The Hacker's Diet *Online*

Web-Based Computer Tools

by John Walker

August 2007

Contents

1 Introduction **1**
- 1.1 Development Environment . . . 1
- 1.2 Application Module Structure . . . 2
- 1.3 Perl Library Module Requirements . . . 2
- 1.4 Local Library Modules . . . 2

2 System Environment Parameters **3**
- 2.1 Program Version . . . 3
- 2.2 Release Date . . . 3
- 2.3 Build Number . . . 3
- 2.4 Build Time . . . 3
- 2.5 Configuration Parameters . . . 3
  - 2.5.1 Beta Test Mode . . . 3
  - 2.5.2 Beta Test Invitation Backdoor . . . 4
  - 2.5.3 CSV Format Version . . . 4
  - 2.5.4 Confirmation signature encoding suffix . . . 4
  - 2.5.5 Master encryption key . . . 4
  - 2.5.6 Monthly log weight range . . . 4
- 2.6 Public Web Addresses . . . 5
- 2.7 Web Installation Directories . . . 5
- 2.8 Host System Properties . . . 7
- 2.9 Web URL Addresses . . . 11

3 `hdCSV.pm` Extended CSV File Parser **13**
- 3.1 Package plumbing . . . 13
- 3.2 ParseCSV . . . 14
- 3.3 EncodeCSV . . . 15

4 `trendfit.pm`: Trend Fitter Object **16**
- 4.1 Package plumbing . . . 16
- 4.2 Constructor . . . 16
- 4.3 Start . . . 17
- 4.4 Add Point . . . 17
- 4.5 Fit Slope . . . 18
- 4.6 Minimum, Maximum, and Mean . . . 18

5 `monthlog.pm`: Monthly Log Object **19**
- 5.1 Package plumbing . . . 19
  - 5.1.1 Constants and conversion tables . . . 20
- 5.2 Constructor . . . 21
- 5.3 Destructor . . . 22
- 5.4 Describe . . . 23

5.5 ComputeTrend . . . 24
5.6 BodyMassIndex . . . 25
5.7 FractionFlagged . . . 26
5.8 Save . . . 27
5.9 Load . . . 28
5.9.1 Parse year, month, and log unit . . . 28
5.9.2 Parse trend carry forward and last modification time . . . 28
5.9.3 Parse daily weight array . . . 29
5.9.4 Parse exercise rung array . . . 29
5.9.5 Parse flag array . . . 30
5.9.6 Parse comments . . . 30
5.10 ToHTML . . . 31
5.10.1 Write HTML table header . . . 32
5.10.2 Generate date column . . . 32
5.10.3 Generate weight, trend, and variance columns . . . 33
5.10.4 Generate exercise rung column . . . 34
5.10.5 Generate flag column . . . 34
5.10.6 Generate comment column . . . 35
5.10.7 Write HTML table footer . . . 35
5.10.8 Format weight to display unit . . . 36
5.10.9 Convert weight from one unit to another . . . 36
5.10.10 Express weight in canonical form . . . 37
5.10.11 HTML generation utility functions . . . 37
5.10.11.1 Test if defined and nonzero . . . 37
5.10.11.2 Return null string if not defined or zero . . . 37
5.10.11.3 Return zero if not defined . . . 38
5.10.11.4 Format as fixed point decimal . . . 38
5.10.11.5 Format as weight . . . 38
5.10.11.6 Format as variance . . . 39
5.11 computeChartScale . . . 40
5.12 plotChart . . . 41
5.12.1 Plotting sparse data . . . 42
5.12.2 Allocate colours for chart . . . 42
5.12.3 Define chart geometry . . . 43
5.12.4 Draw axes for chart and label date axis . . . 43
5.12.5 Plot exercise rung information . . . 44
5.12.6 Determine scale for weight and trend plot . . . 45
5.12.7 Compute diet plan extrema on chart . . . 47
5.12.8 Plot the diet plan if defined and requested . . . 48
5.12.9 Plot weight trend line on chart . . . 48
5.12.10 Plot weight entries as floats and sinkers . . . 49
5.12.11 Label weight axis . . . 50
5.13 updateFromCGI . . . 50
5.13.1 Apply changes to weight . . . 51
5.13.2 Expand weight entry if abbreviated . . . 52
5.13.2.1 Expand weight entry in stones and pounds . . . 53
5.13.3 Apply changes to exercise rung . . . 54
5.13.4 Apply changes to flag . . . 55
5.13.5 Apply changes to comment . . . 56
5.14 ImportCSV . . . 57
5.15 ExportCSV . . . 58
5.16 ExportHDReadCSV . . . 59
5.17 ExportExcelCSV . . . 60

5.17.1 Obtain trend carry-forward for Excel CSV . . . 61
5.18 ExportXML . . . 62
5.19 Monthdays . . . 63
5.20 Previous and next month . . . 64
5.21 Verbose . . . 65
5.22 EncodeComments . . . 66
5.23 DecodeComments . . . 67

6 `history.pm`: Historical Analysis Object **68**
6.1 Package plumbing . . . 69
6.2 Constructor . . . 70
6.3 Get log data for range of days . . . 71
6.4 Analyse Trend . . . 73
6.4.1 Build table of intervals and compute date span of union . . . 74
6.4.2 Examine dates, fitting those within intervals . . . 74
6.5 Draw Chart . . . 75
6.5.1 Scaling and line drawing functions . . . 76
6.5.2 Determine the number of days in the historical interval . . . 77
6.5.3 Find weight and trend extrema in log entries to be plotted . . . 78
6.5.4 Find diet plan extrema on historical chart . . . 79
6.5.5 Plot the diet plan on historical chart . . . 80
6.5.6 Define historical chart geometry . . . 80
6.5.7 Determine vertical weight scaling based on extrema . . . 81
6.5.8 Draw axes for historical chart . . . 82
6.5.8.1 Label date axis at the bottom . . . 83
6.5.8.1.1 Label date axis with years, months, and possibly weeks . . . 84
6.5.8.1.2 Label date axis with year numbers only . . . 85
6.5.9 Label weight axis at the left . . . 86
6.5.10 Label exercise rung axis if any plotted . . . 86
6.5.11 Plot weight and rung data on historical chart . . . 87
6.5.11.1 Plot multiple pixels per day . . . 88
6.5.11.1.1 Plot weight entry as float or sinker . . . 89
6.5.11.2 Plot multiple days per pixel . . . 90
6.5.12 Draw title with date range . . . 91
6.5.13 Draw caption with trend summary . . . 92
6.6 Draw Badge Image . . . 93
6.7 Generate synthetic data . . . 97
6.7.1 Create new month for synthetic data . . . 98
6.7.2 Apply perturbation functions to value . . . 99
6.8 Utility historical queries . . . 99
6.8.1 Last day in database . . . 100
6.8.2 First day in database . . . 101
6.8.2.1 Obtain list of years . . . 101
6.8.2.2 Ensure month is in cache . . . 102
6.8.3 First day of interval . . . 103
6.9 Monthly log cache utilities . . . 103
6.9.1 Fill cache with monthly logs in the date range . . . 104
6.9.2 Write back all items in the cache . . . 105
6.9.3 Empty monthly log cache . . . 105

7 `Aggregator.pm`: Data Aggregator Object **106**
7.1 Package plumbing . . . 106
7.2 Constructor . . . 107
7.3 Retrieve . . . 108
7.3.1 Return log items for aggregation from this user . . . 109

8 `user.pm`: User Object **110**
8.1 Package plumbing . . . 110
8.2 Constructor . . . 110
8.3 Login . . . 113
8.4 Describe . . . 114
8.5 Save . . . 115
8.6 Load . . . 116
8.7 Login Form . . . 117
8.8 New Account Form . . . 119
8.8.1 Encode preset values for use in HTML . . . 120
8.8.2 User login name text field . . . 121
8.8.3 Beta test invitation field . . . 121
8.8.4 Password and password confirmation fields . . . 122
8.8.5 E-mail address text field . . . 123
8.8.6 User's full name optional text fields . . . 123
8.8.7 Height (for body mass index) . . . 124
8.8.8 Weight and energy unit radio buttons . . . 125
8.8.9 Decimal character selection . . . 126
8.8.10 Public name settings . . . 126
8.9 resetPassword . . . 127
8.10 generatePassword . . . 127
8.11 SendMail . . . 128
8.12 Export user information as XML . . . 129
8.13 Export preferences as XML . . . 130
8.14 Export diet plan as XML . . . 131
8.15 enumerateMonths . . . 132
8.16 enumerateYears . . . 133
8.17 dietPlanLimits . . . 134
8.18 generateEncryptedUserID . . . 135
8.19 decodeEncryptedUserID . . . 136
8.20 QuoteUserName . . . 137
8.21 Express number in canonical form . . . 138
8.22 Convert decimal number to localised form . . . 138
8.23 Express number in localised form . . . 139

9 `session.pm`: Session Object **140**
9.1 Package plumbing . . . 140
9.2 Constructor . . . 141
9.3 Describe . . . 142
9.4 Save . . . 143
9.5 Load . . . 144
9.6 Save Active Session . . . 144
9.7 Load Active Session . . . 145
9.8 GenerateSessionID . . . 145

10 cookie.pm: Cookie Object **146**
10.1 Package plumbing . . . 146
10.2 Constructor . . . 147
10.3 Describe . . . 148
10.4 Save . . . 148
10.5 Load . . . 149
10.6 signCookie . . . 149
10.7 generateCookie . . . 150
10.8 expireCookie . . . 150
10.9 storeCookie . . . 151
10.10 testCookiePresent . . . 152
10.11 checkCookieSignature . . . 153
10.12 generateCookieID . . . 153

11 pubname.pm: Public Name Manager Object **154**
11.1 Package plumbing . . . 154
11.2 Constructor . . . 155
11.3 Generate random name . . . 156
11.4 Generate unique name . . . 157
11.5 Assign public name . . . 158
11.6 Find public name . . . 159
11.7 Delete public name . . . 159
11.7.1 Delete existing public name . . . 160
11.8 Describe . . . 160
11.9 Save . . . 161
11.10 Load . . . 161

12 HackDiet.pl: Main CGI Application **162**
12.1 Main program . . . 163
12.1.1 Set handler for termination signals . . . 164
12.1.2 Define requests permissible whilst browsing public account . . . 164
12.1.3 Extract brain-dead Internet Exploder request field . . . 165
12.1.4 Estimate local time at user site . . . 167
12.1.5 Dispatch requests which return non-HTML results . . . 168
12.1.6 Dispatch requests which return HTML result documents . . . 169
12.1.6.1 Login-related transactions . . . 170
12.1.6.2 Account management transactions . . . 170
12.1.6.3 Dispatch administrator requests . . . 171
12.1.7 Return login request form . . . 172
12.1.8 Validate user login request . . . 173
12.1.8.1 Validate user name and password . . . 174
12.1.8.1.1 Reject login: Unknown user name . . . 174
12.1.8.1.1.1 Log failed login attempt in system log . . . 175
12.1.8.1.2 Reject login: Incorrect password . . . 175
12.1.8.2 Close previous session if still open . . . 176
12.1.8.3 Open new session and link to user directory . . . 176
12.1.8.4 Update last login and transaction time . . . 177
12.1.8.5 Add login to history database . . . 177
12.1.8.6 Update persistent login state . . . 178
12.1.9 Main account dispatch page . . . 179
12.1.9.1 Show user name and account being browsed . . . 180
12.1.9.2 Standard navigation bar functions . . . 180
12.1.9.3 Utility functions for regular session . . . 181
12.1.9.4 Browsing public account functions . . . 182

12.1.9.5 Administrator-only functions . . . 183
12.1.9.6 Show build number and date . . . 184
12.1.9.7 Generate assumed identity notification . . . 185
12.1.10 Force re-login if session terminated or invalid . . . 186
12.1.11 Display password reset request form . . . 187
12.1.12 Reset a user's password . . . 188
12.1.12.1 Validate E-mail address agrees with specification in reset request . . . 189
12.1.12.2 Prohibit password reset on read-only account . . . 190
12.1.12.3 Send E-mail confirming password reset . . . 190
12.1.12.4 Return password reset confirmation page . . . 191
12.1.13 Log out user: end session . . . 192
12.1.13.1 Retrieve active session information . . . 193
12.1.13.2 Retrieve user account information . . . 194
12.1.13.3 Sanity check year and month specification . . . 195
12.1.14 Display monthly log . . . 196
12.1.14.1 Determine which monthly log to display . . . 197
12.1.14.2 Read log if in database or create blank log if it's not . . . 197
12.1.14.3 Monthly log title and navigation buttons . . . 198
12.1.14.4 Set monthly log property variables . . . 199
12.1.14.4.1 Define "cachebuster" argument . . . 199
12.1.14.5 Generate hidden monthly log property fields . . . 200
12.1.14.6 Fill in trend carry-forward from most recent previous log, if required . . . 201
12.1.14.7 Display trend summary below monthly chart . . . 202
12.1.14.8 Monthly log control panel . . . 203
12.1.14.8.1 Administrator object dump selection . . . 204
12.1.14.9 Dump objects if requested by administrator . . . 205
12.1.15 Update monthly log . . . 206
12.1.15.1 Write updated log item back to database . . . 207
12.1.15.1.1 Update Web page badge . . . 207
12.1.16 Display calendar navigation page . . . 208
12.1.16.1 Generate table of yearly calendars . . . 209
12.1.16.2 New monthly log creation form . . . 210
12.1.17 Display CSV import request form . . . 211
12.1.17.1 CSV file upload import form . . . 212
12.1.17.2 CSV direct upload import form . . . 213
12.1.18 Import uploaded CSV log entries . . . 214
12.1.18.1 Import log items from XML database file . . . 217
12.1.18.1.1 Process XML daily log entry . . . 218
12.1.18.2 Dump XML database file . . . 219
12.1.18.3 Import log items from CSV database file . . . 220
12.1.18.4 Check for Excel CSV record . . . 221
12.1.18.4.1 Parse Excel CSV date field . . . 222
12.1.18.4.2 Load or create monthly log containing Excel CSV record . . . 223
12.1.18.4.3 Set monthly log entry from Excel CSV record . . . 224
12.1.18.5 Check for Palm/HDREAD CSV record . . . 225
12.1.18.5.1 Set monthly log entry from Palm CSV record . . . 226
12.1.18.6 Write back logs modified by database import . . . 226
12.1.18.7 Append summary of records imported . . . 227
12.1.19 Configure Web page status badge . . . 228
12.1.19.1 Interval selection . . . 229
12.1.19.2 Form action buttons . . . 230
12.1.20 Update Web page status badge . . . 231
12.1.20.1 Show updated badge configuration . . . 232

12.1.20.2 Show prototype XHTML code to display badge . . . 233
12.1.21 Recalculate trend carry-forward for all logs for a user . . . 234
12.1.22 Quit browsing another account . . . 235
12.1.23 Download monthly log as CSV file . . . 235
12.1.24 Download monthly log as XML file . . . 236
12.1.25 Export log database . . . 237
12.1.25.1 Selection of months to export from database . . . 238
12.1.26 Process database export . . . 239
12.1.26.1 Determine first and last days in database . . . 240
12.1.26.2 Export database in XML format . . . 241
12.1.26.3 Export database as Hacker's Diet Online CSV . . . 242
12.1.26.4 Export database as Palm Eat Watch CSV . . . 243
12.1.26.5 Export database as Legacy Excel Eat Watch CSV . . . 244
12.1.27 Request paper log forms . . . 245
12.1.27.1 Selection of months for paper logs . . . 246
12.1.28 Generate paper log forms . . . 247
12.1.28.1 Generate paper log form for month . . . 248
12.1.29 Download backup copy of all logs for user . . . 249
12.1.30 Generate monthly chart . . . 250
12.1.30.1 Specify Content-type for PNG image . . . 250
12.1.31 Trend analysis . . . 251
12.1.31.1 Emit trend anlysis page . . . 252
12.1.31.2 Add standard intervals to analysis list . . . 252
12.1.31.3 Process custom interval specification, if any . . . 253
12.1.31.4 Output trend analysis report for intervals evaluated . . . 254
12.1.31.4.1 Output table rows for each interval analysed . . . 255
12.1.31.4.2 Label trend report for custom interval . . . 256
12.1.31.5 Generate form fields for custom trend interval . . . 257
12.1.31.5.1 Custom start and end date selection boxes . . . 258
12.1.31.5.1.1 Custom trend start date . . . 259
12.1.31.5.1.2 Custom trend end date . . . 260
12.1.32 Diet calculator . . . 261
12.1.32.1 Set primary diet calculator fields from user object . . . 262
12.1.32.2 Generate warning if JavaScript disabled in diet calculator form . . . 263
12.1.32.3 Report warnings from static diet calculator update . . . 263
12.1.32.4 Generate array of years for diet calculator selection . . . 264
12.1.32.5 Preset diet calculator start and end dates . . . 264
12.1.32.6 Generate diet calculator form . . . 265
12.1.32.6.1 Diet calculator: calorie balance . . . 265
12.1.32.6.2 Diet calculator: initial weight . . . 266
12.1.32.6.3 Diet calculator: goal weight . . . 266
12.1.32.6.4 Diet calculator: desired weight change . . . 267
12.1.32.6.5 Diet calculator: weight change per week . . . 267
12.1.32.6.6 Diet calculator: diet duration . . . 268
12.1.32.6.7 Diet calculator: start and end dates . . . 269
12.1.32.7 Diet calculator form action buttons . . . 270
12.1.32.8 Calculate dependent variables from primary variables . . . 270
12.1.32.9 Override diet calculator primary fields from form fields . . . 271
12.1.32.10 Perform static update of diet calculator . . . 272
12.1.32.10.1 Static change to energy unit . . . 273
12.1.32.10.2 Static change to daily balance . . . 273
12.1.32.10.3 Static change to weight unit . . . 273
12.1.32.10.4 Static change to initial weight . . . 274

12.1.32.10.5 Static change to goal weight . . . 274
12.1.32.10.6 Static change to desired weight change . . . 274
12.1.32.10.7 Static change to weight change per week . . . 275
12.1.32.10.8 Static change to diet duration in weeks . . . 275
12.1.32.10.9 Static change to diet duration in months . . . 276
12.1.32.10.10 Static change to start date . . . 276
12.1.32.10.11 Static change to end date . . . 277
12.1.33 Save diet calculator settings . . . 278
12.1.34 Request historical chart . . . 279
12.1.34.1 Emit historical chart request form . . . 280
12.1.34.2 Generate form fields for custom chart interval . . . 281
12.1.34.2.1 Embed historical chart image in request/result page . . . 281
12.1.34.2.1.1 Determine range of dates to plot in historical chart . . . 282
12.1.34.2.2 Set variables to default to previous request settings . . . 283
12.1.34.2.3 Generate option items for years in database . . . 285
12.1.34.2.4 Generate option items for months . . . 285
12.1.34.2.5 Generate option items for days . . . 286
12.1.34.2.6 Generate option items for chart sizes . . . 286
12.1.35 Generate historical chart . . . 287
12.1.36 Create new user account request . . . 288
12.1.36.1 Propagate handheld setting to subsequent forms . . . 288
12.1.37 Process new user account request . . . 289
12.1.37.1 Validate user name for new account . . . 290
12.1.37.2 Validate beta test invitation code . . . 290
12.1.37.3 Cancel used beta test invitation code . . . 291
12.1.37.4 Create the new user account . . . 291
12.1.37.4.1 Store settings for user account . . . 292
12.1.37.4.2 Update user account information . . . 293
12.1.37.5 Report errors in new account request and re-issue form . . . 294
12.1.38 Modify user account request . . . 295
12.1.39 Reject setting query or change from cookie-based login . . . 296
12.1.40 Process user account modification . . . 297
12.1.40.1 Log changes to account settings . . . 298
12.1.40.2 Report errors in account modification request and re-issue form . . . 299
12.1.41 Forget all persistent logins . . . 300
12.1.42 List publicly-visible accounts . . . 301
12.1.42.1 Obtain list of public accounts . . . 302
12.1.42.2 Generate table of public accounts . . . 303
12.1.43 Provide browse access to public account . . . 304
12.1.43.1 Look up public account and verify it exists . . . 305
12.1.44 Request invitation codes . . . 306
12.1.45 Generate invitation codes . . . 307
12.1.46 Display administrator account manager . . . 309
12.1.46.1 Obtain list of open accounts . . . 313
12.1.47 Provide administrator access to user account . . . 314
12.1.48 Verify that user has administrator privilege . . . 315
12.1.49 Process administrator database purge . . . 316
12.1.50 Process administrator account delete . . . 318
12.1.51 Display administrator session manager . . . 320
12.1.51.1 Obtain list of open sessions . . . 321
12.1.51.2 Generate table of open sessions . . . 322
12.1.52 Force termination of user session . . . 323
12.1.52.1 Validate administrator password . . . 324

12.1.52.2 Confirm a session is selected . . . 325
12.1.52.3 Validate specified session . . . 326
12.1.53 Display administrator persistent login manager . . . 327
12.1.53.1 Obtain list of persistent login tokens . . . 328
12.1.53.2 Generate table of persistent logins . . . 328
12.1.54 Delete a persistent login token . . . 329
12.1.54.1 Confirm a persistent login is selected . . . 330
12.1.55 Display administrator global statistics . . . 331
12.1.55.1 Request log records from the aggregator and compute global statistics . . . 332
12.1.55.2 Generate global statistics for open accounts . . . 333
12.1.55.3 Display global statistics mean trend change . . . 334
12.1.55.4 Compute global statistics gain and loss extrema . . . 335
12.1.55.5 Display global statistics gain and loss extrema . . . 336
12.1.55.6 Display global statistics log update frequency . . . 337
12.1.55.7 Receive log records from the aggregator for global statistics . . . 338
12.1.55.7.1 Update global statistics overall trend analysis . . . 339
12.1.55.7.2 Compute global statistics trend analysis for previous user . . . 340
12.1.56 Generate synthetic data for user account . . . 341
12.1.56.1 Generate synthetic data as specified in form . . . 342
12.1.56.2 Generate synthetic data specification form . . . 343
12.1.56.2.1 Synthetic data start date . . . 344
12.1.56.2.2 Synthetic data end date . . . 345
12.1.56.2.3 Synthetic data field selection . . . 346
12.1.56.2.4 Synthetic data start and end values . . . 346
12.1.56.2.5 Table of perturbation functions . . . 347
12.1.57 Send a feedback message . . . 348
12.1.57.1 Generate feedback message composition form . . . 349
12.1.57.2 Show preview of message being composed . . . 350
12.1.57.2.1 Check spelling in subject and message . . . 351
12.1.57.3 Enumerate feedback message categories . . . 352
12.1.58 Send a message from the feedback form . . . 353
12.1.58.1 Send mail to feedback address . . . 354
12.1.58.2 Send copy to the submitter . . . 355
12.1.58.3 Show feedback message in reply page . . . 356
12.1.59 Delete entire log database . . . 357
12.1.59.1 Emit shrill warning about what is about to transpire . . . 358
12.1.59.2 Generate form permitting user to back up database . . . 358
12.1.59.3 Generate destructive operation confirmation form . . . 359
12.1.60 Process database delete . . . 360
12.1.60.1 Reject deletion request when user name and password fail to match . . . 361
12.1.60.2 Reject deletion request when confirmation code fails to match . . . 361
12.1.60.3 Backup user account before destructive operation . . . 362
12.1.60.4 Generate confirmation of database deletion . . . 362
12.1.61 Close this user account . . . 363
12.1.61.1 Reject request if logs remain in database . . . 364
12.1.61.2 Warn user about consequences of closing account . . . 364
12.1.62 Process user account close . . . 365
12.1.62.1 Reject account close for user name or password mismatch . . . 366
12.1.62.2 Reject account close for confirmation code mismatch . . . 366
12.1.62.3 Reject account close if logs remain in the database . . . 367
12.1.63 Generate test output page . . . 367
12.1.64 Emit diagnostic for undefined query . . . 367
12.1.64.1 Dump CGI environment and parsed arguments . . . 368

12.1.65 JavaScript debugging console . . . 368
12.1.66 Template . . . 368
12.2 Global declarations . . . 369
12.3 Perl language modes . . . 369
12.3.1 Default parameter settings . . . 370
12.3.2 Global variables . . . 370
12.3.3 Process command line options . . . 371
12.3.4 Validate option specfications . . . 371
12.4 XHTML generation . . . 371
12.4.1 Mime Content-type specification . . . 372
12.4.2 Local time zone offset field . . . 372
12.5 Utility Functions . . . 373
12.5.1 Propagate trend through user's monthly logs . . . 374
12.5.1.1 Identify log where recomputation begins . . . 375
12.5.1.2 Load first log into memory . . . 375
12.5.1.3 Propagate trend to next log . . . 376
12.5.1.3.1 Convert trend to weight unit in this log, if different . . . 376
12.5.1.3.2 Write modified log back to database . . . 377
12.5.1.3.3 If requested, canonicalise weight entries in log . . . 377
12.5.2 Append entry to transaction history log . . . 378
12.5.3 Update time of user's last transaction . . . 379
12.5.4 Return time of user's last transaction . . . 379
12.5.5 Test whether user has a session active . . . 380
12.5.6 Parse weight value . . . 381
12.5.7 Parse signed weight value . . . 381
12.5.8 Wrap long lines onto multiple lines . . . 382
12.5.9 Print command line help information . . . 383
12.5.10 Minimum, Maximum, Sign, and Round functions . . . 384
12.5.11 Execute system command . . . 385
12.5.12 Edit Unix time value to ISO 8601 local date and time . . . 385
12.5.13 Convert characters in a string to hexadecimal . . . 386
12.5.14 Test if month is the current month . . . 386
12.5.15 Draw text in a chart . . . 387
12.5.16 Encode international domain name . . . 388
12.5.17 Test domain valid for E-mail . . . 388
12.5.18 Parse CGI arguments . . . 389
12.5.19 Supply default values for undefined variables . . . 390
12.5.20 Read line from persistent object file . . . 390

13 Cluster file system support **391**
13.1 Display cluster configuration . . . 392
13.1.1 Display summary of cluster transaction queue . . . 393
13.2 Enqueue cluster synchronisation transaction . . . 394
13.2.1 Copy a file to the cluster members . . . 395
13.2.2 Delete a file from cluster members . . . 395
13.2.3 Create a directory on cluster members . . . 395
13.2.4 Delete a directory from cluster members . . . 396
13.2.5 Recursively delete a directory from cluster members . . . 396
13.3 Cluster synchronisation process . . . 397
13.3.1 Assume group and user identity of cluster synchronisation process . . . 398
13.3.2 Save process ID of cluster synchronisation job . . . 398
13.3.3 Activate cluster synchronisation log file if configured . . . 399
13.3.4 Cycle active log file if `HUP` signal received . . . 399

13.3.5 Process queued cluster synchronisation transactions . . . 400
13.3.5.1 Execute cluster synchronisation transaction . . . 401
13.3.5.2 Recover from failure of a cluster synchronisation transaction . . . 402
13.3.5.3 Determine if failed transaction should be retried . . . 403
13.3.6 Execute cluster synchronisation command . . . 404
13.3.6.1 Check for errors we deem harmless to cluster synchronisation . . . 405
13.3.7 Output a log message . . . 405
13.3.8 Conditionally un-taint a variable . . . 406
13.3.9 Display if variable is tainted . . . 406

