Recommended: OFF.

GuideLimit

The maximum number of documents a guide search will return. This just sets a default value; the users can override this. Recommended: 100.

GuideType

Historically, there was a change in the WAIS query type around April 1997; the old WAIS query type was called 'OLD', and the newer one 'NEW'. It was necessary to set things up so that a transitional period was available between the old and new WAIS query syntaxes. **OLD** was used prior to April 1997;; **BOTH** was used during a several-month transition period; **NEW** was used thereafter. This worked fine until the new XML-based guide system was deployed, at which point it became necessary to handle the transitional phase between WAIS and the new guide implementation (which couldn’t be called **NEW**, because **NEW** had already been co–opted to mean the "new" WAIS syntax) and to be ready for the migration away from WAIS to be complete.

There are therefore now **4** different guide types:

- OLD - the older WAIS syntax: "Data_Center=NSIDC and Parameter=SNOW_DEPTH". *Don’t use.*
- NEW - the newer WAIS syntax: "arch7 and parm1006". *Don’t use.*
- BOTH - the *WAIS* transitional syntax: "arch7 or Data_Center=NSIDC and parm1006 or Parameter=SNOW_DEPTH". *Don’t use.*
- XML - "datacenter/nsicd and parameter/snow_depth". *Don’t use **yet**.*
- TRANSITIONAL - run both XML and WAIS searches, automatically dispatching the search to the right server(s). *Use this one.*

At some point in the future, when the transitional period is over, we will want to change over to **XML** and eliminate the WAIS-based code altogether,

LogLevel

Logging level for IMS event logging. The IMS developers will inform you as to any other settings required; for now, please leave this as-is. Recommended: "normal".

RandomContext

Context used to determine "next letter" for random passwords. "6" seems to provide passwords which are bizarrely memorable, but not actual English.

RandomLength

Length of randomly-generated passwords. "9" seems long enough to be pretty secure, but short enough to still be memorizable.

SecureServer

Whether or not you ar using a secure server (SSL support). Since this support is currently *very* experimental and extremely buggy, we recommend you leave this as "0" (*not* using SSL).

SiteAdmin

What user account on this host should site-specific mail go to?

SpidexTimeout

The length of time we should wait before giving up on a new (XML-based)

UpdateEvery

How often to update search status, in seconds.

Note: This value is dependent on a number of things; if your users' browsers don't finish receiving and rendering the CommStat pages before the update starts fetching the page again, the system will appear to hang until the search completes.

Recommended: 15 seconds (with debug off) or 1 minute (with debug on).

4.1.5 External URLs/addresses

You probably won't need to change these.

ExplorerUrl

URL for the "Microsoft Internet Explorer" button.

HomeEmail

Who to e-mail with questions/comments about the Web Gateway system. This is the address used by the "Comments" form.

HomeUrl

URL of the Web Gateway home page.

HdfInfoUrl

URL pointing to info on the HDF format, and available tools.

GcmdHomeUrl

URL for the GCMD home page. A hyperlink to GCMD is provided so novice users can read up on it.

GcmdGetdifUrl

URL for the GCMD "getdif" script. This is the CGI script used to retrieve a DIF when you request "brief info" on a subject.

ImsDefnUrl

URL for the IMS definitions directory, with trailing '/'.

ImsHomeUrl

URL for the IMS home page.

ImsGlossUrl

URL for the IMS Glossary of Terms.

NetscapeUrl

URL for the "Netscape Now!" button.

4.1.6 Run-time variables

These are not configuration variables *per se*, since they are computed at run-time. However, they are used so often in generating file and directory names that it is essential to give them special mention.

PID

This is not even a real variable; it's just our way of referring to the current process id.

SessionId

This holds the **session id**, which is the unique identifier for the current user session.

```
$SessionId = '806269173-1138'
```

SessionDir

This holds the **session directory**, which is the temporary directory set aside for the use of the current user session.

```
$SessionDir = '/tmp/imswww-usr/806269173-1138'
```

4.2 Environment variables

The principle **SETUP** variables will be available as environment variables to processes running under the gateway. Here are the environment variables for the principle configuration variables:

- **TopDir** is **$IW_TOP_DIR**
- **SessionDir** is **$IW_SESSION_DIR**
- **SessionId** is **$IW_SESSION_ID**

The following are also defined, but are deprecated as of v.1.2 (they are all derivable from $IW_TOP_DIR):

- **CgiDir** is **$IW_CGI_DIR**
- **DocsDir** is **$IW_DOCS_DIR**
- **LibDir** is **$IW_LIB_DIR**

As you can see, the name for the environment variable corresponding to a given **SETUP** variable is straightforward: instead of “MixedCase” style, use the same words in “CAPS_AND_UNDERSCORES” style, with the prefix “`IW_`”.

In addition, the following environment variables will also be available:

GAEADATA_DIR (IMS)

This variable will tells the client where to get the configuration files. This variable should be the same for every instances of the backends.

USRDATA_DIR (IMS)

This variable contains the directory that holds the user information of the client. It actually holds user-specific information that the same user can access in different sessions. It has information like user profiles and user saved results.

GAEATMP_DIR (IMS)

This variable contains the directory that holds the session-specific information, i.e. information that will be deleted after a user finishes a session. It will holds the syslog file, the messages passed back and forth between the client and server, and all the GDB files.

SYSLOG_LEVEL (IMS)

This variable decides the log message level.

4.3 CGI tags

The following are the input tags which are common to all Web Gateway CGI scripts.

4.3.1 dskey

A “data set key”, which is a brief way of specifying a data set in the search results.

```
dskey = "6/203"
```

This is of the form:

```
archive-tag/dataset-tag
```

The presence of this CGI tag usually indicates that the requested operation should be done in the context of the given data set(s).

4.3.2 gkey

A “granule key”, which is a brief way of specifying a granule in the search results.

```
gkey = "6/203/AVH19920912_2033"
```

This is of the form:

```
archive-tag/dataset-tag/granule-id
```

The presence of this CGI tag usually indicates that the requested operation should be done in the context of the given granule(s).

4.3.3 mode

Mode of operation. This is a way of having the same CGI script perform several different functions, based on a form tag.

```
mode = "GRANLIST"
```

This “mode” mechanism is necessary because in HTML 2.0, forms may only be submitted to a single URL, although in some cases it is necessary to have different “submit” buttons on the form do very different things. An example is the “Results” form, where users can choose to re-sort, order, or do a granule list.

The valid values of this tag vary with respect to the script it is submitted to.

If this tag is not supplied, the CGI script will typically have an assumed default.

4.3.4 select?* and "msubmit"

This special form of CGI tag allows us to use “image” inputs in a simple way to mimic the behavior of multiple “submit” buttons.

Prior to Internet Explorer 4.5, it was possible to encode part of the query in the name of a INPUT TYPE=IMAGE tag, like this:

```
select?tag1=value1&tag2=value2&...&tagN=valueN
```

For example:

```
<INPUT
    TYPE=IMAGE
    SRC="/~imswww/pub/docs/btns/sub-glist.gif"
    NAME="select?mode=GRANLIST&submode=REV">
```

When this was clicked, the image was treated as a server-side image map, and the CGI script invoked in this was received the click coordinates via that input’s NAME... as you would expect, like this:

```
select?mode=GRANLIST&submode=REV.x = 20
select?mode=GRANLIST&submode=REV.y = 32
```

However, IE 4.5 *broke* this; it does not handle server-side image maps properly. all you get is two null x= and y= tags, without the select information encoded anymore.

The necessary change was to add a hidden field following every image that was to be used as an "msubmit" button. Each msubmit button is given a unique name, and the hidden field following it is also given the same name. It so happens that the bad x and y tags are embedded in the incoming CGI POST stream exactly at the point where the image button appeared. If we have a hidden field directly after the button, we can locate the hidden data we really wanted, since it occurs immediately after the bad tags. IW::CGI.pm handles all this internally, but it’s described here in case another browser release breaks it. Again.

The Web Gateway scripts recognize such CGI inputs on initiallization, and they decode them, adding the “enclosed” tag/value pairs, as though the script had actually received them... like this:

```
mode = GRANLIST
submode = REV
```

In addition, the coordinates of the click are made accessible via the special “selected” tag:

```
selected.x = 20
selected.y = 32
```

As you can see, it does not make sense to have more than one “submit?” tag sent to a CGI script. Their sole purpose is to behave like a SUBMIT button which contains enough context information to alert the script exactly *which* submit button was pressed.

4.3.5 sid

Session id. If supplied, this tag indicates to the CGI script that it is resuming an existing session.

```
sid = "806950388-419"
```

It is an error for this tag to indicate a session that does not exist.

It is typically an error for the tag not to be supplied to a CGI script in the gateway; exceptions are modes like SRCHFORM (the default if no mode is given), ADMIN, BROWSERTEST, and HELP.

4.3.6 submode

This is a recent refinement of the **mode** tag. Whereas the "mode" tag indicates which Perl module to load, the "submode" tag indicates what initial action should be taken.

4.3.7 u

The **uniquifier** tag (q.v.).

4.4 IMS Data Dictionary

This document, which contains the full definitions of all the standard ODL messages in the V0 protocol, is a Web Gateway developer’s near-and-dear friend.

It is available on-line in directory: **ftp://killians.gsfc.nasa.gov/pub/dd/**. Get the .ascii file.

Also useful is the on-going document of proposed ODL extensions, at: **http://harp.gsfc.nasa.gov/documents/odl_extensions.html**.

4.5 Formats

This section discusses file/data formats which are used extensively by the system.

4.5.1 Chunks

This is an informal, lightweight ASCII-based means of representing and communicating structured information. It is basically like ODL, but optimized and intended for machines – not humans – to use.

For example, here is an **ODL** group:

```
GROUP = "*root*"
  NAME       = "John"
  AGE        = 21
  WEIGHT     = 17.5
  GROUP      = "ADDRESS"
    STREET     = "Yoyodyne Propulsion Labs,\n
                  123 Johnstown Road,\n
                  Suite 1"
    CITY       = "Grover's Mill"
    STATE      = NJ
  END_GROUP  = "ADDRESS"
END_GROUP = "*root*"
```

And here is the equivalent chunk structure:

```
--
NAME: John
AGE: 21
WEIGHT: 17.5
BEGIN ADDRESS
   STREET: Yoyodyne Propulsion Labs,
         : 123 Johnstown Road,
         : Suite 1
   CITY: Grover's Mill
   STATE: NJ
END ADDRESS
```

Notice the use of ":" in continuation lines. A sequence-string in ODL is represented in chunk-syntax as a string with \001\n sequences throughout, indicating split marks:

```
FISHERMEN: Wynken\001
         : Blynken\001
         : Nod
```

4.5.2 HDF

HDF (Hierarchical Data Format) is a binary format for storing structured information in files. It is used by EOSDIS for browse images.

For reading/writing HDF files from C, we currently use the NCSA-developed "HDF" library. we are forced to use both the HDF4 and the HDF5 libraries, because there are browse files out there encoded in both ways.

4.5.3 ODL

ODL (Object Description Language) is an ASCII-based "parameter=value language" (PVL) used at NASA for the storage and exchange of high-level attributes of satellite data (known as "metadata"). It is used by the IMS for message-passing and configuration files.

Since the system was designed previous to the introduction of XML, ODL was chosen as the language for representing and transmitting structured data. Future efforts will probably turn to XML for this purpose.

Example

Here's a simple ODL tree... an ODL **OBJECT** that might represent a person:

```
OBJECT           = PERSON
  NAME             = "Doe, Jean"
  SMOKER           = N
  AGE_YEARS        = 34
  WEIGHT_LBS       = 145.0
  LAST_UPDATE      = 1997-09-21T12:34:567.8Z
END_OBJECT       = PERSON
```

So far we can see the basic simple ODL types: STRING, SYMBOL, INTEGER, REAL, and DATETIME. Parameter names are on the left.

OBJECTs (and their syntactic equivalent, GROUPs, which together are known as "Aggregate" structures) can be nested:

```
OBJECT           = PERSON
  NAME             = "Doe, Jean"
  SMOKER           = N
  AGE_YEARS        = 34
  WEIGHT_LBS       = 145.0
  LAST_UPDATE      = 1997-09-21T12:34:567.8Z

  GROUP            = CONTACT
    NAME             = "Doe, Pat"
```

```
      RELATION          = "spouse"
      PHONE             = "232-555-3435"
    END_GROUP         = CONTACT

    GROUP             = CONTACT
      NAME              = "Deere, Zhan"
      RELATION          = "mother"
      PHONE             = "312-555-6899"
    END_GROUP         = CONTACT

  END_OBJECT        = PERSON
```

As you can see, the names of sub-OBJECTs and sub-GROUPs (and parameters too, I suppose) need not be distinct.

Every parameter in ODL can technically have *one or more* values; when a parameter has multiple values, they are represented as *sequences*. Here's an example, using a strict line-oriented syntax:

```
GROUP             = CONTACT
  NAME              = "Deere, Zhan"
  RELATION          = "mother"
  PHONE             = (
     "312-555-6899",
     "312-555-9977",
     "310-555-0010")
END_GROUP         = CONTACT
```

Notice the commas, and the opening and closing parens.

Supporting software

For reading/writing ODL in C, we currently use the externally-developed "ODLC" library. It is very buggy, namespace unclean, and produces poorly-formatted (and sometimes erroneous) output... but, it is what is most-widely in use until a substitute can be developed.

Chapter 5

Coding standards/practices

This section discusses the basic coding philosphies and practices used in the implementation of the system. Before you write any code for the system, you should read this.

5.1 Dictionary of entity names/prefixes

The following are the terms that will be used throughout the system to represent various entities. Where possible, you should use them, *as shown*, to name variables, constants, etc. You may modify case, etc. as to your liking.

5.1.1 General terminology

arch

Anything pertaining to an archive.

archid

Archive id. This is the abbreviated name for the DAACs: e.g., “GSFC”, “ORNL”, etc. Archive ids contain no whitespace.

archtag

Archive tag. This is the numeric code for a particular archive: e.g., “6” might signify “GSFC”.

atag

Same as "archtag".

ds

Anything pertaining to a data set.

dsid

Data set id. This is the name for a particular data set: e.g., “AMAZON RIVER BASON PRECIPITATION”.

dskey

Data set key. This is the “abbreviated” indentifier for a particular data set, using numeric codes for the archive and data set.

In C code, this will commonly be a structure; in CGI scripts, the items are written out and separated by slashes (e.g., “6/240”): this is called an “ASCII dskey”.

dstag

Data set tag. This is the numeric code for a particular data set: e.g., “240” might signify “AMAZON RIVER BASON PRECIPITATION” in the context of archive “GSFC”.

dtag

Same as "dstag".

gid

Granule id.

gkey

Granule key. This is the “abbreviated” indentifier for a particular granule, using numeric codes for the archive and data set: in actuality, it is the “dskey” coupled with the “gid”.

In CGI scripts, the items are written out and separated by slashes (e.g., “6/240/1990Tx111222.dat”): this is called an “ASCII gkey”.

mode

Should hold the CGI tag of the same name.

pk (or pkg)

Anything pertaining to a package.

pkid (or pkgid)

Package id. This is the name for a particular package: e.g., “*” or “AMZNRVRBSNPRCPTTN”.

pkkey

Package key. This uniquely identifies a package.

sid

Session id (although in entity logs this is the system id).

submode

Should hold the CGI tag of the same name.

5.1.2 Keys, tags, and ids (general)

Consider the following definitions:

- A **key** is a text string which uniquely identifies an entity (e.g., a data set, or a granule) in the system. For example, the **data set key** `"6/240"` uniquely identifies a particular EOSDIS data set, and the **granule key** `"6/240/THX1138"` uniquely identifies a particular EOSDIS granule (which happens to be in that data set).
- A **tag** is a part of a key. Usually, tags in a key are separated by slashes. Each tag provides a certain piece of information; for example, in the granule key `"6/240/THX1138"`...
  - The `"6"` is the *archive-tag*. It identifies a data center, like GSFC.
  - The `"240"` is the *dataset-tag*. It identifies a single data set at data center `"6"`.
  - The `"THX1138"` is the *granule tag*. It identifies a single granule in data set `"6/240"`.

  Note that two different data sets at two different archives *can have the same dataset-tag*. That is why you need the *data set key* when you want a globally-unique identifier.
- An **id** is a human-readable equivalent of a tag. for example, archive-tag “6” might have archive-id “GSFC”.

See the data dictionary to find out how we name variables which hold keys or tags.

