Summary of Bugs Database

Jeremy Kepner

September, 1993

1. Introduction

1.1 Document Purpose

This document is meant to provide a summary of the development as well as an outline of the use of the routines written to create the current version of the bugs database.

1.2 Brief Project History

Development of the Bugs database began in May of 1992. Initially, a very simple database model was used. By July the decision to use Sybperl as the main development environment had been made. In October the routines for parsing bug reports were complete, which allowed further experimentation in the model design. The main points in the current design were in place by January, 1993. At this time it became apparent that a simple interface would be necessary. The current Gopher interface was implemented in March. The database has been functioning automatically since then with only minor changes.

1.3 Section Description

Section 2 Project Goals briefly summarize the past and current goals of the bugs database. Section 3 Code Structure gives an outline of the structure and organization of the code. Section 4 Life Cycle gives a more detailed description of the specific routines of the database by presenting the complete life-cycle of a bug report. Section 5 is a set appendices giving attributes of the database.

2. Project Goals

2.1 Original Goals

Originally the Sybperl implementation of the bugs database was to completely replace the old bug tracking system made up of Bourne shell scripts. In addition to storage, the new system would handle creating and editing bugs, as well as provide more diverse access to the data.

2.2 Current Goals

With the impending arrival of Scopus, the role of the Sybperl database has changed. Currently, it assists the Bourne shell database with storage and retrieval. The combination of the Gopher interface with Sybperl has made retrieval of bugs both faster and easier. Thus, for now, the Sybperl database acts as a bridge between the old Bourne shell system and new Scopus system and should ease the transition between the two. Furthermore, the Sybperl database serves as an example for other database problems. The combination of Perl, Sybase, and Gopher provides one of the fastest, most easily maintainable solutions to database management.

3. Code Structure

3.1 Overview

The general approach to the development of the Sybperl bugs database has been to use Perl wherever possible. Perl provides greater flexibility and power than SQL. It is much easier to create and edit Perl scripts than to create and edit corresponding SQL procedures. The advantages of using Perl are even more apparent when, as in this case, a significant amount of information is being read in from text files. Thus, only in situations where speed is crucial have SQL routines been used.

3.2 Perl Environment

All the code relating to the database can be found in the directory: `/home/sunspark/jeremy/bugdb/`. The main sub-directories are:

`cron/ insert/ old/ output/ priority/ reset/ sybperl/`

One line descriptions of most of the files in `bugdb/` can be found in Appendix A. The source code for Sybperl can be found in the `sybperl/` sub-directory. From the users perspective, Sybperl adds two functions to Perl. First, the function "dblogin", which handles logging into Sybase. Second, the function "sql" which sends a string to Sybase (assumed to be SQL commands) and returns the results in the form of a Perl array.

The Sybase structure of the database is stored in the `reset/` directory. Calling the Perl routine `reset_db.pl` will execute a complete reset of the database. Each table, view, index, trigger and procedure is deleted and then recreated and initial values are inserted. Descriptions of these initial values can be found in Appendix C.

The `insert/` directory contains the Perl scripts that read in bug reports, decides which pieces go where, and creates the appropriate SQL statements to do so. The primary routine is `insert_bug.pl`, which in turn calls `insert_bug.plsub` or `update_bug.plsub`. `insert_bug.pl` is designed to insert bugs in large numbers. Normally, a file containing a list of bugs to be inserted is provided as an argument. `insert_bug.pl` examines each bug and decides which subroutine to call depending upon whether or not the bug already exists in the database.

The `priority/` directory handles the prioritizing and categorizing of bugs and contains a complete facility for doing this task interactively. Currently, much of this code is unused. The two most important routines are `allbugs.pl` and `update_priorities.pl`. `allbugs.pl` examines the entire bug directory and

prints out lists of bugs and their status in a form suitable for `insert_bug.pl`. In addition, it can check the modification date of a bug. `update_priorities.pl` takes a list of bugs and their priorities and categories (e.g. `~don/bugs/PRIORITY`) and updates the appropriate tables in the database.

An important aspect of the Sybperl bugs database is that it updates itself automatically. The `cron/` directory contains a program called `end-of-day` which calls `end-of-day.pl`. `end-of-day` is run every weekday at 8:00 p.m. This program hops around the various directories of bugdb doing what needs to be done to make the database current. Primarily, it calls `allbugs.pl` with a time-stamp file to generate a list of bugs that have been created/modified since the time-stamp file was last touched. The list of bugs is then passed to `insert_bug.pl`.

3.3 Sybase Environment

Sybase organizes data using a relational model. The fields used in the bug reports have been broken down so that each field becomes a distinct table (see Appendix B). In this distributed fashion a bug becomes simply a list of pointers the various rows in the tables. Currently, there are five "core" tables: products, releases, bugs, details, and people. These represent the five most important concepts in the database. At the root is the products table, e.g., DataViews, DynaGraphX, XDesigner, etc... There can be many products. Adding a new product is as simple as adding a new row to the products table. Each product can have many releases, but each release must be connected to a particular product. Similarly, a release can have many bugs, but a bug must point to a particular release. Details contain the actual text of the bug, e.g. problem, detailed description, work around, solution, etc. ... The relationship of details to bugs is the same as bugs to releases and releases to products—many-to-one. The situation with respect to people is

reversed. Each person could have many details to their credit, but a particular detail can be authored by only one person.