14 HTML utilities **407**
14.1 Write XHTML prologue . . . 408
14.2 Generate XHTML navigation bar . . . 410
14.3 Write XHTML epilogue . . . 412
14.4 Quote text for inclusion in HTML . . . 413

15 XML utilities **414**
15.1 Generate XML prologue . . . 415
15.2 Generate XML epilogue . . . 415
15.3 QuoteXML . . . 416
15.4 TextXML . . . 416
15.5 UNIX time to ISO date and time . . . 417

16 Julian date utilities **419**
16.1 Julian date constant definitions . . . 420
16.2 Julian date support functions . . . 420
16.3 Gregorian leap year computation . . . 421
16.4 Gregorian date to Julian day number . . . 421
16.5 Julian day to Gregorian date . . . 422
16.6 Julian day to day of week . . . 423
16.7 Civil time to Julian day fraction . . . 423
16.8 Julian day fraction to civil time . . . 424
16.9 UNIX time to Julian day and fraction . . . 424
16.10 Julian day and fraction to UNIX time . . . 424
16.11 UNIX time to civil date and time . . . 425
16.12 Julian day and fraction to RFC 822 time and date . . . 425
16.13 Julian day and fraction to RFC 3339 time and date . . . 426
16.14 Julian day and fraction to old HTTP cookie time and date . . . 426

17 Documentation in POD format **427**
17.0.1 Options . . . 428
17.0.1.1 `--copyright` . . . 428
17.0.1.2 `--help` . . . 428
17.0.1.3 `--verbose` . . . 428
17.0.1.4 `--version` . . . 429

18 `HackDietBadge.pl`: Return badge for user **430**
18.1 HackDietBadge . . . 431

19 `jig.pl`: Test Jig **433**

20 `Bowdler.pl`: Bowdlerise Source for Publication **434**

21 `bump`: Increment build number **435**

22 XHTML Style Sheet **436**
22.1 Global document properties . . . 436
22.2 Links . . . 437
22.3 Headings and titles . . . 438
22.4 Blocks of text . . . 439
22.5 Block text styles . . . 439
22.6 Inline text decoration . . . 440
22.7 Fieldsets . . . 441
22.8 Images . . . 441
22.9 Canvas . . . 442
22.10 Lists . . . 442
22.11 Navigation . . . 443
22.12 Paper log forms . . . 444
22.13 Tables . . . 445
22.13.1 Page title table . . . 445
22.13.2 Sign in and account management tables . . . 446
22.13.3 Monthly log table . . . 447
22.13.4 Trend analysis table . . . 448
22.13.5 Navigation bar table . . . 449
22.13.6 Calendar Navigation Tables . . . 450
22.13.7 Persistent login manager table . . . 450
22.13.8 Feedback message table . . . 451
22.13.9 Global statistics tables . . . 452

23 XHTML Handheld Style Sheet **453**
23.1 Global document properties . . . 453

24 JavaScript Utilities **454**
24.1 Global definitions . . . 454
24.2 Unicode text entity definitions . . . 455
24.3 Document load-time processing . . . 455
24.4 Warn if document accessed insecurely . . . 455
24.5 Record unsaved changes . . . 456
24.6 Document departure processing . . . 456
24.7 Format weight to display unit . . . 457
24.8 Parse weight . . . 458
24.8.1 Parse signed weight . . . 458
24.9 Trend fitting utilities . . . 459
24.9.1 Start fit . . . 459
24.9.2 Add Point . . . 459
24.9.3 Fit Slope . . . 459
24.10 Expand abbreviated weight entry . . . 460
24.10.1 Expand abbreviated stones and pounds entry . . . 461
24.11 Create canvas to draw in chart image . . . 462
24.12 Handle resize of window . . . 463
24.13 Recompute after weight change . . . 464
24.13.1 Check for implausibly large change in weight . . . 465
24.13.2 Undraw trend values starting at this day . . . 466
24.13.3 Plot weight on chart image . . . 467
24.13.4 Extract scaling information for chart . . . 468
24.13.5 Find most recent trend value before this day . . . 469
24.13.6 Update the trend and variance for this and subsequent days . . . 470
24.13.7 Plot the updated trend . . . 471
24.13.8 Fit a linear trend and update weight and energy balance . . . 472

24.13.9 Update the mean and most recent body mass index . . . 473
24.14 Change exercise rung field . . . 474
24.14.1 Plot exercise rung on chart image . . . 475
24.15 Change comment field . . . 476
24.16 Diet calculator support . . . 476
24.16.1 Load diet calculator values . . . 477
24.16.2 Recalculate diet calculator values . . . 478
24.16.2.1 Set date selection . . . 479
24.16.3 Change energy balance . . . 479
24.16.4 Change energy unit . . . 480
24.16.5 Change starting weight . . . 480
24.16.6 Change weight unit . . . 481
24.16.7 Change goal weight . . . 481
24.16.8 Change desired weight gain/loss . . . 482
24.16.9 Change weekly weight gain/loss . . . 482
24.16.10 Change weeks duration . . . 483
24.16.11 Change months duration . . . 483
24.16.12 Change start date . . . 484
24.16.12.1 Get selected date . . . 484
24.16.13 Change end date . . . 485
24.16.14 Change plot diet plan in chart . . . 485
24.17 Validate feedback form . . . 486
24.18 Reset keyboard focus . . . 486
24.19 External link window management . . . 487
24.20 Determine local time zone offset . . . 488
24.21 Express number in canonical form . . . 488
24.22 Height specification unit conversion . . . 488
24.22.1 Centimetres . . . 489
24.22.2 Feet . . . 490
24.22.3 Inches . . . 491
24.23 Handle change to weight unit in new account creation . . . 492
24.24 Replace a text node in an HTML document . . . 492
24.25 Update a variance field . . . 493
24.26 Recompute flagged fraction . . . 493
24.27 Password strength estimation . . . 494
24.27.1 *Ad hoc* tests for bad passwords . . . 495
24.27.2 Character frequency table . . . 496
24.28 Password match indication . . . 497
24.29 Mathematical functions . . . 497
24.30 Debugging console support . . . 498

25 XML Database Export Document Type Definition **499**
25.1 Overall document structure . . . 500
25.2 User information . . . 501
25.3 Preferences . . . 502
25.4 Diet Plan . . . 503
25.5 Monthly Log . . . 504

26 XML Database Export Style Sheet **505**
26.1 Overall document structure . . . 505
26.2 Epoch of export . . . 506
26.3 Account information . . . 506
26.4 User identification . . . 507
26.5 Preferences . . . 508

26.6 Diet plan . . . 509
26.7 Monthly logs . . . 510

27 `webapp.html`: Main Web Page **513**
27.1 HTML Header Section . . . 514

28 `Makefile` **515**
28.1 Extract source code from Nuweb . . . 516
28.2 Source distribution . . . 517
28.3 Documentation . . . 518
28.4 Testing . . . 519
28.5 Clean-up . . . 519
28.6 Installation . . . 520
28.7 Source installation . . . 521

29 Indices **522**
29.1 Files . . . 522
29.2 Macros . . . 522
29.3 Identifiers . . . 531

30 Feature Requests **536**

31 Development Log **537**

Chapter 1

Introduction

The Hacker's Diet *Online* is an interactive Web application implemented as a Common Gateway Interface (CGI) application in Perl. Interactivity is enhanced by a JavaScript component which provides error-checking and real-time data entry feedback, but JavaScript is not required to use the application. The program is 100% "Unicode clean"—any Unicode character may be used in any text field.

No back-end database is required. All user data are stored in the Unix file system in flat ASCII files. Charts are generated directly from Perl using the `GD` module.

This program is in the public domain. It may be used by any person in any manner without any restrictions whatsoever.

1.1 Development Environment

This program was developed using the Literate Programming methodology with the Nuweb programming tool. To build this program, you will need to download and install that utility on your system.

The server-side application requires a recent version of Perl with full Unicode support; this version has been developed and tested on Perl 5.8.5.

The program build process is automated using GNU/Linux `make`, using a `Makefile` defined within this program.

To view the documentation, you will need a LaTeX system with the `xdvi` utility. To update the PDF documentation, you must also have the `pdftex` system; most modern TeX distributions, such as TeX Live, include these components.

The latest version of this program is always available from:

`http://www.fourmilab.ch/hackdiet/online/`

1.2 Application Module Structure

In addition to the `HackDiet.pl` main application, the following modules are defined in this program. All are kept in a `HDiet` subdirectory, along with other material needed by the application

`Aggregator`	Cross-account data aggregation (Chapter 7).
`Cluster`	Cluster file system support (Chapter 13).
`cookie`	Utilities for managing persistent logins (Chapter 10).
`hdCSV`	Utilities for encoding and parsing our Unicode-extended CSV files (Chapter 3).
`history`	Generation of historical charts and trend analyses (Chapter 6).
`html`	Facilities for generating XHTML files (Chapter 14).
`Julian`	Julian date utilities (Chapter 16).
`monthlog`	Monthly log object: represents one month's log for a user (Chapter 5).
`pubname`	Object which manages the pseudonyms assigned to users who grant public access to their records (Chapter 11).
`session`	Session object: represents an open session by a user (Chapter 9).
`trendfit`	Object which manages the fitting of linear trends to data (Chapter 4).
`user`	User object: represents the account and settings for a user account (Chapter 8).
`xml`	Utilities for generating and parsing XML files (Chapter 15).

1.3 Perl Library Module Requirements

The following Perl modules are required by this program. If they are not present in the Perl configuration on your server, you can download and install them from CPAN.

- `Crypt::CBC`
- `Crypt::OpenSSL::AES`
- `Data::Dumper`
- `Digest::SHA1`
- `Encode`
- `Fcntl`
- `File::Temp`
- `GD`
- `Getopt::Long`
- `Socket`
- `Sys::Syslog`
- `Time::HiRes`
- `Time::Local`
- `XML::LibXML`
- `XML::LibXML::Common`

1.4 Local Library Modules

The following Perl library modules are included in the source distribution but not defined within this program. They are CPAN. library modules which have either been specially modified for use in this application, or are included to avoid the need to install them on the Web server which runs this program. If the server's library includes the standard versions of one or more of these modules, no harm will be done—the local copies will be used in all cases.

- `CGI`
- `Digest::Crc32`
- `IDNA::Punycode`
- `Text::CSV`

Chapter 2

System Environment Parameters

Set the following parameters to correspond to the system on which you're installing the software.

2.1 Program Version

⟨ Version 3a ⟩ ≡
```
1.0◇
```
Macro referenced in 371a, 383, 427, 517, 521.

2.2 Release Date

⟨ Release Date 3b ⟩ ≡
```
August 2007◇
```
Macro referenced in 371a, 383, 427, 513.

2.3 Build Number

⟨ Build Number 3c ⟩ ≡
```
5113
◇
```
Macro referenced in 184, 354.

2.4 Build Time

⟨ Build Time 3d ⟩ ≡
```
2011-07-27 20:48 UTC
◇
```
Macro referenced in 184, 354.

2.5 Configuration Parameters

2.5.1 Beta Test Mode

If "Beta test" is set to 1, beta testing features, including the creation of new user accounts only by invitation, will be enabled. If set to 0, the system will be open to the general public and special test features will be disabled.

⟨ Beta test 3e ⟩ ≡
```
0◇
```
Macro referenced in 118, 119, 171, 179, 183, 184, 290b, 291a, 306, 307.

2.5.2 Beta Test Invitation Backdoor

The following magic word serves to bypass the need for a beta test invitation. It is intended purely for developer use in creating test accounts without using up invitations. It should be disabled (by being set to the null string) for production.

⟨ Beta test backdoor 4a ⟩ ≡
```
'Beta luck next time'◇
```
Macro referenced in 290b.

2.5.3 CSV Format Version

This value is included in the second line of all CSV monthly logs we export. This allows us to provide upward compatibility with logs written by earlier versions of the applications should the need arise to change the format.

⟨ CSV Format version 4b ⟩ ≡
```
1.0◇
```
Macro referenced in 58.

2.5.4 Confirmation signature encoding suffix

We require the user to enter a randomly-generated confirmation signature to confirm dangerous and destructive operations. The user's entry is passed to the execution transaction as a CGI argument, and the SHA1 signature of the correct confirmation code as another. To avoid its being obvious that we're passing the signature, and to deter possible attacks by signature collision, we "salt" the confirmation code by appending the following string before computing the signature for encoding and verification. You should change this suffix to a secret value which you do not disclose to users of your server.

⟨ Confirmation signature encoding suffix 4c ⟩ ≡
```
"Sodium Chloride"◇
```
Macro referenced in 149c, 153a, 359, 360, 365, 394, 401.

2.5.5 Master encryption key

The master encryption key is used to encrypt turn-around documents such as the opaque user name identities used in badge generation. This key is 512 bits in length and generated with Fourmilab's HotBits radioactive random number generator. This key is replaced with a default value by the `Bowdler` program—be sure to insert your own randomly-generated key here and guard it against those who would compromise your site.

⟨ Master encryption key 4d ⟩ ≡
```
"Super duper top secret!"◇
```
Macro referenced in 135, 136.

2.5.6 Monthly log weight range

Monthly logs must be scaled to a range wider than that of the weight and trend data which appears in them to permit addition of new entries outside the extrema. The following definitions specify the *minimum* monthly log weight ranges for logs in kilograms and pounds, respectively. This is the range which will be used for a log in which the minimum and maximum of existing entries are equal. The range will be expanded by the difference between the minimum and maximum of existing entries.

⟨ Monthly Log Weight Range in Kilograms 4e ⟩ ≡
```
1◇
```
Macro referenced in 4f, 45.

⟨ Monthly Log Weight Range in Pounds 4f ⟩ ≡
```
(⟨ Monthly Log Weight Range in Kilograms 4e ⟩ * 2)◇
```
Macro referenced in 45.

2.6 Public Web Addresses

The following definitions specify the URLs through which the public accesses the application. They are used where we need fully-qualified cross-links.

⟨ Web Document Home 5a ⟩ ≡
```
/hackdiet/online◇
```
Macro referenced in 163, 196, 233, 440.

Web URL of the static documents for the application.

2.7 Web Installation Directories

⟨ Book Directory 5b ⟩ ≡
```
/server1/pub/www.fourmilab.ch/web/hackdiet◇
```
Macro referenced in 5d.

⟨ Production Book Directory 5c ⟩ ≡
```
/server/pub/www.fourmilab.ch/web/hackdiet◇
```
Macro referenced in 5e.

This is the Web home directory for *The Hacker's Diet* book. All static documents belonging to this application are placed within the "Web Directory" defined below, which is usually a subdirectory of the Book Directory.

⟨ Web Directory 5d ⟩ ≡
⟨ Book Directory 5b ⟩`/online◇`
Macro referenced in 515.

⟨ Production Web Directory 5e ⟩ ≡
⟨ Production Book Directory 5c ⟩`/online◇`
Macro referenced in 515.

This is the Web directory into which the static Web documents for the application (help files, JavaScript, style sheets, images, etc.) are placed.

⟨ CGI Installation Directory 5f ⟩ ≡
```
/server1/bin/httpd/cgi-bin◇
```
Macro referenced in 515.

⟨ Production CGI Installation Directory 5g ⟩ ≡
```
/server/bin/httpd/cgi-bin◇
```
Macro referenced in 515.

The Perl application is installed in the Web server's CGI programs directory as defined above. This is the installation destination as seen from the development system on which this program is built which may not (for example, in the case of a server farm with a deployment machine or NFS mounting of the directories on a server) necessarily be the same as the directory from which the Web server executes the program.

⟨ CGI Execution Directory 5h ⟩ ≡
```
/server/bin/httpd/cgi-bin◇
```
Macro referenced in 6a.

This is the directory from which the HTTP server executes CGI programs, as seen from the server itself.

⟨ CGI Support Directory 6a ⟩ ≡
⟨ CGI Execution Directory 5h ⟩`/HDiet`◇

Macro referenced in 6bc, 369a.

This directory, normally a subdirectory of the main "CGI Execution Directory" is where static support files used by the application (for example, Perl modules, fonts, etc.) are placed. This is independent of the database directory defined below.

⟨ TrueType Font Directory 6b ⟩ ≡
⟨ CGI Support Directory 6a ⟩`/Fonts`◇

Macro referenced in 83, 387.

This directory contains TrueType font definitions used in plotting charts. Each font is expected to have an extension of "`.ttf`".

⟨ Image and Icon Directory 6c ⟩ ≡
⟨ CGI Support Directory 6a ⟩`/Images`◇

Macro referenced in 93b, 431.

This directory contains PNG images and icons incorporated in images we generate.

⟨ Executable Installation Directory 6d ⟩ ≡
`/server1/bin/hackdiet`◇

Macro referenced in 515.

⟨ Production Executable Installation Directory 6e ⟩ ≡
`/server/bin/hackdiet`◇

Macro referenced in 515.

⟨ Database Directory 6f ⟩ ≡
`/server/pub/hackdiet`◇

Macro referenced in 6ghi, 7abc, 11e, 163, 401.

All of the application's data are kept in this directory and its subdirectories. This directory may be located anywhere on the Web server, but must be readable and writeable by the user ID under which CGI programs run.

⟨ Session Directory 6g ⟩ ≡
⟨ Database Directory 6f ⟩`/Sessions`◇

Macro referenced in 176ab, 192, 193, 235a, 304, 314, 321, 322, 323, 326, 365, 380.

Active sessions are denoted by files in this directory. This is usually placed beneath the Database Directory, but needn't be if you wish to be eccentric.

⟨ Users Directory 6h ⟩ ≡
⟨ Database Directory 6f ⟩`/Users`◇

Macro referenced in 78, 102, 104, 105a, 108, 109, 132, 133, 173, 174a, 176ab, 177a, 188, 192, 194, 197b, 201, 207ab, 223, 226b, 231, 241, 242, 243, 244, 249, 290a, 291b, 293, 303, 311, 313, 314, 317, 319, 351, 360, 365, 374, 375b, 376a, 377a, 378, 379ab, 380, 431, 433.

User accounts are stored as subdirectories with the encoded user's name within this directory. This is usually a subdirectory of the Database Directory, but need not be.

⟨ Public Name Directory 6i ⟩ ≡
⟨ Database Directory 6f ⟩`/Pubname`◇

Macro referenced in 156, 157, 158, 159a, 160a, 302.

Public names are generated and managed using files in this directory. This is usually a subdirectory of the Database Directory, but need not be.

⟨ Beta Test Invitations Directory 7a ⟩ ≡
    ⟨ Database Directory 6f ⟩`/Invitations`◇

Macro referenced in 290b, 291a, 308.

Beta test invitations are stored as files in this directory. When one is used to successfully create a new user account, it is deleted.

⟨ Backups Directory 7b ⟩ ≡
    ⟨ Database Directory 6f ⟩`/Backups`◇

Macro referenced in 362a.

Backups of user directories before destructive operations are performed are saved in this directory. If "Backups Directory" is the null string, no backups will be made.

⟨ Cluster Transaction Directory 7c ⟩ ≡
    ⟨ Database Directory 6f ⟩`/ClusterSync`◇

Macro referenced in 392, 393, 394, 400.

If clustering is configured (see "Cluster Member Hosts" below), this directory will be used to store the pending cluster synchronisation transactions generated by this host. If this is the null string or no cluster member hosts are listed, or the named directory does not exist, clustering will be disabled.

2.8 Host System Properties

⟨ Perl directory 7d ⟩ ≡
    `/usr/bin/perl`◇

Macro referenced in 13, 16a, 19, 69, 106, 110, 140, 146, 154, 163, 391, 397a, 407, 414, 419, 431, 433, 434.

This is the absolute path to the directory in which Perl is installed.

⟨ Maximum File Length 7e ⟩ ≡
    `255`◇

Macro referenced in 137.

This specification defines the maximum number of characters in a file name (independent of any other possible restriction in length of a complete path name). This is used to restrict the length of file names we generate based on user names. If the encoded user name exceeds this length, it is limited to this length by truncation to this length less 40 characters, then appending the SHA1 hash of the of the full name before truncation.

⟨ Cluster Member Hosts 7f ⟩ ≡
    `server0.fourmilab.ch server1.fourmilab.ch server2.fourmilab.ch`◇

Macro referenced in 391, 397a.

To enable cluster replication, list the space-separated fully qualified domain names of the cluster members above. These names *must* agree with those passed to CGI programs via the `SERVER_NAME` environment variable, and must function as destination addresses in the `scp` and `ssh` commands. If the list is null, clustering will be disabled.

⟨ Cluster Synchronisation Time Interval 7g ⟩ ≡
    `30`◇

Macro referenced in 397a.

Usually cluster synchronisation is triggered by the receipt of a `SIGUSR1` signal sent by the process which enqueues the transaction. In case one of these signals is not received, or if a synchronisation operation fails, a periodic sweep of the queued transactions will be performed at the interval in seconds given above.

⟨ Cluster Transaction Retry Time Interval 8a ⟩ ≡
```
45◇
```
Macro referenced in 404.

If a cluster synchronisation transaction fails, the destination host is placed in a list of failed hosts. Transactions for that host will be queued until the above timeout in seconds has elapsed, after which they will be retried. This interval should be a multiple of the synchronisation time interval given above.

⟨ Cluster Failed Transaction Retry Interval 8b ⟩ ≡
```
60◇
```
Macro referenced in 402.

Cluster synchronisation transactions which fail due to race conditions or other transient causes will be retried opportunistically as other transactions arrive at the interval in seconds specified above. Note that if the transaction rate is low, the actual retry time will be that of the Cluster Synchronisation Time Interval should no other transactions arrive.

⟨ Cluster Failed Transaction Maximum Retries 8c ⟩ ≡
```
5◇
```
Macro referenced in 402.

A failed cluster synchronisation transaction will be retried this number of times (at the Cluster Failed Transaction Retry Interval or greater), after which the transaction will be abandoned as hopeless and deleted from the transaction directory.

⟨ Cluster Synchronisation Signal 8d ⟩ ≡
```
USR1◇
```
Macro referenced in 394, 397a.

When a transaction is placed on the cluster synchronisation queue, the following signal will be sent to the synchronisation process to inform it that work is available. This is usually `SIGUSR1`, but you can change it if necessary. The signal is specified as the Perl symbolic key for the `%SIG` hash.

⟨ Cluster Synchronisation Process ID File 8e ⟩ ≡
```
/server/run/ClusterSync/ClusterSync.pid◇
```
Macro referenced in 394, 397a, 398b.

The process ID of the cluster synchronisation process will be written to the file named above to allow transaction processors to send signals to alert it that work for it has been queued.

⟨ Cluster Synchronisation Log File 8f ⟩ ≡
```
/server/log/hackdiet/ClusterSync.log◇
```
Macro referenced in 399ab.

The cluster synchronisation process will write its log to the file specified above. To disable logging, define this as the null string. The log file will be closed and re-opened when the cluster synchronisation process receives a `HUP` signal, permitting cycling of the log.

⟨ Cluster Synchronisation User ID 8g ⟩ ≡
```
apache◇
```
Macro referenced in 398a.

⟨ Cluster Synchronisation Group ID 9a ⟩ ≡

```
apache◇
```

Macro referenced in 398a.

If initially started as the super-user (for example, from an `init` script), the cluster synchronisation process will assume the group and user identity (both effective and real) given above and perform all operations as that user. This should usually be set to the identity under which your Web server runs CGI programs. If the null string is specified, no identity change will be performed, nor will the change be attempted unless the synchronisation process is initially running as super-user.

⟨ Characters Permissible in File Names 9b ⟩ ≡

```
[\w\x{c0}-\x{d6}\x{d8}-\x{dd}\x{df}\x{e0}-\x{f6}\x{f8}-\x{fd}\x{ff}]◇
```

Macro referenced in 137.

This is a regular expression which matches ISO 8859-1 characters which are permissible in system file names. All characters which are not matched by this expression (and all Unicode character with code points of U+0100 and above) will be encoded for appearance in file names by expressing the code point in hexadecimal and enclosing the value in the delimiters defined below, which *must not* be included in the permissible characters defined above.

If you include space among the permissible characters, file names containing spaces will be generated. If space is not permitted, it is encoded as:

⟨ Encoding for Space in File Name Characters 9c ⟩ ≡

```
+◇
```

Macro referenced in 137.

which, of course, must not be a permissible character.

If the underlying system, like most modern server file systems, permits almost any characters in file names, the setting of the variable above is a trade-off between closer correspondence between the name the user types to log in and the name of the directory containing their file on the one hand, and potential difficulty in entering such file names on the part of the system administrator on the other. On the gripping hand, in most cases you can cut and paste file name characters from directory listings or wild-card problematic characters, so difficulty in entering them from the keyboard isn't such a big thing. Restricting the permissible characters to pure ASCII alphanumerics is tempting, but it means that names with ISO accented characters will turn into ugly file names.

One crucial issue is that if the underlying file system does not distinguish upper and lower case characters (the technical name for such a file system is a "pile of crap"), then you *must* restrict the permissible characters to one letter case (usually lower, as they're more common in typical user names), and force the other to be encoded. Otherwise, you'll encounter name collisions between names which differ only in the case of one or more letters.

⟨ Left Delimiter for Quoted File Name Characters 9d ⟩ ≡

```
{◇
```

Macro referenced in 137.

⟨ Right Delimiter for Quoted File Name Characters 9e ⟩ ≡

```
}◇
```

Macro referenced in 137.

⟨ Maximum Text Input Field Length 9f ⟩ ≡

```
4096◇
```

Macro referenced in 35a, 118, 121ab, 122, 123ab, 187, 312, 320, 327, 359.

The following defines the absolute path used to invoke the Sendmail mail transfer agent on the system. This is used by the `user::sendMail` method to send electronic mail to a user's configured address.

⟨ Path to Invoke Sendmail 10a ⟩ ≡

```
/usr/lib/sendmail◇
```

Macro referenced in 128, 354, 355.

Mail sent to users will contain the following "From" address by default. It is usually configured to a bit-bucket address which discards inadvertent replies.

⟨ From address for mail sent to users 10b ⟩ ≡

```
noreply\@fourmilab.ch◇
```

Macro referenced in 128, 354, 355.

When a user requests a password reset, a new password will be assigned whose length is as specified below.

⟨ Address for feedback E-mail 10c ⟩ ≡

```
bitbucket@fourmilab.ch◇
```

Macro referenced in 354.

Feedback E-mail is sent to the above address. This is generally a "wave through" address which bypasses junk mail filters.

⟨ Maximum line length in feedback E-mail messages 10d ⟩ ≡

```
64◇
```

Macro referenced in 351, 356.

Message body lines in feedback E-mail messages will be wrapped (if possible, only breaking lines at white space) so as not to exceed the above line length.

⟨ Categories of feedback messages 10e ⟩ ≡

```
        '(Not specified)',
        'Problem report',
        'Recommendation for change',
        'Suggestion for new feature',
        'How do I...?',
        'Documentation or usage question',
        'General comment'
    ◇
```

Macro referenced in 370b.