5.1.3 Archive tags, keys, and ids

archive-tag

This is an integer, currently 0-255. It identifies an archive (data center) like GSFC, MSFC, etc. Here are some examples:

```
6
42
150
```

The number itself is taken from the IK library’s `IK_ArchiveTag` type, by treating the one-byte identifier as an unsigned integer.

archive-key

Since an archive-tag uniquely identifies an archive in the system, an archive-key is identical to an archive-tag.

archive-id

This is a short abbreviation of the data center name. Here are some examples:

```
GSFC
MSFC
NSIDC
```

5.1.4 Data set tags, keys, and ids

dataset-tag

This is an integer, currently 0-65535. It identifies a single data set with respect to a given archive. Here are some examples:

```
6
42
150
65234
```

The number itself is taken from the IK library's `IK_DatasetTag` type, by treating the two-byte identifier as an unsigned integer.

data set key

This uniquely identifes a data set in the system. It is composed like this:

```
archive-tag "/" dataset-tag
```

`Here are some examples:`

```
6/240
1/3
0/0
244/56512
```

Note: Note that two different data sets at two different archives can have the same dataset-tag. That is why you need the *data set key* when you want a globally-unique identifier.

dataset-id

This is the data set name, which in many cases is unique across all data centers, but in some cases (where granules from the same data set are stored at multiple DAACs) is not. Here are some examples of data set ids:

```
1995 MONTHLY PRECIPITATION MEANS
GLOBAL AVERAGE TEMPERATURES
```

5.1.5 Granule tags, keys, and ids

granule-tag

This is an arbitrary text string (probably without whitespace.) It identifies a single granule with respect to a given data set. Here are some examples:

```
950912
PZ00346.123+A
ERM000234-11111/aer4
```

The granule-tag is equivalent to the granule-id in the IK library's `IK_Granule` structure.

granule-id

This is identical to the granule-tag, and the term "granule-id" is used in preference.

granule-key

This uniquely identifes a granule in the system. It is composed like this:

```
archive-tag "/" dataset-tag "/" granule-id
```

`Here are some examples:`

```
6/240/950912
1/3/PZ00346.123+A
0/0/XY/Z
244/56512/ERM000234-11111/aer4
```

Note: Note that two different granules in two different data sets can have the same granule-id. That is why you need the *granule-key* when you want a globally-unique identifier.

5.1.6 Package tags, keys, and ids

package-tag

As with granules, a **package-tag** is identical to the **package-id**, and hence, we will simply use the term package-id.

package-id

This is an arbitrary text string (probably without whitespace) used to identify a packaging option of a given granule. They are taken directly from the `package_id` array in the `IK_Granule` structure. There are two kinds of package-id:

- **If the package-tag is "*",** that indicates that a package consisting of the single granule is available. In this case, both the dataset-package-key and the unique-package-key are of the form...

  ```
  archive-tag "/" dataset-tag "/*/" granule-id
  ```

- **If the granule package id is any other string,** that indicates that a package of that same name (which is unique to the granule's data center) contains the granule. In this case, the **dataset-package-key** is of the form...

  ```
  archive-tag "/" dataset-tag "/" package-tag
  ```

  And the **unique-package-key** (a.k.a the merged-package-key) is of the form...

  ```
  archive-tag "/M/" package-tag
  ```

  which, as you can see, is just the result of replacing the dataset-tag with an `"M"` (thus "merging" package keys which differ only in the dataset-tag).

psuedo-package-key

Finally, there is the notion of a **psuedo-package-key**, which is used internally by the software when the user chooses a package for a granule. You don't really need to worry about it much: it is simply a **dataset-package-key**, but with the optional case of:

```
archive-tag "/" dataset-tag "/*"
```

...which represents the set of all single-granule packages in the given data set. This pseudo-package key provides a way of saying "all single-granule packages in this data set."

5.2 Namespaces

A namespace, in general coding parlance, is a subset of all possible names (for variables, functions, etc.) which has been assigned to a particular module in an application, so that names in that module will not conflict with names in other modules. E.g.: in the GDBM library, all functions begin with `"gdbm_"`.

To avoid collisions (and confusion) between the names of Web Gateway files, functions, environment variables, etc., and the names of non-Web-Gateway entitites, the Web Gateway project defines namespace conventions for its different implementation languages.

5.2.1 For C code

All Web Gateway-specific development work in C is to use the `"IW_"` namespace (meant to indicate IMS/WWW, the original name of the system), in the same manner used by the existing IMS code. E.g., here are possible entries for an abstract data type, `IW_Thing`:

```
typedef struct IW_Thing {             /* the type (a structure) */
    int this;
    int that;
} IW_Thing;

IW_Thing *IW_ThingNew(void);          /* constructor */
int       IW_ThingAttr(void);         /* attribute access */
void      IW_ThingSetAttr(int);       /* attribute setting */
void      IW_ThingHaveSomeFun(void);  /* you get the idea */
void      IW_ThingFree(IW_Thing *);   /* destructor */

#define IW_ThingCONSTANT 0            /* a constant for this package */

extern int IW_ThingGlobalVar;         /* a global variable */
```

N.B.: Jhon and I differ stylistically on function naming: I prefer to use a rigid namespace for `IW_ThingAction`; he likes the form `IW_ActionThing`. Use either one, but be consistent.

It is recommended that you follow these conventions even for entities with static linkage; it will make your code clearer, and make namespace conflicts even less likely.

5.2.2 For Perl code

All Web Gateway-specific development work in Perl is to use the `"IU"` or `"IW"` "MixedCase" namespaces for packages.

There are different naming conventions for different kinds of entities in the IU/IW libraries:

- *Local variables* should begin with a lowercase letter.
- *Global variables* are mixed-case with a leading capital.
- *Global 'constants'* are all uppercase.
- *Class methods* are lowercase with underscores.
- *Instance methods* are lowercase with underscores.
- *Non-method subroutines* may be anything.

Module-private entities (constants, globals, methods, subroutines) may signify their privateness by a single leading underscore, but this is completely optional.

As an example, here are possible subroutines for a package implementing data type, IW::Thing:

```
IW::Thing::do_stuff()                 # package utility

IW::Thing->new(This=>1, That=>2)      # class method

$thing->attr()                        # attribute access
$thing->attr($newvalue)               # attribute setting
$thing->have_some_fun()               # etc.

$IW::Thing::CONSTANT          # a constant for this package

$IW::Thing::PublicGlobalVar   # a "public" global variable

$IW::Thing::_PrivateGlobalVar # "private" globals could have uglier names,
                              # indicating ``hands off" to outsiders.
```

Whether you use arrays or hashes as your method parameters is entirely up to you, on a case-by-case basis. However, if you use hashes, your tags should be mixed-case with a leading capital.

5.2.3 CGI tags

CGI tags will be all lowercase, digits, and hyphens only.

Since we are not linking the Web Gateway system with external modules that attempt to intercept and handle the CGI tags, there is no need for an "iw" namespace for the tags.

To facilitate the construction of simple hyperlinks between pages, it is important to limit the length of URLs; thus, "common" tags like "sid" are kept as short as possible.

5.2.4 Environment variables

All Web Gateway-specific environment variables are to be prefixed with `IW_`. They will be in the uppercase-and-underscores style:

```
IW_SESSION_DIR = ...
IW_SESSION_ID = ...
IW_TOP_DIR = ...
```

5.3 Coding conventions

This is a large project, with many developers. It is *vital* that the coding practices be consistent if the system is to be maintainable.

5.3.1 General

Your best bet is to look at some of the more-stable pieces of code in the system, and follow the conventions therein. That means:

- **Module-level documentation.** Put a POD header in every Perl module you write. A person should be able to get a sense of what the module does/provides at a glance.
- **Commenting your code.** Yes, you have to. The rest of us are not mind readers, and you may not be around a year from now when your code needs changing. Nearly every `if` and `while` block should be commented; don't go 30 lines with nary a comment. Be sure to put comments about your subroutines/functions as well, showing arguments and return value where not entirely obvious.
- **Proper indentation practice.** Two to four space indentation per level, no "cuddling" of "else"s, please. Make sure that all code at the same nesting level lines up perfectly.
- **Whitespace.** Whitespace is your friend. Separate blocks of code in long subroutines with blank lines, putting a comment above each block.

> ***Note:*** If you don't know Emacs, then you're making life harder on yourself and the rest of us. Emacs has modes for editting Perl and C source which will automatically indent your code and check the balancing of parentheses. *Plus,* when running under X, it can be made to shown keywords, comments, string-literals, etc. all in different colors. Highly recommended!

5.3.2 For C code

If implementing under the `"IW_"` namespace in C, all source code should follow the following form (as much as possible) for a package/abstract data type `IW_Thing`:

IW_Thing.h

Defined constants, typedefs, function prototypes, external declarations.

IW_Thing.c

Function/global variable definitions.

5.3.3 For Perl code

Modules

All modules which expect to be run by the WWW gateway under a CGI script should go in the **IW::** (IMS Web) library.

More-generic modules should go in the **IU::** (IMS Universal) library.

The number of modules in the system and its complexity sometimes required multi-level nesting. For example, IW::Foo::Bar is implemented as module `IW/Foo/Bar.pm`.

All Perl source code should follow the following form (as much as possible) for a package/abstract data type; e.g., IW::Thing:

A Perl module `IW::Thing` should reside in `IW/Thing.pm`, and should have the following general form. Notice in particular the banner around the name in the subroutine header, and the use of POD markup.

```
 package IW::Thing;

 =head1 NAME

 IW::Thing -- brief description

 =head1 DESCRIPTION

 Detailed description goes here.
 You can also have other sections: WARNING, NOTES, etc.

 =head1 AUTHOR

 Principle author(s).

 =head1 RCS INFO

$Id: coding-conventions.ipod,v 1.2 2002/11/25 20:20:08 joe Exp $

=cut

use strict;
use vars qw(@ISA);      # as appropriate

use IU...;              # IU modules

use IW...;              # IW modules

#------------------------------
#
# Globals
#
#------------------------------
```

```
# Any globals you have...
my $Global;

#------------------------------------------------------------
# new NAME, AGE
#------------------------------------------------------------
# I<Class method, constructor.>
# Create and return a new IW::Thing.
# NAME defaults to "anonymous".  AGE defaults to 0.
#
sub new {
   ...
}

#------------------------------------------------------------
1;
```

Executables

You can use the .pl extension on Perl executables if you wish (.cgi if the executable is a CGI script).

Chapter 6

Installation

6.1 Installation requirements

You will need the following to obtain, install, and run the EOS Data Gateway:

- A Unix platform running: **Solaris 2.4/2.5**, **IRIX 5**, **IRIX 6**, or **Linux**. The versions noted are *minimum* levels; the EDG is running successfully on IRIX 6.4 and IRIX 6.5, for example. No support for VMS, MacOS, or MSWindows is planned.
- At least 0.5 to 1 GB of disk space should be enough for the Web Gateway software, session directories, cached browse images, etc. More is strongly recommended.
- Some form of the **tar** utility.
- **GNU's "gzip"**.
- **GNU's "make"**, also known as "gmake".
- An **ANSI C compiler**.
- A full set of **ANSI- and POSIX-compliant C libraries.** If you've got the ANSI compiler, you probably have the libraries, but we just thought we'd list this explicitly.
- The **HDF C library,** version 3.3. or 4.1 (version 4.0 has a bug which affects the display of our integrated browse products so we don't recommend installing 4.0). This is for converting HDF browse products to GIFs for integrated browse.

  You should build all the libraries, and create the link `"/usr/local/src/HDF"` to wherever the top-level directory (e.g., `HDF3.3R1`) resides.

- The **HDF5 C library,** version 5-1.4.4, needed for converting HDF5 browse products to GIFs for integrated browse.

  You should build all the libraries, and create the link `"/usr/local/src/HDF5"` to wherever the top-level directory (e.g., `HDF5-1.4.4`) resides.

- An **HTTPD** that supports CGI scripts. We recommend **Apache 1.3.9** or better.

> ***Warning:*** Note that we do not require an SSL-capable HTTPD at present; secure services have been put on the back burner until some later date.

- **Perl 5.6.1** is the minimum required Perl version level.

  The **GDBM_File** module must either be compiled in or (preferably) available on autoload.

  The EDG has been tested with Perl 5.6.1.

- **FreeWAIS 0.5**. Newer versions of FreeWAIS will not work. This is for performing the guide searches.

- A real, unadulterated installation of **GDBM 1.5** or better. This is for linking with Perl, to use `GDBM_File`. **Do not use the custom IMS version of GDBM for the Perl module!** It will almost certainly give you major headaches, and not work.

- The `Expat` XML parser, to be used in conjunction with the Perl `XML::Parser` module (see below). We have successfully built and used `expat-1.95.1` on IRIX 6.5; you may have to find an `Expat` suitable for your system.

- The `XML::Parser` Perl module, which *must* be installed into your Perl by your system administrator. We have found that attempting to build XML::Parser outside of the system Perl libraries to be difficult and unrewarding (it's a pain, and it often doesn't work, for no discernable reason).

  We have been successful *only* with `XML::Parser 2.30`; we *do not* recommend any other version.

- The `Storable` Perl module, which *must* be installed into your Perl by your system administrator. `Storable` is another module which calls the Perl internals directly, and we have found that it works best when built into the system Perl libraries. It sometimes refuses to work at all when built outside, for no easily diagnosable reason.

6.2 Obtaining it

Obtaining a copy of the EOS Data Gateway system requires an agreement with NASA. If you would like to get a copy of the EOS Data Gateway please contact our NASA Task Representative.

6.3 Preflight checklist

If you are installing the distribution for the first time, or if you are installing a new major version (e.g., upgrading from 1.x to 2.x), you should go through this next section carefully.

6.3.1 Step 1: Make sure you have Perl.

Make sure you have installed **Perl 5.6.1**. The `GDBM_File` module must work, preferably on autoload. The recommended path is either `/usr/bin/perl` or `/usr/local/bin/perl`.

6.3.2 Step 2: Make sure `XML::Parser` is installed in your Perl.

Make sure that your system administrator has installed `XML::Parser` for you. Execute the following command in the shell to verify that it is installed:

```
/path/to/your/perl -MXML:Parser -e 'print "OK\n"'
```

The command should print "OK". If it says:

```
Can't locate XML/Parser.pm in @INC (@INC contains: ...
```

and then lists a bunch of paths, `XML::Parser` is not installed. If it says anything else, talk to your system administrator.

6.3.3 Step 3: Make sure `Storable` is installed in your Perl.

Make sure that your system administrator has installed `Storable` for you. Execute the following command in the shell to verify that it is installed:

```
/path/to/your/perl -MStorable -e 'print "OK\n"'
```

The command should print "OK". If it says:

```
Can't locate Storable.pm in @INC (@INC contains: ...
```

and then lists a bunch of paths, `Storable` is not installed. If it says anything else, talk to your system administrator.

6.3.4 Step 4: Create your `wwwuser` account and groups.

Create an account called `"wwwuser"` for the HTTPD. You will run the HTTPD under this account.

Create a new group, also called `"wwwuser"`. Both `"wwwuser"` and `"imswww"` should be members of this group. This group allows you to avoid giving "write-anybody" permissions to some of the EDG files.

If you prefer to use another account (e.g., "nobody") instead of `"wwwuser"`, be sure that you add this user to the `"wwwuser"` group.

> ***Warning:*** Remember, if you choose not to set up the `"wwwuser"` *group*, you will have to have some files with "write-anybody" permissions. You must judge which you feel is a greater security risk; we recommend creating the group.

6.3.5 Step 5: Set up your `httpd.conf`.

Making sure that your HTTPD can access the EOS Data Gateway documents and scripts is one of the more complicated parts of the installation procedure, since it's very site-dependent.

Fortunately, you usually only have to do it once, no matter how many times you upgrade. The following steps describe what you're trying to achieve, and give specific examples of how that can be done.

1. Set up the ports on which you want the server to listen.

    ```
    Listen 80
    ```

    > ***Note:*** If you are running Apache as `wwwuser`, you will not be able to use port numbers < 1024. We recommend 8080 in this case; you may choose any other that you desire. Remember that this port number will have to be added to the `PrimaryPort` directive in the EDG `SETUP` file. The `SecurePort` directive is ignored by the EDG, so ou can ignore it too.

2. Set up the HTTPD to run CGI scripts as user `"wwwuser"` This should be a non-privileged user, like `"nobody"`.

    ```
    User wwwuser
    Group wwwuser
    ```

    > ***Warning:*** Your HTTPD should run CGI scripts as a *non-privileged* user. If your installation runs CGI scripts as `root`, you have a security hole a mile wide at your site. *Don't do it!*

3. Set things up to make the paths to the EDG as simple as possible.

```
Redirect /imswelcome http://<your_host_name_here>/ims-bin/pub/imswelcome
```

4. Enable the use of user `public_html` directories, and set them up so that `.htaccess` files can specify password files, and so symbolic links will be followed.

   In addition, block any protocol which could overwrite things in the `public_html` directory.

```
UserDir public_html
<Directory /*/public_html>
    AllowOverride FileInfo AuthConfig Limit Options
    Options MultiViews Indexes FollowSymLinks IncludesNoExec ExecCGI
    <Limit GET POST OPTIONS PROPFIND>
        Order allow,deny
        Allow from all
    </Limit>
    <Limit PUT DELETE PATCH PROPPATCH MKCOL COPY MOVE LOCK UNLOCK>
        Order deny,allow
        Deny from all
    </Limit>
</Directory>
```

5. Set up the necessary `ScriptAlias` directives to allow the EDG to execute its CGI scripts.

```
ScriptAlias /cgi-bin/ /<path_to_your_httpd_here>/cgi-bin/
ScriptAlias /ims-bin/ /<path_to_imswww_home_dir_here>/cgi-bin/
```

6. Set up needed `Environment Variable` to be used with the HDF5 libraries. The path needs to be put into the httpd.conf file. To be on the safe side place the following directive below the AddModule section. mod_env.so must also be loaded.

   Here is the apache directive.

```
SetEnv  LD_LIBRARY_PATH "/usr/lib:/usr/local/lib:/<path_to_hdf5_libraries_here>/lib"
```

Your `httpd.conf` is now ready. To test this configuration out, issue the following commands as `'wwwuser'`:

```
/path/to/apache/bin/apachectl stop
/path/to/apache/bin/apachectl start
```

6.3.6 Step 6: Make sure you have the supporting programs/libraries

Certain software packages are needed to compile *and/or* link *and/or* run the EOS Data Gateway. However, to reduce the size of the distribution package (and because you may already have them installed on your system), they are not included with EOS Data Gateway. Please review the **Installation requirements** if you haven't already done so.