Additional descriptive information is assigned to a bug through the use of various "side" tables (see Appendix B). The side tables consist of lists of possible bug attributes: priorities (high, medium, low, ...); statuses (open, closed, not a bug, ...); types (problem, detailed description, work around, solution, ...); keywords (x-windows, VUobject, memory, ...). In most cases, a particular attribute will have many associated bugs and a bug can have many attributes. For example, each category, by definition should have more than one bug, but each bug should not be limited to being a member of only one category. These many-to-many relationships are resolved with "link" tables. Link tables are very simple, consisting only of pointers to the tables they are linking together.

3.4 Gopher Environment

The Perl routines for handling gopher are found in the `output/` directory. There are two sets of gopher routines. Those found in `output/gopher/` and those in `output/gopher.etc/` that correspond to different menus in Gopher. Currently, all the Gopher interaction with the database is handled by the scripts in `gopher.etc/`, with the exception of those items found under the "Test New Features" menu, which are handled by the scripts in `gopher/`. The `gopher.etc/` scripts are executed from the `~don/gopher-data/bug-info/bin/` directory where copies of these programs reside. The `gopher/` scripts are executed via "wrapper" scripts found in `~don/gopher-data/bug-info/test/bin/`. The purpose of `gopher/` was originally to replace `gopher.etc/` with an entirely new set of scripts that took advantage of the library (`lib/`) that was written to make database programming much easier. The `gopher/` scripts are much simpler, but now only serve as example to the usage of `lib/`.

3.5 Library Environment

After writing the `gopher.etc/` scripts it became apparent that further development in this fashion would only result in larger files with a great deal of duplication. The routines in `lib/` were written to eliminate further duplication of effort and to provide a standard structure for future development. The `lib/` directory consists of:

```
bugdb.pllib  assoc/  commands/  io/  list/ sybperl/
```

The library is invoked with the Perl statement:

```
require "${BUGDBLIB}/bugdb.pllib";
```

which compiles and makes available all the routines in the library. The sub-directories contain the library functions, broken down by category. For a more complete description of the library, please Appendix D.

4. Life Cycle

4.1 Database Creation

Instructions for restarting the database from scratch can be found in: `bugdb/reset/reset_instructions`. This section only describes what takes place in `reset_db.pl`, which begins by logging into the database (it assumes the database "bugs" exists) and proceeds to call `&reset_table`, `&reset_view`, `&reset_procedure`, `&reset_trigger`, and `&reset_index` for each object in the database. These procedures work in pretty much the same way. Each object is first deleted from the database and then the appropriate sub-directory is searched for a corresponding file which contains the SQL code to recreate the object. For example, `&reset_table("bugs")` reads in a file called `reset/table/bugs`, which contains the SQL statements for creating the bugs table. After reading in the SQL file, the commands are sent to the database.

4.2 Bug insertion

As mentioned earlier, the principle programs for inserting bugs are `insert_bug.pl` and the subroutines `insert_bug.plsub` and `update_bug.plsub`. The most common method for inserting a bug is:

```
insert_bug.pl -q -f bugs_file
```

The -q option stands for "query" and tells `insert_bug.pl` that it should first query the database to see if this bug already exists. `bugs_file` is typically output from the `priorities/allbugs.pl` script. The bug report is then read in by either `insert_bug.plsub` or `update_bug.plsub` and each field is put into a Perl variable. If the contents of the Perl variables are not empty, then the appropriate SQL statement for adding the information to the database is constructed and executed.

4.3 Categories and Priorities

The task of prioritizing and categorizing begins with the updating of the `~don/bugs/PRIORITY` file. When the priority file is changed, it is read in by `update_priorities.pl`, which changes to the single character abbreviations to the priority and category names found in the database. In the case of priorities, the prior_id field in the bugs table is updated with the corresponding value from the priorities table. The updating of categories begins by deleting all the catlinks for the bug and then recreating them for each category to which the bug has been assigned.

4.4 Status Updates

The Bourne shell bugs system represents a change of status by copying a bug to a new directory. Changing directories causes the file itself to be modified and is noted by the `allbugs.pl` routine. Because the bug report itself might have changed, updating a bug consists of first deleting and reinserting all the details of

the bug report. An additional output of the `allbugs.pl` script is the directories of the bug (i.e. the status). If the status of the bug has changed, then the appropriate statlink is created.

4.5 The Cron Job

The processes outlined in sections 4.2, 4.3 and 4.4 are carried out each weekday by the cron job found in `cron/end-of-day`. The reader is encouraged to examine `end-of-day` and `end-of-day.pl` as this is the best way to understand the daily cycle of the database.