The above categories are presented to the user when sending a feedback message. You are free to add categories or change the order, as long as the "(Not specified)" item remains first in the list.

⟨ Command to check spelling 10f ⟩ ≡

```
'aspell list --encoding=utf-8 --mode=none | sort -u'◇
```

Macro referenced in 351.

This command will be used to check the spelling of feedback messages (including the subject line) when the user opts to preview them. The command must accept and emit UTF-8 encoded data, but there is no requirement that the output be formatted in any particular way; it is just taken as a list of words split arbitrarily across lines.

⟨ Length of automatically generated passwords 10g ⟩ ≡

```
8◇
```

Macro referenced in 188.

When the user performs a password reset, a new password of this length will be generated.

⟨ Cookie name 11a ⟩ ≡
```
'HDiet'◇
```
Macro referenced in 151, 172, 178.

Cookies for persistent logins will be generated with the name above.

⟨ Default cookie retention time 11b ⟩ ≡
```
(90 * 24 * 60 * 60)◇
```
Macro referenced in 147.

This value is the default retention time for login cookies, in seconds. The standard value is 90 days, which remembers the login token for about three months. Note that a new cookie is assigned at each login, so this only affects how long a "remember me" is effective when the user has not logged in from the browser in which the cookie has been set.

⟨ Domain for cookies 11c ⟩ ≡
```
.fourmilab.ch◇
```
Macro referenced in 150ab.

⟨ Path for cookies 11d ⟩ ≡
```
/cgi-bin/HackDiet◇
```
Macro referenced in 150ab.

The path and domain above is used to tag cookies set by the "remember me" facility.

⟨ Remember Me Directory 11e ⟩ ≡
⟨ Database Directory 6f ⟩`/RememberMe◇`
Macro referenced in 151, 152, 300, 328a, 329.

Directory in which cookies for persistent logins are stored.

2.9 Web URL Addresses

⟨ Site home URL 11f ⟩ ≡
```
http://www.fourmilab.ch◇
```
Macro referenced in 11g, 190b, 233, 409, 427.

This is the absolute URL of the hosting site, which is used for the link on the site logo. This should be an absolute URL and specify the HTTP protocol. Otherwise, if the user is running the application over HTTPS and goes to the home page, the connection will remain secure, which is unnecessary and needlessly burdens the server.

⟨ Book home URL 11g ⟩ ≡
⟨ Site home URL 11f ⟩`/hackdiet◇`
Macro referenced in 11h, 409, 427.

This is the home directory of *The Hacker's Diet* book. It is usually based upon the site home, and for the reasons given above should also specify the HTTP protocol.

⟨ Application documentation URL 11h ⟩ ≡
⟨ Book home URL 11g ⟩`/online/hdo.html◇`
Macro referenced in 409.

This is *The Hacker's Diet Online* application documentation.

The URL (either relative or absolute) used to invoke the CGI program is specified below:

⟨ URL to invoke this program 12a ⟩ ≡

```
/cgi-bin/HackDiet◇
```

Macro referenced in 12b, 118, 179, 180b, 181, 182, 183, 189, 190ab, 195, 196, 198, 209, 211, 212, 213a, 233, 234, 239, 249, 251, 281b, 297, 299, 304, 305, 314, 315, 316, 317, 318, 319, 324, 325, 326, 330, 353, 360, 361ab, 362b, 364a, 365, 366ab, 367ac, 410a, 455c.

The following `method` and `action` will be used to submit forms for processing. The `action` must name the path to the CGI program which responds to requests.

⟨ Form processing action and method 12b ⟩ ≡

```
method="post" action="⟨ URL to invoke this program 12a ⟩"◇
```

Macro referenced in 117, 181, 182, 183, 185, 187, 191, 196, 210, 214, 228, 237, 245, 257, 261, 280, 288a, 294, 295, 299, 301, 306, 307, 309, 320, 327, 331, 341, 349, 357, 358b, 363.

Chapter 3

hdCSV.pm Extended CSV File Parser

We import and export database files in an extended dialect of "comma-separated value" almost-flat files. Fields in these files need not be quoted unless they:

- Have leading or trailing spaces.
- Contain a comma or quote (") character.
- Include a character not in the ISO-8859-1 graphic character set.

If one or more of these conditions applies, the field will be enclosed in ASCII quote characters, with any embedded quotes expanded to two consecutive quotes. Non-graphic characters and Unicode characters above U+00FF are encoded as in a Perl string: as \x{*hexval*}; backslashes are escaped as two consecutive backslashes.

3.1 Package plumbing

"HDiet/hdCSV.pm" 13 ≡

```
#! ⟨ Perl directory 7d ⟩

    ⟨ Perl language modes 369b ⟩

    package HDiet::hdCSV;

    require Exporter;

    our @ISA = qw(Exporter);
    our @EXPORT = qw(parseCSV encodeCSV);
    1;
◇
```

File defined by 13, 14, 15.

3.2 ParseCSV

The `parseCSV` function is called with a CSV record (which may contain trailing white space, including an end of line sequence). Fields from the record are parsed and stored into an array, which is returned in list context.

"HDiet/hdCSV.pm" 14 ≡

```
sub parseCSV {
    my ($s) = @_;

    my @fields;
    my $f;

    $s =~ s/\s+$//;
    $s =~ s/,$/,""/;

    while ($s ne '') {
        if ($s =~ s/^\s*"((?:""|[^"])*)"\s*,?//) {

            $f = $1;
            $f =~ s/""/"/g;
            my $uf = '';
            while ($f =~ s/^(.)//) {
                my $c = $1;
                if ($c eq "\\") {
                    if ($f =~ s/\\//) {
                        $uf .= "\\";
                    } elsif ($f =~ s/x{([0-7a-fA-F]+)}//) {
                        $uf .= chr(hex($1));
                    } else {
                        print(STDERR "Undefined backslash escape \\$f in CSV record.\n");
                    }
                } else {
                    $uf .= $1;
                }
            }
            $f = $uf;
        } else {
            $s =~ s/^\s*([^,]*),?//;
            $f = $1;
            $f =~ s/\s+$//
        }

        push(@fields, $f);
    }

    return @fields;
}
```

◇

File defined by 13, 14, 15.

3.3 EncodeCSV

The `encodeCSV` is called with a list of fields to be encoded into a CSV file, which may either be a sequence of arguments or an array argument. The single-line CSV record is returned, with no end of line sequence appended.

"HDiet/hdCSV.pm" 15 ≡

```
sub encodeCSV {
    my $f;
    my $s = '';

    while (defined($f = shift)) {

        #   Encode any non-ISO-8859-1 graphic characters
        #   (including wide characters) as hexadecimal escape
        #   sequences and force any backslashes in the
        #   string.

        my $ef = '';
        my $forced = 0;
        while ($f =~ s/^(.)//) {
            my $o = ord($1);
            if (($o < 32) ||
                (($o >= 127) && ($o < 161)) ||
                ($o > 255)) {
                $ef .= sprintf("\\x{%lx}", $o);
                $forced = 1;
            } elsif ($1 eq "\\") {
                $ef .= "\\\\";
                $forced = 1;
            } else {
                $ef .= $1;
            }
        }

        #   If the field contains leading or trailing white
        #   space, an embedded comma, quote, or an escaped character,
        #   force quotes within it and enclose in quotes.

        if (($ef =~ m/^\s/) || ($ef =~ m/\s$/) ||
            ($ef =~ m/,/) || ($ef =~ m/"/) || $forced) {
            $ef =~ s/"/""/g;
            $ef = '"' . $ef . '"';
        }

        $s .= $ef . ',';
    }

    $s =~ s/,$//;
    return $s;
}
```

◇

File defined by 13, 14, 15.

Chapter 4

`trendfit.pm`: Trend Fitter Object

The `trendfit` object fits a linear trend to a sequence of values via linear regression with the least squares method.

4.1 Package plumbing

"HDiet/trendfit.pm" 16a ≡

```
#! ⟨ Perl directory 7d ⟩

    ⟨ Perl language modes 369b ⟩

    package HDiet::trendfit;

    require Exporter;

    our @ISA = qw(Exporter);
    our @EXPORT = qw(new start addPoint fitSlope);
    1;
◇
```

File defined by 16ab, 17ab, 18ab.

4.2 Constructor

A new `trendfit` object is created by calling the `new` constructor.

"HDiet/trendfit.pm" 16b ≡

```
    sub new {
        my $self = {};
        my ($invocant) = @_;
        my $class = ref($invocant) || $invocant;

        bless($self, $class);

        $self->start();

        return $self;
    }
◇
```

File defined by 16ab, 17ab, 18ab.

4.3 Start

The `start` method resets the `trendfit` object to compute a new trend. This is called by the constructor, so you needn't explicitly call this method unless you've already added some points and wish to start over.

"HDiet/trendfit.pm" 17a ≡

```
        sub start {
            my $self = shift;

            $self->{n} = 0;
            $self->{s1} = $self->{s2} = $self->{s3} = $self->{s4} = 0;
            $self->{min} = 1E308;
            $self->{max} = -1E308;
        }
    ◇
```

File defined by 16ab, 17ab, 18ab.

4.4 Add Point

The `addPoint` method adds one or more point values specified by the argument to the trend we're fitting.

"HDiet/trendfit.pm" 17b ≡

```
        sub addPoint {
            my $self = shift;

            my $v;
            foreach $v (@_) {
                $self->{s1} += ($self->{n} + 1) * $v;
                $self->{s2} += ($self->{n} + 1);
                $self->{s3} += $v;
                $self->{s4} += ($self->{n} + 1) ** 2;
                $self->{n}++;
                $self->{min} = ::min($self->{min}, $v);
                $self->{max} = ::max($self->{max}, $v);
            }
        }
    ◇
```

File defined by 16ab, 17ab, 18ab.

4.5 Fit Slope

The `fitSlope` method fits a linear trend to the points supplied so far and returns its slope. You are free to continue adding points after returning the trend. If too few points have been specified to fit a linear trend, we return a slope of zero.

"HDiet/trendfit.pm" 18a ≡

```
        sub fitSlope {
            my $self = shift;

            my $denom = (($self->{s4} * $self->{n}) - ($self->{s2} ** 2));
            return 0 if $denom == 0;
            return (($self->{s1} * $self->{n}) - ($self->{s2} * $self->{s3})) /
                    $denom;
        }
```

◇

File defined by 16ab, 17ab, 18ab.

4.6 Minimum, Maximum, and Mean

The `minMaxMean` method returns a list containing the minimum, maximum, and mean values of the points added so far. This may be called at any point and does not disturb the computation of subsequently added points.

"HDiet/trendfit.pm" 18b ≡

```
        sub minMaxMean {
            my $self = shift;

            return ($self->{min}, $self->{max}, ($self->{s3} / $self->{n}));
        }
```

◇

File defined by 16ab, 17ab, 18ab.

Chapter 5

monthlog.pm: Monthly Log Object

The monthlog object encapsulates almost all of the mechanics of processing monthly weight and exercise logs.

5.1 Package plumbing

"HDiet/monthlog.pm" 19 ≡

```
#! ⟨ Perl directory 7d ⟩

    ⟨ Perl language modes 369b ⟩

    use HDiet::trendfit;

    package HDiet::monthlog;

    use HDiet::hdCSV;
    use HDiet::Julian qw(WEEKDAY_NAMES :DEFAULT);
    use HDiet::html;
    use HDiet::xml;
    use GD;

    require Exporter;
    our @ISA = qw(Exporter);
    our @EXPORT = qw(
                      WEIGHT_KILOGRAM WEIGHT_POUND WEIGHT_STONE
                      ENERGY_CALORIE ENERGY_KILOJOULE
                      WEIGHT_CONVERSION ENERGY_CONVERSION
                      CALORIES_PER_WEIGHT_UNIT
                      );
    our %EXPORT_TAGS = (
                           units => [ qw(WEIGHT_KILOGRAM WEIGHT_POUND WEIGHT_STONE
                                         ENERGY_CALORIE ENERGY_KILOJOULE) ]
                       );
    1;

    ⟨ Constants and conversion tables 20 ⟩

    ⟨ Minimum, Maximum, and Sign functions 384 ⟩

◇
```

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.1.1 Constants and conversion tables

We define the following symbolic constants for quantities used in the monthly log object. Conversion tables between the various weight and energy units are also defined as constants.

⟨ Constants and conversion tables 20 ⟩ ≡

```
use constant FILE_VERSION => 1;
use constant WEIGHT_KILOGRAM => 0;
use constant WEIGHT_POUND => 1;
use constant WEIGHT_STONE => 2;
use constant WEIGHT_UNITS => [ "kilogram", "pound", "stone" ];
use constant DELTA_WEIGHT_UNITS => [ "kilogram", "pound", "pound" ];
use constant DELTA_WEIGHT_ABBREVIATIONS => [ "kg", "lb", "lb" ];
use constant WEIGHT_ABBREVIATIONS => [ "kg", "lb", "st" ];
use constant CALORIES_PER_WEIGHT_UNIT => [ 7716, 3500, 3500 ];

use constant WEIGHT_CONVERSION => [
#   Entries for pounds and stones are identical because
#   even if stones are selected, entries in log items are
#   always kept in pounds.
#
#  To:         kg              lb             st
#                                                      From
          [ 1.0,            2.2046226,     2.2046226    ],  #   kg
          [ 0.45359237,     1.0,           1.0          ],  #   lb
          [ 0.45359237,     1.0,           1.0          ]   #   st
];


use constant ENERGY_CALORIE => 0;
use constant ENERGY_KILOJOULE => 1;
use constant ENERGY_UNITS => [ "calorie", "kilojoule" ];
use constant ENERGY_ABBREVIATIONS => [ "cal", "kJ" ];
use constant CALORIES_PER_ENERGY_UNIT => [ 1, 0.239045 ];
use constant ENERGY_CONVERSION => [
#
#   To:         cal         kJ                From
             [   1.0,        4.18331  ],     #   cal
             [   0.239045,   1.0      ]      #   kJ
];

use constant RUNG_MAX => 48;
```

◇

Macro referenced in 19.

5.2 Constructor

A new `monthlog` object is created by calling the `new` constructor. The following arguments may be supplied, all of which are optional; omitted arguments are filled in with zeroes or blanks as appropriate.

login_name	User login name
year	Gregorian year
month	Month number (1–12)
log_unit	Weight unit
trend_carry_forward	Trend at start of month
last_modification_time	UNIX time of last modification

"HDiet/monthlog.pm" 21 ≡

```
sub new {
    my $self = {};
    my ($invocant, $login_name, $year, $month, $log_unit,
        $trend_carry_forward, $last_modification_time) = @_;
    my $class = ref($invocant) || $invocant;

    $login_name = '' if !defined($login_name);
    $year = 0 if !defined($year);
    $month = 0 if !defined($month);
    $log_unit = WEIGHT_KILOGRAM if !defined($log_unit);
    $trend_carry_forward = 0 if !defined($trend_carry_forward);
    $last_modification_time = 0 if !defined($last_modification_time);

    bless($self, $class);

    $self->{version} = FILE_VERSION;

    #   Initialise instance variables from constructor arguments
    $self->{login_name} = $login_name;
    $self->{year} = $year;
    $self->{month} = $month;
    $self->{log_unit} = $log_unit;
    $self->{trend_carry_forward} = $trend_carry_forward;
    $self->{last_modification_time} = $last_modification_time;

    $self->{weight} = [];                   # Create empty weight array
    $self->{rung} = [];                     # Create empty exercise rung array
    $self->{flag} = [];                     # Create empty flag array
    $self->{comment} = [];                  # Create empty comment array
    $self->{trend} = [];                    # Create empty trend array
    $self->{verbose} = 0;                   # Default to non-verbose mode

    return $self;
}
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.3 Destructor

The destructor cleans up when an object is deallocated. Actually, regular reference count garbage collection handles everything just fine without a destructor, but I've provided for one in case we do something fancy with the database which may eventually require one.

"HDiet/monthlog.pm" 22 ≡

```
sub DESTROY {
    my $self = shift;

    if ($self->{verbose}) {
        print("monthlog: Destructor invoked\n");
    }
    undef($self->{weight});
    undef($self->{rung});
    undef($self->{flag});
    undef($self->{comment});
    undef($self->{trend});

}
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.4 Describe

The `describe` method prints a primate-readable description of the monthly log on the file handle (default `STDOUT`) given by the argument.

"HDiet/monthlog.pm" 23 ≡

```
sub describe {
    my $self = shift;
    my ($outfile) = @_;

    if (!(defined $outfile)) {
        $outfile = \*STDOUT;
    }

    print($outfile "MONTHLOG Version: $self->{version}\n");
    print($outfile "  Login: '$self->{login_name}'  Year: $self->{year}  " .
        "Month: $self->{month}  " .
        "Log unit: " . WEIGHT_UNITS->[$self->{log_unit}] ."\n");
    print($outfile "  Trend carry-forward: $self->{trend_carry_forward}\n" .
        "  Last modification time: " . localtime($self->{last_modification_time}) .
        "\n");
    print($outfile "  Days in month: " . $self->monthdays() . "\n");

    for (my $i = 1; $i <= $self->monthdays(); $i++) {
        my $dw = defined($self->{weight}[$i]) ? sprintf("%6.2f", $self->{weight}[$i]) : "      ";
        my $dt = defined($self->{trend}[$i]) ? sprintf("%6.2f", $self->{trend}[$i]) : "      ";
        my $dr = defined($self->{rung}[$i]) ? sprintf("%2d", $self->{rung}[$i]) : "  ";
        my $df = defined($self->{flag}[$i]) ? sprintf("%1d", $self->{flag}[$i]) : " ";
        my $dc = defined($self->{comment}[$i]) ? "  $self->{comment}[$i]" : "";

        printf($outfile "   %2d  $dw  $dt  $dr  $df$dc\n", $i);
    }
}
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.5 ComputeTrend

The `computeTrend` method calculates the exponentially smoothed moving average trend for days in the month, starting with the trend carried forward from the previous month (or the first entry in this log, is no trend carry forward is specified). The slope of a linear regression fit to the trend (truncated at the last specified data in the month) is returned.

If there are insufficient data points in the log from which to extrapolate a trend, a slope of zero is returned.

"HDiet/monthlog.pm" 24 ≡

```
sub computeTrend {
    my $self = shift;

    my $t = $self->{trend_carry_forward};
    my $n = $self->monthdays();

    if ($t == 0) {
        for (my $i = 1; $i <= $n; $i++) {
            if (defined($self->{weight}[$i]) && ($self->{weight}[$i] > 0)) {
                $t = $self->{weight}[$i];
                last;
            }
        }
    }

    if ($t > 0) {
        for (my $i = 1; $i <= $n; $i++) {
            if (defined($self->{weight}[$i]) && ($self->{weight}[$i] > 0)) {
                $t = $t + (($self->{weight}[$i] - $t) / 10);
            }
            $self->{trend}[$i] = $t;
        }
    }

    my $nd = $n;

    while (($nd >= 0) && (!defined($self->{weight}[$nd]))) {
        $nd--;
    }

    if ($nd <= 1) {
        return 0;
    }

    my $fitter = HDiet::trendfit->new();
    for (my $i = 1; $i <= $nd; $i++) {
        $fitter->addPoint($self->{trend}[$i]);
    }
    return $fitter->fitSlope();
}
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.6 BodyMassIndex

This method computes the Body Mass Index (BMI), which is defined as w/h^2 where w is the weight in kilograms and h the height in metres. The first argument specifies the user's height in centimetres, which is how we store it in the `user` object.

If called with no second argument, the `bodyMassIndex` method computes the mean body mass index for the month. If called with a second argument, it returns the body mass index for that day. If no weight entry exists for that day (or for the entire month, if the mean is requested), or the user has not specified their height, zero is returned. If called with a negative second argument, the body mass index for the last day of the month for which a weight entry is present will be returned (or zero if the log is entirely empty).

Note that the body mass index is always computed from the *trend*, not the unsmoothed weight; this makes it more stable and consistent. The standard interpretation of body mass index numbers for adults is as follows:

BMI	Weight Status
< 18.5	Underweight
18.5–24.9	Normal
25.0–29.9	Overweight
≥ 30.0	Obese

"HDiet/monthlog.pm" 25 ≡

```
sub bodyMassIndex {
    my $self = shift;

    my ($height, $day) = @_;

    $day = 0 if !defined($day);
    return 0 if $height == 0;

    my $n = $self->monthdays();
    my $weight = 0;

    if ($day <= 0) {
        my $nd = 0;
        for (my $i = 1; $i <= $n; $i++) {
            if (defined($self->{weight}[$i]) && ($self->{weight}[$i])) {
                if ($day < 0) {
                    $weight = $self->{trend}[$i];
                    $nd = 1;
                } else {
                    $weight += $self->{trend}[$i];
                    $nd++;
                }
            }
        }
        $weight /= $nd if $nd > 0;
    } else {
        $weight = $self->{weight}[$day] if defined($self->{weight}[$day]);
    }

    $weight *= WEIGHT_CONVERSION->[$self->{log_unit}][WEIGHT_KILOGRAM];
    return sprintf("%.1f", $weight / ($height / 100) ** 2);
}
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.7 FractionFlagged

The `fractionFlagged` method returns the fraction (between 0 and 1) of the days in the log whose flag field are checked,

"HDiet/monthlog.pm" 26 ≡

```
sub fractionFlagged {
    my $self = shift;

    my $n = $self->monthdays();
    my $nf = 0;

    for (my $i = 1; $i <= $n; $i++) {
        if ($self->{flag}[$i]) {
            $nf++;
        }
    }

    return $nf / $n;
}
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.8 Save

The `save` method writes the log item to the already-open file handle passed as the argument. All archival information in the log is preserved by a `save` followed by a `load`.

"HDiet/monthlog.pm" 27 ≡

```
sub save {
    my $self = shift;
    my ($outfile) = @_;

    #   File format version number
    print($outfile "$self->{version}\n");
    #   Year, Month, Log unit
    print($outfile "$self->{year},$self->{month},$self->{log_unit}\n");
    #   Trend carry-forward, Last modification time
    print($outfile "$self->{trend_carry_forward},$self->{last_modification_time}\n");
    my $md = $self->monthdays();
    #   Weight array
    for (my $i = 1; $i <= $md; $i++) {
        print($outfile (dnz($self->{weight}[$i]) ? $self->{weight}[$i] : ''));
        print($outfile (($i < $md) ? ',' : "\n"));
    }
    #   Rung array
    for (my $i = 1; $i <= $md; $i++) {
        print($outfile (dnz($self->{rung}[$i]) ? $self->{rung}[$i] : ''));
        print($outfile (($i < $md) ? ',' : "\n"));
    }
    #   Flag array
    for (my $i = 1; $i <= $md; $i++) {
        print($outfile (dnz($self->{flag}[$i]) ? $self->{flag}[$i] : ''));
        print($outfile (($i < $md) ? ',' : "\n"));
    }

    #   Comments
    print($outfile $self->encodeComments() . "\n");
}
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.9 Load

The `load` reads a file from the argument file handle in the format produced by `save` and stores the values into the monthly log object, which is assumed to be a new void object. It does not, for example, undefine existing values in the weight array when those values are unspecified in the file being loaded. We start by validating the file version.

"HDiet/monthlog.pm" 28a ≡

```
sub load {
    my $self = shift;
    my ($infile) = @_;

    my $s = in($infile);

    if ($s != FILE_VERSION) {
        die("monthlog::load: Incompatible file version $s");
    }
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.9.1 Parse year, month, and log unit

The second record in the file gives the year and month this log represents and the weight unit used within.

"HDiet/monthlog.pm" 28b ≡

```
    $s = in($infile);
    $s =~ m/^(\d+),(\d+),(\d+)$/ || die("monthlog::load: Error parsing year, month, log unit");
    $self->{year} = $1;
    $self->{month} = $2;
    $self->{log_unit} = $3;
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.9.2 Parse trend carry forward and last modification time

The third record contains the trend value carried forward from the previous log item (or zero if this is the first log in the user database), and the UNIX `time` of the last modification to the log.

"HDiet/monthlog.pm" 28c ≡

```
    $s = in($infile);
    $s =~ m/^([\d\.]+),(\d+)$/ || die("monthlog::load: Error parsing trend carry forward, last modificatio
    $self->{trend_carry_forward} = $1;
    $self->{last_modification_time} = $2;
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.9.3 Parse daily weight array

The fourth record contains the daily weight entries in the unit given by the second record. Weight entries are decimal numbers separated by commas. Unspecified entries are void.

"HDiet/monthlog.pm" 29a ≡

```
            my $md = $self->monthdays();

            $s = in($infile);
            for (my $i = 1; $i <= $md; $i++) {
                $s =~ s/^([\d\.]*),?// || die("monthlog::load: Error parsing weight for day $i");
                if ($1 ne '') {
                    $self->{weight}[$i] = $1;
                }
            }
            if ($s ne '') {
                die("monthlog::load: Residual characters ($s) after parsing weights");
            }

            $self->computeTrend();      # Fill in daily trend now that weights are known
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.9.4 Parse exercise rung array

The fifth record gives the exercise rung for each day of the month, specified as a number from 1 to `RUNG_MAX`, separated by commas. Blank entries are void.

It waw previously possible for a craftily coded CSV import to create a rung value with one or more leading blanks, which would cause our strict parser to fail. To avoid application crashes from this, we ignore any blanks in the rung record.

"HDiet/monthlog.pm" 29b ≡

```
            $s = in($infile);
            $s =~ s/\s//g;
            for (my $i = 1; $i <= $md; $i++) {
                $s =~ s/^(\d*),?// || die("monthlog::load: Error parsing rung for day $i");
                if ($1 ne '') {
                    $self->{rung}[$i] = $1;
                }
            }
            if ($s ne '') {
                die("monthlog::load: Residual characters ($s) after parsing rungs");
            }
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.9.5 Parse flag array

The sixth record specifies the general purpose flag items, which are the number 1 if the day is flagged and a void entry otherwise.

"HDiet/monthlog.pm" 30a ≡

```
            $s = in($infile);
            for (my $i = 1; $i <= $md; $i++) {
                $s =~ s/^(\d*),?// || die("monthlog::load: Error parsing flag for day $i");
                if ($1 ne '') {
                    $self->{flag}[$i] = $1;
                }
            }
            if ($s ne '') {
                die("monthlog::load: Residual characters ($s) after parsing flags");
            }
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.9.6 Parse comments

Finally, the seventh record specifies the comments for days in the month, if any, in the format created by `encodeComments`. This format is optimised for the common case of identical comments for multiple days in the month.

"HDiet/monthlog.pm" 30b ≡

```
            $s = in($infile);
            $self->decodeComments($s);
        }

        ⟨ Read line from persistent object file (30c monthlog ) 390b ⟩
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.10 ToHTML

The `toHTML` method generates an HTML table containing the data in the current monthly log. The first argument is the file handle to which the table should be written. The second and third arguments specify the range of days which should be rendered as editable form fields rather than static data. If an all-static table is desired, these arguments may be omitted or set to zero. If `$browse_public` is true, a completely static table will be generated and no comments will be shown.