6.3.7 Step 7: Make sure you have a user account "imswww".

It doesn't matter what group this account goes in, although putting it in the same group as your HTTPD `wwwuser` is not a bad idea. It will make setting permissions on directories easier and safer.

For the sake of argument, let's say that `~imswww` is `/home/imswww`.

If you're curious, you might want to read the section (in the design document) entitled "**Why require a ~imswww account?**"

6.4 Installing it

If you are reading this, you have probably uncompressed and de-tarred the distribution. **Don't do anything further until you read these directions.**

6.4.1 Step 1: Go through the pre-flight checklist.

Please go through the **preflight checklist** carefully.

In particular, you *must* build your HTTP server in accordance with the instructions. You should review the pre-flight checklist when installing or reinstalling; the requirements may have changed since your last install.

6.4.2 Step 2: Unpack distribution in ~imswww.

As user `imswww`, move the distribution `.tar.gz` (or `.tgz`) file to the `~imswww` directory, uncompress, and untar it. Like so:

```
gzip -d distfile.tar.gz  or
gunzip distfile.tgz

tar -xvf distfile.tar
```

Warning: All files in the distribution should be owned by `~imswww/`. If you are not logged in as `~imswww`, do so now, and bring the files over from wherever they currently are.

You should now have a directory structure like this:

```
~imswww/
   |
   |--- imswww-VERSION/
   :       |
           |--- bin/ ...
           |--- cgi-bin/ ...
           |--- docs/ ...
           |--- src/ ...
           |--- lib/ ...
           :
```

If this is not the first time you've done this, you may have directories for other versions of EOS Data Gateway at the top level.

6.4.3 Step 3: Unpack the support files in ~imswww

You need to set the support files if you don't have them from a previous installation.

As user `imswww`, move the support_files `.tar.gz` (or `.tgz`) file to the `~imswww` directory, uncompress, and untar it.

Warning: All files in the support_files directory should be owned by `~imswww/`. If you are not logged in as `~imswww`, do so now, and bring the files over from wherever they currently are.

You should now have a directory structure like this:

```
~imswww/
   |
   |--- support_files/
   :       |
           |--- sup_files/ ...
           |--- dist_files/ ...
           :
```

All EOS V0 clients, including the EOS Data Gateway, depend on a directory tree of special supporting files. These files come in two sets. The first set is sometimes referred to as the `"distribution files"` found in support_file/dist_files/ is a static set of files. The second set of files are sometimes called the `"support files"` and sometimes called the `"valids"` and are found in support_files/sup_files.

Be sure that imswww has read and write access to this directory and also to the parent directory of support_files.

6.4.4 Step 4: “Configure” the system.

You are now ready to give EOS Data Gateway the information it needs to *compile and link* the gateway programs.

Note: If you are building on a Solaris machine you may need to include "/usr/ccs/bin" in your path.

If you have previous version of the web gateway installed on your system you may want to look at the **myconfig** script used to build that version.

Edit the script called `"myconfig"`, which should look like this:

```
#!/bin/sh
# myconfig -- invoke configure script

./configure \
    --with-debug \
    --verbose \
    --with-perl=/usr/local/bin/perl5.6.1 \
```

As a first cut supply just the Perl path.

Change to the top-level directory of the distribution and invoke the `configure` utility, like so:

```
cd ~/imswww-VERSION
```

Now run the myconfig script:

```
 ./myconfig
```

This may take some time, as the `configure` script will try and determine certain characteristics of your system: what C compiler to use, what functions are available, etc. Once it is done, it will go through your distribution and generate certain files needed for the build (primarily Makefiles) from the corresponding `*.in` (configure INput) files.

If at any point you think that `configure` has made an undesirable choice, you should interrupt it and restart it, supplying the correct values in the `"myconfig"` script. Type:

```
configure --help
```

and look for the `-with-`*`package`* options at the end of the listing.

Warning: If you are re-running `configure` because it guessed wrong, make sure to delete the `"config.cache"` file... otherwise, it may use the same cached values.

6.4.5 Step 5: "Make" the system.

From the top-level directory of the distribution, invoke your GNU `make` program, like so (for clarity, we will refer to it as `"make"`, but you should use whatever command is appropriate, like `"gmake"`):

```
cd ~/imswww-VERSION
make install >& make_install.log &
```

This will compile the system, copy the binaries over to their appropriate locations, create necessary symbolic links, etc.

When the make install is done, carefully read through the make_install.log file looking for `Error` messages. Numerous `Warnings` are common and can usually be ignored, if an error occurred then the problem must be fixed and make install rerun.

Note: There will be a warning during the installation of libwww-perl which implies that an FTP module was needed but not found. Since the EDG doesn't access any `ftp://...` links internally, it doesn't need this module and this isn't a problem

6.4.6 Step 6: Create symbolic links.

The following links make the installation visible to the web via the ~imswww/public_html directory. You need to verify that user public html directories on your system are called public_html before proceeding with this step. If the user public html are called something different then use that name instead of public_html.

In the home directory of `"imswww"` create the following directory structure for the HTTPD to access if they don't already exist from a previous installation.

```
cd ~imswww

mkdir public_html

cd ~imswww/public_html
```

```
mkdir cgi-bin

cd  ~imswww/public_html/cgi-bin
```

Create links for this installation:

```
cd ~imswww

ln -s imswww-VERSION VERSION
(For example: ln -s imswww-1.5 1.5)

cd ~imswww/public_html:

ln -s ~imswww/imswww-VERSION/docs VERSION
(For example:  ln -s ~imswww/imswww-1.5/docs 1.5 )

cd ~imswww/public_html/cgi-bin:

ln -s ~imswww/imswww-VERSION/cgi-bin VERSION
(For example: ln -s ~imswww/imswww-1.5/cgi-bin 1.5)
```

6.4.7 Step 7: Generate and apply the `SETUP` file.

You are now ready to give EOS Data Gateway the information it needs to *install and run* the gateway. From the top-level directory of the distribution, check the `setup` utility. **Edit this file and make sure that the Perl path on the second line is correct for your site.** Then, invoke the `setup` utility, like so:

```
./setup -make
```

This will create a file called `SETUP` with default values. **Edit this file to make certain it contains the correct information for your site.**

Required:

At a minimum you need to change the following lines to the appropriate values for your site (full descriptions of all the values are described elsewhere in this document):

AdminCrypt

The crypted password for the Gateway administrators to use, exactly as you'd see it in an /etc/passwd file. This password will be provided by the EOS Data Gateway administrators. *For example:*

```
AdminCrypt      "encrypted-pw-provided"
```

Warning: The special value "NO PASSWORD" allows ANYONE in!!! PLEASE, DO NOT USE AN ACTUAL SYSTEM PASSWORD FOR THIS!

CgiUrl

Set this to "/ims-bin/VERSION/". *For example:*

```
CgiUrl         "/ims-bin/1.5/"
```

DocsUrl

Set this to "/~imswww/VERSION/". *For example:*

```
DocsUrl        "/~imswww/1.5/"
```

EcsProfileSmc

Set this to "SMC-UPG". *For example:*

```
EcsProfileSmc   "SMC-UPG"
```

GaeaDataDir

Set this to the full path to your valids. This MUST match the directory you created in Step 3. *For example:*

```
GaeaDataDir "/home/imswww/support_files/sup_files"
```

GaeaDistribDir

Set this to the full path to your distribution support files. This MUST match the directory you created in Step 3.

```
GaeaDistribDir "/home/imswww/support_files/dist_files"
```

MorePaths

Be sure it includes the location of your `wais` software. *For example:*

```
MorePaths   "/usr/bsd:/usr/local/bin:/usr/local/wais/bin"
```

PerlHttpdPort

Choose a port on which to run the Perl httpd. is recommended that you run the Perl httpd. Running the Perl httpd will significately speed up the performance of the system, by eliminating the need to reload the Perl libraries for each CGI process. At the current time, we do not recommend running with PerlHttpdPort set, because we have been having some problems with it.

```
PerlHttpdPort "8123"
```

We recommend setting it to:

```
PerlHttpdPort ""
```

Warning: Port numbers are very site specific. Check with your system administrator to get an appropriate port number.

HomeEmail

Set the e-mail address to where comments are sent. This should be set to the Data Center User services group where the machine is hosted.

For example:

```
HomeEmail  "edc@eos.nasa.gov"
```

HttpdHost

This is the hostname that will be used in URLs pointing back into the system, and also for email sent to/from this site; leave it empty ("") for the default hostname.

For example:

```
HttpdHost "harp.gsfc.nasa.gov"
```

MaxGransDefault

This is the number of granules returned per dataset when a user issues a query.

For example:

```
MaxGransDefault  "100"
```

UseNonSpatial

This is a switch **(Y or N)** used if this installation supports the special ODL flag for including non-spatial granules in the results. If set to **Y**, a checkbox option will be displayed on the search form within the Choose Additional Options, ALL Options section. The default value is empty string or N.

For example:

```
UseNonSpatial  "Y"
```

UseNonTemporal

This is a switch **(Y or N)** used if this installation supports the special ODL flag for including non-temporal granules in the results. If set to **Y**, a checkbox option will be displayed on the search form within the Choose Additional Options, ALL Options section. The default value is empty string or N.

For example:

```
UseNonTemporal "Y"
```

SiteAdmin

Set the e-mail address to where system error messages (`"dying gasps"`) and other information is sent.

For example: SiteAdmin "imswww"

SiteName

Set the name you wish to use for your installation.

For example:

```
SiteName  "Goddard Space Flight Center"
```

(Optional) Site Administration :

DisableByDiscipline

For special case installations of the EDG, non-standard valids may not have all the information needed to drive the *By Discipline* feature on the search form. For those installations you can disable the *By Discipline* feature by setting this flag to "YES". Leave blank to enable *By Discipline*.

For example:

```
DisableByDiscipline "YES"
```

SiteSpecDir

Specify the directory you will use to hold site specific info and documents for the gateway, such as welcome_header.html.

For example: SiteSpecDir "/usr/people/imswww/site_specific"

PolicySecurity

Normally set to "US Government" - this will cause a link to be created on the welcome page to the US Government security policy. This is required for all US installations.

However installations at non-US sites may want to substitute their own security policies. In which case, set this to "local". If you don’t want any security policy link on the welcome page then set it to blank "".

PolicyPrivacy

Normally set to "US Government" - this will cause a link to be created on the welcome page to the US Government privacy policy. This is required for all US installations.

However installations at non-US sites may want to substitute their own privacy policies. In which case, set this to "local". If you don't want any privacy policy link on the welcome page then set it to blank "".

XML Guide Search (Spidex) Configuration

You will need to check the Guide Configuration Page, located at

This page specifies the values you'll need to enter for `SpidexUrl` and `SpidexConf`.

(Optional) Persistent User Accounts:

Beginning with release 2.0, the EOS Data Gateway can be set up so that user accounts can be established. These accounts permit a user's session to be active for an extended period of time.

To enable User Accounts (Persistent User Sessions):

HavePersistentSessions

Allow the users to have Persistent directories on the server. The default is "NO", change it to "YES" to enable this option.

PersistentDir

Base location of the Persistent directories , for instance :

```
"/imsdev/persistentUser/"
```

Please don't have this directory "tied" to a particular version of the Gateway. We recommand the directory reside under ~imswww/.

In most cases, the Web Gateway's defaults can be used for the following :

AccountMax

How much disk space should be allowed for each user account. This maximum limit should be enough to handle the results of a large search. If a user exceeds this size and they are not signed-in, their results will be purged. It they are signed-in they will be warned. Specify a size with units of Mega Bytes: e.g., "20mb". Recommend: "20mb".

For example: AccountMax "20mb"

AllAccountsMax

How much disk space should be allowed for ALL user accounts. This maximum limit should be quite large, since each account can be typically 1-5 MB (although could be smaller or bigger). Specify a size with units of Mega Bytes or Giga Bytes: e.g., "500mb" or "2gb". Recommend: "1gb".

For example: AllAccountsMax "1gb"

CriticalAllAccountsMax

How much disk space should be allowed for ALL user accounts before more drastic measures are invoked to clear space. This maximum limit should be quite large, since each account can be typically 1-5 MB (although could be smaller or bigger). It should generally be larger than AllAccountsMax. Specify a size with units of Mega Bytes or Giga Bytes: e.g., "500mb" or "2gb". Recommend: "2.5gb".

For example: CriticalAllAccountsMax "10gb"

Warning: Again, accounts sizes are very site specific. The size of the accounts depends on capabilities of the system as well as the usage of the EOS Data Gateway.

GuideKillTime

Under CRITICAL disk space conditions (account usage exceedingCriticalAllAccountsMax), if guide search results are not accessed for this period of time, then they will be deleted - even from accounts where the user is signed in. Specify a time, optional spaces, and units: e.g. "5m" for 5 minutes. Recommend: "15 m".

For example: GuideKillTime "15m"

LogoutTime

How long should a logged in account be inactive before we log the user out? Used by imscleanaccounts.pl. Specify integer time, optional blanks, and units. Recommend: "1 d" for one day.

For example: LogoutTime "1 d"

ResultsLifetime

How long can results be kept in a user account, that is not accessed, before they are deleted (to free up disk space and keep info from becoming out-of-date)? Used by imscleanaccounts.pl. Specify integer time, optional blanks, and units: e.g., "10d" for 10 days. Recommend: "14 d" for fourteen days, the valids cycle.

For example: ResultsLifetime "14 d"

SecuredDir

Location of the secured subdirectory of cgi-bin (for persistent logins). Defaults to "secured".

For example: SecuredDir "secured"

(Optional) Detailed Map:

If you want to enable the more detailed map for the map selection applet, you must follow the following directions.

MapCgiScript

This is the map server script used by the Geo selection java map. By default this is set to "imswdbtile.cgi" which is a low resolution map with maximum zoom of 16x (state level) and has the smallest map database.

For Example:

```
MapCgiScript "imswdbtile.cgi"
```

Set to "drawglobe.cgi" for higher resolution maps and more features.

For Example:

```
MapCgiScript "drawglobe.cgi"
```

For this option you have a choice of installing one of three resolution levels. Pick the resolution level by setting the MapResolution.

Possible values for MapResolution are:

MEDIUM

Uses the lowres and medres databases (25MB total) and allows zooming to 64x (cities and major roads).

HIGH

Uses the lowres, medres and highres databases (237MB total) and allows zooming to 500x (towns and secondary roads).

ULTRA

Uses the lowres, medres, highres, and ultrares databases (1331MB total) and allows zooming to 1000x (neighborhoods and residential roads).

Here are the directions for installing the map data bases:

1. Read the SETUP file and find the location of the support_files directory as indicated by the setup variable GaeaDistribDir.
2. Switch to the directory of GaeaDistribDir.

   For example: cd ~imswww/support_files/dist_files

3. ftp lyta.gsfc.nasa.gov
   Name: anonymous
   Password: your-email-address
4. cd outgoing/edg
5. For all resolutions:
   get map_symbols.tar.gz
   get map.tar.gz
6. For HIGH and ULTRA resolution maps:
   get highres.tar.gz
7. For ULTRA resolution maps:
   get ultrares.tar.gz
8. quit
9. Use "gunzip" or "gzip -d" to uncompress the files:
   gunzip map.tar.gz
   gunzip map_symbols.tar.gz
   gunzip highres.tar.gz
   gunzip ultrares.tar.gz
10. Untar the files:
    tar xvf map.tar
    tar xvf map_symbols.tar
    Untar highres and ultra database, if needed, in the map directory:
    cd map
    tar xvf ../highres.tar
    tar xvf ../ultrares.tar

Apply the SETUP file:

When you are done editing the SETUP file, save it. Then, *apply* the contents of SETUP by running:

```
./setup -apply
```

6.4.8 Step 8: Transfer the valids.

Updates to the valids are provided by the EOS Data Gateway Team every two weeks. You obtain the latest set of valids by executing the `"run_autoxfer.pm"` script supplied with the web gateway. **Run it now to get the valids:**

```
~imswww/imswww-VERSION/lib/perl/run_autoxfer.pm  force
```

In the step 11 you will set this script up to run each night in a cron job, to automatically keep your valids up to date.

The command line argument 'force' is new with version 2.0 of the EOS Data Gateway. It should be used **ONLY** during the installation run and **NOT** in the cron job.

6.4.9 Step 9: Make valids file for e-mail ordering.

The wwwvalids.conf file is used by the system for e-mail ordering. Which data centers and data sets available of e-mail ordering along with the e-mail address to which send orders are stored here. The editable text version must be converted into a gdb file for use by the system. The gdb version of the file must be recreated on each system as gdb files have platform dependences in them.

```
cd ~imswww/imswww-VERSION/lib/conf
cp ~imswww/support_files/sup_files/wwwvalids.conf ./

../../bin/mkvalids
```

If wwwuser is set up as a group that is shared with imswww you should do the following:

```
chgrp wwwuser wwwvalids.gdb
chgrp wwwuser wwwvalids.gdb..LCK
chmod 664 wwwvalids.gdb
chmod 664 wwwvalids.gdb..LCK
chgrp wwwuser misrData.gdb
chgrp wwwuser misrData.gdb..LCK
chmod 664 misrData.gdb
chmod 664 misrData.gdb..LCK
cd ../..
chgrp wwwuser log
chmod 775 log
chmod 755 src/sf_client/autoxfer
cd tmp
chgrp wwwuser imswww-usr
chmod 775 imswww-usr
```

If wwwuser is not set up this way you must:

```
chmod 666 lib/conf/wwwvalids.gdb
chmod 666 lib/conf/wwwvalids.gdb..LCK
```

6.4.10 Step 10 (Optional): Make site specific directory and files.

To tailor the gateway welcome screen to your site, we have provided a mechanism for you to do this.