Appendices

Appendix A: Descriptions of bugdb/

Appendix B: Database Structure

Appendix C: Tables, Initial Values, Views, Indexes, Triggers, and Procedures

Appendix D: bugdb.pllib

Appendix A: Description of contents of bugdb/

Lines	*Filename*	*Description*
	cron/	Daily scripts and time stamp files.
	insert/	Perl scripts for inserting bugs.
	old/	Old versions of perl scripts.
	output/	Various tools for accesing the database.
	priority/	Utilities for prioritizing bugs.
	reset/	Scripts and SQL files for rebuilding/resetting database.
	sybperl/	Source and executable of sybperl.
	cron/	
0	do_these	Bugs to insert today.
6	end-of-day*	Wrapper for end-of-day.pl.
116	end-of-day.pl*	Inserts todays bugs and update priorities.
0	last_time	Time stamp file indicating last time bugs were inserted.
	insert/	
156	insert_bug.pl*	Reads in a list of bugs and inserts/updates.
324	insert_bug.plsub	Reads one bug report, parses and inserts.
51	insert_detail.plsub	Inserts a detail of a bug.
72	insert_keywords.plsub	Creates keywords for a detail.
35	nultst.plsub	Checks to see if a string has any data.
	spell/	Unused routines for checking spelling of keywords.
50	sql_in_list.plsub	Formats a list of items in a manner suitable for SQL.
	test_data/	Lists of bugs for test inserting.
316	update_bug.plsub	Reads one bug report, parses and updates.
	insert/spell/	
36	ispell.pl*	Runs ispell spelling program.
54	open2.pl	Program from the Perl library for handling processes.
14	rootify*	Creates a pipe to ispell for getting root of keywords.
	insert/test_data/	
49	testbugs	A sample of 49 DataViews bugs (7.0 - c9.1).
21	testbugs.small	A sample of 21 DataViews bugs (7.0 - c9.1).
3	testbugs.tiny	A sample of DataViews bugs (9.0 - b9.0).
	output/	
	gopher/	Gopher scripts based on library.
	gopher.etc/	Gopher scripts (mirrored in ~don/gopher-data/bug-info/bin).
	lib/	Library of perl routines for accessing database.
	output/gopher/	
	commands/	Scripts that display information in gopher.
	wrappers/	Menu names (mirrored on ~don/gopher-data/bug-info/test).
	output/gopher/comm...	
4	bug_report*	Bourne Shell wrapper script for bug_report.pl.
51	bug_report.pl*	Retrieves, formats for gopher, and prints a bug.
4	bug_search*	Sets options and stderr for bug_search.pl.
94	bug_search.pl*	Executes a search, and creates a gopher menu.
4	build_search*	Sets options and stderr for build_search.pl.
320	build_search.pl*	Displays and executes menus for creating a database search.

127	cached_searches.pl*	Menus for predefined gopher searches.
4	keyword_info*	Sets options and stderr for keyword_info.pl.
111	keyword_info.pl*	Retrieves and displays info on selected keywords.
4	save_bugs*	Sets options and stderr for save_bugs.pl.
131	save_bugs.pl*	Saves output from bug_search in file.
4	standard_search*	Sets options and stderr for standard_search.pl.
127	standard_search.pl*	Displays and executes predefined searches.

	output/gopher/wrappers/	
3	bug-lists-by-priority...	Calls standard_search.
3	bug-menu*	Calls build_search.
7	lookup-by-bug-numb*	Calls bug_report.
3	search-by-keyword*	Calls keyword_search.
3	search-by-priority*	Calls priority_search.
	output/gopher.etc/	
98	README	Description of this directory.
546	bug_menu.pl*	Display and executes menus for creating searches.
263	bug_report.pl*	Displays a bug report for a given bug.
378	bug_search.pl*	Executes a bug search.
90	keyword_OR_search.pl*	Executes seperate searches on each of several keywords.
44	keyword_report.pl	Gets info on a keyword.
192	keyword_search.pl*	Executes a single search on a list of keywords.
122	priority_search.pl*	Display and execute priority searches.
50	sql_in_list.plsub	Procedure for formatting a list of words.
	output/lib/	
520	bugdb.pllib	Loads sybperllibrary from sub-directories.
	io/	High level library functions.
	assoc/	Utilities for processing asssociative arrays.
	list/	Utilities for processing lists.
	sybperl/	Sybperl scritps for logging into and accessing Sybase.
	output/lib/assoc/	
31	abbr2full.plsub	Converts abbreviated keys to full names.
42	argv2assoc.plsub	Converts @ARGV to associative array.
27	assoc2argv.plsub	Converts an associative array to an @ARGV.
28	assoc_hmt2hmt.plsub	Puts a new head-middle-tail around each item.
32	assoc_hmt2sep.plsub	Removes head-middle-tail and puts in control-a
32	assoc_sep2hmt.plsub	Removes control-a and puts in head-middle-tail.
29	full2abbr.plsub	Converts full names to abbreviations.
71	full_abbr.plsub	Lookup tables for abbr2full and full2abbr.
	output/lib/io/	
128	bug_report.plsub	For a given bug, returns bug report in an associative array.
277	bug_search.plsub	Takes a search and returns a list of bugs.
81	bug_table.plsub	Queries a table for a list of allowable values.
75	bugs_summary.plsub	Returns a summary on a given bug.
54	exec_sql.plsub	Executes an SQL statement as owner (allows edit).
31	gopher_menu.plsub	Take a list of commands and print it as a gopher menu.
74	keyword_info.plsub	Get information on keywords.
133	print_report.plsub	Take a report assoic array and print it out.
49	run_sql.plsub	Executes an SQL statement as browser.
110	status_search.plsub	Search for bugs based on their status.
	output/lib/list/	
41	hmt2hmt.plsub	head-middle-tail string -> head-middle-tail string.
29	hmt2list.plsub	head-middle-tail string -> perl list.
38	hmt2sep.plsub	head-middle-tail string -> control-a seperated string.
30	list2hmt.plsub	perl list -> head-middle-tail string.
32	list2sep.plsub	perl list -> control-a seperated string.
29	list2sepb.plsub	perl list -> control-b seperated string.
42	quotify.plsub	take a perl list around each item that has spaces in it.