"HDiet/monthlog.pm" 31 ≡

```
sub toHTML {
    my $self = shift;

    my ($fh, $efirst, $elast, $display_unit,
        $decimal_character, $browse_public,
        $printFriendly, $monochrome) = @_;

    $efirst = 0 if !defined($efirst) || $printFriendly;
    $elast = 0 if !defined($elast) || $printFriendly;

    my $printfix = ($printFriendly ? 'pr_' : '') . ($monochrome ? 'mo_' : '');

    my $n = $self->monthdays();

    my $logToDisplayUnit = WEIGHT_CONVERSION->[$self->{log_unit}][$display_unit];

    ⟨ Write HTML table header 32a ⟩

    my $lastweight;
    for ($lastweight = $n; $lastweight >= 1; $lastweight--) {
        if (znd($self->{weight}[$lastweight])) {
            last;
        }
    }

    my $wday = jd_to_weekday(gregorian_to_jd($self->{year}, $self->{month}, 1));
    for (my $i = 1; $i <= $n; $i++) {
        my $edit = (!$browse_public) && ($i >= $efirst) && ($i <= $elast);

        print($fh "<tr>\n");

        ⟨ Generate date column 32b ⟩
        ⟨ Generate weight, trend, and variance columns 33 ⟩
        ⟨ Generate exercise rung column 34a ⟩
        ⟨ Generate flag column 34b ⟩
        if (!$browse_public) {
            ⟨ Generate comment column 35a ⟩
        }

        print($fh "</tr>\n");
    }

    ⟨ Write HTML table footer 35b ⟩

}
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.10.1 Write HTML table header

The HTML table header is written, along with the column headings.

⟨ Write HTML table header 32a ⟩ ≡

```
    print $fh <<"EOD";
<table border="border" class="${printfix}mlog">
<tr>
<th colspan="2">Date</th>
<th>Weight</th>
<th>Trend</th>
<th>Var.</th>
<th>Rung</th>
<th>Flag</th>
EOD

    if (!$browse_public) {
        print $fh <<"EOD";
<th>Comments</th>
EOD
    }

    print $fh <<"EOD";
</tr>
EOD
```

◇

Macro referenced in 31.

5.10.2 Generate date column

The date (day within month and weekday abbreviation) is written as the first column in the table.

⟨ Generate date column 32b ⟩ ≡

```
    print($fh "<th>$i</th>\n");     # Day
    print($fh "<td>" . substr(WEEKDAY_NAMES->[$wday], 0, 3) . "</td>\n"); # Weekday
    $wday = ($wday + 1) % 7;
```

◇

Macro referenced in 31.

5.10.3 Generate weight, trend, and variance columns

The next three columns give that day's weight, the trend value, and the colour-coded variance between the weight and trend. The trend is left blank for days between the last weight entry and the end of the log, and the variance is displayed only for days with a weight entered.

⟨ Generate weight, trend, and variance columns 33 ⟩ ≡

```
# Weight
print($fh "<td>");
if ($edit) {
    print($fh "<input type=\"text\" name=\"w$i\"  id=\"w$i\" size=\"6\" value=\"" .
        wgt(znd($self->{weight}[$i]) * $logToDisplayUnit,
            $display_unit, $decimal_character) .
        "\" onchange=\"changeWeight($i);\" />" .
    "<input type=\"hidden\" id=\"W$i\" value=\"" .
    (znd($self->{weight}[$i]) ? fixo(znd($self->{weight}[$i]) * $logToDisplayUnit, 4)
                              : '') . "\" />");
} else {
    print($fh wgt(znd($self->{weight}[$i]) * $logToDisplayUnit,
        $display_unit, $decimal_character));
}
print($fh "</td>\n");

# Trend
print($fh "<td id=\"t$i\">" .
    wgt(($i > $lastweight) ? undef :
        (znd($self->{trend}[$i]) * $logToDisplayUnit), $display_unit, $decimal_character, 1) .
        "<input type=\"hidden\" id=\"T$i\" value=\"" . fixo(znd($self->{trend}[$i]) *
        $logToDisplayUnit, 4) . "\" />" .
    "</td>\n");

 # Variance
my $var = (defined($self->{weight}[$i]) && defined($self->{trend}[$i]) &&
            ($self->{weight}[$i] > 0) && ($self->{trend}[$i] > 0)) ?
                (($self->{weight}[$i] - $self->{trend}[$i]) * $logToDisplayUnit) : undef;
print($fh "<td class=\"r\"><span id=\"v$i\" class=\"" . $printfix .
             ((defined($var) && (sprintf("%.1f", $var) !~ m/^\-?0\.0$/)) ?
                (($var < 0) ? "g" : "r") : "bk") . "\">" .
                var($var, $decimal_character) . "</span></td>\n");
```

◇

Macro referenced in 31.

5.10.4 Generate exercise rung column

Next comes the exercise rung, a number between 1 and `RUNG_MAX`, or blank if no exercise was done that day.

⟨ Generate exercise rung column 34a ⟩ ≡

```
        print($fh "<td>");
        if ($edit) {
            print($fh "<input type=\"text\" name=\"r$i\" id=\"r$i\" size=\"3\" value=\"" .
                bnd($self->{rung}[$i]) . "\" onchange=\"changeRung($i);\" />");
        } else {
            print($fh bnd($self->{rung}[$i]));
        }
        print($fh "</td>\n");
    ◇
```

Macro referenced in 31.

5.10.5 Generate flag column

Following the exercise rung is the utility flag column, which is shown as a check mark or blank. All we do with the flag is show the percentage of flagged days in the month. If we're generating a non-editable "Printer friendly" form, encode checked flags in `hidden` input fields so we don't lose them when the user clicks the "Update" button.

⟨ Generate flag column 34b ⟩ ≡

```
        print($fh "<td>");
        if ($edit) {
            print($fh "<input type=\"checkbox\" name=\"f$i\" id=\"f$i\" onclick=\"updateFlag($i);\"" .
                ($self->{flag}[$i] ? " checked=\"checked\"" : "") . " />");
        } else {
            if ($self->{flag}[$i]) {
                print($fh "<input type=\"hidden\" name=\"f$i\" id=\"f$i\" value=\"checked\" />");
            }
            print($fh $self->{flag}[$i] ? "&#10004;" : "");
        }
        print($fh "</td>\n");
    ◇
```

Macro referenced in 31.

5.10.6 Generate comment column

Finally, we come to the comment, which is simply arbitrary text.

⟨ Generate comment column 35a ⟩ ≡

```
        print($fh "<td>");
        my $cmt = quoteHTML(defined($self->{comment}[$i]) ? $self->{comment}[$i] : "");
        if ($edit) {
            print($fh "<input type=\"text\" name=\"c$i\" id=\"c$i\" size=\"60\" " .
                      "maxlength=\"⟨ Maximum Text Input Field Length 9f ⟩\" " .
                      "value=\"$cmt\" onchange=\"changeComment($i);\" />");
        } else {
            print($fh $cmt);
        }
        print($fh "</td>\n");
    ◇
```

Macro referenced in 31.

5.10.7 Write HTML table footer

Close the current HTML table.

⟨ Write HTML table footer 35b ⟩ ≡

```
        print $fh <<"EOD";
    </table>
    EOD
    ◇
```

Macro referenced in 31.

5.10.8 Format weight to display unit

The `editWeight` function returns a string with its first weight argument formatted appropriate for the display unit specified by the second argument.

"HDiet/monthlog.pm" 36a ≡

```
sub editWeight {
    my ($weight, $unit, $dchar) = @_;

    $dchar = '.' if !defined($dchar);
    my $sgn = ($weight < 0) ? "-" : "";
    my $w = abs($weight);
    my $sw;
    if ($unit == WEIGHT_STONE) {
        $sw = sprintf("%s%d %2.1f", $sgn, int($w / 14), $w - (int($w / 14) * 14));
    } else {
        $sw = sprintf("%s%.1f", $sgn, $w);
    }
    $sw =~ s/\./$dchar/;
    return $sw;
}
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.10.9 Convert weight from one unit to another

The `convertWeight` function converts the `$weight` argument in the `$from` unit to a weight in the `$to` units. If is perfectly valid to call this function with the same from and to units; this has the salutary effect of converting the value into canonical form.

"HDiet/monthlog.pm" 36b ≡

```
sub convertWeight {
    my ($weight, $from, $to) = @_;

    $weight = canonicalWeight($weight * WEIGHT_CONVERSION->[$from][$to]);

    return $weight;
}
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.10.10 Express weight in canonical form

The `canonicalWeight` function expresses a weight quantity in our canonical form of at most two decimal places, with trailing zero decimal places removed, and no decimal if the value is integral.

"HDiet/monthlog.pm" 37a ≡

```
sub canonicalWeight {
    my ($weight) = @_;

    $weight = sprintf("%.2f", $weight);

    $weight =~ s/(\.[^0]*)0+$/$1/;
    $weight =~ s/\.$//;

    return $weight;
}
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.10.11 HTML generation utility functions

The following little functions simplify generating HTML from fields in the log.

5.10.11.1 Test if defined and nonzero

"HDiet/monthlog.pm" 37b ≡

```
sub dnz {
    my ($s) = @_;

    return defined($s) && ($s > 0);
}
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.10.11.2 Return null string if not defined or zero

"HDiet/monthlog.pm" 37c ≡

```
sub bnd {
    my ($s) = @_;

    return (defined($s) && ($s > 0)) ? $s : '';
}
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.10.11.3 Return zero if not defined

"HDiet/monthlog.pm" 38a ≡

```
        sub znd {
            my ($s) = @_;

            return (defined($s) && ($s > 0)) ? $s : 0;
        }
    ◇
```

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.10.11.4 Format as fixed point decimal

The first argument is formatted as a fixed point quantity with the number of decimal places given by the second. Trailing zeroes in the decimal part are discarded, as well as trailing decimal points on integral quantities.

"HDiet/monthlog.pm" 38b ≡

```
        sub fixo {
            my ($v, $places) = @_;
            my $s = sprintf("%.${places}f", $v);
            $s =~ s/0+$//;
            $s =~ s/\.$//;
            return $s;
        }
    ◇
```

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.10.11.5 Format as weight

The first argument is formatted as a weight in the unit given by the second or blank if not defined. The third argument specifies the user's preferred decimal separator character. If the fourth argument is nonzero, "` `" will be generated as a place-holder if the field is void.

"HDiet/monthlog.pm" 38c ≡

```
        sub wgt {
            my ($s, $dunit, $dchar, $nbsp) = @_;

            return (defined($s) && ($s > 0)) ? editWeight($s, $dunit, $dchar) :
                ($nbsp ? ' ' : '');
        }
    ◇
```

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.10.11.6 Format as variance

The argument is formatted as a signed weight variance, or blank if not defined. Zeroes are output without a sign. Note that the HTML "`&minus;`" sign is used for negative quantities, and consequently these numbers cannot be directly parsed by Perl. The second argument specifies the user's preferred decimal separator character,

"HDiet/monthlog.pm" 39 ≡

```
        sub var {
            my ($s, $dchar) = @_;

            my $v;
            if (defined($s)) {
                $v =  (($s < 0) ? "&minus;" : "+") . sprintf("%.1f", abs($s));
                $v = '0.0' if $v =~ m/\D0\.0$/;
                $v =~ s/\./$dchar/;
            } else {
                $v = ' ';
            }
            return $v;
        }
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.11 computeChartScale

The `computeChartScale` method is used to obtain the X and Y scale factors for the chart embedded in a monthly log document. These are used by the JavaScript code to plot entries made in log on the chart. The method re-uses the code of the actual chart plotter (`plotChart`, below) to auto-scale the plot, and then simply returns a string with the relevant scale factors suitable for embedding in a hidden form field of the chart document.

"HDiet/monthlog.pm" 40 ≡

```
sub computeChartScale {
    my $self = shift;
    my ($width, $height, $display_unit, $dietcalc) = @_;

    $width = 640 if !defined($width);
    $height = 480 if !defined($height);

    my $logToDisplayUnit = WEIGHT_CONVERSION->[$self->{log_unit}][$display_unit];

    ⟨ Define chart geometry 43a ⟩

    ⟨ Determine scale for weight and trend plot 45, ... ⟩

    return $bX . ',' .
           sprintf("%.4f", $pixelsPerDay) . ',' .
           $bY . ',' .
           $weightMin . ',' .
           ($extentY - ($bottomMargin + $topMargin)) . ',' .
           sprintf("%.4f", $weightMax - $weightMin) . ',' .
           RUNG_MAX;

}
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.12 plotChart

The `plotChart` method writes a PNG chart for the data in the month on the file handle specified by the first argument. The second and third arguments specify the width and height of the chart. The output file defaults to `STDOUT`, and the width and height default to 640 by 480 pixels if omitted. The chart is scaled and labeled with the `$display_unit` given by the argument.

"HDiet/monthlog.pm" 41 ≡

```
sub plotChart {
    my $self = shift;
    my ($outfile, $width, $height, $display_unit, $dchar,
        $dietcalc, $printFriendly, $monochrome) = @_;

    if (!(defined $outfile)) {
        $outfile = \*STDOUT;
    }

    $width = 640 if !defined($width);
    $height = 480 if !defined($height);

    my $logToDisplayUnit = WEIGHT_CONVERSION->[$self->{log_unit}][$display_unit];

    ⟨ Define chart geometry 43a ⟩

    my $img = new GD::Image($width, $height);

    $img->interlaced('true');

    ⟨ Allocate colours for chart 42 ⟩

    $img->filledRectangle($leftMargin + (-$axisOffset) + 1, $topMargin,
                    $leftMargin + ($pixelsPerDay * ($self->monthdays() - 1)),
                        $topMargin + ($extentY - ($topMargin + $bottomMargin)) + ($axisOffset + 1), $grey)

    my $lday = 0;
    for (my $i = $self->monthdays(); $i >= 1; $i--) {
        if (dnz($self->{weight}[$i])) {
            $lday = $i;
            last;
        }
    }

    ⟨ Draw axes for chart and label date axis 43b ⟩
    ⟨ Plot exercise rung information 44 ⟩

    ⟨ Determine scale for weight and trend plot 45, ... ⟩
    ⟨ Plot the diet plan if defined and requested 48a ⟩
    if ($lday > 0) {
        ⟨ Plot weight trend line on chart 48b ⟩
        ⟨ Plot weight entries as floats and sinkers 49 ⟩
    }
    ⟨ Label weight axis 50a ⟩

    print($outfile $img->png());

}
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.12.1 Plotting sparse data

The weight and exercise rung data we plot in charts may be arbitrarily sparse—the user may have failed to record the weight, the date may be in the future or before the user began logging, and the user may have skipped exercise one or more days. All of the items we plot are discrete quantities: they have meaning only at the time they were measured, so plotting them on a line graph is simply a convenience which makes them easier to visualise. Here are the conventions we use to plot lines through potentially sparse data points.

1. Today's point is defined.
   Call its co-ordinates (cx, cy).
   (a) Yesterday's point is defined ($ly \neq -1$).
   Plot a line from its co-ordinates (lx, ly) to (cx, cy). Set $(lx, ly) = (cx, cy)$.
   (b) Yesterday's point is not defined ($ly = -1$).
      i. This is the last day in the chart.
      Let lx be the X co-ordinate of the next to last day in the chart. Plot a line from (lx, cy) to (cx, cy).
      ii. This is not the last day in the chart.
      Set $(lx, ly) = (cx, cy)$.
2. Today's point is absent.
   Call its X co-ordinate cx.
   (a) Yesterday's point is defined ($ly \neq -1$).
   Plot a line from (lx, ly) to (cx,ly). Set $(lx, ly) = (-1, -1)$.
   (b) Yesterday's point is not defined ($ly = -1$).
   Do nothing.

5.12.2 Allocate colours for chart

We use an indexed-palette image format for the charts. Here we allocate the colours we're going to use, noting that the first one we allocate automatically becomes the background colour for the image.

⟨ Allocate colours for chart 42 ⟩ ≡

```
#   First colour allocated is background
my $white =  $img->colorAllocate(255, 255, 255);
my $grey = ($printFriendly || $monochrome) ?
               $white :
               $img->colorAllocate(160, 160, 160);
my $black =  $img->colorAllocate(  0,   0,   0);
my $red =    $monochrome ? $black : $img->colorAllocate(255,   0,   0);
my $green =  $monochrome ? $black : $img->colorAllocate(  0, 255,   0);
my $yellow = $monochrome ? $black : ($printFriendly ?
    $img->colorAllocate(192, 192,   0) : $img->colorAllocate(255, 255,   0));
my $blue =   $monochrome ? $black : $img->colorAllocate(  0,   0, 255);
my $dkgrey = $img->colorAllocate(128, 128, 128);
```

◇

Macro referenced in 41, 75, 93b.

5.12.3 Define chart geometry

The following variables specify the layout of items within the chart. They are basically twiddle knobs which are adjusted in the interest of appearance.

⟨ Define chart geometry 43a ⟩ ≡

```
        my ($fontLineHeight, $fontCharHeight) = (20, 10);
        my ($leftMargin, $rightMargin, $topMargin, $bottomMargin) =
            ($fontCharHeight * (($display_unit == WEIGHT_STONE) ? 6 : 5),
            $fontCharHeight * 3, 10, $fontCharHeight * 3);
        my ($axisOffset, $tickSize, $sinkerSize) = (3, 5, 4);

        my ($topLeftX, $topLeftY) = (0, 0);
        my ($extentX, $extentY) = ($width, $height);

        my $pixelsPerDay = ($extentX - ($leftMargin + $rightMargin)) / ($self->monthdays() - 1);

        my ($bX, $bY) = ($topLeftX + $leftMargin, (($topLeftY + $extentY) - $bottomMargin));
    ◇
```

Macro referenced in 40, 41.

5.12.4 Draw axes for chart and label date axis

⟨ Draw axes for chart and label date axis 43b ⟩ ≡

```
        #   X axis
        $img->line($bX - $axisOffset, $bY + $axisOffset + 1,
                   $bX + ($pixelsPerDay * ($self->monthdays() - 1)), $bY + $axisOffset + 1, $black);
        #   Y axis
        $img->line($bX - $axisOffset, $bY + $axisOffset + 1,
                   $bX - $axisOffset, $bY - ($extentY - (($topMargin + $bottomMargin))), $black);

        #   Date axis labels
        for (my $i = 0; $i < $self->monthdays(); $i += 3) {
            main::drawText($img, $i + 1, 'Times', 12, 0,
                $topLeftX + $leftMargin + ($pixelsPerDay * $i),
                (($topLeftY + $extentY) - ($bottomMargin - $topMargin)), 'c', 't', $black);
        }

        #   Ticks on date axis
        for (my $i = 1; $i < $self->monthdays(); $i++) {
            $img->line($bX + ($pixelsPerDay * $i), $bY + $axisOffset,
                       $bX + ($pixelsPerDay * $i), ($bY + $axisOffset) - $tickSize, $black);
        }
    ◇
```

Macro referenced in 41.

5.12.5 Plot exercise rung information

If the log contains any exercise rung entries, plot the rung information for the month and draw the rung scale at the right of the chart. This code uses the algorithm for plotting sparse data described in section 5.12.1 above.

⟨ Plot exercise rung information 44 ⟩ ≡

```
my $lrung;
my ($lx, $ly) = (-1, -1);

for (my $i = 1; $i <= $self->monthdays(); $i++) {
    if (dnz($self->{rung}[$i])) {
        my $rt = $self->{rung}[$i];
        $lrung = $rt;

        my ($cx, $cy) = ($bX + ($pixelsPerDay * ($i - 1)),
            ($bY - int((($rt - 1) * ($extentY - ($bottomMargin + $topMargin))) / RUNG_MAX)));

        if ($ly >= 0) {
             $img->line($lx, $ly, $cx, $cy, $blue);
             ($lx, $ly) = ($cx, $cy);
        } else {
            if ($i == $self->monthdays()) {
                $lx = $bX + ($pixelsPerDay * ($i - 2));
                $img->line($lx, $cy, $cx, $cy, $blue);
            } else {
                ($lx, $ly) = ($cx, $cy);
            }
        }
    } else {
        if ($ly >= 0) {
            my $cx = $bX + ($pixelsPerDay * ($i - 1));
            $img->line($lx, $ly, $cx, $ly, $blue);
            ($lx, $ly) = (-1, -1);
        }
    }
}

#   Draw labels for exercise rung scale

if ($lrung) {
    $img->line($bX + ($pixelsPerDay * ($self->monthdays() - 1)),
               $bY + $axisOffset + 1,
               $bX + ($pixelsPerDay * ($self->monthdays() - 1)),
               $bY - ($extentY - (($topMargin + $bottomMargin))), $black);

    for (my $i = 1; $i <= RUNG_MAX; $i = (int($i / 6) * 6) + 6) {
        main::drawText($img, $i, 'Times', 12, 0,
            $bX + ($pixelsPerDay * ($self->monthdays() - 1)) + 8,
            $bY - (int((($i - 1) * ($extentY - ($bottomMargin + $topMargin))) / RUNG_MAX)), 'l', 'c', $bla
        if ($i > 1) {
            $img->line($bX + ($pixelsPerDay * ($self->monthdays() - 1)) - $tickSize,
                       $bY - int((($i - 1) * ($extentY - ($bottomMargin + $topMargin))) / RUNG_MAX),
                       $bX + ($pixelsPerDay * ($self->monthdays() - 1)),
                       $bY - int((($i - 1) * ($extentY - ($bottomMargin + $topMargin))) / RUNG_MAX), $blac
        }
    }
}
```

◇

Macro referenced in 41.

5.12.6 Determine scale for weight and trend plot

Scan the weight and trend entries for the month and determine extrema. These are then used to automatically scale the weight and trend plot to fit on the chart.

⟨ Determine scale for weight and trend plot 45 ⟩ ≡

```
my ($weightMin, $weightMax) = (1e308, 0);
my ($trendMin, $trendMax) = (1e308, 0);

for (my $i = 1; $i <= $self->monthdays(); $i++) {
     if (dnz($self->{weight}[$i])) {
        $weightMax = max($weightMax, $self->{weight}[$i] * $logToDisplayUnit);
        $weightMin = min($weightMin, $self->{weight}[$i] * $logToDisplayUnit);
        $trendMax = max($trendMax, $self->{trend}[$i] * $logToDisplayUnit);
        $trendMin = min($trendMin, $self->{trend}[$i] * $logToDisplayUnit);
     }
}

$weightMin = min($weightMin, $trendMin);
$weightMax = max($weightMax, $trendMax);

if ($self->{trend_carry_forward} > 0) {
    $weightMin = min($weightMin, $self->{trend_carry_forward} * $logToDisplayUnit);
    $weightMax = max($weightMax, $self->{trend_carry_forward} * $logToDisplayUnit);
}

⟨ Compute diet plan extrema on chart 47 ⟩
if ($plan_start_day > 0) {
    $weightMin = min($weightMin, min($plan_start_weight, $plan_end_weight));
    $weightMax = max($weightMax, max($plan_start_weight, $plan_end_weight));
}

#   If no weights at all have been specified, scale the chart to encompass
#   the union of the 5% to 95% percentile points of adult males and females
#   as published at:    http://www.halls.md/chart/height-weight.htm
if ($weightMin > $weightMax) {
    if ($display_unit == WEIGHT_KILOGRAM) {
        $weightMin = 40;
        $weightMax = 120;
    } else {
        $weightMin = 100;
        $weightMax = 265;
    }
} else {
    #   Provide a buffer zone around extrema for new entries
    $weightMax += (($display_unit == WEIGHT_KILOGRAM) ?
        ⟨ Monthly Log Weight Range in Kilograms 4e ⟩ :
        ⟨ Monthly Log Weight Range in Pounds 4f ⟩) / 2;
    $weightMin -= (($display_unit == WEIGHT_KILOGRAM) ?
        ⟨ Monthly Log Weight Range in Kilograms 4e ⟩ :
        ⟨ Monthly Log Weight Range in Pounds 4f ⟩) / 2;
}

my $maxLabelRows = int(($extentY - ($topMargin + $bottomMargin)) / $fontLineHeight);
```

◇

Macro defined by 45, 46.
Macro referenced in 40, 41.

Find a display scale power and factor which permits a suitable number of labels on the weight axis. We do this using units like a vintage Tektronix oscilloscope: increasing powers of 1, 2, and 5.

⟨ Determine scale for weight and trend plot 46 ⟩ ≡

```
my $factor = 0;
my $vunit = 1;
my $power = 1;
my @factors = (1, 2, 5);

$weightMin *= 10;
$weightMax *= 10;
$weightMin = int($weightMin);
$weightMax = int($weightMax);

while (int(($weightMax - ($weightMin - ($weightMin % $vunit))) / ($factors[$factor] * $power)) > $maxLabel
    $factor++;
    if ($factor > 2) {
        $factor = 0;
        $power *= 10;
    }
    $vunit = $factors[$factor] * $power;
}

#   There's no point using a finer-grained unit than we
#   plot decimal places for weight.

if (($vunit < 10) && ($self->{log_unit} == WEIGHT_STONE)) {
    $vunit = 10;
}
if (($vunit < 100) && ($self->{log_unit} == WEIGHT_STONE)) {
    $vunit = 100;
}

#   Round weight unit minimum to even unit multiple

$weightMin -= $weightMin % $vunit;

#   Offset by one unit at top and bottom to avoid collision
#   with axes.

$weightMin -= $vunit;
$weightMax += $vunit;
$weightMin /= 10;
$weightMax /= 10;
$vunit /= 10;
```

◇

Macro defined by 45, 46.
Macro referenced in 40, 41.

5.12.7 Compute diet plan extrema on chart

If the user has established a diet plan and requested that it be plotted in charts, determine the segment of the plan which falls within this chart. There are three possibilities. If the plan starts after the end of this chart, we plot nothing. If the plan begins during the period of this chart, we plot from the beginning of the plan to the end of the month. If the plan ends within this month, we plot a flat line from the date of the end of the plan to the end of the month, on the assumption that the user intends to maintain the goal weight of the plan after its accomplishment.

Note that the start and goal weight for the diet plan are always kept in kilograms in the `user` object, so we convert them to the display weight unit being used in the chart here.

⟨ Compute diet plan extrema on chart 47 ⟩ ≡