First, create a site specific directory if you don't already have one. It must have the same name specified in SiteSpecDir in the SETUP file from Step 7 above.

```
mkdir ~imswww/site_specific
```

You can now create a file in this directory for any text, links, graphics, or html compatible information that you want to appear at the top of the Welcome page. The file **MUST** be called:

```
welcome_header.html
```

You can now create a file in this directory for any text, links, graphics, or html compatible information that you want to appear at the bottom of the Welcome page. The file **MUST** be called:

```
welcome_footer.html
```

If you set one or both of PolicySecurity or PolicyPrivacy to "local" in your SETUP file, then you need to create a local security policy file and/or local privacy policy file in your site specific directory.

For PolicyPrivacy = "local" the file **Must** be called:

```
input_privacy.html
```

For PolicySecurity = "local" the file **Must** be called:

```
input_security.html
```

6.4.11 Step 11: Set up crontab for imsstats, imscleanup, autoxfer, makewelcome, and (optionally) imscleanaccounts.

The wwwuser and imswww accounts should have cron jobs running to handle periodic execution of some support scripts. Check your unix documentation for the proper use of crontab.

Crontab for wwwuser

All actions (events) are logged by the EOS Data Gateway software. We would appreciate if you could send us a copy of your EOS Data Gateway logs. NASA is very interested in knowing how many people are using the system and what features are being used most often. The information in the logs is important to us in helping us build better future EOS Data Gateways. Simply add an entry for the program **imsstats** to the **crontab** for the ~wwwuser account to be run once a day.

The EOS Data Gateway has an **imscleanup** script which should be run periodically (about every 15 minutes) to clean up cache files and stale user sessions. The most reliable way to invoke it, *by far*, is as a `cron` job.

(Optional) If your site is set up to have persistent user accounts, then a separate cleanup script, **imscleanaccounts.pl**, should be run periodically, about once per hour. Again, it is easiest to do this in a `cron` job.

To setup the three scripts to run periodically, login as user `"wwwuser"` and add the **imsstats** and **imscleanup** routines to the crontab. For example:

```
# Once a day mail EOS Data Gateway logs to the imswww-stat account at GSFC:
0 1 * * * /usr/people/imswww/pub/bin/imsstats -cron >/dev/null

# Cleanup the EOS Data Gateway's session files -- 2.0 area:
0,15,30,45 * * * * /home/imswww/2.0/lib/bin/imscleanup -cron >/dev/null
# Cleanup the EOS Data Gateway's session files -- 2.1 area:
1,16,31,46 * * * * /home/imswww/2.1/lib/bin/imscleanup -cron >/dev/null
```

Please note that you need to have an entry in the crontab to run **imscleanup** for each of your installations of the web gateway. As versions of the web gateway are installed or deleted you need to add or remove the appropriate crontab entry. You don't need to cleanup installations that are not being used. Only one entry is needed for **imstats** as we are only interested in the log files from whatever is the current public installation.

If you have persistent user accounts in your EOS Data Gateway, then the cron job should run **imscleanaccounts** once per hour. You only need one of these for the persistent user accounts, even if you have multiple installations. The entry in the crontab should look like this:

```
# Once per hour cleanup persistent user accounts.
2 * * * * /home/imswww/pub/lib/bin/imscleanaccounts.pl -cron >/dev/null
10 3 * * * /usr/people/imswww/pub/bin/buildBridge >/dev/null
```

Crontab for imswww

The welcome pages contain a "What's New" section which needs to be updated periodically. To do this, you need to rerun the `makewelcome` script. This script runs quickly, accessing a Web page from another server and copying part of its text into the welcome page. We recommend that this be done daily; updates are not that frequent, but it is more convenient to run this script on a daily basis.

To set up the `makewelcome.pm` program, log in as user `"imswww"` and add the routine to the crontab. For example:

```
# Get the current "What's New" entries:
0 4 * * * /usr/people/imswww/pub/lib/perl/makewelcome.pm >/dev/null
```

The `"valids"` you obtained in step 8 are updated by the EOS Data Gateway Team every two weeks. To be sure to keep your valids updated, the **run_autoxfer.pm** script should be run periodically. This script runs very quickly. It can take less than one second if no new valids are available. Therefore, we recommed that you simply add an entry for the script **run_autoxfer.pm** to the **crontab** for the ~imswww account to be run once a day.

To setup the **run_autoxfer.pm** program, login as user `"imswww"` and add the routine to the crontab. For example:

```
# Get the latest valids for the public installation:
0 3 * * * /usr/people/imswww/pub/lib/perl/run_autoxfer.pm -cron >/dev/null
```

NOTE:

The cronjob program run_autoxfer copies the wwwvalids.conf file included in the valids distribution into the EDG installation. However one set of support_files and therefore one run_autoxfer program often supports multiple EDG installations, this means that wwwvalids.conf only gets updated in the EDG installation which actually ran the transfer. Other installations which also use the same support_files are not updated with the new wwwvalids.conf. In the event that an installation is sharing the valids with another installation, transfer the latest wwwvalids.conf file using the following command:

```
autowwwvalids -l supportFileDir -t topEDGInstDir
```

Where `"autowwwvalids"` is the script that performs the transfer. It should be located in the /lib/bin directory of the installation.

`"-l" and "-t"` are command arguments needed by the script.

`"supportFileDir"` is the absolute path of the support file directory.

`"topEDGInstDir"` is the absolute path of the top level directory for the installation where the wwwvalids.con file will be copied.

For example:

```
# Once a day check/transfer the latest wwwvalids.conf file
0 2 * * * /usr/people/imswww/pub/lib/bin/autowwwvalids -l /usr/people/imswww/support_files/sup_fi
```

6.4.12 Step 12 (Optional): Starting the custom httpd Perl script server.

Warning: We do not recommend running with the custom httpd Perl script server at this time.

If you did not set the variable **PerlHttpdPort** in the setup step and you wish to run the custom httpd go back to **Step 7: Generate and apply the SETUP file.** and do so. If you did not set the variable **PerlHttpdPort** and do not want to run the custom httpd you need not do anything.

To reduce the time it takes to run CGI Perl scripts, a custom httpd server `imshttpd` is run. The Perl server has the the Perl modules preloaded, so when a user requests a CGI-generated page in the EOS Data Gateway the load time for the Perl is reduced. Non-CGI pages are served via your systems standard httpd server.

A `imshttpd` is run for each installation of the EOS Data Gateway on the system. Different versions of the EOS Data Gateway do not and can not use the same imshttpd server. Each server needs its own port number (which was set in the SETUP step).

Each version has its own imshttpd:

```
imswww-2.3 on port 8126,
imswww-2.4 on port 8127.
```

Warning: Port numbers are very site specific, we can not tell you what would be an appropriate port number to use. Port 8123 maybe used by other software on your system. Check with your system administrator to get an appropriate port number.

To start the imshttpd, login to the account `"wwwuser"`. and execute:

```
~imswww/VERSION/lib/bin/imshttpd
```

A successfully started imshttpd will end with the message:

```
Bottom line: OPERATION SUCCEEDED!
```

You will want to have your system administrator add the starting of the imshttpd's to the system init file so that they will be automatically restarted when the system is rebooted.

6.4.13 Step 13: Try out the installation

Get into your favorite WWW browser, and open the URL:

```
http://<your_host_name_here>/~imswww/VERSION/imswelcome
```

For example:

```
http://harp.gsfc.nasa.gov/~imswww/1.5/imswelcome
```

You should see the “welcome” page.

6.5 Making your new installation public

Once you have tested you new installation you will want to make it `"public"`. While you may have several installations of the EOS Data Gateway on your system at once, only one of them is available to the public. This is the version which is available through a standard URL for the EOS Data Gateway (For Example: http://eos.nasa.gov/imswelcome) that is advertised to your users.

6.5.1 Step 1: Warn users

If you already have an installation of the web gateway installed on your system which is being used by the public you may want to warn users of up coming down time.

To warn users of an upcoming change in the system create the file **warn.txt** in the directory **~imswww/VERSION/lib/conf** of **the current public system** and put into the file your warning message. For example:

```
The web gateway will be down from 10:00am to 11:00am EST
next Monday morning for a software upgrade. Please use
one of our other installations if you need to use the
system during then.
```

The warning message will be printed at the top of each screen.

6.5.2 Step 2: Take the system down

If you already have an installation of the web gateway installed on your system which is being used by the public you will want to take the current system down in a graceful manner. If you were to make the switch to a new version without using the **down.txt** while the system is being used the users will suddenly start getting errors because the old system disappeared from under them.

```
cd ‘imswww/VERSION/lib/conf
```

To take the system down create the file **down.txt** in the directory **~imswww/VERSION/lib/conf** of **the current public system** and put in the file your system down message. For example:

```
The web gateway is down for a software upgrade
it will be backup at 11:00 am EST today.
Please use one of our other installations if you
need to use the system during this time.
```

```
An error message page with the text of the down message will
be displayed if a user tries access the system while the down
message is in affect.
```

6.5.3 Step 3: Modify setup

As user `imswww` cd ~imswww/imswww-VERSION and edit the `"SETUP"` file. Change the following entries to point the pub area:

```
CgiUrl         "/ims-bin/pub/"

DocsUrl        "/~imswww/pub/"

WelcomeUrl     "http://<your_host_name_here>/~imswww/pub/imswelcome/index.html"
```

Also in SETUP you need change the `StatsId` from the test one `"v0ieww00"` to the one assigned to your installation by the EOS Data Gateway team.

```
StatsId        "v0ieww02" (for example)
```

After you have made the changes above to your SETUP file you need to `apply` the changes:

```
setup -apply
```

6.5.4 Step 4: Clean up logs from testing and create public log link

To avoid any problems with the logs from testing being included in the official statistics, please delete the old log files:

```
cd ~imswww/imswww-VERSION/log

rm *.statlog
```

```
The log file for the current date will be recreated the first time
some uses the system.
```

Optionally you can create a permanent area outside of the version directory tree for log files, for example:

```
mkdir ~imswww/logs/pub
```

Then when a version is made "public" create a link from the ~imswww/imswww-VERSION/log to the permanent log area, for example:

```
cd ~imswww/imswww-VERSION
mv log log~
ln -s ~imswww/logs/pub log
```

The benefit of doing this is all the log files for the "public" version of the system are then kept in this one "permanent" area instead of being scattered amoung individual log directories of the different versions.

6.5.5 Step 5: Create public link

Delete the existing pub link (if this is the first time you have installed the EOS Data Gateway, then you won't have to delete it):

```
cd ~imswww
rm pub
```

Now create the link:

```
ln -s imswww-VERSION pub    # for example: ln -s imswww-1.5 pub
```

The newly installed version of the EOS Data Gateway should be accessible via your public URL.

6.6 Verify the installation

Purpose: Verify the correct installation of the Web Gateway. Does not test every page in the system, but rather tests selected functions to verify that base level routines and software has been installed and configured properly. For example, testing "Detailed Document" search verifies that the wais software is working properly.

Application: Web Gateway Version 3.4

1. Examine Welcome Page:
   - Check Web Gateway Version number at the bottom of the page.
   - Examine the header and footer customizations for errors.
   - If the site wants user accounts enabled, verify the links for Login and Create are displayed.
2. Map generation - Verify the CGI scripts used to generate the various maps are working and the WDBTILE database is accessible.
   - Java Map:
     Bring up the Java enabled Map. Verify the map GIFs are visible. Make a selection and hit OK. Verify that a summary of your selection is shown on the Search form.
   - Equatorial Map:
     Select a region using the Equatorial Map.
   - North Pole Stereo:
     Select a region using the North Polar Stereo map.

3. Valids - Verify the support files are installed properly and they are the correct type:
    - Go to the Data Center Dependent Valids page and verify that only "CINTEX OPS" centers are listed.
4. Detail Document (Guide) - Verify wais is working. Note that if wais is not working you do not get an error message, instead you will get zero hits. You must verify wais by retrieving at least one document.
    - Go to the Data Center Valids selection page and get a Detailed Document (guide) document for GHRC.
5. Definition - Verify "gethostbyname" function is available:
    - Go to the Sensor Valids selection page and get a Definition for sensor "ACCELEROMETER".
6. Upload Query:
    - Upload the "All Data Centers" query.
7. Inventory Search:
    - Send inventory search to all data centers.
    - To test partial results, start a search to multiple data centers. As soon as some granules come in, hit the View results button.
8. Browse - Verify hdf2gif is working.
    - Select View Browse for one of the Granules.
9. Coverage Map Graph - Verify it runs.
    - Select a granule with non-global coverage (such as AVHRR data from DLR).
    - Do a time graph for the same granules.
10. E-mail ordering - Verify wwwvalids.conf is accessible:
    - Put a granule from a CINTEX site (such as AVHRR data from DLR) into the shopping cart. Once in the shopping cart it should have a "e-mail" ordering icon next to it.
11. Upload PREFS:
    - Upload a user preferences file. Use one that includes the switches for Admin mode.

12. Admin Mode:
    - Verify that a password has been set for Admin Mode.
    - Examine the Crontab for the wwwuser account. Verify the imsstats, imscleanup and imscleanaccounts routines are running.
    - Check the StatsID by looking at the log files.
    - Verify that you can examine active sessions by clicking on the "User sessions, active..." button.
    - Verify that you can examine previous sessions by clicking on the "User sessions, stat logs..." button.
13. Comments (mail working):
    - Send a test comment to verify that mail is working.
14. User Accounts
    - If the site has enabled user accounts, create an account and then login to it.

Chapter 7

Configuration

This section deals with the configuration (or reconfiguration) of the Web Gateway. Configuration is done...

- When you first install the system.
- When you have to make slight adjustments to the Web Gateway because of changes in your local directory structure (or certain URLs).

7.1 The "configure" script

The `configure` script resides at the root of your Web Gateway installation directory tree. It sets up certain files that Web Gateway needs before the system can be compiled: Makefiles, for example.

You will usually *never* need to re-run `configure`, unless you are doing a recompile of the system.

The `configure` script is a Bourne shell script, automatically generated by GNU's `autoconf[igure]` program from a special source file, `configure.in`, that normally resides only with the developers. The `configure` script is incredibly complex, and not meant to be touched by human hands.

7.2 The "setup" script

The `setup` script resides at the root of your Web Gateway installation directory tree. It sets up certain files that the Web Gateway needs before the system can be run.

You will need to re-run `setup` if you change the contents of your **SETUP** file; otherwise, your changes will not take effect.

7.3 The "SETUP" file

The `SETUP` file resides at the root of your Web Gateway installation directory tree. It contains the **configuration variables** which point to important directories, URLs, and other information needed by the Web Gateway at run-time.

Warning: If you change the contents of the `SETUP` file, you *must* re-run

```
    setup -apply
```

to make sure that your changes get propagated throughout the distribution. You will *not* need to do a recompile.

If you don't see the `SETUP` file when you first unpack the Web Gateway, don't panic: you just have to run `setup`: it will give you instructions on how to generate a `SETUP` file with reasonable defaults.

Chapter 8

Administration

This chapter covers the day-to-day administration of the system. It does *not* cover installation, reconfiguration, or upgrade of the system; go to the **Installation** chapter for that.

8.1 Making sure "cleanup" is going well

If you have installed **imscleanup** as a cron job (which you should have), be warned that fatal errors which prevent the running of the program will be emailed *to whatever user account the cron job is running under*, which will necessarily be *whatever user account the CGI scripts run as*.

If, therefore, the cleanup script does not appear to be running, you should examine this account's mailbox from time-to-time (particularly after upgrades or reconfigurations) to find out why.

8.2 The custom HTTP daemon

We are no longer using HTTP daemon, so please do not use it.

8.3 Using the Administration features of the Web Gateway

There are some administration features that can be accessed via the Web Gateway. You must have the ADMIN password to use these features.

8.3.1 Setting up access to the ADMIN mode

Step 1: Add the adminstration flag to your User Prefs file

Opoen up your "User Prefs" file with an ordinary text editor (if you don't have a "User Prefs" file, just go to the Gateway, select "User Prefs", fill out the form, and download it). There is a GROUP called WWW_DISPLAY (it'll have stuff where the "..." is):

```
GROUP = WWW_DISPLAY
...
END_GROUP
```

Inside this group (between the GROUP and END_GROUP), add the line:

```
ADMINISTRATOR = "Y"
```

so you have something like:

```
GROUP = WWW_DISPLAY
    ADMINISTRATOR = "Y"
...
END_GROUP
```

Now save the file.

Step 2: Upload your new User Prefs file

Press the "User Prefs" button to get to the User Preferences Screen. Press the appropriate button to upload your User Prefs file.

If your site's SETUP file already has an entry under **AdminCrypt**, you can skip to step 6 now. Otherwise, you'll need to set up the ADMIN password...

Step 3: Encrypt the ADMIN password

Encrypt the site-wide ADMIN password by pressing the "Encrypt Admin Password" button. Enter your password twice as directed.

Step 4: Enter the encrypted password into your `SETUP` file.

Enter the encrypted password between the double quotes:

```
AdminCrypt      ""
```

in the `~imswww/VERSION/SETUP` file.

Step 5: Run setup -apply

Execute:

```
~imswww/VERSION/setup -apply
```

At this point, your gateway site now has a password.

Step 6: Press the "Admin" button and log in

In the navigator at the top of the page, you should see an ADMIN button. Press it. Login as the administrator. Once you have logged in to the Adminstation system you can access the main functions (keep reading...)