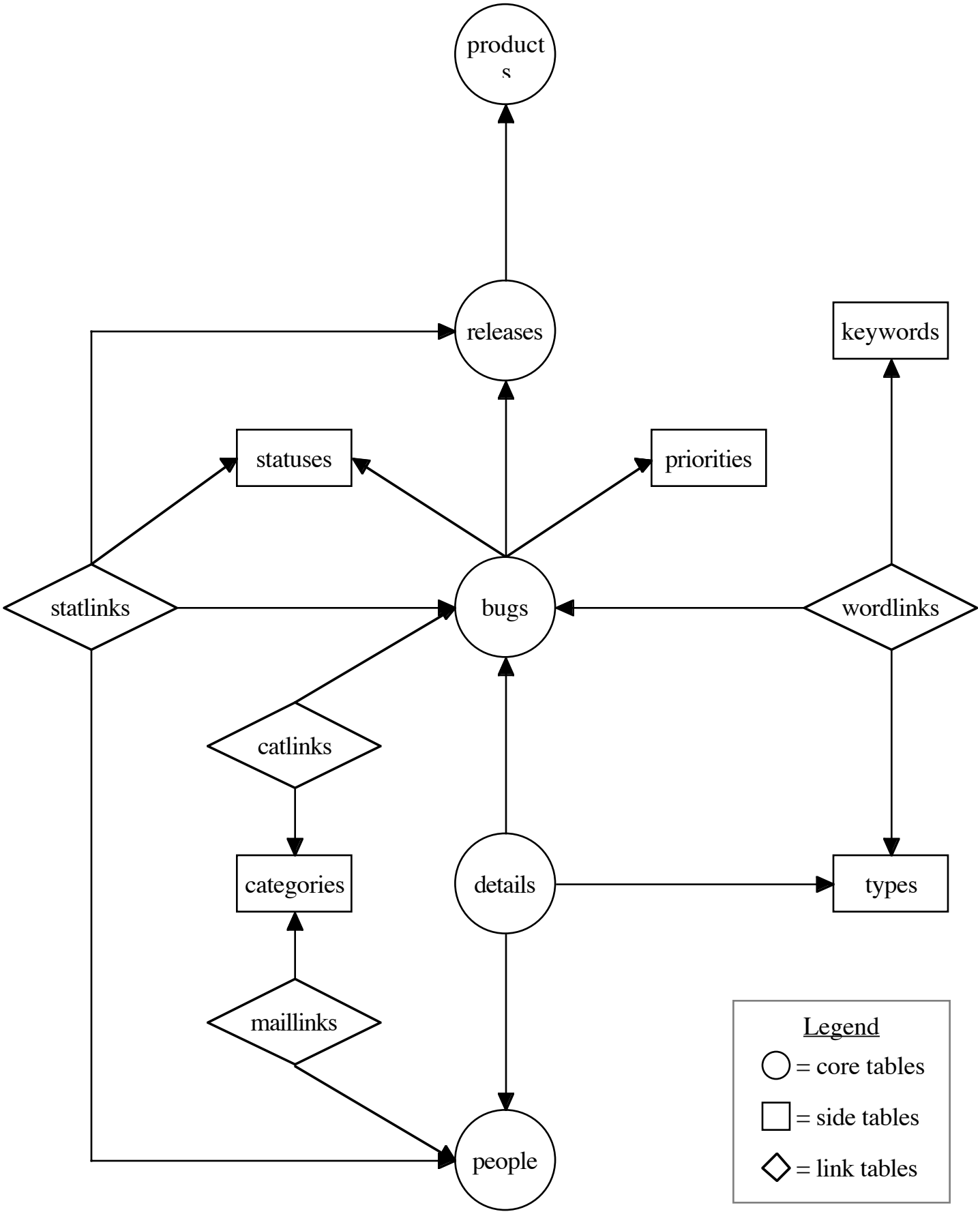
31	sep2hmt.plsub	control-a string -> head-middle-tail string.
24	sep2list.plsub	control-a string -> perl list.
21	sepb2list.plsub	control-b string -> perl list.
	output/lib/sybperl/	
87	sql.pl	Executes an sql statement and formats the results.
135	sybdb.ph	Sybperl function mapping?
17	sybperl.pl	Sets Sybperl environment variables.