```
#   Julian day of start and end of month
my ($mjdstart, $mjdend) = (gregorian_to_jd($self->{year}, $self->{month}, 1),
    gregorian_to_jd($self->{year}, $self->{month}, $self->monthdays()));
my ($pjdstart, $pjdend) = (max($mjdstart, $$dietcalc[0]), min($mjdend, $$dietcalc[2]));
my ($plan_start_day, $plan_start_weight,
    $plan_end_day, $plan_end_weight) = (-1) x 4;
if (defined($$dietcalc[0])) {
    #   If plan starts before the end of the month, we shall plot it
    if ($pjdstart <= $mjdend) {
        if ($$dietcalc[2] <= $mjdstart) {
            #   Plan ends before start of month; plot flat line at end weight
            $plan_start_day = 1;
            $plan_end_day = $self->monthdays();
            $plan_start_weight = $plan_end_weight = $$dietcalc[3] *
                WEIGHT_CONVERSION->[WEIGHT_KILOGRAM][$display_unit];
        } else {
            (undef, undef, $plan_start_day) = jd_to_gregorian($pjdstart);
            (undef, undef, $plan_end_day) = jd_to_gregorian($pjdend);
            $plan_start_weight = $$dietcalc[1] + (($$dietcalc[3] - $$dietcalc[1]) *
                (($pjdstart - $$dietcalc[0]) / ($$dietcalc[2] - $$dietcalc[0])));
            $plan_end_weight = $$dietcalc[1] + (($$dietcalc[3] - $$dietcalc[1]) *
                (($pjdend - $$dietcalc[0]) / ($$dietcalc[2] - $$dietcalc[0])));
            $plan_start_weight *= WEIGHT_CONVERSION->[WEIGHT_KILOGRAM][$display_unit];
            $plan_end_weight *= WEIGHT_CONVERSION->[WEIGHT_KILOGRAM][$display_unit];

        }
    }
}
```

◇

Macro referenced in 45.

5.12.8 Plot the diet plan if defined and requested

If a segment of the diet plan appears within this chart, plot it. If the plan ends within this month, draw a flat line at the goal weight from the end of the plan to the last day in the month.

⟨ Plot the diet plan if defined and requested 48a ⟩ ≡

```
        if ($plan_start_day > 0) {
            my $sy = int((($plan_start_weight - $weightMin) *
                ($extentY - ($bottomMargin + $topMargin))) / ($weightMax - $weightMin));
            my $ey = int((($plan_end_weight - $weightMin) *
                ($extentY - ($bottomMargin + $topMargin))) / ($weightMax - $weightMin));

            $img->setStyle($yellow, $yellow, $yellow, $yellow,
                           gdTransparent, gdTransparent, gdTransparent, gdTransparent);
            $img->line($bX + ($pixelsPerDay * ($plan_start_day - 1)), $bY - $sy,
                       $bX + ($pixelsPerDay * ($plan_end_day - 1)), $bY - $ey, gdStyled);
            if ($plan_end_day < $self->monthdays()) {
                $img->line($bX + ($pixelsPerDay * ($plan_end_day - 1)), $bY - $ey,
                           $bX + ($pixelsPerDay * ($self->monthdays() - 1)), $bY - $ey, gdStyled);
            }
        }
```

◇

Macro referenced in 41.

5.12.9 Plot weight trend line on chart

The weight trend is plotted as a solid red line.

⟨ Plot weight trend line on chart 48b ⟩ ≡

```
        for (my $i = 1; $i < $lday; $i++) {
            my $oy = int((($self->{trend}[$i] * $logToDisplayUnit) - $weightMin) *
                ($extentY - ($bottomMargin + $topMargin))) / ($weightMax - $weightMin));
            my $ny = int((($self->{trend}[$i + 1] * $logToDisplayUnit) - $weightMin) *
                ($extentY - ($bottomMargin + $topMargin))) / ($weightMax - $weightMin));

            $img->line($bX + ($pixelsPerDay * ($i - 1)), $bY - $oy,
                       $bX + ($pixelsPerDay * $i), $bY - $ny, $red);
        }
```

◇

Macro referenced in 41.

5.12.10 Plot weight entries as floats and sinkers

Individual weight log entries are plotted as blue diamonds, filled with yellow if the date is flagged and white otherwise. If the weight is above or below the trend line for that day, a green line is drawn to connect it to the trend and indicate whether the day's weight is a "float" pulling the trend up or a "sinker" dragging it down.

⟨ Plot weight entries as floats and sinkers 49 ⟩ ≡

```
        for (my $i = 1; $i <= $self->monthdays(); $i++) {
             if (dnz($self->{weight}[$i])) {
                my $px = $pixelsPerDay * ($i - 1);
                my $ty = int((($self->{trend}[$i] * $logToDisplayUnit) - $weightMin) *
                            ($extentY - ($bottomMargin + $topMargin))) / ($weightMax - $weightMin));
                my $wy = int((($self->{weight}[$i] * $logToDisplayUnit) - $weightMin) *
                            ($extentY - ($bottomMargin + $topMargin))) / ($weightMax - $weightMin));
                my $offset = $wy - $ty;

                if (($offset < -$sinkerSize) || ($offset > $sinkerSize)) {
                     my $dy = sgn($offset);

                    $img->line($bX + $px, $bY - ($ty + $dy),
                               $bX + $px, $bY - ($wy + (($offset > 0) ? -$sinkerSize : $sinkerSize)), $green);
                }

                #   Fill float/sinker with white or yellow, if it's flagged.

                for (my $j = -$sinkerSize; $j <= $sinkerSize; $j++) {
                    my $dx = abs($j) - $sinkerSize;

                    $img->line($bX + $px - $dx, $bY - ($wy + $j),
                               $bX + $px + $dx, $bY - ($wy + $j),
                               $self->{flag}[$i] ? $yellow : $white);
                }

                #   Trace the outline of the float/sinker in blue

                $img->line($bX + $px - $sinkerSize, $bY - $wy,
                           $bX + $px, $bY - ($wy - $sinkerSize), $blue);
                $img->line($bX + $px, $bY - ($wy - $sinkerSize),
                           $bX + $px + $sinkerSize, $bY - $wy, $blue);
                $img->line($bX + $px + $sinkerSize, $bY - $wy,
                           $bX + $px, $bY - ($wy + $sinkerSize), $blue);
                $img->line($bX + $px, $bY - ($wy + $sinkerSize),
                           $bX + $px - $sinkerSize, $bY - $wy, $blue);
            }
        }
```

◇

Macro referenced in 41.

5.12.11 Label weight axis

Draw the labels for the weight axis, according to the scale determined above.

⟨ Label weight axis 50a ⟩ ≡

```
        for (my $plotw = $weightMin; int($plotw * 10 + 0.5) <= int($weightMax * 10 + 0.5); $plotw += $vunit) {
            my $wy = int((($plotw - $weightMin) *
                        ($extentY - ($bottomMargin + $topMargin))) / ($weightMax - $weightMin));
            main::drawText($img, editWeight($plotw, $display_unit, $dchar), 'Times', 12, 0,
                $leftMargin - 8, $bY - $wy, 'r', 'c', $black);
            if ($plotw > $weightMin) {
                $img->line($bX - $axisOffset, $bY - $wy,
                           $bX + (-$axisOffset + $tickSize), $bY - $wy, $black);
            }
    #print("$plotw $wy\n");
        }
    ◇
```

Macro referenced in 41.

5.13 updateFromCGI

The `updateFromCGI` method is called with a reference to a hash containing the arguments submitted via a CGI request containing a monthly log table generated by `toHTML`. The fields in the hash are examined and any changes from the current state of the log are applied. A list is returned containing, as the first element, the total number of changes made, followed by the number of changes to weights, runs, flags, and comments respectively.

"HDiet/monthlog.pm" 50b ≡

```
        sub updateFromCGI {
            my $self = shift;
            my ($h) = @_;

            my ($change_weight, $change_rung, $change_flag, $change_comment) = (0, 0, 0, 0);
            my $days = $self->monthdays();

            for (my $d = 1; $d <= $days; $d++) {
                my $k;

                ⟨ Apply changes to weight 51 ⟩
                ⟨ Apply changes to exercise rung 54 ⟩
                ⟨ Apply changes to flag 55 ⟩
                ⟨ Apply changes to comment 56 ⟩
            }

            my $changes = $change_weight + $change_rung + $change_flag + $change_comment;

            return ($changes, $change_weight, $change_rung, $change_flag, $change_comment);
        }
    ◇
```

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.13.1 Apply changes to weight

The following code applies any changes the user has made to items in the weight column. It must cope with the user's having specified weights in stones and pounds (while entries in the weight array are pounds, even when the unit it set to stones), and differences in the display and log units for weight. We determine the display unit for this log from the hidden "`du`" field placed in the update form submitted by the user. We allow either the period or comma as a decimal character.

⟨ Apply changes to weight 51 ⟩ ≡

```
$k = "w$d";
if (defined($$h{$k})) {
    my $w = $$h{$k};

    $w =~ s/,/./g;

    ⟨ Expand weight entry if abbreviated 52 ⟩

    $w =~ s/[^\d\s\.]//g;
    $w =~ s/^\s+//;
    $w =~ s/\s+$//;
    #   If specification is stones and pounds, convert to pounds
    if (($w ne '') && ($$h{du} == WEIGHT_STONE)) {
        if ($w =~ m/\s*(\d+)\s+([\d\.]+)/) {
            $w = ($1 * 14) + $2;
        }
    }

    if ($w ne '') {
        $w = convertWeight($w, $$h{du}, $self->{log_unit});
    }

    if (($w eq '') && (znd($self->{weight}[$d]) != 0)) {
        undef($self->{weight}[$d]);
        $change_weight++;
    } elsif (($w ne '') && ($w ne znd($self->{weight}[$d]))) {
        $self->{weight}[$d] = $w;
        $change_weight++;
    }
}
```

◇

Macro referenced in 50b.

5.13.2 Expand weight entry if abbreviated

The user may abbreviate a weight entry in a variety of forms, ranging from a single period which copies the most recent entry, replacing the decimal place or last digit and decimal place, to Byzantine complexity of abbreviated entries in stones and pounds described in section 5.13.2.1 below.

⟨ Expand weight entry if abbreviated 52 ⟩ ≡

```
my $wa = $w;
$wa =~ s/^\s+//;
$wa =~ s/\s+$//;

if (($$h{du} == WEIGHT_STONE) && ($wa !~ m/\d*\.\d*/)) {
    $wa = '';
}

if (($wa eq '.') || ($wa =~ m/^\.\d+$/) ||
        ($wa =~ m/^\d(\.\d*)?$/) ||
        (($$h{du} == WEIGHT_STONE) && ($wa =~ m/^\d\d\.\d*$/))) {
    my $p = 0;
    my $lw;
    for (my $j = $d - 1; $j >= 1; $j--) {
        if (defined($$h{"w$j"}) && ($$h{"w$j"} =~ m/^\d/)) {
            $lw = $$h{"w$j"};
            $lw =~ s/,/./g;
            $p = 1;
            last;
        }
    }

    if ($p) {
        if ($wa eq '.') {
            $w = $lw;
        } else {
            if ($$h{du} == WEIGHT_STONE) {
                ⟨ Expand weight entry in stones and pounds 53 ⟩
            } else {
                if ($wa =~ m/^\.\d+$/) {
                    $w = int($lw) + $wa;
                } elsif ($wa =~ m/^\d(\.\d*)?$/) {
                    $w = (int($lw / 10) * 10) + $wa;
                }
                $$h{$k} = $w;
            }
        }
    }
}
```

◇

Macro referenced in 51.

5.13.2.1 Expand weight entry in stones and pounds

When the weight unit is set to stones an abbreviation may be used to change the pounds and decimal place of the previous stone and pound display just as when the units are pounds. In addition, when the display unit is set to stones, if the previous entry has a pounds field between 10 and 13 and the user enters a single digit, decimal character, and optional decimal digit, the action taken depends on the units digit entered. If it's between 0 and 3, it replaces the last digit of the pounds in the last entry, but if the digit is 4 or greater (which is invalid in a stones and pounds display), that digit replaces the two digit pounds field in the previous entry. This reduces the scribbling required when the weight happens to fluctuate around X stones 10. In addition, when the display unit is stones, the user can enter two digits followed by the decimal character and an optional decimal digit to replace the pounds field of the last stones and pounds entry; the decimal character must be entered to distinguish the entry from one denoting an even number of stones.

⟨ Expand weight entry in stones and pounds 53 ⟩ ≡

```
$lw =~ m/^(\d+)\s+(\d*\.?\d*)$/;
my ($stones, $pounds) = ($1, $2);

if ($pounds >= 10) {
    if ($wa < 4) {
        if ($wa =~ m/^\.\d+$/) {
            $pounds = int($pounds) + $wa;
        } else {
            $pounds = ((int($pounds  / 10)) * 10) + $wa;
        }
    } else {
        $pounds = $wa;
    }
} else {
    if ($wa =~ m/^\.\d+$/) {
        $pounds = int($pounds) + $wa;
    } else {
        $pounds = $wa;
    }
}
$w = "$stones $pounds";
$$h{$k} = $w;
```

◇

Macro referenced in 52.

5.13.3 Apply changes to exercise rung

Apply any changes the user has made in the exercise rung column to the rung array. Rung numbers are constrained to the range 1–RUNG_MAX, with blank indicating no entry for the day.

A single period for the rung causes the most recent non-blank rung to be copied into the field; if none exists, the period is discarded and the rung left blank.

⟨ Apply changes to exercise rung 54 ⟩ ≡

```
$k = $$h{"r$d"};

if (defined($k) && ($k =~ m/^\s*([\.,\+\-])\s*$/)) {
    my $cop = $1;
    for (my $j = $d - 1; $j >= 1; $j--) {
        if (defined($$h{"r$j"}) && ($$h{"r$j"} ne '')) {
            $k = $$h{"r$j"};
            $k++ if $cop eq '+';
            $k-- if $cop eq '-';
            last;
        }
    }
}

if (defined($k)) {
    $k =~ s/\D//g;          #   Delete non-digit characters
    if ($k =~ m/^\d/) {
        $k = 1 if $k < 1;
        $k = RUNG_MAX if $k > RUNG_MAX;
    }
    $$h{"r$d"} = $k;

    if (($k eq '') && (znd($self->{rung}[$d]) != 0)) {
        undef($self->{rung}[$d]);
        $change_rung++;
    } elsif (($k ne '') && ($k ne znd($self->{rung}[$d]))) {
        $self->{rung}[$d] = $k;
        $change_rung++;
    }
}
```

◇

Macro referenced in 50b.

5.13.4 Apply changes to flag

If the state of the utility flag for the day has changed, update the item in the flag array. Because flags are checkboxes, and browsers to not usually send unchecked values, we have to handle the case where the user has unchecked a flag which was previously set in the database. For each undefined flag, we test if the flag for the corresponding day is set and, if so, clear it. We also include a test for notional eccentric browsers which send unchecked boxes with a blank value field, just in case.

⟨ Apply changes to flag 55 ⟩ ≡

```
        if (defined($$h{"f$d"})) {
            $k = $$h{"f$d"};

            if (($k eq '') && (znd($self->{flag}[$d]) != 0)) {
                undef($self->{flag}[$d]);
                $change_flag++;
            } elsif (($k ne '') && (znd($self->{flag}[$d]) == 0)) {
                $self->{flag}[$d] = 1;
                $change_flag++;
            }
        } else {
            if (znd($self->{flag}[$d]) != 0) {
                undef($self->{flag}[$d]);
                $change_flag++;
            }
        }
```

◇

Macro referenced in 50b.

5.13.5 Apply changes to comment

Apply any changes the user has made in the comment column. Anything goes as a comment, except that trailing white space is automatically deleted unless the comment consists of just a period followed by white space. To keep this from being confused with a ditto in a static update, this case is canonicalised as a period followed by a single space.

A field with a single period causes the most recent non-blank comment to be copied into this field. If there is no previous item, the period is simply left in place, and may serve to ditto a comment for a previous day when it is entered.

⟨ Apply changes to comment 56 ⟩ ≡

```
if (defined($$h{"c$d"})) {
    $k = $$h{"c$d"};
    if (($k eq '.') || ($k eq ',')) {
        for (my $j = $d - 1; $j >= 1; $j--) {
            if (defined($$h{"c$j"}) && ($$h{"c$j"} ne '')) {
                $k = $$h{"c$j"};
                $$h{"c$d"} = $k;
                last;
            }
        }
    }
    if ($k =~ m/^\.\s+$/) {
        $k = '. ';
    } else {
        $k =~ s/\s+$//;
    }
    if (($k eq '') && defined($self->{comment}[$d])) {
        undef($self->{comment}[$d]);
        $change_comment++;
    } elsif (($k ne '') &&
        ((!defined($self->{comment}[$d])) || ($k ne $self->{comment}[$d]))) {
        $self->{comment}[$d] = $k;
        $change_comment++;
    }
}
```

◇

Macro referenced in 50b.

5.14 ImportCSV

The `importCSV` method extracts fields from a strictly compliant extended CSV file of the form created by our `hdCSV` module and inserts them into the object arrays for the specified day, first verifying that the month and year specified in the record actually agree with those for this log. Non-CSV records (defined as those which do not contain an ISO-8601 YYYY-MM-DD date) are ignored. The return value is 0 if the record was ignored, 1 if it was inserted into the log.

"HDiet/monthlog.pm" 57 ≡

```
sub importCSV {
    my $self = shift;

    my $s = shift;
    my ($date, $weight, $rung, $flag, $comment) = parseCSV($s);

    #   Ignore any line without a strictly compliant date

    if ($date =~ m/^(\d\d\d\d)\-(\d\d)\-(\d\d)$/) {
        my ($yy, $mm, $dd) = ($1, $2, $3);
        if (($yy != $self->{year}) ||
            ($mm != $self->{month}) ||
            ($dd < 1) || ($dd > $self->monthdays())) {
            die("Bogus CSV import date for $self->{year}-$self->{month}: $date");
        }
        $weight =~ s/\s//g;
        $self->{weight}[$dd] = $weight if ($weight ne '');
        $rung =~ s/\s//g;
        $self->{rung}[$dd] = $rung if ($rung ne '');
        $flag =~ s/\s//g;
        $self->{flag}[$dd] = $flag if ($flag ne '');
        $self->{comment}[$dd] = $comment if ($comment ne '');
        return 1;
    }
    return 0;
}
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.15 ExportCSV

The `exportCSV` method writes the monthly log to a CSV file on the file handle given by the first argument (`STDOUT` if it is omitted) in a format strongly reminiscent of that used by `hdread`, but extended to handle Unicode and non-graphic characters as defined in our `hdCSV` package.

"HDiet/monthlog.pm" 58 ≡

```
        sub exportCSV {
            my $self = shift;

            my ($fh) = @_;

            print $fh <<"EOD";
    Date,Weight,Rung,Flag,Comment\r
    StartTrend,$self->{trend_carry_forward},$self->{log_unit},$self->{last_modification_time},$self->{last_modifi
    EOD

            for (my $i = 1; $i <= $self->monthdays(); $i++) {
                my $csv = encodeCSV(sprintf("%04d-%02d-%02d", $self->{year}, $self->{month}, $i),
                                    bnd($self->{weight}[$i]),
                                    bnd($self->{rung}[$i]),
                                    znd($self->{flag}[$i]),
                                    (defined($self->{comment}[$i]) ? $self->{comment}[$i] : ''));
                print($fh "$csv\r\n");
            }
        }
    ◇
```

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.16 ExportHDReadCSV

The `ExportHDReadCSV` method exports a monthly log to the file handle argument in the format used by the Palm Eat Watch desktop utility `hdread`. This is very similar to our own native format, but it does not support Unicode characters and uses Excel-style quoting and escaping of string fields.

macros.

"HDiet/monthlog.pm" 59 ≡

```
        sub exportHDReadCSV {
            my $self = shift;

            my ($fh) = @_;

            my $tcf = sprintf("%.4f", znd($self->{trend_carry_forward}));
            my $wu = ucfirst(WEIGHT_UNITS->[$self->{log_unit}]) . 's';
            my $mon = $::monthNames[$self->{month}];
            print $fh <<"EOD";
    Date,Weight,Rung,Flag,Comment
    StartTrend,$tcf,$self->{log_unit},$self->{last_modification_time},$self->{last_modification_time}
    EOD
            for (my $i = 1; $i <= $self->monthdays(); $i++) {
                my $cmt = '';
                if (defined($self->{comment}[$i])) {
                    $cmt = $self->{comment}[$i];
                    $cmt =~  s/([\x{00}-\x{1F}-\x{80}-\x{9F}\x{100}-\x{FFFF}])/sprintf("&#x%x;", ord($1))/eg;
                    $cmt =~ s/"/""/g;
                    if ($cmt =~ m/[\s",]/) {
                        $cmt = '"' . $cmt . '"';
                    }
                }
                print($fh sprintf("%04d-%02d-%02d", $self->{year}, $self->{month}, $i) . ',' .
                          bnd($self->{weight}[$i]) . ',' .
                          bnd($self->{rung}[$i]) . ',' .
                          znd($self->{flag}[$i]) . ',' .
                          $cmt . "\n");
            }
        }
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.17 ExportExcelCSV

The `ExportExcelCSV` method writes the monthly log to a CSV file on the file handle given by the first argument (`STDOUT` if it is omitted) in a format compatible (via cut and paste into a pre-created yearly log template) with the legacy Excel Eat Watch macros.

"HDiet/monthlog.pm" 60 ≡

```
        sub exportExcelCSV {
            my $self = shift;

            my ($fh) = @_;

            ⟨ Obtain trend carry-forward for Excel CSV 61 ⟩

            my $wu = ucfirst(WEIGHT_UNITS->[$self->{log_unit}]) . 's';
            my $mon = $::monthNames[$self->{month}];
            print $fh <<"EOD";
    Date,,Weight,Trend,Variance,,Rung,Flag\r
    ,,,,,,,\r
    ,,$wu,,$mon $self->{year},,,\r
    Trend carry forward:,,,$tcf,,,,\r
    EOD

            my $wd = jd_to_weekday(gregorian_to_jd($self->{year}, $self->{month}, 1));

            for (my $i = 1; $i <= $self->monthdays(); $i++) {
                my $cmt = '';
                if (defined($self->{comment}[$i])) {
                    $cmt = $self->{comment}[$i];
                    $cmt =~  s/([\x{00}-\x{1F}-\x{80}-\x{9F}\x{100}-\x{FFFF}])/sprintf("&#x%x;", ord($1))/eg;
                    $cmt =~ s/"/""/g;
                    $cmt = '"' . $cmt . '"';
                }

                print($fh sprintf("%d/%d/%02d", $self->{month}, $i, $self->{year} % 100) . ',' .
                    WEEKDAY_NAMES->[$wd] . ',' .
                    ((bnd($self->{weight}[$i]) eq '') ? (($cmt eq '') ? '---' : $cmt) : sprintf("%.1f", $self->{we
                    ((bnd($self->{trend}[$i]) eq '') ? '' : sprintf("%.1f", $self->{trend}[$i])) . ',' .
                    sprintf("%.2f", znd($self->{weight}[$i]) - znd($self->{trend}[$i])) . ',' .
                    sprintf("%.1f", $tcf) . ',' .
                    bnd($self->{rung}[$i]) . ',' .
                    (znd($self->{flag}[$i]) ? '1' : ''). "\r\n");

                $wd = ($wd + 1) % 7;
                if (znd($self->{trend}[$i])) {
                    $tcf = $self->{trend}[$i];
                }
            }
        }
    ◇
```

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.17.1 Obtain trend carry-forward for Excel CSV

CSV format doesn't understand our convention of zero denoting no trend carryforward. We fill in an unspecified trend carryforward with the first specified weight in the month. The trend cary-forward is rounded to two decimal places.

⟨ Obtain trend carry-forward for Excel CSV 61 ⟩ ≡

```
my $tcf = $self->{trend_carry_forward};
if ($tcf == 0) {
    for (my $i = 1; $i <= $self->monthdays(); $i++) {
        if (znd($self->{weight}[$i])) {
            $tcf = $self->{weight}[$i];
            last;
        }
    }
}
$tcf = sprintf("%.2f", $tcf);
```

◇

Macro referenced in 60.

5.18 ExportXML

The `exportXML` method writes the monthly log as an XML `monthlog` element on the file handle given by the first argument (`STDOUT`. The character encoding of the XML is UTF-8, and it is the responsibility of the caller to ensure that the file handle has that output discipline in effect. The caller is assumed to have already established the structure of the XML file to which the log will be appended.

"HDiet/monthlog.pm" 62 ≡

```
sub exportXML {
    my $self = shift;

    my ($fh, $safe) = @_;

    my $wu = WEIGHT_UNITS->[$self->{log_unit}];
    my $lm = timeXML($self->{last_modification_time});
    my $nd = $self->monthdays();

    print $fh <<"EOD";
<monthlog version="1.0">
    <properties>
        <year>$self->{year}</year>
        <month>$self->{month}</month>
        <weight-unit>$wu</weight-unit>
        <trend-carry-forward>$self->{trend_carry_forward}</trend-carry-forward>
        <last-modified>$lm</last-modified>
    </properties>
    <days ndays="$nd">
EOD

    for (my $i = 1; $i <= $self->monthdays(); $i++) {
        my $sweight = textXML('weight', bnd($self->{weight}[$i]), $safe);
        my $srung = textXML('rung', bnd($self->{rung}[$i]), $safe);
        my $sflag = textXML('flag', bnd($self->{flag}[$i]), $safe);
        my $scomment = textXML('comment', (defined($self->{comment}[$i]) ? $self->{comment}[$i] : ''), $sa

        print $fh <<"EOD";
        <day>
            <date>$i</date>
            $sweight
            $srung
            $sflag
            $scomment
        </day>
EOD
    }
    print $fh <<"EOD";
    </days>
</monthlog>
EOD
}
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.19 Monthdays

The utility `monthdays` returns the number of days in the month represented by this log. It is mostly intended for internal use, but it is exported so users of logs may call it if required. It can also be called as a function independent of a `monthlog` object by passing two arguments giving the year and month.

"HDiet/monthlog.pm" 63 ≡

```
sub monthdays {
    my ($year, $month);

    if ($#_ == 0) {
        my $self = shift;
        ($year, $month) = ($self->{year}, $self->{month});
    } else {
        ($year, $month) = @_;
    }

    if ($year == 0) {
        return 0;
    }

    #   Thirty days hath September, ...
    my @monthdays = ( 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 );

    if ($month == 2) {
        if ((($year % 4) != 0) ||
            ((($year % 100) == 0) && ($year % 400) != 0)) {
            return 28;
        }
        return 29;
    } else {
        return $monthdays[$month];
    }
}
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.20 Previous and next month

The `previousMonth` and `nextMonth` methods return the year and month of the previous and next months respectively as a list of two numbers. Note that this does not imply that said month is present in the user database from which this month originated; that's up for the caller to determine. These functions may also called directly from the package with two arguments giving the month and year for which the previous or next month is required.

"HDiet/monthlog.pm" 64 ≡

```
sub previousMonth {
    my ($year, $month);

    if ($#_ == 0) {
        my $self = shift;
        ($year, $month) = ($self->{year}, $self->{month});
    } else {
        ($year, $month) = @_;
    }

    $month--;
    if ($month < 1) {
        $year--;
        $month = 12;
    }

    return ($year, $month);
}

sub nextMonth {
    my ($year, $month);

    if ($#_ == 0) {
        my $self = shift;
        ($year, $month) = ($self->{year}, $self->{month});
    } else {
        ($year, $month) = @_;
    }

    $month++;
    if ($month > 12) {
        $year++;
        $month = 1;
    }

    return ($year, $month);
}
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.21 Verbose

If called with an argument `verbose` method sets the verbosity switch on (for an argument of 1) or off (argument 0). If called with no argument, the current setting is returned.