8.3.2 Custom HTTPD

When you press the "Custom HTTPD" button you can do the following functions:

- CHECK THE STATUS of the custom EOSDIS V0 Web Gatway HTTPD
- START A NEW USER SESSION under the custom EOSDIS V0 Web Gatway HTTPD
- START A NEW ADMIN SESSION under the custom EOSDIS V0 Web Gatway HTTPD

8.3.3 User sessions

The "User sessions" button allows you to browse user sessions and to delete the session directories in an emergency.

8.3.4 Browsefile Cache

The "Browsefile Cache" button allows you to browse the cache of browse files on your system. You can also do the following functions:

- Modify the shelf life and the linger time of the browse files.
- Also modify the comfort size of the cache.
- Purge cache until it is below the comfort size.
- Deactivate and re-activate selections.

Delete individual browse files or all of the browse files in case of emergency.

8.3.5 Crontab entries

The "Crontab entries" button allows you to browse and edit the crontab file for the HTTPD user account. This is useful for setting up the cron jobs for **imscleanup**, **imscleanaccounts.pl**, and `imsstats`.

8.4 Bringing the system down

Occasionally, you will have to off-line the Web Gateway... for upgrades, fixes, etc. Here is how to do it with minimal impact to your users.

8.4.1 Warning users of impending downtime

Since the gateway may be used by users around the world, there is no “good time” to suddenly bring the system down: midnight EST falls inside of the Japanese work day. So, it is best to warn users as far ahead of time as possible.

As of v.1.2, the way to do this is to create the file **LibDir**/conf/warn.txt. It should contain, in plain ASCII text, a short message like:

```
The system will be taken down at 12:00 GMT today.
Please finish what you are doing before then.
```

The message will appear at the top of every interface screen.

8.4.2 Bringing the system down

It may occasionally be desirable to bring the system down for upgrades, testing, or other reasons. Currently, one way to do this is to create the file **LibDir**/conf/down.txt. It should contain, in plain ASCII text, the reason that the system has been brought down, and when it is expected to come back up, like this:

```
The system is down for upgrade to v.1.2.
It should be back up again by 14:00 EST (19:00 GMT).
You may wish to go to one of the other gateway sites.
```

The presence of this file is detected by every CGI script during initial setup; it is exists, an error document is returned, containing the file’s contents.

8.4.3 Backdoor: testing a "down" system

There should be a way to do this, so that priveleged users can use the system while everyone else sees it as "down". This has not yet been implemented.

Chapter 9

Change Log

9.1 Version 3.6 (October 2003)

- Added buttons to the search form to change between Primary, Data Granule and Granule ID Searches. In 3.5 and earlier versions the search type was changed by clicking a link in the navigator bar. Since links do not process form data any changes to the search form since the last form button was clicked was not saved. This became more of an issue when we added the javascript based "By Discipline" feature to the search form as any selections made using the javascript were not being saved if the user changed search types. Now if the user makes some selections or changes on the search form they will be saved when they click on one of the toggle buttons to change the search type.
- Enhancements requested by LPDAAC to Landsat 7 Floating Scene selection.
  - Display the path and starting/ending row values for the selected scene (in tenths for the row values).
  - Allow the user to slide the selection box in tenths of a row. So, if the swath covers path 29, rows 22 through 30, as the user moves the selection box along the swath, the values move from 22.0 to 22.1 to 22.2, etc. up to 30.0. The selection box should "snap" to these values. (Upper boundary, lower boundary, and center of the selected scene can still be adjusted separately)
  - Still show the lat/lon values, but show them in degrees and minutes.
  - For the compute cost, use the lat/lon decimal degree values from the selected scene and send these to ECS.
  - When submitting the order to ECS (PRODUCT_REQUEST), adjust the lat/lon values "outward" so that the ordering box is slightly larger than what they saw on the user interface. So, if they had selected rows 24.5 through 27.3, you'd send ECS the decimal degree lat/lon values that correspond to rows 24.45 through 27.35 (i.e. add .05 of a WRS row to

  each end before sending the lat/lon values to ECS - this will add a pad to ensure the user gets what they wanted).
  - On the EDG floating scene subsetter window, a list of the approximate scene corner coordinates (UL, UR, LR, LL) is currently displayed for the user's selected scene. However, the scene center is not included, and this has caused difficulties for some users. Please add the selected scene center (lat-long) to the list of coordinates that are displayed.
  - Add the user's selected path/row values somewhere in the PRODUCT_REQUEST ODL? That way if someone ordered something and we needed to resend the order, we'd know what values were used for the order.

- Added the ability to show MISC_URL in Package groups.
- Added a column in the Shopping Cart to display ASTER On-Demand product names if the user selects these package options.
- Modified to inhibit the display of FIXED specialized criteria if the OBSCURED flag is set in package selection.
- Modified the system to show all browse images vertically on the page.

9.2 Version 3.5 (December 2002)

- Revamped the default keyword selection function on main search page to allow users to identify data sets based on scientific discipline and topic
- Integrated text-based search into keyword selection page
- Added support for use of grid row/column values with subsetting service
- Limit the size of order requests based on volume of order or number of granules ordered
- Added support for ECS data pools and ASTER on-demand processing
- Enable users to pre-select ECS and Landsat-7 media options before entering shopping cart
- Update browse to support latest version of HDF5 libraries
- Reconfigured calendar loading to bypass loading if calendar host machine is not responding
- Various fixes to Landsat-7 floating scene tool

9.3 Version 3.4 (September 2002)

- Perl 5.6.1 is now required to run the system.
- Support for display of granule footprints from AMSR-E instrument
- Introduction of a special search type ("Browse Search") that will be used exclusively to search for data from the AIRS instrument
- Addition of a special data access key for customers wishing to order data from NASDA/Japan
- Interface support for external subsetter tools being developed by the University of Alabama
- Textual changes to "alt tags" and links to make the EDG comply with Regulation 508 - Access by Users with a disability
- Based on the size of a granule to be ordered, present the user with only those media that have capacity sufficient enough to hold the granule.
- A browse image request can now be submitted from the Granule Attributes page via a new "Available" link.
- When the "Request Sample" function is invoked (also know as FTP Browse), an intermediate page pops up to clarify the meaning of "Request Sample".

9.4 Version 3.3 (May 2001)

- Changes were made to support the new Spidex/Isearch based guide search engine in addition to the WAIS search for guide (Detailed) documents.
- Seamless User Registration between EDG and ECS was added. This enables users to create a new account on EDG and ECS at the same time.
- A link to a Gazetteer was added. The Gazetteer allows the user to enter a place name and get back the latitude/longitude coordinates of the location.
- Users can customize what types of metadata are returned from a search and displayed in the result tables.
- The ability to enter a point location was added.
- The ability to display a browse palette was added to the system. It was added primarily to support AIRS browse, but it will work with any 8 bit images.
- A username for DORRAN was added to support the accounting system at the EDC DAAC.

- For search by data granule ID, a change was made to only display the data sets from data centers that allow search by data granule ID.
- A caption that includes the data set name and granule id was added to browse images.
- Improvements to the ordering of partial scenes for Landsat 7 were made.
- An ultra high resolution map data base to support the java geographic selection page was added.
- Use of the Terra Products page is now integrated with EDG accounts.
- Added support for HDF5 image formats.

9.5 Version 3.2 (January 2001)

- Data set lookup tool is integrated into the primary data search page. There are now two methods by which valids can be chosen: via the lookup tool or via a pulldown menu.
- The requirement to choose at least one valid from either "Data set", "Sensor", or "Parameter" attribute lists was removed. A valid from any attribute list is sufficient for a search.
- An optional map that can be used to visualize the location of a typed-in lat/lon or path/row is available in the geographic area selection.
- Summary document search page was redesigned to make it consistent in look and feel to all other search pages.
- Required user information that must be supplied as part of the data ordering process is now presented as starred (*) red headings. The '*' will improve the clarity of the order form for colorblind EDG users.
- New links were added to the navigation section: EDG tutorial is now accessible here, as well as information about HDF software tools.
- "Miscellaneous URL" display capability was added to the search results pages.
- The box for setting the maximum number of Detailed documents to be returned in a detailed document search was moved to the User Preferences section.
- A link to the "Terra Products Page" was added to the EDG Home Page. This Products page provides an alternative path for entering the EDG.
- Local Granule ID is displayed on the Order Summary Page.
- DAAC contact information is displayed on the Order Failed page when an order fails to that DAAC.

- Corrected multiple bugs that resulted in inoperative submit buttons under a MacIntosh/Internet Explorer 5.0 environment.
- Corrected bug that prevented the "Day Only" or "Night Only" flag from being sent in a search.

9.6 Version 3.1 (September 2000)

Changes to support ECS:

- Modifications were made to the Floating Scene Applet for Landsat 7 to include a flag that allows the user to order an entire subinterval, if the entire subinterval was available for order.
- Modifications were made to support flags for include non-spatial and include non-temporal granules. Return non-spatial only and return non-temporal only will be handled by Product Specific Attributes (PSAs).
- Modifications were made so that users can select the attributes of metadata that are returned in search results. This feature has not yet been implemented by ECS so this feature is currently turned off. The customizing of the "ALL" or "Default" flags was moved to the customize results page.
- Code changes were made to ensure that data set, source and sensor disclaimers are working correctly for ECS data sets.
- Code was added to support the display of Guide URLs, Misc URLs and Data URLs.
- Changes were made to add the capability to have a media handling fee. This fee will be applied once per each type of media for each DAAC (data center) that has this feature.
- A feature was added to accept Julian date format (year-day of year) in addition to the standard month-day-year format.
- Changes were made to send the list of data sets along with the other attributes chosen. This should speed up searches for the ECS data centers.
- Changes were made to calculate the approximate MISR granule center point latitude and longitude for granule coverage map display.

Changes to support usability improvements:

- Search by Granule ID and search by Local Granule ID were modified to use the improved search form.

- The Order Manifest was hidden from public view. Also the session directories were hidden from public view from the browser.
- The detailed document search (guide search) was modified to use the improved search form.

9.7 Version 3.0 (April 2000)

Changes to support ECS:

- A Java applet was developed to allow users to subset Landsat 7 Floating Scenes to the desired area. This included the development of a Price Estimate Message that is used to determine the cost and number of scan lines that the user is requesting.
- A flag was developed so that the granules returned from ECS can have the default set of metadata returned or ALL possible metadata returned with each data granule returned.
- ECS User Registration was implemented, but it is turned off in the present build. It was turned off because the DAACs were not ready to implement ECS User Registration at this time.
- Code was developed to support the Ftp Push order option. A password widget was added to allow typing a password (without displaying it) for ftp sites that need a password.
- Code was added to display the Local Granule ID in addition to the regular Granule ID on the granule list, folder and shopping cart.
- Code was added to display a warning when a user adds a data granule with a "failed" quality flag to the shopping cart.
- Code was added to display complex MISR data granule footprints and their associated path and block numbers on Covermap.

Changes to support usability improvements:

- This was a major overhaul of the search creation form. We also eliminated the simple form and replaced it with a simple search widget available from every page.
- Code was developed to help the user customize the results pages (granule list, data set list, and folder). Also, code was developed to support printing a text report.
- Code was added to allow a user to click on the "Forgot my password button" and a new password generated and emailed to the owner of the persistent account.

- Code was developed to remove the CGI login screen after the browser popup login. Now there is only the browser popup.
- Code was added to save a history of orders within a session to allow easy product status requests.
- Changes were made to the geoselection map applet to show state boundaries at 4x zoom instead of 16x zoom.
- Code was developed to make a more user friendly display and manipulation of the locations of the data granules on a map. Added panning capability on granule Covermap.
- Code was added to display the order options along with the cost for each granule and display it on the granule attributes page.
- On the Communication Status page change the indicators to only show a fail status if the communication with the data center failed, not if there was no match found.

9.8 Version 2.4 (Sept. 1999)

- The EDG can now be setup to run under a secure web server. Transactions between the user's web browser and the EDG will be encrypted, resulting in increased privacy of the transactions between the user and the EDG system.

  Communications between the EDG and the data servers can now be encrypted. As data servers are upgraded to use encryption, the system will become more private and reduce the likelyhood of malicious tampering of the communication messages exchanged between the EDG and data servers.
- The size of the data files in bytes is displayed on the order options page if available.

  A per order handling fee for some data sets is now displayed and included in the cost calculation.
- Changed colors used on the orthographic map and selection box to make them easier to see.
- All of the Geo options are now available from the simple search form.
- When saving search results to a database style file on the user's computer, carriage returns in the comment field were messing up the table structure of the file and geographic coverage was not being included.
- Performance improvement in getting definitions from the simple search form.

  Implemented cost estimate message which will be used in future releases of the system.

Made changes to support subsetting of Landsat 7 subinterval data into floating scenes - not available to users in this release but lays the ground work for this capability in the future.

9.9 Version 2.3 (July 1999)

- The welcome page was redesigned. If the user's browser supports java script then a java script enabled page is displayed with popup text for each link on the welcome page. If no java script then a plain html page is shown. A section on "What's new" in the system was added and is dynamically updated.
- The navigator menu was streamlined - the gif images were replace by "text" links. The inline contextual help was dropped as well as the status message for each item in the navigator. Links are no longer greyed out if the option isn't active yet, instead text is displayed suggesting what steps need to be done to activate the option.
- Long lists of data sets and data granules are now paginated. A pagination menu is displayed at the bottom of the screen with links to the next "pages" of the listing. Selections and the last page displayed are remembered if you leave the page and come back.
- Previously you could only get to the granule list by going to the data set listing and rebuilding the granule listing. Now have a granule list link which takes you to the most recent granule listing.
- Dropped the small equatorial map option as this was redundant in function to the full size equatorial map.
- New and improved maps in the java map applet for geo search region selection. The new maps have more detail, more zoom levels and more overlay options including WRS overlays.
- Now have the option to display a disclaimer message when data is placed in the shopping cart. Done primarily for TRMM data.
- Added the day/night option to the search form.
- The geo selection applet (Mapplet) user interface was redesigned to be cleaner and smaller.
- ECS authentication and profile retrieval have been dropped until ECS is ready to support them.
- The simple search form was updated based on user feedback. Added definition button for the term list and boolean options ("Contains ANY", "Contains ALL", and "Match Exactly") for the data set catalog search.
- The order submitted screen and order summary e-mail message were updated so the user can more easily determine the status of the order.

9.10 Version 2.2 (Feb. 1999)

- ECS attributes are now supported using the extended search criteria.
- Product specific attributes are displayed on the granule attributes page.
- Users can now retrieve and update their ECS profile on the ECS profile server from the web client.
- Users can now set up their web client account to use the same username and password as their ECS account. In addition users can now change the name and password of their web client account.
- Integrated browse HDF 24 bit raster images are converted to JPEG format instead of GIF, so that color resolution is not lost.
- A number of small changes were made to the granule listing and order options to help support ordering of Landsat 7 level 1 data.
- Made several changes to the Dependent Valids Applet to improve performance and usability.
- The first twenty characters of the Data set comments are now displayed on the results page with a link to the full data set comment on the data set attributes page. Data sets without any granules are also listed instead being filtered out so the use can get a better idea of why no granules matched.

9.11 Version 2.1 (Oct. 1998)

- The Simple Search was revised based comments on the prototype fielded in release 2.0. The Simple Search screen is now the default search screen with an option to switch to the Advanced (full featured) search screen if desired.
- A new Navigation bar was placed on the left side of every screen, this replaces the button bar previously at the top of the screen. The navigator outlines the basic steps in searching and ordering data and provides an indicator as to the user's progress through the system. In addition, there are context sensitive status messages and help.
- A new Java Applet for selecting dependent valids was added. Users can now select from all 7 dependent valids categories using just one screen. Selections are validated against previous selections for all categories. The validated query is displayed and has incompatible selections indicated. Users have the option of switching to the non-java CGI based valids selection screen.
- Users can now view the partial results of a search while the search is still running. They can then switch back to the search status screen to check on the status of the search or redisplay the partial results to see addition results using

"View Search" and "View Results" Options in the Navigation bar. A contextual status message in the Navigator indicates when partial results are available for viewing.

- This version has been updated and tested for year 2000 compliance.
-
  - Perl 5.004_04 is now required to run the system.
  - Now support IRIX 6.5 on SGI, in addition to IRIX 6.4, Solaris and Linux.
  - Now support IRIX -n32 compiler option.

9.12 Version 2.0 (Mar. 1998)

- Added a prototype search form which has basic free text search interface. The user types in a string which is then compared against all the of V0 valids to find any matches. Matching terms are then mapped to their corresponding data sets. The user then has the option of doing an Inventory Search on the selected data sets.
- The following terminology changes were recommended by the usability improvement team. The purpose is to replace ESDIS jargon with more descriptive terms.

Old Term:	New Term:
Inventory Search	Data Search and Order
Guide Search	Detailed Document Search
Directory Search	Summary Document Search
Granule	Data Granule
Brief Info	Summary Document
Detailed Info	Detailed Documents
Package	Ordering Options

- Users now have the option of becoming a registered user. This creates a persistent (permanent) user session which they can reconnect with at anytime via a password controlled login. Things such as user information, searches, search results, etc. are saved for the user. imscleanaccounts.pl is a script run in the wwwuser crontab that purges old files from user accounts and checks disk usage.
- The data granule coverage map now has the ability to display granule footprints which are described using g-rings. See the **G-ring Specification** for more details.
- Users can now retrieve their billing and accounting information for needed for ordering from ASF separate from doing an inventory search.