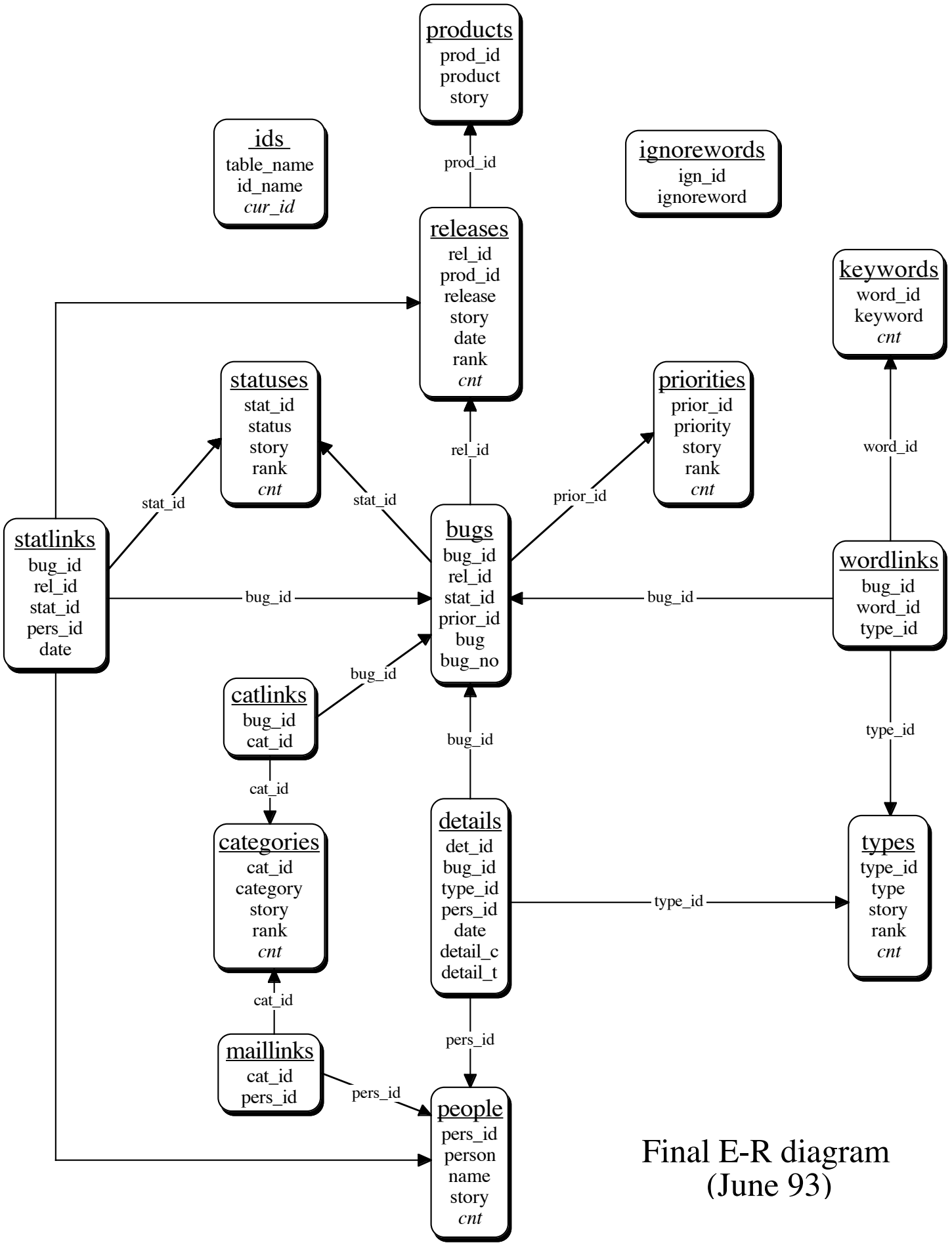
	priority/	
30	CATEGORY	List of bug category and priority abbreviations.
23	CATEGORY.old	Original bug category and priority abbreviations.
23	DGX_KEY	DynaGraphX categories and priorities abbreviations.
105	DGX_PRIORITY	List of DynabGraphX bugs with a priority or category.
2	DGX_PRIORITY.old	Last list of DynabGraphX bugs with a priority or category.
30	DV_KEY	DataViews categories and priorities abbreviations.
23	DV_KEY.old	Original DataViews categories and priorities abbreviations.
23	EO_KEY	EO categories and priorities abbreviations.
0	EO_PRIORITY	List of EO bugs with a priority or category.
0	EO_PRIORITY.old	Previous list of EO bugs with a priority or category.
23	XD_KEY	XDesiginer categories and priorities abbreviations.
0	XD_PRIORITY	List of XDesigner bugs with a priority or category.
0	XD_PRIORITY.old	Previous list of XDesigner bugs with a priority or category.
196	allbugs.pl*	Generates list of bugs with status, category, priority, etc.
6	prioritize*	Wrapper for prioritize.pl
180	prioritize.pl*	Takes a list of bugs and intereactively cretes priority file.
3	prioritize_dgx*	Calls prioritize for DynaGraphX.
3	prioritize_dv*	Calls prioritize for DataViews.
3	prioritize_eo*	Calls prioritize for EO.
3	prioritize_xd*	Calls prioritize for XDesigner.
131	update_priorities.pl*	Updates priorites in the database.
	reset/	
	indexes/	Directory of SQL statements to create indexes.
	procedures/	Directory of SQL statements for creating procedures.
125	reset_db.pl*	Resets the entire database.
44	reset_index.plsub	Perl function for resetting indexes.
418	reset_initial.plsub	Resets initial values of each table.
115	reset_instructions	Procedure for resetting database from scratch.
125	reset_one.pl*	Copy of reset_db.pl that resets only one item.
44	reset_procedure.plsub	Perl function for resetting procedures.
57	reset_table.plsub	Perl function for resetting tables.
334	reset_trigger.plsub	Perl function for resetting triggers.
44	reset_view.plsub	Perl function for resetting view.
	tables/	Directory of SQL statements for creating tables.
	triggers/	Directory of SQL statements for creating triggers.
	views/	Directory of SQL statements for creating views.
	reset/indexes/	
3	bugs_bug_ind	Index on bug field in bugs table.
3	bugs_ind	Index on bug_id field in bugs table.
3	bugs_ord_ind	Index on bug_no and rel_id in bugs table (for sorting).
3	details_ind	Index on det_id in details table.
3	keywords_ind	Index on key_id in keywords table.
3	statlinks_ind	Index on bug_id in statlinks table.
3	statlinks_p_ind	Index on pers_id in statlinks table.
3	wordlinks_ind	Index on bug_id, type_id, word_id in wordlinks table.
	reset/procedures/	
43	bugs_ins	Inserts a bug into bugs.
43	bugs_upd	Updates a bug in bugs table.
51	statlinks_ins	Creates a statlink.