"HDiet/monthlog.pm" 65 ≡

```
sub verbose {
    my $self = shift;

    my $v;
    if ($v = shift) {
        $self->{verbose} = $v;
    }
    if ($self->{verbose}) {
        print("monthlog: Verbose = $self->{verbose}\n");
    }
    return $self->{verbose};
}
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.22 EncodeComments

Since many logs contain duplicate comment fields, we compress the comments in the database to a string containing a list of days on which a given comment appears and the quoted comment string. Quotes within comments are expanded to two consecutive quotes. The `encodeComments` method takes no argument and returns the string containing the compressed comments for the month.

"HDiet/monthlog.pm" 66 ≡

```
sub encodeComments {
    my $self = shift;

    my $mdays = $self->monthdays();
    my $enc = '';
    my @cmt = @{$self->{comment}};

    for (my $i = 1; $i <= $mdays; $i++) {
        if (defined($cmt[$i]) &&
            ($cmt[$i] ne '')) {
            $enc .= $i;                 # Comment appears on this day first
            for (my $j = $i + 1; $j <= $mdays; $j++) {
                if (defined($cmt[$j]) &&
                    ($cmt[$i] eq $cmt[$j])) {
                    $enc .= ",$j";      # Comment also appears on this day
                    $cmt[$j] = '';      # Wipe it out since it's been handled as a duplicate
                }
            }
            my $ct = $cmt[$i];
            $ct =~ s/"/""/g;
            $enc .= "\"$ct\"";
        }
    }
    return $enc;
}
```

◇

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

5.23 DecodeComments

The `decodeComments` method takes a string argument containing comments compressed by `encodeComments`, extracts them, and stores their values in the `comment` array of the log. Any existing comments for days in the encoded comments (but not for other days) are overwritten.

"HDiet/monthlog.pm" 67 ≡

```
        sub decodeComments {
            my $self = shift;

            my $ecom = shift;

            while ($ecom =~ s/^([\d,]+)"((?:[^"]|"")+)"//) {
    #print("Days: $1  Comment: ($2)\n");
                my ($days, $comment) = ($1, $2);

                $comment =~ s/""/"/g;
                while ($days =~ s/(\d+),?//) {
                    $self->{comment}[$1] = $comment;
                }

            }
        }
    ◇
```

File defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.

Chapter 6

`history.pm`: Historical Analysis Object

The `history` object performs historical analysis of data for a user, including generation of historical charts and long-term trend analyses.

6.1 Package plumbing

"HDiet/history.pm" 69 ≡

```
#! ⟨ Perl directory 7d ⟩

    ⟨ Perl language modes 369b ⟩

    use HDiet::monthlog qw(:units);
    use HDiet::trendfit;

    package HDiet::history;

    use HDiet::Julian qw(WEEKDAY_NAMES :DEFAULT);
    use HDiet::Cluster;
    use GD;

    require Exporter;

    use constant MIN_VALUE => -1E100;
    use constant MAX_VALUE => 1E100;

    ⟨ Minimum, Maximum, and Sign functions 384 ⟩

    our @ISA = qw(Exporter);
    our @EXPORT = ( );

    my ($width, $height, $leftMargin, $rightMargin, $topMargin, $bottomMargin,
        $xAxisLength);
    my ($start_jd, $end_jd);
    my ($wgt_min, $wgt_max);
    my ($img);
    my %logs;                    # Monthly log cache
    my @years;                  # List of years in the user database

    my $fitter;
    my $lastFitDay;
    my ($nDays, $tFlags);

    1;
```

◇

File defined by 69, 70, 71, 72, 73, 75, 76, 93ab, 94abc, 95, 96, 97, 100, 101a, 103.

6.2 Constructor

A new `history` object is created by calling the `new` constructor. It is called with the user object whose history is to be analysed and the name of the directory in which that user's history is kept. (We actually could determine the latter from the former, but since all callers have already done this, there's no reason not to just supply it as a constructor argument.)

"HDiet/history.pm" 70 ≡

```
        sub new {
            my $self = {};
            my ($invocant, $user, $user_file_name) = @_;
            my $class = ref($invocant) || $invocant;

            bless($self, $class);

            #   Initialise instance variables from constructor arguments
            $self->{user} = $user;
            $self->{user_file_name} = $user_file_name;

            return $self;
        }
```

◇

File defined by 69, 70, 71, 72, 73, 75, 76, 93ab, 94abc, 95, 96, 97, 100, 101a, 103.

6.3 Get log data for range of days

Obtain weight, trend, and rung for a given range of days, if available, When returning data for more than one day, the arithmetic mean of days for which entries are present is returned. The weight and trend results are expressed in the current `display_unit` in the user preferences. This function may only be called after the monthly logs for all dates in the requested range have been loaded into the `%logs` cache.

"HDiet/history.pm" 71 ≡

```
sub getDays {
    my ($jdstart, $ndays, $ui) = @_;

    my ($wsum, $tsum, $rsum, $flgc, $wtd, $rd) = (0, 0, 0, 0, 0, 0);

    for (my $i = 0; $i < $ndays; $i++) {
        my ($yy, $mm, $dd) = jd_to_gregorian($jdstart);
        my $m = $logs{sprintf("%04d-%02d", $yy, $mm)};

        if ($m) {
            if ($m->{weight}[$dd]) {
                $wsum += $m->{weight}[$dd] * HDiet::monthlog::WEIGHT_CONVERSION->[$m->{log_unit}][$ui->{di
                my $dtrend = $m->{trend}[$dd] * HDiet::monthlog::WEIGHT_CONVERSION->[$m->{log_unit}][$ui->
                $tsum += $dtrend;
                $wtd++;
            }

            if ($m->{trend}[$dd]) {
                if ($jdstart > $lastFitDay) {
                    $fitter->addPoint($m->{trend}[$dd] *
                        HDiet::monthlog::WEIGHT_CONVERSION->[$m->{log_unit}][$ui->{display_unit}]);
                }
            }

            if ($m->{rung}[$dd]) {
                $rsum += $m->{rung}[$dd];
                $rd++;
            }

            if ($m->{flag}[$dd]) {
                $flgc++;
            }
            if ($jdstart > $lastFitDay) {
                $nDays++;
                $tFlags++ if $m->{flag}[$dd];
                $lastFitDay = $jdstart;
            }
        }
        $jdstart++;
    }
```

◇

File defined by 69, 70, 71, 72, 73, 75, 76, 93ab, 94abc, 95, 96, 97, 100, 101a, 103.

If one or more days were present in the range, compute the mean weight, trend, and rung and return to the caller.

"HDiet/history.pm" 72 ≡

```
            if ($wtd > 0) {
                $wsum /= $wtd;
                $tsum /= $wtd;
            } else {
                $wsum = undef;
                $tsum = undef;
            }
            if ($rd > 0) {
                $rsum /= $rd;
            } else {
                $rsum = undef;
            }

            return ($wsum, $tsum, $rsum, $flgc);
        }
```

◇

File defined by 69, 70, 71, 72, 73, 75, 76, 93ab, 94abc, 95, 96, 97, 100, 101a, 103.

6.4 Analyse Trend

The `analyseTrend` method computes the slope of the linear regression best fit to one or more date intervals specified by a list passed as an argument which contains the start and end dates for each interval required. The result is returned as a list of sequences of four items: slope, minimum, maximum, and mean values for the trend over the time intervals requested. If no log entries exist within one or more requested ranges, `undef` will be returned as the slope for that interval.

The trend slopes for all date ranges requested are computed in a single pass over a date range inclusive of all, but computation is not optimised for widely-saparated disjoint intervals. In fact, almost all callers of this method use it to compute nested intervals (last week, last month, last quarter, etc.), so it's probably not worth the bother of adding the complexity it would require to skip dates in gaps.

"HDiet/history.pm" 73 ≡

```
        sub analyseTrend {
            my $self = shift;

            my (@intervals) = @_;

            my ($ui, $user_file_name) = ($self->{user}, $self->{user_file_name});

            my ($start_date, $end_date) = ('9999-99-99', '0000-00-00');

            ⟨ Build table of intervals and compute date span of union 74a ⟩

            ⟨ Determine the number of days in the historical interval 77 ⟩
    #print("Inclusive interval: $start_date - $end_date  $start_jd - $end_jd  $dayspan days\n");

            ⟨ Fill cache with monthly logs in the date range 104 ⟩

    #use Data::Dumper;
    #print(Dumper(\@interval));

            #   Instantiate a fitter for each interval we're watching
            my @fitter;
            for (my $i = 0; $i <= $#interval; $i++) {
                $fitter[$i] = HDiet::trendfit->new();
            }

            $fitter = HDiet::trendfit->new();
            $lastFitDay = 0;
            ($nDays, $tFlags) = (0, 0);

            ⟨ Examine dates, fitting those within intervals 74b ⟩

    #print(Dumper(\@fitter));

            my @slopes;
            for (my $i = 0; $i <= $#fitter; $i++) {
                push(@slopes, $fitter[$i]->fitSlope());
                push(@slopes, $fitter[$i]->minMaxMean());
            }

    #print(Dumper(\@slopes));

            return @slopes;
        }
    ◇
```

File defined by 69, 70, 71, 72, 73, 75, 76, 93ab, 94abc, 95, 96, 97, 100, 101a, 103.

6.4.1 Build table of intervals and compute date span of union

Walk through the argument pairs and build the `@interval` table, each entry of which is an array of two Julian day numbers giving the start and end of the interval. As we're doing this, we keep track of the first and last dates in the union of the intervals, which we'll use to determine the portion of the database we need to scan.

⟨ Build table of intervals and compute date span of union 74a ⟩ ≡

```
my @interval;
for (my $i = 0; $i <= $#intervals; $i += 2) {
    die("history::analyseTrend: Interval[$i] ($intervals[$i] - $intervals[$i + 1]) out of order")
        if $intervals[$i] gt $intervals[$i + 1];
    $start_date = $intervals[$i] if $intervals[$i]  lt $start_date;
    $end_date = $intervals[$i + 1] if $intervals[$i + 1] gt $end_date;
    $intervals[$i] =~ m/^(\d+)(?:\-(\d+))?(?:\-(\d+))?$/ ||
        die("history::analyseTrend: invalid intervals[$i] start date intervals[$i]");
    my $int_start_jd = gregorian_to_jd($1, $2, $3);

    $intervals[$i + 1] =~ m/^(\d+)(?:\-(\d+))?(?:\-(\d+))?$/ ||
        die("history::drawChart: history::analyseTrend: invalid intervals[$i + 1] start date intervals[$i
    my $int_end_jd = gregorian_to_jd($1, $2, $3);
    push(@interval, [$int_start_jd, $int_end_jd]);
}
```

◇

Macro referenced in 73.

6.4.2 Examine dates, fitting those within intervals

Iterate over all dates in the union of the requested intervals. For each which has a trend value available, include it in the regression fit for each interval within which the interval falls.

One subtlety is how we handle missing trend values. Until we've seen a date with a valid trend, we ignore days entirely; this handles requests which begin before the first log entry in the database. Afterward, missing trend values, which usually result when an entire month has no log entries, are faked by filling in the last valid trend value we saw before the gap.

⟨ Examine dates, fitting those within intervals 74b ⟩ ≡

```
my $lastTrend = 0;

for (my $cdate = $start_jd; $cdate <= $end_jd; $cdate++) {
    my ($weight, $trend) = getDays($cdate, 1, $ui);
    $trend = $lastTrend if (!$trend) && $lastTrend;
    if ($trend) {
        for (my $i = 0; $i <= $#interval; $i++) {
            if (($cdate >= $interval[$i][0]) && ($cdate <= $interval[$i][1])) {
                $fitter[$i]->addPoint($trend);
            }
        }
        $lastTrend = $trend;
    }
}
```

◇

Macro referenced in 73.

6.5 Draw Chart

The `drawChart` method plots an historical chart with the span specified by the arguments.

"HDiet/history.pm" 75 ≡

```
sub drawChart {
    my $self = shift;

    my ($outfile, $start_date, $end_date, $ww, $hh, $dietcalc,
        $printFriendly, $monochrome) = @_;

    my ($ui, $user_file_name) = ($self->{user}, $self->{user_file_name});

    ($width, $height) = ($ww, $hh);

    if (!(defined $outfile)) {
        $outfile = \*STDOUT;
    }

    $fitter = HDiet::trendfit->new();
    $lastFitDay = 0;
    ($nDays, $tFlags) = (0, 0);

    ⟨ Determine the number of days in the historical interval 77 ⟩
    ⟨ Find weight and trend extrema in log entries to be plotted 78 ⟩
    ⟨ Find diet plan extrema on historical chart 79 ⟩
    ⟨ Define historical chart geometry 80b ⟩

    $img = new GD::Image($width, $height);
    ⟨ Allocate colours for chart 42 ⟩
    $img->interlaced('true');

    $xAxisLength = $width - ($leftMargin + $rightMargin);
    $img->filledRectangle($leftMargin + (-$axisOffset) + 1, $topMargin,
                          $leftMargin + $xAxisLength + $axisOffset,
                          $topMargin + $height - ($topMargin + $bottomMargin) + ($axisOffset - 1), $grey);

    ⟨ Draw axes for historical chart 82b ⟩

    ⟨ Determine vertical weight scaling based on extrema 81a, ... ⟩

    ⟨ Plot weight and rung data on historical chart 87 ⟩

    ⟨ Plot the diet plan on historical chart 80a ⟩

    ⟨ Empty monthly log cache 105b ⟩

    print($outfile $img->png());


}
```

◇

File defined by 69, 70, 71, 72, 73, 75, 76, 93ab, 94abc, 95, 96, 97, 100, 101a, 103.

6.5.1 Scaling and line drawing functions

The following utility functions are used in the `drawChart` method above to scale weight and exercise rung quantities to the dimensions of the chart and draw lines within the plot area.

"HDiet/history.pm" 76 ≡

```
        sub WeightToY {             # Map weight to vertical pixel position
            my ($w) = @_;
            return int((($w - $wgt_min) * ($height - ($bottomMargin + $topMargin))) / ($wgt_max - $wgt_min));
        }

        sub RungToY {               # Map exercise rung to vertical pixel position
            my ($r) = @_;
            return int((($r - 1) * ($height - ($bottomMargin + $topMargin))) / HDiet::monthlog::RUNG_MAX);
        }
        sub PlotScaleLine {         # Transform plot area co-ordinates into absolute
            my ($x1, $y1, $x2, $y2) = @_;
            return ($leftMargin + $x1, ($height - $bottomMargin) - $y1,
                    $leftMargin + $x2, ($height - $bottomMargin) - $y2);
        }

        sub PlotLine {              # Plot a line given plot area co-ordinates and colour
            my ($rx1, $ry1, $rx2, $ry2, $colour) = @_;
            die("Colour missing from call to PlotLine") if !defined($colour);
            my ($x1, $y1, $x2, $y2) = PlotScaleLine($rx1, $ry1, $rx2, $ry2);
            $img->line($x1, $y1, $x2, $y2, $colour);
        }
```

◇

File defined by 69, 70, 71, 72, 73, 75, 76, 93ab, 94abc, 95, 96, 97, 100, 101a, 103.

6.5.2 Determine the number of days in the historical interval

The horizontal scale of the chart is determined by the number of days it spans. Parse the start and ending date arguments, filling in defaults for omitted fields, and determine the days spanned by the chart by taking the difference of their Julian day numbers.

⟨ Determine the number of days in the historical interval 77 ⟩ ≡

```
$start_date =~ m/^(\d+)(?:\-(\d+))?(?:\-(\d+))?$/ || die("history::drawChart: invalid start date $start_da
my ($start_y, $start_m, $start_d) = ($1, $2, $3);
$start_m = 1 if !defined($start_m);
$start_d = 1 if !defined($start_d);
$start_jd = gregorian_to_jd($start_y, $start_m, $start_d);

$end_date =~ m/^(\d+)(?:\-(\d+))?(?:\-(\d+))?$/ || die("history::drawChart: invalid end date $end_date");
my ($end_y, $end_m, $end_d) = ($1, $2, $3);
$end_m = 12 if !defined($end_m);
if (!defined($end_d)) {
    $end_jd = gregorian_to_jd($end_y, $end_m + 1, 1) - 1;
    $end_d  = (jd_to_gregorian($end_jd))[2];
}
$end_jd = gregorian_to_jd($end_y, $end_m, $end_d);

my $dayspan = (($end_jd + 1) - $start_jd);
```

◇

Macro referenced in 73, 75, 97.

6.5.3 Find weight and trend extrema in log entries to be plotted

Scan the logs present in the database between the start and end dates and determine the extrema of trend and weight. These are used to establish the vertical scale for the chart. In the process, a hash named `%logs` is created with keys of the year and month and values of the log objects for those months.

⟨ Find weight and trend extrema in log entries to be plotted 78 ⟩ ≡

```
my ($cur_y, $cur_m) = ($start_y, $start_m);
my ($first_day, $last_day) = ($start_d, 31);
my ($weight_min, $weight_max, $trend_min, $trend_max, $rung_min, $rung_max) =
        (MAX_VALUE, MIN_VALUE, MAX_VALUE, MIN_VALUE, MAX_VALUE, MIN_VALUE);
my ($trend_mean, $trend_last, $trend_ndays) = (0, 0, 0);

for (my $monkey = sprintf("%04d-%02d", $start_y, $start_m);
     $monkey le sprintf("%04d-%02d", $end_y, $end_m);
     $monkey = sprintf("%04d-%02d", $cur_y, $cur_m)) {
    if (-f "⟨ Users Directory 6h ⟩/$user_file_name/$monkey.hdb") {
        open(FL, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/$monkey.hdb") ||
            die("Cannot open monthly log file ⟨ Users Directory 6h ⟩/$user_file_name/$monkey.hdb");
        my $mlog = HDiet::monthlog->new();
        $logs{$monkey} = $mlog;
        $mlog->load(\*FL);
        close(FL);

        if (($cur_y == $end_y) && ($cur_m == $end_m)) {
            $last_day = $end_d;
        }

        my $logToDisplayUnit = HDiet::monthlog::WEIGHT_CONVERSION->[$mlog->{log_unit}][$ui->{display_unit}
        for (my $i = $first_day; $i <= min($mlog->monthdays(), $last_day); $i++) {
            if (defined($mlog->{weight}[$i])) {
                $weight_min = min($mlog->{weight}[$i] * $logToDisplayUnit, $weight_min);
                $weight_max = max($mlog->{weight}[$i] * $logToDisplayUnit, $weight_max);
            }
            if (defined($mlog->{trend}[$i])) {
                $trend_min = min($mlog->{trend}[$i] * $logToDisplayUnit, $trend_min);
                $trend_max = max($mlog->{trend}[$i] * $logToDisplayUnit, $trend_max);
                $trend_last = $mlog->{trend}[$i] *
                    HDiet::monthlog::WEIGHT_CONVERSION->[$mlog->{log_unit}][HDiet::monthlog::WEIGHT_KILOGR
                $trend_mean += $trend_last;
                $trend_ndays++;
            }
            if (defined($mlog->{rung}[$i])) {
                $rung_min = min($mlog->{rung}[$i] * $logToDisplayUnit, $rung_min);
                $rung_max = max($mlog->{rung}[$i] * $logToDisplayUnit, $rung_max);
            }
        }

        if ($mlog->{trend_carry_forward} > 0) {
            $trend_min = min($trend_min, $mlog->{trend_carry_forward} * $logToDisplayUnit);
            $trend_max = max($trend_max, $mlog->{trend_carry_forward} * $logToDisplayUnit);
         }
    }

    ($cur_y, $cur_m) = HDiet::monthlog::nextMonth($cur_y, $cur_m);
    $first_day = 1;
}

($wgt_min, $wgt_max) = (min($weight_min, $trend_min), max($weight_max, $trend_max));
```

◇

Macro referenced in 75.

6.5.4 Find diet plan extrema on historical chart

If the user has established a diet plan and requested that it be plotted in charts, determine the segment of the plan which falls within this chart. There are three possibilities. If the plan starts after the end of this chart, we plot nothing. If the plan begins during the period of this chart, we plot from the beginning of the plan to the end of the month. If the plan ends within this month, we plot a flat line from the date of the end of the plan to the end of the month, on the assumption that the user intends to maintain the goal weight of the plan after its accomplishment.

⟨ Find diet plan extrema on historical chart 79 ⟩ ≡

```
my ($pjdstart, $pjdend) = (max($start_jd, $$dietcalc[0]), min($end_jd, $$dietcalc[2]));
my ($plan_start_day, $plan_start_weight,
    $plan_end_day, $plan_end_weight) = (-1) x 4;
if (defined($$dietcalc[0])) {
    #   If plan starts before the end of the month, we shall plot it
    if ($pjdstart <= $end_jd) {
        if ($$dietcalc[2] <= $start_jd) {
            #   Plan ends before start of chart; plot flat line at end weight
            $plan_start_day = $start_jd;
            $plan_end_day = $end_jd;
            $plan_start_weight = $plan_end_weight = $$dietcalc[3];
        } else {
            $plan_start_day = $pjdstart;
            $plan_end_day = $pjdend;
            $plan_start_weight = $$dietcalc[1] + (($$dietcalc[3] - $$dietcalc[1]) *
                (($pjdstart - $$dietcalc[0]) / ($$dietcalc[2] - $$dietcalc[0])));
            $plan_end_weight = $$dietcalc[1] + (($$dietcalc[3] - $$dietcalc[1]) *
                (($pjdend - $$dietcalc[0]) / ($$dietcalc[2] - $$dietcalc[0])));
        }
    }
    $plan_start_weight *= HDiet::monthlog::WEIGHT_CONVERSION->[HDiet::monthlog::WEIGHT_KILOGRAM][$ui->{dis
    $plan_end_weight *= HDiet::monthlog::WEIGHT_CONVERSION->[HDiet::monthlog::WEIGHT_KILOGRAM][$ui->{displ
    if (min($plan_start_weight, $plan_end_weight) > 0) {
        $wgt_min = min($wgt_min, min($plan_start_weight, $plan_end_weight));
    }
    if (max($plan_start_weight, $plan_end_weight) > 0) {
        $wgt_max = max($wgt_max, max($plan_start_weight, $plan_end_weight));
    }
}
```

◇

Macro referenced in 75.

6.5.5 Plot the diet plan on historical chart

If a segment of the diet plan appears within this chart, plot it.

⟨ Plot the diet plan on historical chart 80a ⟩ ≡

```
    if ($plan_start_day > 0) {
        my $sx = int(($xAxisLength * ($plan_start_day - $start_jd)) / ($end_jd - $start_jd));
        my $sy = WeightToY($plan_start_weight);
        my $ex = int(($xAxisLength * ($plan_end_day - $start_jd)) / ($end_jd - $start_jd));
        my $ey = WeightToY($plan_end_weight);

        $img->setStyle($yellow, $yellow, $yellow, $yellow,
                       gdTransparent, gdTransparent, gdTransparent, gdTransparent);
        PlotLine($sx, $sy, $ex, $ey, gdStyled);
        if ($plan_end_day < $end_jd) {
            PlotLine($ex, $ey, $xAxisLength, $ey, gdStyled);
        }
    }
```

◇

Macro referenced in 75.

6.5.6 Define historical chart geometry

The following variables specify the layout of items within the chart. They are basically twiddle knobs which are adjusted in the interest of appearance.

⟨ Define historical chart geometry 80b ⟩ ≡

```
    $width = 640 if !defined($width);
    $height = 480 if !defined($height);

    my ($fontLineHeight, $fontCharHeight) = (20, 10);
    my $fontCharWidth = 8;
    ($leftMargin, $rightMargin, $topMargin, $bottomMargin) =
        ($fontCharHeight * (($ui->{display_unit} == HDiet::monthlog::WEIGHT_STONE) ? 6 : 5),
        $fontCharHeight * 3, $fontCharHeight * 2, int($fontCharHeight * 6));
    my ($axisOffset, $tickSize, $sinkerSize) = (3, 5, 4);

    my ($topLeftX, $topLeftY) = (0, 0);
    my ($extentX, $extentY) = ($width, $height);

    my $pixelsPerDay = int(($extentX - ($leftMargin + $rightMargin)) / ($dayspan - 1));
    my $daysPerPixel = int(($dayspan - 1) / ($extentX - ($leftMargin + $rightMargin)));

    my ($bX, $bY) = ($topLeftX + $leftMargin, (($topLeftY + $extentY) - $bottomMargin));
```

◇

Macro referenced in 75.

6.5.7 Determine vertical weight scaling based on extrema

Based on the weight and trend extrema we've found in the log items for the requested date range, scale the vertical axis to include the extrema with an easy to read scale based on factors of 1, 2, or 5, whichever best fits the data and the chart height.

If there's only one day's worth of data or all the entries in the log are identical (it *can* happen), it's possible for `$wgt_min` and `$wgt_max` to be identical, which will result in division by zero in the autoscaling and plotting code. If this is the case, we arbitrarily expand the extrema by one tenth of a weight unit to plot the data centred on the chart.

⟨ Determine vertical weight scaling based on extrema 81a ⟩ ≡

```
 if ($wgt_min == $wgt_max) {
        $wgt_min -= 10;
        $wgt_max += 10;
    }

    my $maxLabelRows = ($height - ($topMargin + $bottomMargin)) / $fontLineHeight;
```

◇

Macro defined by 81ab, 82a.
Macro referenced in 75.

Find a display scale power and factor which permits a suitable number of labels on the weight axis. We do this using units like a vintage Tektronix oscilloscope: increasing powers of 1, 2, and 5.

⟨ Determine vertical weight scaling based on extrema 81b ⟩ ≡

```
    $wgt_max = int($wgt_max * 100);
    $wgt_min = int($wgt_min * 100);
    my $factor = 0;
    my $vunit = 1;
    my $power = 1;
    my @factors = (1, 2, 5);

    while ((($wgt_max - ($wgt_min - ($wgt_min % $vunit))) / ($factors[$factor] * $power)) > $maxLabelRows) {
        $factor++;
        if ($factor > 2) {
            $factor = 0;
            $power *= 10;
        }
        $vunit = $factors[$factor] * $power;
    }
```

◇

Macro defined by 81ab, 82a.
Macro referenced in 75.

There's no point in using a finer-grained weight unit than the one we use to plot weights. Adjust the weight unit if it's smaller than the label increment.

⟨ Determine vertical weight scaling based on extrema 82a ⟩ ≡

```
if ($vunit < 100) {
    $vunit = 100;
}
$vunit /= 100;

$wgt_min -= $wgt_min % $vunit;
$wgt_max = $wgt_max / 100;
$wgt_min = $wgt_min / 100;
```

◇

Macro defined by 81ab, 82a.
Macro referenced in 75.

6.5.8 Draw axes for historical chart

The X and Y axes are drawn to the full extent of the left and bottom edges of the plot area. Date labels are plotted on the X axis in a variety of styles depending upon the relationship between the interval being plotted and the size of the chart.

⟨ Draw axes for historical chart 82b ⟩ ≡

```
#   Y axis
PlotLine(-$axisOffset, -$axisOffset, -$axisOffset, $height - ($topMargin + $bottomMargin), $black);