- Users can now check on the status of orders placed through the Web Gateway. If the order was placed to a data center which supports the order status ODL message then the request will be sent via ODL. If not, then the order status message is sent via e-mail the data center user services.
- If a user enters an invalid valids keyword string, the system will now prompt the user if they wish to go ahead and do a search using that string anyhow. If yes, the new "valid" is temporarily added to the valids. This feature is designed to allow daac testers and developers to test new data sets in their servers without the corresponding information being in the valids yet.

  To try out this feature:

  1. Click on **Compose a Search**.
  2. Click on the **User Prefs** button at the top of the page.
  3. Scroll down to the 'Interface Preferences' section and set **May I use Dynamic Valids in generating a query?** to **YES**. Scroll down and click on the **Ok1 Accept Display Options Only!** button.
  4. Click on the **Search Form** button at the top of the page.
  5. Click on the **Edit** button under 'Data Set'. Scroll down to the **Type in** section. Enter an invalid data set name (for example `cheese`) and click on the **Apply** button.
  6. You will see the error message: *Sorry... there are no compatible valids for Data Set: cheese*. Click on the **ADD** button. You have now temporarily added the keyword `cheese` as a data set valid.
-
  - Now have support for the ECS authenticator.
  - User can now set the maximum time that a search will run. Once the **Max time** is reached then the search is automatically stopped and any partial results displayed.
  - For data centers which support it, users can now order multiple copies of the same line item.

9.13 Version 1.9 (Nov. 1997)

- Added option to select the geographic search region using a Java enabled map (Mapplet). The mapplet has the following features:
  - "Rubber band" selection of search box on map with the corresponding coordinates of the selected region shown on the mapplet.
  - "Zoom" to higher resolution and detailed maps.
  - "Pan" of map/globe using slider bars or by entering new center lat/lon.
  - Display options for:

    - Rivers and lakes overlay.
    - Political boundaries overlay.
    - Changing lat/lon grid size.
  - A reference globe to show the where you are. Helps you keep track of where you are as you zoom and pan.

- The system was thoroughly tested for compatibility with Microsoft Internet Explorer on a number of platforms. Major changes were made to the Communications status screen to use client pull instead of server push. In addition a number of smaller changes were made through out the system to improve usability for MSIE uses.

- All extraneous functions were removed from the shopping cart. Only those functions needed for ordering data remain. The idea is to help the user focus on what needs to be done to order data and not distract him.

- New compact (tabular) version for the granule and data set listing. A primary action for the listing is highlighted to aid the user in deciding what to do next.

- The guide document is automatically displayed if only one hit is returned from a guide search. The user now has the option of changing on the search form the default maximum limit of 40 guide documents per data center per search.

- Support has been added for the new BROWSE_URL ODL group. The "View Browse" button now shows either Integrated Browse, Browser URL or both, depending on the type supplied by the data center.

- Added link on Welcome page to GCMD. Added new mode to the web gateway which builds an inventory search based on input from the GCMD. A user can do a search in GCMD to find a list of data set DIFs and link from there to the VO IMS web gateway. An inventory search using data sets and any geo selection criteria is automatically created.

- Added support for two new packaging options.
  - ORDER_OPTIONS which is used to specify how data will be processed.
  - DATASET_ORDER_OPTIONS for data set based ordering. The user can specify a set of criteria which will be used for selecting the specific granules to be included in her order.

-
  - Made postal (zip) code non-mandatory.
  - Granules which can not be ordered are marked with "Can't be ordered" with a link to an explanation page and appropriate data center contact information.
  - All clicks on data center supplied URLs (data set home page, data, miscellaneous, and browse) are now logged by the system, so we can if a user "left" the system by following a link to a data center web page.

  - The current search region is shown on the granule coverage map.
  - Added support for Annually repeating time periods.
  - Implemented work-a-round for the problem of multiple line items (packages) for the same item in orders. ASF had requested that users be allowed to select multiple package options for the same granule, however some data centers can't handle this. The multiple packages option can now be turned on or off on a per data center basis. Data centers which want this option can have it turned on and data centers which don't want or support it can have it turned off.
  - Added new test script for testing the Java Mapplet.

- - Integrated snap shot of the latest IK software.
  - Added infrastructure for supporting Java applets within a CGI based system.
  - Includes new ODL library which fixes:
    - The backslash n problem.
    - Limit on string length (long URL problem).
    - Crash if sequence string too long, such as selecting all parameters in the valids.

9.14 Version 1.8 (July 1997)

-

- Preliminary support for extended search criteria has been added. The extended search criteria categories and variants are fully dependent with the core attributes. Extended search criteria can be specified in one 5 different ways:
  - fill-in-the-blank (__________________)
  - fill in a range (from _____ to _____)
  - select one number from a range
  - select one item from a list
  - select one or more items from a list

- On the main search form a new selection labeled **Other Criteria** was added. This takes the user to a page for selecting which extended search attribute categories he wants. The user is then taken to the variant selection page. The extended search categories selected by the user and the choices made within each of the categories are displayed on the main search form.

Data Set Based Ordering:

- Added support for ordering data sets using subsetting criteria from either a inventory search or directory search. Data set which are orderable are labeled as being **Orderable** on the data set listing page. For a data set to be orderable the data center must supply packaging information for the data set. Orderable data sets can be placed in the shopping cart. Packages and optionally subsetting criteria can be selected for each data set. Orders can be placed for data sets and/or granules to the data centers.

Saving Search Results to text file:

- From the Data Set Listing page the user can save the search results to a text file local to his machine by using the "Save to Disk" button.

Multiple Data Center Contact information:

- After an order has been place one or more daac contact information are displayed for each data center.

Usability Improvements:

- The main search form was extensively redesigned to make it more compact and easier to use. Feedback from usability testing with the USWG was taken into consideration. The overall functionality remains the same except for the addition of the Extended search criteria.
- The page for used for selecting valids was redesigned to make it less confusing and to more clearly show the relationship between filters and choices. Again feedback from the usability testing was the main driver for the changes.
- The work flow for ordering data was streamlined. Numbered steps lead the user through the ordering process.

Odds and Ends:

- Default WRS in path/row changed to WRS-2.
- "All Compatible Data_Center_ID" changed to "All Compatible Data Centers" on valids selection page.
- Removed single item guide search option from the Parameter selection page.
- Guide and definitions removed from filters for all attributes.
- Revised help pages for search building and ordering.
- The status code comment is displayed after ftp browse and order.

Behind the scenes:

- The infrastructure for handling how a search query is passed and stored was completely redone. The main driver for this was to add support for the extended search criteria. The query state is now stored on the server instead of being passed from page to page using form tags as before. All pages which deal with with the query state had to be rewritten.
- Infrastructure for passing query state between java applets and cgi forms.
- Status of orders and ftp browse are now read from the ODL directly instead the Rx*.ODL. Status code comments are now displayed for each of these.
- Support for "Both" poles flag added to granule coverage map.
- Fixed date padding in granule temporal coverage generation to fix "bad" date error.
- Fixed race condition between imscleanup and a brand new empty session directory.
- New version of HTML::Stream.pm installed.
- Definitions retrieval now looks in the same directories as the X-client.

9.15 Version 1.7 (April 1997)

Search Form:

- In the search attribute selection pages provided a menu for jumping to other attribute selection page, so the user can go to the next step of building his search without going back up to the search form after each attribute selection step.
- Reorganized search form flow and generation to support the four search types (Guide, Inventory, Directory, and Granule ID).

Granule Id Search:

- Added new search option for search by granule ID. To allow users to type in a granule ID if known.

Geoselection widget:

- Support for entering a Path/Row specification for geospatial selection area. Converted to a Lat/Lon range for data centers that don't understand path/row.
- Polar stereographic map projection for geospatial area selection.
- Coordinate entry boxes, keep displayed coordinates in the same format as the user originally entered his coordinates.

Order Subsetting:

- Provides temporal, spatial, and criteria based subsetting as part of granule packaging options.

Directory Search:

- Add URL extensions to Directory Search results.
- Fast Directory Search (drop preloading of GCMD diff info).

Granule footprint coverage map:

- Implemented color blending code were footprint over lap
- Detail level of stamps controlled separately from big gif.
- Detail level of footprint increases with zoom.
- Granule footprints displayed using multiple colors.

Guide Search:

- Made changes needed for the "numeric" mapping of guide keywords. The system now supports both old and new style document indexes.
- Guide Search based selection of multiple valids.
- Free text guide search box.

User Prefs:

- User is now able to select display preferences without having to fill in rest of the form.
- Changed field validation in User Prefs so it doesn't accept fields with just a tab or space.

Behind the scenes:

- ODL is now read directly via IK hooks.
- Increased length of DAAC status code comment to 256 chars.
- Installed new IK s/w with improved communications polling.
- Separate stats page for each Web Gateway installation.

- Communication Status Page updated for Browse and FTP so it checks status code and displays error gracefully instead of dying.
- Validation of fields in User Comments.
- Changed FTP browse to be recorded as "productreq" events.
- Changed dying gasp to short form for "back" button errors in the shopping cart.

9.16 Version 1.6 (Jan 1997)

Search Screen:

- Search Type Option - Inventory or Directory Search. Now support directory search.
- Geographic coordinate entry boxes - a variety of formats are now supported.
- Campaign and Processing Level attributes added.
- Max granule limit changed to 1000.

Valids Selection Screens:

- Text entry box which allows wild cards using "*" and multiple strings. (Short cut for experts: AIR* followed by a carriage return will select all values which match AIR and take you directly back to the search form).
- Filter selection.
- "Select All" and "Select None" buttons for quick marking of items.
- Items selected by the user but made invalid by further dependent valids selections are marked with a "*".

Search Results Screen:

- Daac status_code_comment (if provided by the daac) is now displayed.
- Daac name is now a hyper link to Daac contact information.

Data set List:

- Added data set listing for directory search results.
- Button to create an Inventory Search based on the data sets selected by the user from an directory search results data set listing.

- "Add to Workbench" and "Remove from Workbench" buttons displayed on inventory results data set listing.
- "Links" ICON on Data set Listing to:
    - DATA_URL: link directly to the data if available.
    - DATASET_HOME_PAGE: link to data set specific URL.
    - DATASET_SPECIFIC_SEARCH: link to data set specific search tools.
    - MISC_URL: any other link a daac might like displayed.
- Filter out Data sets w/ no granules from inventory results.
- Display "number_of_granule_hits" from inventory results.

Data set Attributes Page (Inventory Search):

- Added data set status_code_comment.
- Added campaigns/processing level to granule/data set attributes.
- Fix format of "comment" on data set attributes.
- Added links (multiple entries supported for each type) for:
    - DATA_URL: link directly to the data if available.
    - DATASET_HOME_PAGE: link to data set specific URL.
    - DATASET_SPECIFIC_SEARCH: link to data set specific search tools.
    - MISC_URL: any other link a daac might like displayed.
- Display "number_of_granule_hits".
- Data set-Specific Contact Address.

Granule Listings (GranList and Workbench):

- "Add to Workbench" and "Remove from Workbench" buttons.
- Added "LINKS" Icons to Granule Listing.
- Granules are now displayed in two lists, first the selected granules then the unselected granules.
- Map or Time display modes now keep the full list of granules (not just the selected list of granules).
- "On Workbench" and "In Cart" icons show which granules have been put on the Workbench or in the Shopping Cart.

- Added button for FTP browse.

Granule Attributes Page:

- Previous and next arrows allow stepping through the list granules.
- Added campaigns/processing level attributes.
- Fix format of "comment".
- Added links (multiple entries supported for each type) for:
  - DATA_URL: link directly to the data if available.
  - MISC_URL: any other link a daac might like displayed.
- FTP browse button.

FTP browse:

- Added support for FTP browse.
- Added support for DATA_URL if returned by a FTP browse request.

UserPrefs:

- Added Champaign and Processing Level to list attributes which can be selected by the user in "User Prefs" for display on the Data Set and Granule Listings.

Granule Coverage Map:

- Support for footprints in different colors, with blending in overlap areas.
- Added previous and next arrows at the top of the map to make it easier compare coverage between granules. There are two modes either step through the list of selected granules one at a time or add/subtract next granule from the map. The "current" granule is displayed in red and the other granules on the map in gold.
- Each granule displayed on the map is marked on the granule list by a square of color matching that of its footprint.
- The default centering of the map is now the center of the first or current granule on the map instead of 0 lat 0 long.
- Fixed bug for coverage maps that include the poles.

Granule Temporal Coverage:

- Each granule displayed on the graph is marked on the granule list by a square of color matching that on the graph.

Workbench:

- Created new screen called the "Workbench" which is an area where users can put granules for closer examination without having to put them into the shopping cart. Granules stay in the workbench between searches within the same session.

  From the workbench the user can do all the activities for browsing and comparing granules (View Browse, Map, Time, Attributes, etc.). At any time the user can add or remove granules to/from the workbench and add or remove granule to/from the shopping cart.

New Menu Buttons:

- New more compact and more readable menu buttons for the Data Set List, Granule List, Workbench and Shopping Cart.

Behind the scenes:

- Valids will now be automatically updated by the autoxfer script.
- Removed hardcoding of wais serves for guide search.
- Made it easier for other installations of the web gateway to customize the welcome page.
- Save the granule.gdb and dataset.gdb files between searches so that ViewBrowse will work for granules from previous searches that are stored in the Workbench or Cart.
- Now read the ODL directly via hooks into the IK layer - this makes it easier to add support for ODL extensions to the web gateway without having to change the IK layer for each extension.
- WDBTILE is now read from the valids directory instead of being part of the web gateway distribution.
- Added several features to make it easier to track down errors.
- A number of common user errors are now handled in a more graceful manner and don't generate an error mail message.
- A dummy V0 server was set so the ODL extensions can be tested.

9.17 Version 1.5b

This is a minor set of changes and bug fixes to 1.5a.

- Made the software for mailing logs to GSFC and statistics page generation part of the distribution.
- Intergrated IK 6.0.16 library.
- Folded in changes needed for Solaris and Linux ports into the baseline code. There shouldn't be a need anymore for separate distributions for each platform. The one distribution should now build on IRIX, Solaris and Linux with no or minimal changes. To work-around Perl Flock bug on Solaris made all flock calls from perl to be exclusive locks.
- Fixed bug in cost estimation in building a order.
- Makewelcome script was rewritten to remove hardcoded references and read them from setup. If the variable **PerlHttpPort** is left blank in SETUP then the standard httpd is used instead of the custom imshttpd.
- Upload and Download features made to look the same between the Search and Prefs pages.
- Hardcoded references removed from ModeINFO and put in setup.
- Fixed granule list bug in COVERMAP and COVERTIME. All the granules were displayed if none were selected - now a warning message is given if no granules are selected before doing COVERMAP or COVERTIME.
- Added event logging to all modes.
- Added standard event logging to Definition, Guide, Comment, and Help. These were overlooked in the 1.5 rewrite.
- Updated Design Documentation. Improved Installation instructions and added section on system administration.
- Added support in COVERMAP for filled ranges.

9.18 Version 1.5a

This was a major rewrite of the entire system, with added functionality as well.

- Order-by-e-mail is now supported, for the benefit of the CINTEX sites which do not yet transmit packaging information for their granules.

- User preferences and search constraints may now be uploaded via the common file-upload mechanism (if your browser supports it). Both files are now in ODL format; in addition, the standard XGaea preferences files may be uploaded into the Web Gateway.
- User authentication (based on the AUTHENTICATION_KEY in the user preferences) is now supported. This allows user account information to be used in the data ordering process.
- Definition files are now fetched from the master site (currently harp) and cached locally.
- Browse images are now stored in an encapsulated, near-fixed-size cache. Future releases may make use of a pure fixed-size cache.
- The coverage map is now generated by new, sphere-quadtree-based code, which fills in the data coverage on the globe, rather than merely outlining it.
- The WAIS-based guide search has been improved.
- A special mode, ADMIN, was introduced to make maintenance of the system a little easier. This will be more fully documented in 1.6.
- A small Perl HTTPD, imshttpd, was written to speed up execution of the system by bypassing the parse-and-load phase. *This is an interim solution* until the new Perl Compiler is able to compile the system.
- Major, major improvements were made in the communication status page, both to allow it to be presented in the browse and order stages, and to make it more reliable.
- The imsclient program was modified to *not* open the data set/granule GDBM files read-only when doing a browse.
- The system now uses the 1.6 release of the IK libraries (up from 1.4).
- All code was reorganized into the IU:: (IMS Utilities) and IW:: (IMS Web) namespaces. Many of the modules may be reused in other applications.
- The entire system has been placed under RCS.
- Numerous configuration options have been placed in the SETUP file.
- Documentation source has been migrated to the top of the search form.

9.19 Version 1.4

This was an interim release that corrected several bugs. No real functionality was added; that was deferred to the 1.5 release.

However, some of the backend functionality was migrated into the new IU:: module namespace.