60	wordlinks_ins	Creates a wordlink.

	reset/tables/	
7	bugs	Core bug table.
6	categories	List ofdifferent bug categories.
3	catlinks	Links bugs to categories.
8	details	Stores bug details.
4	ids	Stores current id for each table.
3	ignorewords	List of words to ignore.
4	keywords	List of keywords.
3	maillinks	Links categories to people.
6	people	List of people.
6	priorities	List of bug priorities.
4	products	List of products.
8	releases	List of product releases.
6	statlinks	Links statuses with bugs, releases, and people.
6	statuses	List of statuses.
6	types	List of detail types.
4	wordlinks	Links words with bugs.
	reset/triggers/	
24	bugs_itrg	Bugs insert trigger.
15	catlinks_dtrg	Catlinks delete trigger.
19	catlinks_itrg	Catlinks insert trigger.
15	details_dtrg	Details delete trigger.
21	details_itrg	Details insert trigger.
44	statlinks_itrg	Statlinks insert trigger.
14	wordlinks_dtrg	Wordlinks delete trigger.
18	wordlinks_itrg	Wordlinks insert trigger.
	reset/views/	
8	bugs_view	View bugs with lookup on all dependant tables.
5	catlinks_view	View catlinks with lookup on bugs and categories.
7	details_view	View details with lookup on bugs, types and people.
5	maillinks_view	View maillinks with lookup on categories and people.
9	statlinks_view	View statlinks with lookup on all dependent tables.
6	wordlinks_view	View wordlinks with lookup on bugs, keywords and types.

Appendix B - Database Structure





Final E-R diagram
(June 93)

Appendix C:

Tables

Initial Values

Views

Indexes

Triggers

Procedures

Tables

Column Name	*Data Type*	*null/ not null*
bugs		
bug_id	id	not null
rel_id	id	not null
stat_id	int	null
prior_id	int	null
bug	varchar(64)	not null
bug_no	int	null
categories		
cat_id	id	not null
category	varchar(64)	not null
story	varchar(255)	null
rank	float	not null
cnt	int	not null
catlinks		
bug_id	id	not null
cat_id	id	not null
details		
det_id	id	not null
bug_id	id	not null
type_id	id	not null
pers_id	id	not null
date	datetime	null
detail_c	varchar(255)	null
detail_t	text	null
ids		
table_name	varchar(64)	not null
id_name	varchar(64)	not null
cur_id	id	not null

ignorewords

ign_id	id	not null
ignoreword	varchar(64))	not null

Tables

Column Name	*Data Type*	*null/ not null*
keywords		
word_id	id	not null
keyword	varchar(64)	not null
cnt	int	not null
maillinks		
cat_id	id	not null
pers_id	id	not null
people		
pers_id	id	not null
person	varchar(64)	not null
name	varchar(64)	null
story	varchar(255)	null
cnt	int	not null
priorities		
prior_id	id	not null
priority	varchar(16)	not null
story	varchar(255)	null
rank	float	not null
cnt	int	not null
products		
prod_id	id	not null
product	varchar(64)	not null
story	varchar(255)	null
releases		
rel_id	id	not null
prod_id	id	not null
release	varchar(64)	null
story	varchar(255)	null

date	datetime	null
rank	float	null
cnt	int	null

Tables

Column Name	*Data Type*	*null/ not null*
statlinks		
bug_id	id	not null
rel_id	id	not null
stat_id	id	not null
pers_id	id	not null
date	datetime	not null
statuses		
stat_id	id	not null
status	varchar(64)	not null
story	varchar(255)	null
rank	float	not null
cnt	int	not null
types		
type_id	id	not null
type	varchar(32)*	not null
story	varchar(255)*	null
rank	float	not null
cnt	int	not null
wordlinks		
bug_id	id	not null
word_id	id	not null
type_id	id	not null

Initial Values

ids

table_name	*id_name*	*cur_id*
bugs	bug_id	1
categories	cat_id	1
catlinks	catlink_cnt	1
details	det_id	1
ignorewords	ign_id	1
keywords	word_id	1
maillinks	maillink_cnt	1
people	pers_id	1
priorities	prior_id	1
products	prod_id	1
releases	rel_id	1
statlinks	statlink_cnt	1
statuses	stat_id	1
types	type_id	1
wordlinks	wordlink_cnt	1

categories

cat_id	*category*
1	ada
2	documentaion
3	example
4	demo
5	graph
6	image
7	known
8	orphan
9	subdrawing...
10	x-window
11	fix-it
12	driver
13	system
14	vms

15 verify

priorities

prod_id	*priority*	*rank*	*story*
1	high	1.0	highest priority, fix asap
2	medium	2.0	would like to fix
3	low	3.0	fix when most convenient
4	dygx	4.0	DynaGraphX bug, no priority
5	port	5.0	proting bug, no priority
6	pending	6.0	waiting to be prioritized
7	none	100.0	no priority at this time

products

prod_id	*product*
1	DataViews
2	DynaGraphX
3	XDesigner
4	EO
5	GraphWidgets

statuses

stat_id	*status*	*rank*	*story*
1	bug	1.0	a bug in this release...
2	not bug	2.0	not a bug in this release...
3	fixed	3.0	fixed in this release...
4	not fixed	4.0	not fixed in this release...
5	verified	5.0	verified as a bug...
6	none	100.0	no status at this time

types

type_id	*type*	*rank*	*story*
1	synopsis	1.0	One Line Summary
2	machine	2.0	Machine and/or OS
3	prod	2.5	Product
4	problem	3.0	Problem
5	description	4.0	Detailed Description
6	test case	5.0	Test Case Instructions
7	workaround	6.0	Workaround