#   X axis
PlotLine(-$axisOffset, -$axisOffset, $xAxisLength + $axisOffset, -$axisOffset, $black);
```

⟨ Label date axis at the bottom 83 ⟩

◇

Macro referenced in 75.

6.5.8.1 Label date axis at the bottom

Label the bottom of the plot with the dates spanned. We have a variety of formats for this, ranging from showing individual weeks all the way to just two digits for a year, with sufficient years between labels so as to avoid overlaps.

⟨ Label date axis at the bottom 83 ⟩ ≡

```
my @ext;

my $font = 'Times';
my $fontFile = "⟨ TrueType Font Directory 6b ⟩/$font.ttf";
@ext =  GD::Image->stringFT($black, $fontFile, 12, 0, 20, 20, "Mar ");
my $cw = $ext[2] - $ext[0];
@ext =  GD::Image->stringFT($black, $fontFile, 12, 0, 20, 20, "M ");
my $scw = $ext[2] - $ext[0];

my $single = 0;
my $flblinc = ((int(($end_jd - $start_jd) / 30) * $cw) + ($xAxisLength - 1)) / $xAxisLength;
my $lblinc = int($flblinc);
$lblinc = 1 if $lblinc < 1;
if ($lblinc > 1) {
    $lblinc = int(((int(($end_jd - $start_jd) / 30) * $scw) + ($xAxisLength - 1)) / $xAxisLength);
    $lblinc = 1 if $lblinc < 1;
    $single = 1;
}

my ($dt_y, $dt_m, $dt_d) = ($start_y, $start_m, $start_d);

my $cjd = gregorian_to_jd($dt_y, $dt_m, $dt_d);

if ($flblinc < 3) {
    ⟨ Label date axis with years, months, and possibly weeks 84 ⟩
 } else {
    ⟨ Label date axis with year numbers only 85 ⟩
}
```

◇

Macro referenced in 82b.

6.5.8.1.1 Label date axis with years, months, and possibly weeks The horizontal size of the plot is such that we can fit at least labels for months in the plot. First we determine whether the months should be expressed as single- or three-letter abbreviations. If single-letter abbreviations are required, the year numbers shown in lieu of January are shown as two digits; otherwise the full year number is plotted. If the plot spans less than a single month, we label day numbers at week boundaries.

⟨ Label date axis with years, months, and possibly weeks 84 ⟩ ≡

```
        while ($cjd <= $end_jd) {
             my $yearStart = $dt_m == 1;
             my $monster;

             if ($yearStart) {
                 $monster = $single ? sprintf("%02d", $dt_y % 100) : $dt_y;
             } else {
                 $monster = substr($::monthNames[$dt_m], 0, $single ? 1 : 3);
             }

             my $pix = $leftMargin + int(($xAxisLength * ($cjd - $start_jd)) / ($end_jd - $start_jd));
             ::drawText($img, $monster, 'Times', 12, 0,
                 $pix, ($height - $bottomMargin) + 8, 'c', 't', $black);
             $img->line($pix, ($height - $bottomMargin) - $tickSize, $pix,
                             ($height - $bottomMargin) + $axisOffset, $black);

             if (($end_jd - $start_jd) < 32) {
                 my ($nt_y, $nt_m) = ($dt_y, $dt_m + 1);
                 if ($nt_m > 12) {
                     $nt_m = 1;
                     $nt_y++;
                 }
                 my $eom_jd = gregorian_to_jd($nt_y, $nt_m, 1);

                 my $md = (int((($dt_d) - 1) / 7) * 7) + 7;
                 for (my $d = gregorian_to_jd($dt_y, $dt_m, $md);
                      $d < ::min($end_jd, $eom_jd); $d += 7, $md+= 7) {
                     $pix = $leftMargin + int(($xAxisLength * ($d - $start_jd)) / ($end_jd - $start_jd));
                     ::drawText($img, $md, 'Times', 12, 0,
                         $pix, ($height - $bottomMargin) + 8, 'c', 't', $black);
                     $img->line($pix, ($height - $bottomMargin) - $tickSize,
                                $pix, ($height - $bottomMargin) + $axisOffset, $black);
                 }
             }

             $dt_m++;
             $dt_d = 1;
             if ($dt_m > 12) {
                 $dt_y++;
                 $dt_m = 1;
             }
             $cjd = gregorian_to_jd($dt_y, $dt_m, $dt_d);
         }
```

◇

Macro referenced in 83.

6.5.8.1.2 Label date axis with year numbers only There is insufficient room on the date axis to plot months. Plot year numbers skipping sufficient years between years plotted so they don't overlap one another. If four digit year labels would require more than one year per label, we fall back on two digit labels.

⟨ Label date axis with year numbers only 85 ⟩ ≡

```
@ext =  GD::Image->stringFT($black, $fontFile, 12, 0, 20, 20, "2999 ");
$cw = $ext[2] - $ext[0];
$lblinc = int(((int(($end_jd - $start_jd) / 365) * $cw) + ($xAxisLength - 1)) / $xAxisLength);
$lblinc = 1 if $lblinc < 1;

$single = 0;
if ($lblinc > 1) {
    @ext =  GD::Image->stringFT($black, $fontFile, 12, 0, 20, 20, "99 ");
    $cw = $ext[2] - $ext[0];
    $lblinc = int(((int(($end_jd - $start_jd) / 365) * $cw) + ($xAxisLength - 1)) / $xAxisLength);
    $lblinc = 1 if $lblinc < 1;
    $single = 1;
}

my $cjd = $start_jd;

if (($start_m != 1) || ($start_d != 1)) {
    $dt_m = $dt_d = 1;
    $dt_y++;
    $cjd = gregorian_to_jd($dt_y, $dt_m, $dt_d);
}

while ($cjd < $end_jd) {
    my $label_x = $leftMargin + int(($xAxisLength * ($cjd - $start_jd)) / ($end_jd - $start_jd));
    ::drawText($img, $single ? sprintf("%02d", $dt_y % 100) : $dt_y, 'Times', 12, 0,
        $label_x, ($height - $bottomMargin) + $tickSize, 'c', 't', $black);
    $img->line($label_x, ($height - $bottomMargin) - $tickSize, $label_x, ($height - $bottomMargin) + $axi
    $dt_y += $lblinc;
    ($dt_m, $dt_d) = (1, 1);
    $cjd = gregorian_to_jd($dt_y, $dt_m, $dt_d);
}
```

◇

Macro referenced in 83.

6.5.9 Label weight axis at the left

Draw labels and ticks for the vertical weight axis at the left. The scale has already been computed in `$vunit`, so all we need to do here is the actual drawing.

⟨ Label weight axis at the left 86a ⟩ ≡

```
        for (my $plotw = $wgt_min; $plotw <= $wgt_max; $plotw += $vunit) {

            my $ws = HDiet::monthlog::editWeight($plotw, $ui->{display_unit}, $ui->{decimal_character});
            my $wy = WeightToY($plotw);
            main::drawText($img, $ws, 'Times', 12, 0,
                $leftMargin - 8, ($height - $bottomMargin) - $wy, 'r', 'c', $black);
            PlotLine(-$axisOffset, $wy, $tickSize - $axisOffset, $wy, $black);
        }
    ◇
```

Macro referenced in 87.

6.5.10 Label exercise rung axis if any plotted

If we've plotted any day with an exercise rung, include the rung scale at the right of the chart. We always plot the value of the last rung in the month, with the rest of the scale adjusted to skip any value which would overwrite the most recent rung. In the most common case where the rung does not change during the month, this provides a precise identification of the run without the need to interpolate between labels.

⟨ Label exercise rung axis if any plotted 86b ⟩ ≡

```
        if ($lrung) {
            #   Rung axis
            PlotLine($xAxisLength + $axisOffset, -$axisOffset, $xAxisLength + $axisOffset, $height - ($topMargin +

            my $RUNG_EXCLUSION_ZONE = 6;    # How many rungs to exclude around last rung in monthly log
                                            # (Should really be calculated from font metrics and window
                                            #  geometry).

            my $ry = RungToY($lrung);
            main::drawText($img, $lrung, 'Times', 12, 0,
                ($width - $rightMargin) + 8, ($height - $bottomMargin) - $ry, 'o', 'c', $black);
            PlotLine($xAxisLength + $axisOffset, $ry, ($xAxisLength + $axisOffset) - $tickSize, $ry, $black);

           for (my $i = 1; $i <= 48; $i = (int($i / 6) * 6) + 6) {
                if (abs($lrung - $i) >= $RUNG_EXCLUSION_ZONE) {
                    $ry = RungToY($i);
                    main::drawText($img, $i, 'Times', 12, 0,
                        ($width - $rightMargin) + 8, ($height - $bottomMargin) - $ry, 'o', 'c', $black);
                    PlotLine($xAxisLength + $axisOffset, $ry, ($xAxisLength + $axisOffset) - $tickSize, $ry, $blac
                }
            }
        }
    ◇
```

Macro referenced in 87.

6.5.11 Plot weight and rung data on historical chart

With the scale having been determined and the axes drawn and labeled, we are finally ready to actually draw the weight and exercise run lines on the chart. There are two cases: if we're plotting a short interval with respect to the width of the chart, there will be multiple horizontal pixels per day's data and we must draw lines to connect the days. If the interval is sufficiently long, we'll have multiple days' data represented by a single pixel, in which case we plot the arithmetic mean of the values in days the pixel encompasses.

⟨ Plot weight and rung data on historical chart 87 ⟩ ≡

```
if ($wgt_max > 0) {
    ⟨ Label weight axis at the left 86a ⟩
    my $lrung = 0;
    my $nFlagged = 0;
    if ($pixelsPerDay > 1) {
        ⟨ Plot multiple pixels per day 88 ⟩
    } else {
        ⟨ Plot multiple days per pixel 90 ⟩
    }

    ⟨ Label exercise rung axis if any plotted 86b ⟩

    ⟨ Draw caption with trend summary 92 ⟩
    ⟨ Draw title with date range 91 ⟩
} else {
    $img->string(gdMediumBoldFont, $leftMargin + 40, $topMargin + int(($height - ($topMargin + $bottomMarg
        "There are no weight log entries in this date range.", $red);
}
```

◇

Macro referenced in 75.

6.5.11.1 Plot multiple pixels per day

The interval being plotted is sufficiently short that when scaled to the width of the plot each day occupies more than one pixel. Iterate over the days and draw line segments for each day.

⟨ Plot multiple pixels per day 88 ⟩ ≡

```
my ($pix, $opix);
my ($lrg, $ltrend);
my ($ow, $owy) = (0, 0);

for (my $cdate = $start_jd; $cdate <= $end_jd; $cdate++) {
    my $pix = int(($xAxisLength * ($cdate - $start_jd)) / ($end_jd - $start_jd));
    my ($weight, $trend, $rung, $flags) = getDays($cdate, 1, $ui);
    $nFlagged += $flags;
    $weight = 0 if !defined($weight);
    $trend = 0 if !defined($trend);

    #   Plot weight
    if ($weight > 0) {
        if ($pixelsPerDay > int($sinkerSize * 1.5)) {
            ⟨ Plot weight entry as float or sinker 89 ⟩
        } else {
            my $nwy = WeightToY($weight);
            if (($ow > 0) && ($weight > 0)) {
                PlotLine($opix, $owy, $pix, $nwy, $dkgrey);
            }
            $ow = $weight;
            $owy = $nwy;
        }
    }

    #   Plot trend
    my $ny = WeightToY($trend);
    if ($ltrend) {
        if ($trend) {
            PlotLine($opix, $ltrend, $pix, $ny, $red);
        } else {
            PlotLine($opix, $ltrend, $pix, $ltrend, $red);
        }
    }
    $ltrend = $ny if $trend;


    if ($lrg) {
        my $rt = $lrg;
        $lrung = $lrg;
        if ($rung) {
            $rt = $rung;
        }
        PlotLine($opix, RungToY($lrg), $pix, RungToY($rt), $blue);
    }
    $lrg = $rung;

    $opix = $pix;
}
```

◇

Macro referenced in 87.

6.5.11.1.1 Plot weight entry as float or sinker If there is sufficient room between the days on the chart, we plot individual weight entries along with the trend.

Individual weight log entries are plotted as blue diamonds, filled with yellow if the date is flagged and white otherwise. If the weight is above or below the trend line for that day, a green line is drawn to connect it to the trend and indicate whether the day's weight is a "float" pulling the trend up or a "sinker" dragging it down.

⟨ Plot weight entry as float or sinker 89 ⟩ ≡

```
my $ty = WeightToY($trend);
my $wy = WeightToY($weight);
my $offset = $wy - $ty;

if (($offset < -$sinkerSize) || ($offset > $sinkerSize)) {
    my $dy = sgn($offset);

    PlotLine($pix, $ty + $dy, $pix, $wy + (($offset > 0) ? -$sinkerSize : $sinkerSize), $green);
}

#   Fill float/sinker with white or yellow, if it's flagged.

for (my $j = -$sinkerSize; $j <= $sinkerSize; $j++) {
    my $dx = abs($j) - $sinkerSize;

    PlotLine($pix - $dx, ($wy + $j),
             $pix + $dx, ($wy + $j),
             $flags ? $yellow : $white);
}

#   Trace the outline of the float/sinker in blue

PlotLine($pix - $sinkerSize, $wy,
         $pix, $wy - $sinkerSize, $blue);
PlotLine($pix, $wy - $sinkerSize,
         $pix + $sinkerSize, $wy, $blue);
PlotLine($pix + $sinkerSize, $wy,
         $pix, $wy + $sinkerSize, $blue);
PlotLine($pix, $wy + $sinkerSize,
         $pix - $sinkerSize, $wy, $blue);
```

◇

Macro referenced in 88.

6.5.11.2 Plot multiple days per pixel

The interval being plotted is sufficiently long with respect to the width of the plot that we're plotting more than one day's data per pixel in the chart.

⟨ Plot multiple days per pixel 90 ⟩ ≡

```
        my $w;
        my $ot = 0;
        my $t;
        my $rg;
        my $oty;
        my ($ow, $owy) = (0, 0);

        for (my $i = 0; $i < $xAxisLength; $i++) {
            my $sDate = $start_jd + ((($end_jd - $start_jd) * $i) / $xAxisLength);
            my $eDate = $start_jd + ((($end_jd - $start_jd) * ($i + 1)) / $xAxisLength);
            my $nd = int($eDate - $sDate);
            $nd = 1 if ($nd == 0);
            my ($weight, $trend, $rung, $flags) = getDays($sDate, $nd, $ui);
            $nFlagged += $flags;

    #####   FIXME -- OPTION TO PLOT WEIGHT AS FLOAT/SINKER BAND ABOVE/BELOW TREND
            #   Plot weight
            $weight = 0 if !defined($weight);
            my $nwy = WeightToY($weight);
            if (($ow > 0) && ($weight > 0)) {
                PlotLine($i - 1, $owy, $i, $nwy, $dkgrey);
            }
            $ow = $weight;
            $owy = $nwy;

            #   Plot trend
            $trend = 0 if !defined($trend);
            my $nty = WeightToY($trend);
            if (($ot > 0) && ($trend > 0)) {
                PlotLine($i - 1, $oty, $i, $nty, $red);
            }
            $ot = $trend;
            $oty = $nty;


            if ($rung) {
                my $ry = RungToY($rung);
                if ($lrung) {
                    PlotLine($i - 1, RungToY($lrung), $i, $ry, $blue);
                }
                $lrung = $rung;
            }
        }
```

◇

Macro referenced in 87.

6.5.12 Draw title with date range

We place a title at the top of the chart to indicate the range of dates it contains. This is the requested range (which corresponds with the date axis), and is not adjusted for absent data at the start and end of the plot.

⟨ Draw title with date range 91 ⟩ ≡

```
my $title = sprintf("%04d-%02d-%02d &#8211; %04d-%02d-%02d",
    $start_y, $start_m, $start_d, $end_y, $end_m, $end_d);
main::drawText($img, $title, 'Times', 12, 0,
    int($width / 2), $topMargin - 4, 'c', 'b', $black);
```

◇

Macro referenced in 87.

6.5.13 Draw caption with trend summary

Add a caption to the chart with the weight and calorie balance determined by the linear regression fit to the trend line over the period charted. We call `analyseTrend` to perform the analysis, since when we're plotting multiple days per pixel we want to compute the trend based on the underlying data, not the values processed by `getDays`, which on small-scale plots may not examine every day in the interval.

⟨ Draw caption with trend summary 92 ⟩ ≡

```
    my (@intervals, @slopes);
    push(@intervals, sprintf("%04d-%02d-%02d", $start_y, $start_m, $start_d),
                     sprintf("%04d-%02d-%02d", $end_y, $end_m, $end_d));
    @slopes = $self->analyseTrend(@intervals);
    my $tslope = $slopes[0];
    my $fracf = $tFlags / $nDays;
    my $sweekly = $self->{user}->localiseDecimal(sprintf("%.2f", abs($tslope) * 7));
    my $caption;
    if ($width < 480) {
#print(STDERR "Narrow $width:  N = $fitter->{n}  Tslope = $tslope  Fracf = $fracf  Sweekly = $sweekly\n");
        $caption = (($tslope > 0) ? "Gain" : "Loss") .
                " $sweekly " .
                HDiet::monthlog::DELTA_WEIGHT_ABBREVIATIONS->[$ui->{display_unit}] .
                "/wk.  " .
                (($tslope > 0) ? "Excess" : "Deficit") .
                sprintf(": %.0f ", abs($tslope) *
                    (HDiet::monthlog::CALORIES_PER_WEIGHT_UNIT->[$ui->{display_unit}] /
                     HDiet::monthlog::CALORIES_PER_ENERGY_UNIT->[$ui->{energy_unit}])) .
                HDiet::monthlog::ENERGY_ABBREVIATIONS->[$ui->{energy_unit}] . "/day" .
                "." .
                (($fracf > 0) ? sprintf("  %.0f%% flag.", $fracf * 100) : '');
    } else {
#print(STDERR "Wide $width:  N = $fitter->{n}  Tslope = $tslope  Fracf = $fracf  Sweekly = $sweekly\n");
        $caption = 'Weekly ' .
                (($tslope > 0) ? "gain" : "loss") .
                " $sweekly " .
                HDiet::monthlog::DELTA_WEIGHT_UNITS->[$ui->{display_unit}] .
                "s.  Daily " .
                (($tslope > 0) ? "excess" : "deficit") .
                sprintf(": %.0f ", abs($tslope) *
                    (HDiet::monthlog::CALORIES_PER_WEIGHT_UNIT->[$ui->{display_unit}] /
                     HDiet::monthlog::CALORIES_PER_ENERGY_UNIT->[$ui->{energy_unit}])) .
                HDiet::monthlog::ENERGY_UNITS->[$ui->{energy_unit}] . "s" .
                "." .
                (($fracf > 0) ? sprintf("  %.0f%% flagged.", $fracf * 100) : '');
    }
    main::drawText($img, $caption, 'Times', 12, 0,
        int($width / 2), $height - 20, 'c', 'b', $black);
    if (($ui->{height} > 0) && ($trend_ndays > 0) && ($trend_last > 0)) {
        $trend_mean /= $trend_ndays;
        $caption = "Body mass index: mean " .
            $self->{user}->localiseDecimal(sprintf("%.1f", $trend_mean / ($ui->{height} / 100) ** 2)) .
            ", most recent " .
            $self->{user}->localiseDecimal(sprintf("%.1f", $trend_last / ($ui->{height} / 100) ** 2)) . ".";
        main::drawText($img, $caption, 'Times', 12, 0,
            int($width / 2), $height - 4, 'c', 'b', $black);
    }
```

◇

Macro referenced in 87.

6.6 Draw Badge Image

The `drawBadgeImage` method creates a "badge" image suitable for inclusion on a Web page which shows, as of the user's most recent log entry, the current weight, energy balance, and rate of weight gain or loss. The PNG image is written to the file handle given by the first argument, or `STDOUT` if it is not defined. The second argument specifies the interval over which the trend should be computed: positive for a number of days, negative for a number of months.

"HDiet/history.pm" 93a ≡

```
sub drawBadgeImage {
    my $self = shift;

    my ($outfile, $trendDays) = @_;

    my ($ui, $user_file_name) = ($self->{user}, $self->{user_file_name});

    if (!(defined $outfile)) {
        $outfile = \*STDOUT;
    }
```

◇

File defined by 69, 70, 71, 72, 73, 75, 76, 93ab, 94abc, 95, 96, 97, 100, 101a, 103.

Create the image and initialise it from the badge template image file. A full colour image is generated, as the antialiasing of the logo and text fonts, combined with our colour coding the trend-derived information, exceeds the 256 colours available in a palette-mapped image. The image is interlaced so as to load more gracefully on a Web page.

"HDiet/history.pm" 93b ≡

```
    ($width, $height) = (200, 78);  # Badge image size
    my ($printFriendly, $monochrome) = (0, 0);
    $img = GD::Image->newFromPng("⟨ Image and Icon Directory 6c ⟩/badgeback.png", 1);
    die("Cannot load image template ⟨ Image and Icon Directory 6c ⟩/badgeback.png") if !$img;
    ⟨ Allocate colours for chart 42 ⟩
    my $dkgreen =  $img->colorAllocate(  0, 160,   0);
    $img->interlaced('true');
```

◇

File defined by 69, 70, 71, 72, 73, 75, 76, 93ab, 94abc, 95, 96, 97, 100, 101a, 103.

Determine the dates of the first and last log entries in the user's database. If there are no log entries at all, we generate a badge with the legend "No Log Entries" in large bold type. Otherwise, we proceed to determine the interval from the most recently logged weight and the start of the requested trend computation interval. If the requested interval exceeds the length of data in the database, the interval is reduced to begin with the first log entry present.

"HDiet/history.pm" 94a ≡

```
            my ($ly, $lm, $ld, $ldu, $lw, $lt) = $self->lastDay();
            my $l_jd = gregorian_to_jd($ly, $lm, $ld);
            my ($s_y, $s_m, $s_d) = $self->firstDay();
            my $s_jd = gregorian_to_jd($s_y, $s_m, $s_d);

            my ($cx, $cy) = (132, 3);

            if (defined($lw)) {

                my (@intervals, @slopes, $tslope, $deltaW, $deltaE);
```

◇

File defined by 69, 70, 71, 72, 73, 75, 76, 93ab, 94abc, 95, 96, 97, 100, 101a, 103.

If more than one day's log entry is present, fit a trend to the data in the interval and compute the energy balance and rate of gain or loss from its slope.

"HDiet/history.pm" 94b ≡

```
                if (($l_jd - $s_jd) > 1) {
                    my ($f_y, $f_m, $f_d) = $self->firstDayOfInterval($ly, $lm, $ld, $trendDays);
                    my $f_jd = gregorian_to_jd($f_y, $f_m, $f_d);
                    push(@intervals, sprintf("%04d-%02d-%02d", $f_y, $f_m, $f_d),
                                     sprintf("%04d-%02d-%02d", $ly, $lm, $ld));
                    @slopes = $self->analyseTrend(@intervals);
                    $tslope = $slopes[0];
                    $deltaW = sprintf("%.2f", abs($tslope) * 7);
                    $deltaW =~ s/\./$ui->{decimal_character}/;
                    $deltaE = sprintf("%.0f", abs($tslope) *
                        (HDiet::monthlog::CALORIES_PER_WEIGHT_UNIT->[$ui->{display_unit}] /
                         HDiet::monthlog::CALORIES_PER_ENERGY_UNIT->[$ui->{energy_unit}]));
                }
```

◇

File defined by 69, 70, 71, 72, 73, 75, 76, 93ab, 94abc, 95, 96, 97, 100, 101a, 103.

Title the badge with the date of the most recent log entry.

"HDiet/history.pm" 94c ≡

```
                main::drawText($img, sprintf("%04d-%02d-%02d", $ly, $lm, $ld),
                    'DejaVuLGCSans-Bold', 10, 0, $cx, $cy, 'c', 't', $black);
                    $cy += 13;
```

◇

File defined by 69, 70, 71, 72, 73, 75, 76, 93ab, 94abc, 95, 96, 97, 100, 101a, 103.

The most recent weight is drawn below the date in a large, bold font. The weight unit follows the weight entry, for example “75.2 kg”, except when stones and pounds are used, in which case the weight will be given like “11 st 11.8 lb”.

"HDiet/history.pm" 95 ≡

```
                my $ws = HDiet::monthlog::editWeight($lw *
                    HDiet::monthlog::WEIGHT_CONVERSION->[$ldu][$ui->{display_unit}],
                    $ui->{display_unit}, $ui->{decimal_character});
                my ($wu, $eu) = (HDiet::monthlog::WEIGHT_ABBREVIATIONS->[$ui->{display_unit}],
                                 HDiet::monthlog::ENERGY_ABBREVIATIONS->[$ui->{energy_unit}]);
                if ($ui->{display_unit} =~ HDiet::monthlog::WEIGHT_STONE) {
                    $ws =~ s/\s/" $wu "/e;
                    $wu = HDiet::monthlog::WEIGHT_ABBREVIATIONS->[HDiet::monthlog::WEIGHT_POUND];
                }
                $ws .= " $wu";
                main::drawText($img, $ws,
                    'DejaVuLGCSans-Bold', 12, 0, $cx, $cy, 'c', 't', $black);
                $cy += 16;
```

◇

File defined by 69, 70, 71, 72, 73, 75, 76, 93ab, 94abc, 95, 96, 97, 100, 101a, 103.

If we were able to compute a trend slope, append lines giving the length of the trend computation period, the daily energy balance, and the weekly weight gain or loss. If no trend is defined due to insufficient data, show a message to that effect. Finally, write the completed badge image to the output file in PNG format.

"HDiet/history.pm" 96 ≡

```
                if (defined($deltaW)) {
                    #   Trend label
                    main::drawText($img,
                        abs($trendDays) . ' ' . (($trendDays > 0) ? 'Day' : 'Month') . 'Trend',
                        'DejaVuLGCSans', 10, 0, $cx, $cy, 'c', 't', $black);
                    $cy += 14;

                    #   Energy balance
                    main::drawText($img,
                        (($tslope <= 0) ? 'Deficit' : 'Excess') . " $deltaE $eu/day",
                        'DejaVuLGCSans', 10, 0, $cx, $cy, 'c', 't',
                        (($tslope <= 0) ? $dkgreen : $red));
                    $cy += 14;

                    #   Weekly weight change
                    main::drawText($img, (($tslope <= 0) ? 'Loss' : 'Gain') . " $deltaW $wu/week",
                        'DejaVuLGCSans', 10, 0, $cx, $cy, 'c', 't',
                        (($tslope <= 0) ? $dkgreen : $red));
                } else {
                    $cy += 18;
                    main::drawText($img, "Trend not defined.",
                        'DejaVuLGCSans', 10, 0, $cx, $cy, 'c', 't', $black);
                }