9.20 Version 1.3

- Added spatial and temporal coverage maps, as supported by the new **imscovermap** and **imscovertime** executables.
- Turned IwGranule and IwDataset into real OO classes, with proper instance variables and access methods.
- Modified entire access to results. Now, granule and data set objects are stored in a Perl-readable GDBM file (`results.gdb`) using the new IwStorable package. **imsgdbdump** was modified to support a "-all" option, which is invoked after a search to dump all the information to the front-end, which creates `results.gdb`. All the other data set/granule access options can now safely be eliminated from **imsgdbdump**.
- Modified **imsclient** interface to do post-processing of search results... this necessitated major changes in the communication-status module(s).
- Added the compilation of some granule information (e.g., sources and sensors) into the data set information. The new information is optionally displayable. This was a long-standing request.
- Moved the geographic selection to the top of the search form.
- *(As per ASF's request:)* Added support for multiple data set/granule sort keys. Currently using only two.
- *(As per ASF's request:)* Removed package id from shopping cart list, and replaced it with cost. Also, added total cost of items in cart to the stats at the top.
- *(As per ASF's request:)* Added display of granule's coverage, and added display of the granule's center lat/long to the granule attributes and (as an option) to the granule listing.
- Shrunk icons slightly, as per ASF's and PW's request to fit more information on screen.
- Added A-Z "jump menu" for parameter, source, etc. selection.
- Dying gasp is now mailed directly to the administrator.

9.21 Version 1.2

Baseline for this change log.

Chapter 10

Technical notes

This section is a sort of addendum which contains information that either (1) has not yet been filed in its appropriate place in the design document, or (2) has no real place, but is too important to bury somewhere.

10.1 Interface for passing a query from CEONet to the V0 Web Client.

CEONet will link to the VO IMS Web Gateway on harp via a cgi post request (a get request is also okay but limited in the number of parameters which can be passed). A query from CEONet to the Web Gateway should be of the following form:

```
http://harp.gsfc.nasa.gov:80/ims-bin/dev/nph-ims.cgi?
mode=CEONET
&md=amprtbcx
&md=msu23rdl93
&north=40
&south=-30
&east=-70
&west=30
&btime=1995-10-02T00:00:02Z
&etime=1996-10-02T00:00:02Z
```

- The tag mode=CEONET tells our system that the request is coming from CEONet.
- One or more GCMD master directory ids are passed to us, using md=amprtbcx. The master directory ids will be added to the VO valids as aliases to the corresponding data sets.

- An optional latitude/longitude bounding box can also be specified by including tags for north, south, east and west in degrees of lat/lon. If no bounding box tags are given then the search region will default to global.
- An optional begin and end time can also be specified using the "btime" and "etime" tags. The search will have no time constraints if no btime and etime tags are given.

10.1.1 Details of Implementation in the Web Gateway:

A new mode call IW::modeCEONET.pm will be created. This is the mode which is called by the CEONET system when they want to pass a query to us.

ModeCEONET does the following

- A query object is created.
- The search type is set to Inventory.
- A geo selection bounding box (range) object is created, using the north, south, east and west cgi parameter tags. The geo selection is defaulted to a global search if no geo tags are passed in.
- A temporal time rage object is created using the "btime" and "etime" tags. The search will have no time constraints if no btime and etime tags are given.
- The master directory entry IDs are temporarly put into the query object as dataset ids. The master directory entry IDs will be stored in the VO valids as aliases to the data set ids, so we use imsdepval to convert the master directory entry IDs (aliases) to a list of VO data set ids. The "real" data set IDs are then stored back into the query object.
- A redirect is issued to mode=SRCHFORM, submode=CEONET to load the main search form.

The new submode CEONET was added to IW::modeASRCHFORM.pm. This submode just prints some informational text at the top of the page saying the query come from CEONET ...blah, blah, blah.... then calls send_page which loads the main search form using the query object we created in modeCEONET.

A new method was added IW::Depval.pm called get_aliases.

10.2 How Data set based ordering works

10.2.1 Reasoning behind data set based ordering (lifted from Helen Conover's ODL extension document)

The current V0 IMS data order paradigm is optimized for high-volume or costly datasets that are typically ordered by the 'granule'. Users can sort through lists of granules, select a few essential granules for purchase, and order them individually. This paradigm is ideal for users needing particular data scenes. However, users who need time-series data are not well served by the IMS. For example, some users want all the data from a particular satellite, for a three year period. These users do not want to do a granule search, browse the data, and select 1000 individual granules for order. They will typically bypass the IMS entirely and place an order directly with User Services. Dataset-based ordering provides a way for the user to order "Dataset X for time period A to B, for geographic area C" through the IMS.

The V0 IMS X Client is very 'granule-centric', and it would likely require a major rewrite to provide both dataset and granule ordering in an intuitive manner. But the Web Gateway is more 'dataset-centric'; it presents search results as a list of datasets. From this dataset list, the user can choose to view guide documents, a directory entry, or a list of granules. Dataset ordering can be provided from this point as well.

10.2.2 Requirements (again lifted from Helen Conover)

This feature should support global or regional time-series data, geographic data that is not time-based, and data that has time/ space-based granules. Granule subsetting can be provided optionally for any of these types of datasets.

- Server can indicate that dataset is orderable on a temporal basis, geographic basis, or both.
- Server can indicate that granules of the dataset can be subset.

If a user chooses to place an order at the dataset level (after either directory or inventory search), the user interface will allow the user to specify:

- Send me all granules from Dataset X that overlap: specified time range (may include annually repeating) – if temporal order available specified geographic area – if geographic order available
- Additionally, subset each data granule as follows: – if granule subsetting supported subset specification

Of course it will be up to the DAAC and its IMS Server to translate this type of order into the granule list for the order desk, and not all DAACs will want to support this order mechanism. However, it may be a more natural way for users to order larger amounts of data.

10.2.3 The ODL message we get from the servers:

Those are a lot of heavy requirements I just listed above, however as it turns out we can piggy back on all the work done for granule-based order subsetting. The **SAME** PACKAGE group used for granule subsetting can be used for data set based ordering and subsetting. All that is needed is for the data center to include as part of the DATASET group one or more PACKAGE groups for the data set and a PACKAGE_ID for the data set which points to those packages which apply to the data set. This is exactly the same mechanism used for granule PACKAGES. The big difference is that both DIRECTORY_RESULTS and INVENTORY_RESULTS can have data set packaging information. For the same data set, packaging information sent by the data center in a directory result **MUST** be exactly the same as the data set packaging information sent in an inventory result (why would they ever need to be different?). If they are **not** the same, the user will see strange things happening, as one search type will over write the package options from the other search type.

The web gateway uses the presence or absence of the PACKAGE_ID keyword in the data set group to decide whether or not the data set is **Orderable**.

Here is an example of the ODL for a DATASET group for which data set - based ordering is available:

```
GROUP                  = DATASET
  DATASET_ID           = "MSFC SSM/I BRIGHTNESS TEMPERATURE GRIDS
      FROM DMSP F13"
  STATUS_CODE          = 1
  MD_ENTRY_ID          = "mif13tbg"
  ORG_CENTER           = "STUB DAAC"
  COMMENT              = ("Daily global 0.5 x 0.5 degree gridded
      Special Sensor", "Microwave/Imager Brightness Temperatures
      from the DMSP F-13", "satellite.  The SSM/I is a 7 channel
      passive microwave", "radiometer system.", "Temporal
      resolution is 1 DAY", "Spatial resolution is 0.5 deg grid",
      "Location is GLOBAL")
  SOURCE_NAME          = ("DMSP-F13")
  SENSOR_NAME          = ("SSM/I")
  PARAMETER            = ("BRIGHTNESS TEMPERATURE")
  PROCESSING_LEVEL     = ’1’
  CAMPAIGN             = "WW2"
  PACKAGE_ID           = ("5678")

  GROUP                  = PACKAGE
    DATA_CENTER_ID       = "STUB"
    DATASET_ID           = "MSFC SSM/I BRIGHTNESS TEMPERATURE GRIDS
      FROM DMSP F13"
    PACKAGE_ID           = "5678"
    NUMBER_OF_GRANULES   = 1
    NUMBER_OF_OPTIONS    = 1

     GROUP                  = PROCESSING_OPTION
       OPTION_ID            = "HIERARCHICAL DATA FORMAT"
       PACKAGE_SIZE         = "17-40 MB"
       NUMBER_OF_MEDIA_TYPE = 4
```

```
     GROUP                  = MEDIA_TYPE
       TYPE_ID              = "FTP"
       NUMBER_OF_MEDIA_FORMAT = 2

      GROUP                = MEDIA_FORMAT
        FORMAT_ID          = "TAR COMPRESSED"
        APPROX_COST        = 2.00
      END_GROUP            = MEDIA_FORMAT

       GROUP                  = MEDIA_FORMAT
         FORMAT_ID            = "COMPRESSED"
         APPROX_COST          = 2.00

         GROUP = DATA_URL
           URL = "ftp://batcave.stx.com/pub/"
           URL_COMMENT = "Come get the data"
         END_GROUP = DATA_URL

        GROUP = DATA_URL
          URL = "ftp://eryq@harp.gsfc.nasa.gov/"
          URL_COMMENT = "Testing the March 5 updates"
        END_GROUP = DATA_URL

       END_GROUP              = MEDIA_FORMAT

     END_GROUP              = MEDIA_TYPE
  END_GROUP              = PROCESSING_OPTION

END_GROUP              = PACKAGE

...
```

10.2.4 Ingest of ODL

Very little had to be changed to the ingest of search results to support data set based ordering.

Since the "data set" packages look exactly like "granule" packages, no distinction need be made between the two. For inventory searches the IK layer handles the "data set" packages just like any other package and puts them in the gaea_tmp area (which is in the user's session directory). For directory results ingest, a call to IK_IngestPackage was added to imsclient since previously there was no package ingest for directory results.

A PackageIds hash was added to the IU::DataSet.pm object to hold the list package IDs for each data set.

10.2.5 Data set Listing Page:

Added a "add" data set to shopping cart button to both the directory and inventory result data set listing page.

DatasetList.pm now checks to see it a "data set" is orderable, i.e. does it have any "PackagesIds" associated with it.

10.2.6 Shopping Cart:

InvStore.pm now supports the idea that a data set can be added to the shopping cart as on orderable entity in their own right and not just as part of putting a granule into the shopping cart. The list of "orderable" data sets is stored in a hash called "JustDatasets". Data sets object can be in the cart either in association with a granule or as orderable objects. Care must be taken when putting granules and their associated data set information into the cart that the granule data set information doesn't over write that of the "orderable" data set with the same name.

Change summary and cost tallying is now based on "items" which are all the orderable items in the cart (sum of granules and orderable datasets).

Two lists are now presented in the shopping cart: a data set list of orderable data sets and a granule list.

Data set and granule sorts work on their respective lists.

Remove now removes either data set or granules from the cart. For "orderable" dataset "remove" just removes the data set name from the "JustDatasets" hash but doesn't remove it from the cart as the data set may be needed by a granule in the cart.

10.2.7 Choose Package:

A new submode was added to ModePKGSLCT.pm to do package selection for data sets. A new funtion in imsgdbdump (IU::Package) was added to get package objects by specifying the data set id.

10.2.8 Order:

Faked out Order::Item to create line item objects for data sets by pretending data setse were granules. This works because the packages and hence line items for both are identical. This does make the code fragule as it will break if the structure of "data set" vs "granule" packages changes in the future.

10.2.9 No support for e-mail ordering:

Currently there is no support data set - based e-mail ordering.

First of all, does it make sense to allow someone to order an entire data set without any package information?

If it is requested we'll added e-mail ordering then. A way of indicating whether or not e-mail ordering is available for an entire dataset will be needed. Probably will need a new field in the wwwvalids.gdb file.

10.3 Integrating GCMD query preview and the Web Gateway.

10.3.1 Overview:

A user starts with the GCMD Query Preview system. By choosing from attributes, geo and temporal information the user can in real time see the approximate number of data sets in GCMD which will satisfy her query. For VO IMS interoperability, there needs to be an option to narrow this list to just those in the VO IMS system. Once the user has created a list of data sets she will be given an option of turning that list of data sets into an inventory search query in the VO IMS Web Gateway.

10.3.2 GCMD to Web Gateway protocol:

The GCMD will link to the VO IMS Web Gateway on harp via a cgi post request (a get request is also okay but limited in the number of parameters which can be passed). A query from GCMD to the Web Gateway should be of the following form:

```
http://harp.gsfc.nasa.gov:80/ims-bin/dev/nph-ims.cgi?
&mode=GCMD
&md=amprtbcx
&md=msu23rdl93
&north=40
&south=-30
&east=-70
&west=30
```

- The tag mode=GCMD tells our system that the request is coming from GCMD.
- One or more master directory ids are passed to us, using md=amprtbcx. The master directory ids will be added to the VO valids as aliases to the corresponding data sets.
- An optional latitude/longitude bounding box can also be specified by including tags for north, south, east and west in degrees of lat/lon.

10.3.3 Details of Implementation in the Web Gateway:

A new mode call IW::modeGCMD.pm was created. This is the mode which is called by the GCMD system when they want to pass a query to us.

ModeGCMD does the following

- A query object is created.
- The search type is set to Inventory.
- A geo selection bounding box (range) object is created, using the north, south, east and west cgi parameter tags. The geo selection is defaulted to a global search if no geo tags are passed in.
- The master directory entry IDs are temporarly put into the query object as dataset ids. The master directory entry IDs will be stored in the VO valids as aliases to the data set ids, so we use imsdepval to convert the master directory entry IDs (aliases) to a list of VO data set ids. The "real" data set IDs are then stored back into the query object.
- A redirect is issued to mode=SRCHFORM, submode=GCMD to load the main search form.

The new submode GCMD was added to IW::modeASRCHFORM.pm. This submode just prints some informational text at the top of the page saying the query come from GCMD ...blah, blah, blah.... then calls send_page which loads the main search form using the query object we created in modeGCMD.

A new method was added IW::Depval.pm called get_aliases.

The nice thing about this implementation is that it can be a hidden feature until we decide to allow queries to come from GCMD.

10.4 G-ring Specification

10.4.1 Purpose:

New ODL type for describing geographic regions. Requested by LARC for TRMM support.

10.4.2 General Description:

A "G-ring" is composed of one or more regions with each region enclosed by a polygon of 3 or more sides (Outer-ring). Each region can have zero, one or more holes which are defined by a polygon of 3 or more sides (Inner-ring). The vertices of the polygons are given by lat/lon pairs listed in clockwise order (Clockwise is from the perspective of an observer looking at the globe from a distance, i.e. satellite view).

The lines joining the vertices of each ring are great circle arcs.

For backwards compatability with older versions of VO clients which do not support the new G_RING_LOC group either an enclosing RANGE_LOC or POLYGON_LOC must be given.

10.4.3 Examples:

Con Gifs:

<P> <img src="../pics/grings.gif" ALT="[B]" align=left> <img src="../pics/60.gif" ALT="[B]" align=left> <P>

```
GROUP    = G_RING_LOC

   # Approximate Center point of granule coverage.
   CENTROID_LAT
   CENTROID_LON
   # One or more outer_rings (regions).
   # First region.
   GROUP       = OUTER_RING
       # Region with 2 holes.
       # lat/lon pairs of outer ring.
       LATITUDE   = [Sequence real]
       LONGITUDE  = [Sequence real]
       # Zero or more inner rings (holes).
       # First hole.
       GROUP     = INNER_RING
           # lat/lon pairs of inner ring
           LATITUDE   = [Sequence real]
           LONGITUDE  = [Sequence real]
       END_GROUP = INNER_RING
       # Second hole.
       GROUP     = INNER_RING
           # lat/lon pairs of inner ring
           LATITUDE   = [Sequence real]
           LONGITUDE  = [Sequence real]
       END_GROUP = INNER_RING
   END_GROUP   = OUTER_RING

   # Second region.
   GROUP       = OUTER_RING
       # Region with no inner rings (holes).
       # lat/lon pair of outer ring
       LATITUDE   = [Sequence real]
       LONGITUDE  = [Sequence real]
```

```
      END_GROUP    = OUTER_RING

   END_GROUP = G_RING_LOC

  # Enclosing POLYGON_LOC for backward compatability

   GROUP = POLYGON_LOC

           LATITUDE  = [Sequence real]
           LONGITUDE = [Sequence real]

   END_GROUP = POLYGON_LOC
```

10.5 Proposal for Order Status Query page.

10.5.1 Purpose:

Allow a user at a later date to check on the status of an order submitted via the web gateway.

10.5.2 Scenerio:

A user places an order to one or more data centers. On the final order status page instruct the user to book mark the page. Then at some later date the user could load the bookmark. The order status query page would be invoked with all the necessary information needed to check on the order already "preloaded" and the query would be sent off to the data centers.

Alternately, the web gateway would have a Order Status Query page which would be accessable from the (main search form?). The user would start a web gateway session. Click on the "check the status on an order" button. The user would then have to type in the order id number and submit the query to the data centers.

10.5.3 Key features/issues:

- A copy of the user's order would have to be stored (for several months?) by the web gateway. We will need to save the REQUEST_ID (Id assigned by the client), web gateway order id (maybe the same as REQUEST_ID), DAAC_ORDER_ID (id given the daac) and and to which data centers the order was sent.
- There will need to be a naughty/nice list of data centers which support the order status query.

- How do we deal with e-mail ordering, send out e-mail?
- Need to figure out how to make the bookmark option work. How can we have the bookmark of a page invoke and different set of events at a later point in time?
- How much do we need to worry about privacy. In principle, if you could check on someoneelse's order if you have a valid order id.
- The IK software hasn't been updated yet to handle the new "PRODUCT_STATUS_REQUEST" and "PRODUCT_STATUS_INFO" messages. This will probably be done no earlier than the end of Aug 97. Will the requests be handled serially (as in orders) or in parallel (as in inventory searches)?
- Imsclient will need to be updated for the new message types.

10.6 Added security for the IMS/WWW gateway

As you know, anything under a user's session subdirectory "pub" (like generated map images), or elsewhere in the imswww-usr directory, is fair game for being loaded by a WWW browser. All other documents (including user preferences, search criteria, etc.) are off-limits via the browser.