8	solution	7.0	Solution
9	modules...	8.0	Modules Affected
10	not fixed	9.0	Why bug is NOT fixed...
11	not a bug	10.0	Why bug is NOT ... a bug

releases

rel_id	*prod_id*	*release*
1	1	6.0
2	1	6.01
3	1	7.0a
4	1	7.0b
5	1	7.0c
6	1	7.0d
7	1	8.0
8	1	a9.0
9	1	b9.0
10	1	9.0
11	1	a9.1
12	1	b9.1
13	1	c9.1
14	1	9.1
15	1	9.1a
16	1	a9.2
17	1	b9.2
18	1	d9.2
19	1	9.2
20	2	dgx0.9
21	2	dgx1.0
22	2	dgx1.1
23	3	xd1.1
24	3	xd1.2
25	3	xd1.3
26	3	xd2.0
27	3	xd2.0f
28	3	xd2.0g
29	3	xd3.0a
30	4	bEO2.0
31	4	EO2.0
32	4	EO3.0
33	1	9.5
34	5	bgw2.0

Views

Columns	*Criteria*
bugs_view	
bugs.[bug,bug_id, rel_id,bug_no]	—
products.product	products.prod_id = releases.prod_id
releases.release	bugs.rel_id = releases.rel_id
statuses.status	bugs.stat_id = statuses.stat_id
priority.priority	bugs.prior_id = priorities.prior_id
catlinks_view	
catlinks.bug_id	—
bugs.bug	catlinks.bug_id = bugs.bug_id
categories.category	catlinks.cat_id = categories.cat_id
details_view	
details.[det_id,date, detail_c,detail_t]	—
bugs.[bug_id,bug]	details.bug_id = bugs.bug_id
types.type	details.type_id = types.type_id
people.person	details.pers_id = people.pers_id
maillinks_view	
categories.category	maillinks.cat_id = categories.cat_id
people.person	maillinks.pers_id = people.pers_id
statlinks_view	
statlinks.date	—
products.product	products.prod_id = releases.prod_id
bugs.bug	stalinks.bug_id = bugs.bug_id
releases.release	stalinks.rel_id = releases.rel_id
statuses.status	stalinks.stat_id = statuses.stat_id
people.person	statlinks.pers_id = people.pers_id
wordlinks_view	
wordlinks.bug_id	—
bugs.bug	wordlinks.bug_id = bugs.bug_id
keywords.keyword	wordlinks.word_id = keywords.word_id
types.type	wordlinks.type_id = types.type_id

Indexes

Name	*Table*	*Column(s)*
bugs_bug_ind	bugs	bug
bugs_ind	bugs	bug_id
bugs_ord_ind	bugs	rel_id,bug_no
details_ind	details	bug_id
keywords_ind	keywords	keyword
statlinks_ind	statlinks	bug_id
statlinks_p_ind	statlinks	pers_id
wordlinks_ind*	wordlinks	word_id,bug_id,type_id

*unique index

Triggers

Table	*Name*	*Execute on*
Description		
bugs	bugs_itrg	insert
decrement ids.cur_id, increment releases.cnt		
catlinks	catlinks_dtrg	delete
decrement categories.cnt		
catlinks	catlinks_itrg	insert
increment ids.cur_id, increment categories.cnt		
details	details_dtrg	delete
decrement types.cnt		
details	details_itrg	insert
increment types.cnt, increment ids.cur_id		
statlinks	statlinks_itrg	insert
increment ids.cur_id, increment statuses.cnt, decrement statuses.cnt, change status of bug, increment people.cnt		
wordlinks	wordlinks_itrg	insert
increment ids.cur_id, increment keywords.cnt		
default	defualt_itrg	insert
increment ids.cur_id		

bugs_ins(@bug,@status,@priority)
Takes a list of bug fields and makes a bug out of it.

bugs_upd(@bug)
Takes a list of bug fields and makes a bug out of it.

statlinks_upd(@bug,@release,@status,@person,@date)
This procedure takes information pertaining
to the status of a bug and makes a statlink
out it. Then it updates the status and bugs tables.

wordlinks_ins(@bug,@type,@word)
Inserts a wordlink

Appendix D: Description of bugdb.pllib

(I) Introduction

This set of libraries implements many of the basic functions of accessing the Bug Database by Perl commands. This makes possible the construction of scripts that select bugs from the database using a variety of searches and format the ouput all without using SQL. The structure of the library is based on a strategy with two goals: (1) to provide a consistent, intuitive means of passing data between the various routines and (2) to use a layered approach that builds up complex functions on top of a basic set of simple tools.

The library consists of three sets of routines. At the lowest level are the functions that reside in the `${BUGDBLIB}/list` directory. These routines are used for handling the most basic units of the library: strings and lists. Most do some variation of the following task: convert a list to a string with the items seperated by some delimiter, and vice-versa. Typically, the delimter used is control-a. On occasions, when we wish to represent a multi-dimensional object as a string, control-b is used as a seperator for the second dimension.

The next layer of the library operates primarily on associative arrays. The routines used in this layer draw extensively on the list functions. Associative arrays are the primary structure of the libary. They are used to store command line arguments, parameter sets for searches, bugs that have been retrieved from the database, and menus. These routines are found in `${BUDBLIB}/assoc` and are primarily used for two tasks: (1) converting command line arguments (@ARGV) to a standardized associative array and back again; (2) processing associative arrays using the list functions.