10.6.1 Protecting files against local access

One thing that should be done is to ensure that all sensitive documents, even if they are off-limits to browsers, are chmod 0600. That will mean that generally, only CGI scripts will be able to read them (currently, users logging into an account the server machine might be able to read them as well).

10.6.2 Using hostname to verify each request

One simple way to make the **pirating** of sessions (stealing them from another browser if you know the sid) even more difficult is to get the host IP address of the requester (from the REMOTE_ADDR environment variable) and store it in the session state when the session is first created... say, in variable **OriginalRemoteAddr**.

We could use this in two places:

> On each new CGI request, the requesting host's IP would be compared to the `OriginalRemoteAddr`. A mismatch causes an immediate permission-denied error.

The **.htaccess** file in the "pub" directory could, instead of providing universal access, **provide access only to that user's host**.

This would likely prevent even the developers from "peeking" into the pub area of another user with a browser... but it would ensure that even a user's "public" files were only accessible from their machine.

10.6.3 Using cookies to verify each request

Another, possibly better way (and now fairly portable) is that *cookies* could be added to the security mechanism. It would work like this:

1. User creates a new session. At this time, a **session cookie** is generated along with the **sid**. **They are not to be identical in any way, and the cookie should be as random as possible.**
2. The cookie stored in the session state, in the variable **DoubleCheck**.
3. The cookie is reported back to the user along with the first response, via the HTTP header line:

   ```
   Set-Cookie: IMSWWW_DOUBLECHECK=(cookie);
               path=/;
               expires=(tomorrow GMT);
   ```

   (The IW::ReplyHead module will need to be extended to handle cookies.)
4. Each new request to the gateway host will have that cookie information in it, which will be accessible to CGI scripts via the HTTP_COOKIE environment variable. On each request, when the session is about to be resumed, a check is made: if the user's browser sent back any cookies at all, we make sure that one of them is an

   ```
   IMSWWW_DOUBLECHECK=(cookie)
   ```

   That matched the cookie stored in the session state. If not, we give a "permission denied" error.

10.7 How searches work

10.7.1 The players, from lowest to highest level

I realize this can get confusing because there are so many players. Let's see if I can summarize the most important ones:

imsclient

A low-level IK executable, written in C.

Queries DAACs, and writes their results in a C-oriented binary format (*dataset.gdb, granule.gdb*). Every 2 seconds, announces the status of each open connection in a single file (*client.stat*).

imsgdbdump (*low-level IK executable*)

The companion low-level IK executable, written in C.

Schooled in the translation of C-oriented binary to textual output. Reads the GDBs, outputting as plain text to the standard output.

imsrunclient

A Perl executable; supervisor of the imsclient/imsgdbdump pair.

Sends *imsclient* off to perform an activity. If the activity was a search, *imsgdbdump* is invoked to interpret what was gleaned from it, and then this script translates the textual output into Perl objects which are stored in a Perl database (*results.gdb*). The status is reported during this **post-processing** phase, in a separate file (*runclient.stat*).

IW::Conn

A Perl object.

Able to diagnose a connection status in the *client.stat*. Can also render itself as a single row of the comm status page.

IW::PostProcess

A Perl object.

Able to diagnose a post-processing status in the *runclient.stat*. Can also render itself as a single row of the comm status page.

IW::Client

A Perl object, containing one IW::Conn object for each connection, and one IW::PostProcess object for each post-processing activity.

The callable Perl interface to the *imsrunclient* executable. Dispatches *imsrunclient* to the task at hand, and checks in from time to time to see whether or not it's still alive, and also how it's doing (or how it did, since the status files (*client.stat, runclient.stat*) are persistent).

IW::CommStat

A Perl object, containing an IW::Client representing the current/last activity.

After IW::Client checks upon the health and well-being of *imsrunclient*, it is IW::CommStat that displays the statuses to the user.

IW::ModeSEARCH

A Perl "mode" handler.

Orders *IW::Client* to commence a search, and then checks in every 15 seconds... at each iteration, *IW::Client* checks if the search is still running, and then *IW::CommStat* is made to output the current page to the user.

The architecture thus is:

```
    IW::ModeSEARCH                     --- uses/spawns
           |                           ... reads/writes
     IW::CommStat                      [ ] executable
           |
      IW::Client ---------------------------.
       /      \                              \
      /        \                              \
  IW::Conn    IW::PostProcess     ..... [imsrunclient] .........
     ^              ^             :        /       \           :
     :              :             :       /         \          :
     :              :             :  [imsclient] [imsgdbdump]  :
client.stat    runclient.stat <...:     :    :      ^          :
     ^                                  :    :      :          :
     :..................................:    V      :          V
                                            dataset.gdb   results.gdb
                                            granule.gdb
```

10.7.2 Algorithm

Basically, you start by creating a new IW::Client object, using the "start" constructor. That constructor will first attempt to "attach" to any existing client, and kill it.

When the "start" constructor succeeds, a copy of the object is placed in the session state. From then on, anyone that wants to "attach" to it will get a full-fledged copy.

10.7.3 Issues

Regrettably, in order to know whether a connection *really* succeeded/failed, we can't just go by the status code: we have to also know what kind of activity was being done (inventory search, dataset search, etc), and then we must check the appropriate status value (number of datasets, number of granules).

In 1.5, a lot of this is scattered among IW::CommStat and IW::Client, and this has led to subtle errors in certain situations.

For 1.6, an attempt is being made to centralize the logic in IW::Client, and maybe even IW::Conn. The success/failure condition at higher levels would then be taken directly from the lower levels, without regard for the actual type of activity being undertaken.

10.7.4 Miscellany

The IMS/WWW standard abbreviations (as used by imsclient) and terms used to denote "activities" are:

```
Abbrev  Full
-----------------------------------------------------
DS      DIRECTORY_SEARCH
FB      FTP_BROWSE
IB      INTEGRATED_BROWSE
IS      INVENTORY_SEARCH
PR      PRODUCT_REQUEST
```

As of 1.6, the abbreviations are *only* used by imsclient; everyone else uses the full names.

10.8 How definition searches are done

10.8.1 Valids-generated query

This is the case where the system is displaying some valid (e.g., the parameter "RELATIVE HUMIDITY (AXP)") and is providing a "definition" button next to it. In this case, the HTTP request will be for a document named:

```
parameter/RELATIVE_HUMIDITY_AXP_.html
```

Notice that the valid name is "munged" by converting each non-alphanumeric-or-hyphen character to an underscore, and by then collapsing all strings of underscores to a single one. That is:

```
$item =~ s/[^A-Z0-9\-]/_/ig;
$item =~ s/_+/_/g;
```

The type is one of

```
campaign
dataset_id
data_center
parameter
sensor
source
processing_level
```

10.8.2 See also

See IW::ModeDEFN (where this documentation should ultimately be placed).

10.9 Event logging

Whenever you submit a form or follow a hyperlink that leads back into the IMS/WWW scripts, the event is logged as an

```
event=mode
```

Think of it as logging the start of a new CGI request (hmmm... maybe we should call it `event=startcgi` instead?).

The CGI tags `mode` and `submode` are logged with this event, to identify what action is about to be done. Also logged is the special `u` tag, also known as the Uniquifier (sounds like a Schwartzenegger movie, doesn't it?).

10.10 Why frames are difficult to implement for this interface

10.10.1 The "simple" scenario

One simple approach that was requested was this:

> Have two frames, with the blue button bar and title in a "title" frame, and the rest of the returned HTML page in a "body" frame. That way, users always see the title and button bar.

```
+-------------------------+
| [] Dataset list         |    Title frame
| [][]  [][][][]  [][][]  |    (fixed)
+-------------------------+
| 4. === === === ===      |
| 5. === === === ===      |    Body frame
| 6. === === === ===      |    (scrollable)
| 7. === === === ===      |
| 8. === === === ===      |
+-------------------------+
```

Sounds nice and simple, right? But consider what happens when you press a button in the blue button bar:

1. You want to generate a new page in the "body" frame, obviously.
2. You *have* to regenerate the "title" frame. Why?

First: the help button is context-sensitive

Second: if you don't, the hyperlinks will suffer cacheing effects; the reason we always generate a new **uniquifier** (u=XXXX) on each page is to *force* browsers to make a new request, regardless of whether or not they think they have the page cached. Remember that bug where you click on the "dataset listing" button, and it comes back immediately with an old dataset listing? That's what we're talking about.

With me so far? Now consider this, right out of the comp.infosystems.www.authoring.cgi FAQ:

```
   --------------------------------------------------------------
   Subject: 3.14 Can I write output to several frames at once?

   A single CGI script can only ever print to one frame.

   However, this limitation may be overcome by using more than one script.
   The first script (the URL of the "submit" button) prints a frameset,
   typically to a "_parent" or "_top" target.   The sources for one or
   more of the frames thus generated may also be CGI scripts, to which
   you can easily pass parameters (eg encoded in URLs with method GET).
*  This hack is definitely not recommended.   If you find yourself wanting
   to update several frames from a single user event, it probably means
   you should review the design of your application at a higher level.

   Warnings:
    1. Don't forget to escape your URLs.
*   2. This technique results in your server being hit by multiple
       concurrent CGI requests.   You'll need LOTS of memory, especially
       if you use a memory-hog like Perl.   It can be a good recipe
       for bringing a server to its knees.

   Javascript is often a valid alternative here, but note just how silly
   it can (and often does) look in a different browser.
   -------------------------------------------------------------
```

The issues here are:

- We *can't* have concurrent requests on the same SID. Part of our design, for safety, is to have each new request *kill* any currently-running request with the same SID.

  This design prevents a lot of weird failures that were occurring when simultaneous requests were competing for single-user resources (like the session.state database). Discarding this scheme would very likely invite disaster (it was not implemented on a whim).

  (Consider: what happens when a user presses a button, and then goes "oops" and quickly presses another one? We can't just wait for the first request to just complete: it might have been a search/browse request. Nor do we want the second request to fail.)

- Quite frankly: when a warning like this appears *in the CGI FAQ*, with phrases like "bringing a server to its knees", I listen to it. We'd effectively be doubling the amount of RAM and CPU time used for each request... all to get a simple frame!

- We could have *all requests* handled by a different CGI script, written in C... call it `imsframe.cgi`. It wouldn't use any of the resources, but it could do the following:

    > Return a FRAMESET document, with 2 frames. The first frame points back to `imsframe.cgi`, requesting a blue button bar. The second frame points to the Perl `nph-ims.cgi`, requesting the body.

  I dunno. Sounds like a looooooooooooooot of work for a feature that has no real scientific merit. And we'd still want to handle frameless browsers, wouldn't we? Or would we?

10.10.2 Other uses of frames

Another possibility is the use of frame in aiding with valids selection, or browsing lists (e.g., dataset list in the top frame, individual datasets in the bottom frame):

```
+-------------------------+
| [] Dataset list         |
| [][]  [][][][]  [][][]  |
|                         |
| 1. === === === ===      |
| 2. === === === ===      |
| 3. === === === ===      |
+-------------------------+
| [] Dataset attributes   |
| [][]  [][][][]  [][][]  |
|                         |
| ...                     |
+-------------------------+
```

This *might* work, since each frame need not worry about updating anything more than a single frame. Notice that we sacrifice display area for this effect.

10.11 How guide searches are done

10.11.1 User-provided text string

This is the case where the user is given a text field to type in their guide search query. In this case, the string

```
"relative humidity"
```

is turned into the WAIS query

```
relative+humidity
```

which will return documents containing *either* the word "relative" *or* the word "humidity".

10.11.2 Valids-generated query

This is the case where the system is displaying some valid (e.g., the parameter "RELATIVE HUMIDITY") and is providing a "detailed info" button next to it. In this case, the WAIS query generated will be:

```
Parameter=RELATIVE_HUMIDITY
```

In this case, the valid name is "munged" by simply converting each space to an underscore.

It may be noted that inside of the IC_Gd* code, it appears that a more-appropriate request would be:

```
Parameter=RELATIVE_HUMIDITY and Document_Type=Parameter
```

However, a sample of the guide documents returned by some of the servers shows that the `Document_Type=...` qualifier is not always present; this means that including that clause in the query would cause the query to return no results in some cases when it should.

10.11.3 See also

See IW::ModeGUIDE (where this documentation should ultimately be placed).

10.12 Keeping search results

10.12.1 Why we need to do it (the browse problem)

Although the **dataset.gdb** and **granule.gdb** files are not used directly by the Perl software, we need to keep *all* their contents around until the session expires. This is because of integrated browse requests... consider:

```
1.  You do a search A, with result set A'.
2.  You put a granule a' from A' into the shopping cart.
3.  You do a search B, with result set B', not containing a'.
4.  You go to the shopping cart, and do an integrated browse on a'.
```

This scenario will fail if `a'` is not kept in the `granule.gdb` file.

10.12.2 The failed attempt

Initially, the plan was to keep the `dataset.gdb` and `granule.gdb` files in place, and just keep adding new directory/inventory search results to them. Special routines in **imsclient** would keep track of what was actually ingested on *this* round.

When this was implemented, several problems were discovered:

- The IK layer seems somewhat biased against us resetting the "ngranule" counts on the dataset objects, and strange errors (involving reports of 0 granule counts when granules were in fact retrieved) were resulting. Since this resetting of counts is crucial to determining how many "new" granules there are, this is a major setback.
- One IK implementer observed that the "visible" flag in datasets – also used in "hiding" the previous search results – is perhaps not well supported throughout the system.
- It was noticed that the granule ingest code will actually delete granules retrieved from prior searches under certain circumstances. This could cause the next browse to break... negating our "solution".

10.12.3 Take two

A second attempt is being made to solve the problem. The idea is this: we keep a "running total" pair of `dataset.gdb/granule.gdb` files around permanently, in session directory `merged` as `merged_dataset.gdb/merged_granule.gdb`. Like this:

```
session-directory/
    |
```

```
|--- dataset.gdb
|--- granule.gdb
|
|--- merged/
:        |
         |-- merged_dataset.gdb
         |-- merged_granule.gdb
```

First we have some terms to define:

The last results

The results that came back from the last search.

The merged results

The current "running total" of all results, as kept in the `merged` directory.

The current results

Whatever's currently sitting in the session directory's dataset/granule GDBs. Note that this might be either the **last results** or the **merged results**, depending on when you look there.

The approach is this:

- **Before each client search activity,** we clear out the current results from the session directory.
- **After each client search activity,** we merge the results of the last search with the merged results.

  Do this by copying every `key=value` pair in the GDB files of the **last results** into the corresponding GDB files of the **merged results**. This will overwrite an old record if it's already there, and simply add a new one if it isn't.

  Once the merging succeeds, the new **merged results** are copied back up into the session directory, becoming the **current results**. This is so that we always have a `dataset.gdb/granule.gdb` pair on hand so that **imsgdbdump** may be called at any time.

  > ***Note:*** We do this stage in as atomic a fashion as possible; it's vital that we always have a good copy of the merged results! Rememeber, it's possible for a "kill" to be delivered to "imsrunclient" at any time; we don't want the merged results to be in a bad state!

- **Before each client non-search activity** (e.g., browse, order), we copy the **merged results** into the session directory, and they now become the **current results**. This allows integrated browses to succeed.

Note: We have to do this! If the last search was interrupted by a "kill" (if it couldn't be stopped gracefully), it probably happened before the merged files were copied back up! Also, remember that IW::Client cleans out the "wreckage" from a failed search, and this includes the GDBs.

Warning: Probably, any activity that uses **imsgdbdump** should look to copy the **merged results** up to the session directory first.
For example, it's possible that the user could start a search (clearing the sesison directory), kill it, and then try to do a package selection. Having **imsgdbdump** (and maybe **imsclient**) be savvy about the merged files will at least ensure that they can "set themselves up" before attempting any operation that requires the GDB files.

10.13 Uniquifier logging

The uniquifier is now logged with the MODE event. Log entries for MODE events now look like:

```
event=mode
sessid=847928329-24446
sid=v0ieww00
event_date=19961113
event_time=184059
addr=fake.name@batcave.stx.com
mode=DSETLIST
submode=
ufrom=928425
upage=928500
```

mode, submode

The mode and submode in the current CGI request we have received... and therefore, the one we are *about* to handle (since we log all mode events *before* actually handling the mode.

ufrom

The `u=` tag from the current CGI request. It identifies what page the user came **from** (because that page was the one that had that `u=` tag embedded in its hyperlinks... see?).

upage

The `u=` tag that will be placed on THIS PAGE. It identifies the page **that we are about to generate**. We will use `u=UPAGE` in all hyperlinks on this page.

So from the above example, to find out what page the user came from, we look at the `ufrom`... it's `928425`, so we'd just scan upwards until we found an event=mode where `upage=928425`.

We can now determine whether or not (and when) a user hit the back button... basically, if we ever see ufrom "jump backwards" to anything but the upage of the very last mode event, we know that the user "went back" to that page. Here's what it would look like (simplified):

```
event=mode
mode=DSETLIST
ufrom=11
upage=20        # the "id" of this DSETLIST page is 20

event=mode
ufrom=20        # ok... we came from the last generated page
upage=37

event=mode
ufrom=37        # ok... we came from the last generated page
upage=52


event=omde
ufrom=20        # AHA!  They went "back" to the DSETLIST!
upage=66
```

Uniquifiers are basically time %1000000, so unless two screens were generated within the same second (odds?) they can't have the same uniqifier. Note that, for obvious reasons, we don't take the last u= tag we got and add 1. Although we COULD do that with the session state... hmmmmmmmmmmm...