The top layer routines--`${BUGDBLIB}/io`--contain the tasks that intereact with the database and the outside world. Each is designed for handling particular

aspects of selection, retrieval, formatting, and output. Using these functions it is possible to create complex database commands fairly quickly.

(II) The Functions

The previous section gave a brief overview of each part of the library. In this section each function is described in detail; the structure of the arguments and results are given as well as an example of the proper usage.

The first set of functions described here are those for formatting strings and lists. This is a common task since the database often require input to be formatted in a comma seperated string. Also, the database returns the results of queries as a list of records where each field in the record is seperated by a control-a. The names of the functions are constructed using a cryptic set of mnemonics. In general, a function name consists of:

<type0> '2' <type1>

which is used to signify that a particular function takes an argument of <type0> and returns it as <type1>. Such a function name should be read as "convert type 0 to type 1". The types are:

list = standard perl list of items

sep = a string of items where each item is seperated by control-a

sepb = a string of items where each item is seperated by control-b

hmt = abbreviation of head-middle-tail which refers to a string representation of a list formatted in the following manner:

$head . $item0 . $tail . $middle . $head . $item1 . $tail . $middle ...

where $head, $middle, $tail are string variables. A typical set of assignments is:

$head = '"'; $middle = ', '; $tail = '"';

which would lead to a string:

"item0", "item1", ... , "itemN"

i.e. each item in the list is sandwiched between $head and $tail and seperated by $middle.

$string = &hmt2hmt($h0,$m0,$t0,$h1,$m1,$t1,$string);

Converts a head-middle-tail string to another head-middle-tail string.

Inverse: hmt2hmt.

Argument: head-middle-tail string fomatted with $h0, $m0, $t0.

Output: head-middle-tail string formatted with $h1, $m1, $t1.

Example: to take a string formatted with '|' as seperators and convert it to a comma seperated list with quotes around each item:

```
$string = &hmt2hmt('','|','','"',', ',','"',$string);
```

@list = &hmt2list($h,$m,$t,$string);

Converts a head-middle-tail string to a list.

Inverse: list2hmt.

Argument: head-middle-tail string formatted with $h, $m, $t.

Output: list of items stripped of formatting.

Example: to convert a comma seperated string with quotes around each item to a list:

```
@list = &hmt2list('"',', ','"',$string);
```

$string = &list2hmt($h,$m,$t,@list);

Converts a list to a head-middle-tail string.

Inverse: hmt2list.

Argument: list of items.

Output: head-middle-tail string formatted with $h, $m, and $t.

Example: to convert a list into '|' seperated string:

```
$string = &list2hmt('','|','',@list);
```

$string = &hmt2sep($h,$m,$t,$string);

Converts a head-middle-tail string to a control-a seperated string.

Inverse: sep2hmt.

Argument: head-middle-tail string formatted with $h, $m, $t.

Output: control-a seperated string.

Example: convert a comma seperated string:

```
$string = &hmt2sep('',', ','',$string);
```

$string = &sep2hmt($h,$m,$t,$string);

Converts a control-a seperated string to a head-middle-tail string.

Inverse: hmt2sep.

Argument: control-a seperated string.

Output: head-middle-tail string formatted with $h, $m, $t.

Example: convert a control-a string to a comma seperated string:

```
$string = &hmt2sep('',', ','',$string);
```

$string = &list2sep(@list);

Converts a list to a control-a seperated string.

Inverse: sep2list.

Argument: a list of items.

Output: control-a seperated string.

@list = &sep2list($string);

Converts a control-a seperated string to a list.

Inverse: list2sep.

Argument: control-a seperated string.
Output: a list of items.

$string = &list2sepb(@list);
Converts a list to a control-b seperated string.
Inverse: sep2list.
Argument: a list of items.
Output: control-b seperated string.

@list = &sepb2list($string);
Converts a control-b seperated string to a list.
Inverse: list2sep.
Argument: control-b seperated string.
Output: a list of items.

@list = &quotify(@list);
Takes a list and puts quotes around any items with spaces. Used primarily for formatting command line arguments.
Inverse: none.
Argument: list of items.
Output: lits of items with quotes around those that have spaces.

Information is passed between procedures in the library via associative arrays. A variety of procedures have been written to facilitate the manipulation of associative arrays. These functions are written primarily to handle two specific situations. First, to convert command line arguments from @ARGV to a standard

form associative array. Second, to change the formatting of the elements of an associative array.

%arguments = &argv2assoc(@argv)

Convert @argv format to an associative array by making '-' items keys and all other items arguments of the keys.

Inverse: assoc2argv.

Argument: list in @argv format.

Output: associative array.

Example:

```
@ARGV = (-opt1,arg1,arg2,-opt2,arg3,arg4);
%args = &argv2assoc(@argv);
$args{'opt1'} = arg1 . control-a  . arg2
$args{'opt2'} = arg3 . control-a .  arg4
```

assoc2argv

abbr2full.plsub

full2abbr.plsub

full_abbr.plsub

assoc_hmt2hmt.plsub

assoc_hmt2sep.plsub

assoc_sep2hmt.plsub

commands:

bug_report.pl

bug_search.pl

build_search.pl

keyword_info.pl

save_bugs.pl

standard_search.pl

io:

bug_report.plsub

bug_search.plsub

bug_table.plsub

build_search.plsub

exec_sql.plsub

gopher_bug.plsub

gopher_menu.plsub

keyword_info.plsub

run_sql.plsub

start_browser.plsub

start_owner.plsub