            } else {
                $cy += int($height / 2) - 12;
                main::drawText($img, 'No Log',
                    'DejaVuLGCSans-Bold', 12, 0, $cx, $cy, 'c', 'c', $black);
                    $cy += 18;
                main::drawText($img, 'Entries',
                    'DejaVuLGCSans-Bold', 12, 0, $cx, $cy, 'c', 'c', $black);
            }

            print($outfile $img->png());
        }
    ◇
```

File defined by 69, 70, 71, 72, 73, 75, 76, 93ab, 94abc, 95, 96, 97, 100, 101a, 103.

6.7 Generate synthetic data

The `syntheticData` method fills the selected field for the given date range with data generated according to its arguments.

"HDiet/history.pm" 97 ≡

```
        sub syntheticData {
            my $self = shift;

            my ($start_date,                    # Start date: YYYY-MM-DD
                $end_date,                      # End date:   YYYY-MM-DD
                $field_name,                    # Name of field to be filled
                $fill_fraction,                 # Fraction of days to fill
                $start_value,                   # Start value
                $end_value,                     # End value
                $format,                        # Format for rounding numbers
                ) = splice(@_, 0, 7);

            my ($ui, $user_file_name) = ($self->{user}, $self->{user_file_name});

            ⟨ Determine the number of days in the historical interval 77 ⟩
            ⟨ Fill cache with monthly logs in the date range 104 ⟩

            my ($ngen, $nskip) = (0, 0);
            for (my $j = $start_jd; $j <= $end_jd; $j++) {
                if (rand() <= $fill_fraction) {
                    my ($cd_y, $cd_m, $cd_d) = jd_to_gregorian($j);
                    my $v = $start_value + ($end_value - $start_value) * (($j - $start_jd) / ($end_jd - $start_jd)

                    ⟨ Apply perturbation functions to value 99 ⟩

                    $v = sprintf($format, $v);
    #print("    $j  $v<br />\n");
                    my $monkey = sprintf("%04d-%02d", $cd_y, $cd_m);
                    if (!defined($logs{$monkey})) {
                        ⟨ Create new month for synthetic data 98 ⟩
                    }
                    $logs{$monkey}->{$field_name}[$cd_d] = $v;
                    $ngen++;
                } else {
    #print("    $j<br />\n");
                    $nskip++;
                }
            }

            ⟨ Write back all items in the cache 105a ⟩

    if (0) {
    print("<pre>\n");
    for my $l (sort(keys(%logs))) {
        $logs{$l} ->describe();
    }
    print("</pre>\n");
    }
            print("<h3>$ngen days generated, $nskip days skipped.</h3>\n");
        }
    ◇
```

File defined by 69, 70, 71, 72, 73, 75, 76, 93ab, 94abc, 95, 96, 97, 100, 101a, 103.

6.7.1 Create new month for synthetic data

The month for which we're generating data does not appear in the database. Create a blank monthly log into which the data will be generated.

⟨ Create new month for synthetic data 98 ⟩ ≡

```
my $mlog = HDiet::monthlog->new();
$logs{$monkey} = $mlog;
$mlog->{login_name} = $ui->{login_name};
$mlog->{year} = $cd_y;
$mlog->{month} = $cd_m;
$mlog->{log_unit} = $ui->{log_unit};
$mlog->{last_modification_time} = 0;
$mlog->{trend_carry_forward} = 0;
```

◇

Macro referenced in 97.

6.7.2 Apply perturbation functions to value

The list of perturbation functions specified by the balance of the argument list are applied to the current value.

⟨ Apply perturbation functions to value 99 ⟩ ≡

```
for (my $n = 0; $n <= $#_; $n++) {

    #   'uniform', <range>
    if ($_[$n] eq 'uniform') {
        $n++;
        $v += rand($_[$n] * 2) - $_[$n];

    #   'gaussian', <range>
    } elsif ($_[$n] eq 'gaussian') {
        $n++;
        my $g = 0;
        for (my $i = 0; $i < 8; $i++) {
            $g += rand();
        }
        $g /= 8;
        $v += ($_[$n] * 2 * $g) - $_[$n];

    #   'sine', <factor>, <period>, <phase>
    } elsif ($_[$n] eq 'sine') {
        my $factor = $_[++$n];
        my $period = $_[++$n];
        my $phase  = $_[++$n];
        $period = 31 if $period eq '';
        $phase = 0 if $phase eq '';
        my $pi = 4 * atan2(1, 1);
        $v += $factor * sin((2 * $pi) * ((($j + $phase) - $start_jd) / $period));

    } elsif ($_[$n] ne '') {
        die("history::syntheticData: Invalid perturbation function $_[$n]");
    }
}
```

◇

Macro referenced in 97.

6.8 Utility historical queries

The following methods, which are primarily intended to be used internally, may also be called by any possessor of a `history` object.

6.8.1 Last day in database

The last day with a weight entry in the database is returned as a list of (year, month, day, display_unit, weight, trend). If the database contains no weight entries at all, `undef` is returned.

"HDiet/history.pm" 100 ≡

```
sub lastDay {
    my $self = shift;

    ⟨ Obtain list of years 101b ⟩

    for (my $y = $#years; $y >= 0; $y--) {
        my @months = $self->{user}->enumerateMonths($years[$y]);
        for (my $m = $#months; $m >= 0; $m--) {
            ⟨ Ensure month is in cache 102 ⟩
            for (my $d = $logs{$months[$m]}->monthdays(); $d >= 1;  $d--) {
                if ($logs{$months[$m]}->{weight}[$d]) {
                    return ($logs{$months[$m]}->{year}, $logs{$months[$m]}->{month}, $d,
                            $logs{$months[$m]}->{log_unit},
                            $logs{$months[$m]}->{weight}[$d],
                            $logs{$months[$m]}->{trend}[$d]);
                }
            }
        }
    }
    return undef;
}
```

◇

File defined by 69, 70, 71, 72, 73, 75, 76, 93ab, 94abc, 95, 96, 97, 100, 101a, 103.

6.8.2 First day in database

The first day with a weight entry in the database is returned as a list of (year, month, day). If the database contains no weight entries at all, `undef` is returned.

"HDiet/history.pm" 101a ≡

```
sub firstDay {
    my $self = shift;

    ⟨ Obtain list of years 101b ⟩

    for (my $y = 0; $y <= $#years; $y++) {
        my @months = $self->{user}->enumerateMonths($years[$y]);
        for (my $m = 0; $m <= $#months; $m++) {
            ⟨ Ensure month is in cache 102 ⟩
            for (my $d = 1; $d <= $logs{$months[$m]}->monthdays();  $d++) {
                if ($logs{$months[$m]}->{weight}[$d]) {
                    return ($logs{$months[$m]}->{year}, $logs{$months[$m]}->{month}, $d);
                }
            }
        }
    }
    return undef;
}
```

◇

File defined by 69, 70, 71, 72, 73, 75, 76, 93ab, 94abc, 95, 96, 97, 100, 101a, 103.

6.8.2.1 Obtain list of years

If it hasn't previously been done, obtain a list of years for which logs are present for this user.

⟨ Obtain list of years 101b ⟩ ≡

```
if ($#years < 0) {
    @years = $self->{user}->enumerateYears();
}
```

◇

Macro referenced in 100, 101a.

6.8.2.2 Ensure month is in cache

Test whether the present month's log is in the cache. If not, bring it in. We are guaranteed at this point that the log for this month is actually in the database.

⟨ Ensure month is in cache 102 ⟩ ≡

```
if (!$logs{$months[$m]}) {
    open(FL, "<:utf8", "⟨ Users Directory 6h ⟩/$self->{user_file_name}/$months[$m].hdb") ||
        die("Cannot open monthly log file ⟨ Users Directory 6h ⟩/$self->{user_file_name}/$months[$m].hdb");
    my $mlog = HDiet::monthlog->new();
    $logs{$months[$m]} = $mlog;
    $mlog->load(\*FL);
    close(FL);
}
```

◇

Macro referenced in 100, 101a.

6.8.3 First day of interval

Given the date of the last day in an interval specified by a number of days (week, fortnight) or months (month, quarter, year), return the date of the first day in the specified interval. Intervals defined in days are specified by positive arguments, while negative arguments specify spans of months. When backing up by a given number of months leaves us at a day number greater than the days in the starting month, we adjust the date to the last day of that month.

This function may be called either as a method of a history object or directly without reference to an object; it references nothing in the object in any case. There is no check as to whether the computed start of interval is before or after the first day in the database.

"HDiet/history.pm" 103 ≡

```
        sub firstDayOfInterval {
            if ($#_ > 3) {
                my $self = shift;
            }
            my ($year, $month, $day, $interval) = @_;
    #print("Fdoi E($interval): $year-$month-$day\n");

            if ($interval >= 0) {
                my $jdEnd = gregorian_to_jd($year, $month, $day);
                ($year, $month, $day) = jd_to_gregorian($jdEnd - $interval);
            } else {
                while ($interval < 0) {
                    ($year, $month) = HDiet::monthlog::previousMonth($year, $month);
                    $interval++;
                }
                if ($day > HDiet::monthlog::monthdays($year, $month)) {
                    $day = HDiet::monthlog::monthdays($year, $month);
                }
            }

    #print("Fdoi X($interval): $year-$month-$day\n");
            return ($year, $month, $day);
        }
    ◇
```

File defined by 69, 70, 71, 72, 73, 75, 76, 93ab, 94abc, 95, 96, 97, 100, 101a, 103.

6.9 Monthly log cache utilities

To avoid repeated loading and decoding of monthly logs, we maintain a cache in the hash `%logs` which is keyed by the year and month in ISO-8601 format, for example "`2004-09`". The following macros perform various operations on the cache.

6.9.1 Fill cache with monthly logs in the date range

For each month in the range of dates between `$start_date` and `$end_date` we load its monthly log, if extant in the database and not already in the cache, into the `%logs` cache. Note that missing logs within the range will not be present in the cache; it's the responsibility of the code which accesses the cache to handle this appropriately. You must have determined the date span in the request (which precomputes several variables needed here) before invoking this code.

⟨ Fill cache with monthly logs in the date range 104 ⟩ ≡

```
my ($cur_y, $cur_m) = ($start_y, $start_m);

for (my $monkey = sprintf("%04d-%02d", $start_y, $start_m); $monkey le sprintf("%04d-%02d", $end_y, $end_m
    if (!$logs{$monkey}) {
        if (-f "⟨ Users Directory 6h ⟩/$user_file_name/$monkey.hdb") {
            open(FL, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/$monkey.hdb") ||
                die("Cannot open monthly log file ⟨ Users Directory 6h ⟩/$user_file_name/$monkey.hdb");
            my $mlog = HDiet::monthlog->new();
            $logs{$monkey} = $mlog;
            $mlog->load(\*FL);
            close(FL);
        }
    }
    ($cur_y, $cur_m) = HDiet::monthlog::nextMonth($cur_y, $cur_m);
}
```

◇

Macro referenced in 73, 97.

6.9.2 Write back all items in the cache

All monthly logs in the cache are written back to the database. This is unconditional—there is no "dirty" flag—because the only circumstance in which we presently do this is after generation of synthetic data and in that case we know that every month in the cache has been changed (except for the odd case of a low probability of filling days which causes us to miss a month entirely). If some future development modifies a sparse set of months in the cache, it may make sense to add a dirty flag and be more selective.

⟨ Write back all items in the cache 105a ⟩ ≡

```
for my $k (keys(%logs)) {
    my $mlog = $logs{$k};
    $mlog->{last_modification_time} = time();
    open(FL, ">:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/$k.hdb") ||
        die("Cannot update monthly log file ⟨ Users Directory 6h ⟩/$user_file_name/$k.hdb");
    $mlog->save(\*FL);
    close(FL);
    clusterCopy("⟨ Users Directory 6h ⟩/$user_file_name/$k.hdb");
}

if (scalar(keys(%logs)) > 0) {
    if ($self->{user}->{badge_trend} != 0) {
        open(FB, ">⟨ Users Directory 6h ⟩/$user_file_name/BadgeImageNew.png") ||
            die("Cannot update monthly log file ⟨ Users Directory 6h ⟩/$user_file_name/BadgeImageNew.png");
        $self->drawBadgeImage(\*FB, $self->{user}->{badge_trend});
        close(FB);
        ::do_command("mv ⟨ Users Directory 6h ⟩/$user_file_name/BadgeImageNew.png ⟨ Users Directory 6h ⟩/$user_fi
        clusterCopy("⟨ Users Directory 6h ⟩/$user_file_name/BadgeImage.png");
    }
}
```

◇

Macro referenced in 97.

6.9.3 Empty monthly log cache

The monthly log cache is emptied. We count on Perl's reference counts to delete all of the associated memory.

⟨ Empty monthly log cache 105b ⟩ ≡

```
%logs = ();
@years = ();
```

◇

Macro referenced in 75.

Chapter 7

Aggregator.pm: Data Aggregator Object

The `Aggregator` object provides access to data belonging to a specified set of user accounts for the purpose of analysis of aggregate data.

7.1 Package plumbing

"HDiet/Aggregator.pm" 106 ≡

```
#! ⟨ Perl directory 7d ⟩

    ⟨ Perl language modes 369b ⟩

    use HDiet::monthlog;

    package HDiet::Aggregator;

    use HDiet::Julian;

    require Exporter;

    our @ISA = qw(Exporter);
    our @EXPORT = ( );

    1;
```

◇

File defined by 106, 107, 108.

7.2 Constructor

A new `Aggregator` object is created by calling the `new` constructor. The constructor is called with the function which will receive the log records returned and the weight unit in which the requester wishes to receive weight and trend values.

"HDiet/Aggregator.pm" 107 ≡

```
sub new {
    my $self = {};
    my ($invocant, $receiver, $weight_unit) = @_;
    my $class = ref($invocant) || $invocant;

    bless($self, $class);

    #   Initialise instance variables from constructor arguments
    $self->{receiver} = $receiver;
    $self->{weight_unit} = $weight_unit;

    return $self;
}
```

◇

File defined by 106, 107, 108.

7.3 Retrieve

The `retrieve` method returns log records within the specified date range for the set of users defined by the following arguments. The `$start_jd` and `$end_jd` arguments specify the inclusive date range to be returned as Julian dates. If `$public_only` is true, only public accounts will be included in the aggregation. If the optional `$user_list` is supplied, only the user names listed in the array reference argument will be included in the aggregation (and only those which are public if `$public_only` is set).

The method returns a list of the total number of account and the number of public accounts. Note that these totals may be larger than the number of distinct accounts in the data returned to the `receiver` function, since accounts which have no log items within the requested date range will never be passed to the receiver.

"HDiet/Aggregator.pm" 108 ≡

```
sub retrieve {
    my $self = shift;
    my ($start_jd, $end_jd, $public_only, $user_list) = @_;

    my $receive = $self->{receiver};
    my ($from_y, $from_m, $from_d) = jd_to_gregorian($start_jd);
    my ($to_y, $to_m, $to_d) = jd_to_gregorian($end_jd);
    my $sdate = sprintf("%04d-%02d-%02d", $from_y, $from_m, $from_d);
    my $edate = sprintf("%04d-%02d-%02d", $to_y, $to_m, $to_d);
    my ($naccts, $npaccts) = (0, 0);

    if (defined($user_list)) {
        my @users = @$user_list;
        for my $u (@users) {
            my $user_file_name = HDiet::user::quoteUserName($u);
            ⟨ Return log items for aggregation from this user 109 ⟩
        }
    } else {
        opendir(CD, "⟨ Users Directory 6h ⟩") ||
            die("Cannot open directory ⟨ Users Directory 6h ⟩");
        for my $user_file_name (grep(!/\.\.?\z/, readdir(CD))) {
            ⟨ Return log items for aggregation from this user 109 ⟩
        }
        closedir(CD);
    }

    return ($naccts, $npaccts);
}
```

◇

File defined by 106, 107, 108.

7.3.1 Return log items for aggregation from this user

Load the `user` object for this account, verify that it is public if the requester wishes only to see public accounts, and then load successive `monthlog` objects within the requested date range and return log items within them.

⟨ Return log items for aggregation from this user 109 ⟩ ≡

```
#my $recret = 0;
    open(FU, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/UserAccount.hdu") ||
        die("Cannot open user account directory ⟨ Users Directory 6h ⟩/$user_file_name/UserAccount.hdu");
    my $ui = HDiet::user->new();
    $ui->load(\*FU);
    close(FU);
    $naccts++;
    $npaccts++ if $ui->{public};
    if ((!$public_only) || $ui->{public}) {
        my ($cur_y, $cur_m, $cur_d) = ($from_y, $from_m, $from_d);
        for (my $j = $start_jd; $j <= $end_jd; ) {
            my $monkey = sprintf("%04d-%02d", $cur_y, $cur_m);
            if (-f "⟨ Users Directory 6h ⟩/$user_file_name/$monkey.hdb") {
                open(FL, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/$monkey.hdb") ||
                    die("Cannot open monthly log file ⟨ Users Directory 6h ⟩/$user_file_name/$monkey.hdb");
                my $mlog = HDiet::monthlog->new();
                $mlog->load(\*FL);
                close(FL);
                for (my $dd = $cur_d; $dd <= $mlog->monthdays(); $dd++) {
                    my ($rw, $rt) = ($mlog->{weight}[$dd], $mlog->{trend}[$dd]);
                    $rw *= HDiet::monthlog::WEIGHT_CONVERSION->[$mlog->{log_unit}][$self->{weight_unit}]
                        if defined($rw);
                    $rt *= HDiet::monthlog::WEIGHT_CONVERSION->[$mlog->{log_unit}][$self->{weight_unit}]
                        if defined($rt);
                    &$receive($ui, $j,
                         $rw,
                         $rt,
                         $mlog->{rung}[$dd],
                         $mlog->{flag}[$dd],
                         $mlog->{comment}[$dd]);
                    $j++;
                    if ($j > $end_jd) {
                        last;
                    }
                }
            }
            $cur_m++;
            $cur_d = 1;
            if ($cur_m > 12) {
                $cur_y++;
                $cur_m = 1;
                $j = gregorian_to_jd($cur_y, $cur_m, $cur_d);
            }
        }
    }
◇
```

Macro referenced in 108.

Chapter 8

user.pm: User Object

The user object represents a user account. All information associated with the user is stored and all account management operations are implemented as methods.

8.1 Package plumbing

"HDiet/user.pm" 110 ≡

```
#! ⟨ Perl directory 7d ⟩

    ⟨ Perl language modes 369b ⟩

    use HDiet::monthlog qw();

    package HDiet::user;

    use Encode qw(encode_utf8);
    use Digest::SHA1  qw(sha1_hex);
    use Crypt::OpenSSL::AES;
    use Crypt::CBC;

    use HDiet::html;
    use HDiet::xml;
    use HDiet::Julian;
    use HDiet::Digest::Crc32;

    require Exporter;
    our @ISA = qw(Exporter);
    our @EXPORT = qw( quoteUserName );
    1;

    use constant FILE_VERSION => 1;

◇
```

File defined by 110, 112, 113, 114, 115, 116a, 117, 118, 119, 127ab, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138ab, 139.

8.2 Constructor

A new user object is created by calling the new constructor.

login_name	User login name
password	Password
account_created	Date and time account created
first_name	First name
last_name	Last name
middle_name	Middle name
e_mail	E-mail address
log_unit	Log weight unit
display_unit	Display weight unit
energy_unit	Energy unit
height	Height in centimetres (for BMI)
calc_calorie_balance	Calorie balance (calculator)
calc_start_weight	Start weight (calculator)
calc_goal_weight	Goal weight (calculator)
calc_start_date	Start date (calculator)
plot_diet_plan	Show diet plan in charts?
current_rung	Current exercise rung
public	Make database publicly visible?
public_name	Name shown to the public
public_since	When account made public
administrator	Does user have administrator privileges?
read_only	Allow read-only access without password?
last_modification_time	UNIX time of last modification
decimal_character	Decimal character (“.” or “,”)
badge_trend	Badge trend interval, 0 for no badge

"HDiet/user.pm" 112 ≡

```
sub new {
    my $self = {};
    my ($invocant, $login_name) = @_;
    my $class = ref($invocant) || $invocant;

    $login_name = '' if !defined($login_name);

    bless($self, $class);

    $self->{version} = FILE_VERSION;

    #   Initialise instance variables
    $self->{login_name} = $login_name;
    $self->{password} = '';
    $self->{password_expires} = 0;
    $self->{account_created} = 0;

    $self->{first_name} = '';
    $self->{last_name} = '';
    $self->{middle_name} = '';
    $self->{e_mail} = '';
    $self->{log_unit} = HDiet::monthlog::WEIGHT_KILOGRAM;
    $self->{display_unit} = HDiet::monthlog::WEIGHT_KILOGRAM;
    $self->{energy_unit} = HDiet::monthlog::ENERGY_CALORIE;
    $self->{height} = 0;
    $self->{calc_calorie_balance} = -500;
    $self->{calc_start_weight} = 0;
    $self->{calc_goal_weight} = 0;
    $self->{calc_start_date} = 0;
    $self->{plot_diet_plan} = 0;
    $self->{current_rung} = 0;
    $self->{public} = 0;
    $self->{public_name} = '';
    $self->{public_since} = 0;
    $self->{administrator} = 0;
    $self->{read_only} = 0;
    $self->{last_modification_time} = 0;
    $self->{decimal_character} = '.';
    $self->{badge_trend} = 0;

    return $self;
}
```

◇

File defined by 110, 112, 113, 114, 115, 116a, 117, 118, 119, 127ab, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138ab, 139.

8.3 Login

A new `user` object can also be created by calling the `login` method with the user name and password. The user record is looked up in the User Directory, and if the password is correct, a new `user` object is constructed and initialised with the user's settings from the record.

"HDiet/user.pm" 113 ≡

```
        sub login {
            my $self = {};
            my ($invocant, $login_name, $password) = @_;
            my $class = ref($invocant) || $invocant;

            bless($self, $class);

            return $self;
        }
```

◇

File defined by 110, 112, 113, 114, 115, 116a, 117, 118, 119, 127ab, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138ab, 139.

8.4 Describe

The `describe` method prints a primate-readable description of the user on the file handle (default `STDOUT`) given by the argument.

"HDiet/user.pm" 114 ≡

```
sub describe {
    my $self = shift;
    my ($outfile) = @_;

    if (!(defined $outfile)) {
        $outfile = \*STDOUT;
    }

    print($outfile "USER Version: $self->{version}\n");
    print($outfile "  Login: '$self->{login_name}'  Password: $self->{password}  " ."\n");
    print($outfile "  Password expires: " . (($self->{password_expires} == 0) ? "Never" :
        localtime($self->{password_expires})) . "\n");
    print($outfile "  First login: " . localtime($self->{account_created}) . "\n");
    print($outfile "  Name:  First '$self->{first_name}'  " .
                   "Middle '$self->{middle_name}'  " .
                   "Last '$self->{last_name}'\n");
    print($outfile "  E-mail: $self->{e_mail}\n");
    print($outfile "  Log unit: " . HDiet::monthlog::WEIGHT_UNITS->[$self->{log_unit}] .
                   "  Display unit: " . HDiet::monthlog::WEIGHT_UNITS->[$self->{display_unit}] .
                   "  Energy unit: " . HDiet::monthlog::ENERGY_UNITS->[$self->{energy_unit}] . "\n");
    print($outfile "  Height: $self->{height}  " .
                   "Log public: " . ($self->{public} ? "Yes" : "No") .
                   "  Administrator: " . ($self->{administrator} ? "Yes" : "No") .
                   "  Read only: " . ($self->{read_only} ? "Yes" : "No") . "\n");
    if ($self->{public}) {
        print($outfile "  Public name: '$self->{public_name}'  Since: " .
            localtime($self->{public_since}) . "\n");
    }
    print($outfile "  Calculator:  Balance: " . $self->{calc_calorie_balance} .
                   "  Start weight: " . $self->{calc_start_weight} .
                   "  Goal weight: " . $self->{calc_goal_weight} . "\n" .
                   "               Plot plan: " . ($self->{plot_diet_plan} ? "Yes" : "No") .
                   "  Start date: " . localtime($self->{calc_start_date}) . "\n");
    print($outfile "  Last modification time: " .
                   localtime($self->{last_modification_time}) . "\n");
    print($outfile "  Decimal character: " . $self->{decimal_character} . "\n");
    print($outfile "  Badge trend interval: $self->{badge_trend}\n");
}
```

◇

File defined by 110, 112, 113, 114, 115, 116a, 117, 118, 119, 127ab, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138ab, 139.

8.5 Save

The `save` method writes the user item to the already-open file handle passed as the argument.

"HDiet/user.pm" 115 ≡

```
        sub save {
            my $self = shift;
            my ($outfile) = @_;

            print $outfile <<"EOD";
    $self->{version}
    $self->{login_name}
    $self->{password}
    $self->{password_expires}
    $self->{account_created}
    $self->{first_name}
    $self->{last_name}
    $self->{middle_name}
    $self->{e_mail}
    $self->{log_unit}
    $self->{display_unit}
    $self->{energy_unit}
    $self->{height}
    $self->{calc_calorie_balance}
    $self->{calc_start_weight}
    $self->{calc_goal_weight}
    $self->{calc_start_date}
    $self->{plot_diet_plan}
    $self->{current_rung}
    $self->{public}
    $self->{public_name}
    $self->{public_since}
    $self->{administrator}
    $self->{read_only}
    $self->{last_modification_time}
    $self->{decimal_character}
    $self->{badge_trend}
    EOD
        }
    ◇
```

File defined by 110, 112, 113, 114, 115, 116a, 117, 118, 119, 127ab, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138ab, 139.

8.6 Load

The `load` method reads a user file from the argument file handle in the format produced by `save`.

"HDiet/user.pm" 116a ≡

```
sub load {
    my $self = shift;
    my ($infile) = @_;

    my $s = in($infile);

    if ($s != FILE_VERSION) {
        die("user::load: Incompatible file version $s");
    }

    $self->{login_name} = in($infile);
    $self->{password} = in($infile);
    $self->{password_expires} = in($infile);
    $self->{account_created} = in($infile);
    $self->{first_name} = in($infile);
    $self->{last_name} = in($infile);
    $self->{middle_name} = in($infile);
    $self->{e_mail} = in($infile);
    $self->{log_unit} = in($infile);
    $self->{display_unit} = in($infile);
    $self->{energy_unit} = in($infile);
    $self->{height} = in($infile);
    $self->{calc_calorie_balance} = in($infile);
    $self->{calc_start_weight} = in($infile);
    $self->{calc_goal_weight} = in($infile);
    $self->{calc_start_date} = in($infile);
    $self->{plot_diet_plan} = in($infile);
    $self->{current_rung} = in($infile);
    $self->{public} = in($infile);
    $self->{public_name} = in($infile);;
    $self->{public_since} = in($infile);;
    $self->{administrator} = in($infile);
    $self->{read_only} = in($infile);
    $self->{last_modification_time} = in($infile);
    $self->{decimal_character} = in($infile, '.');
    $self->{badge_trend} = in($infile, 0);
}
```

⟨ Read line from persistent object file (116b `user`) 390b ⟩

◇

File defined by 110, 112, 113, 114, 115, 116a, 117, 118, 119, 127ab, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138ab, 139.

8.7 Login Form

The `login_form` method writes the HTML form presented to a user who wishes to log in. The form is written to the file handle argument, which defaults to `STDOUT` if not specified. The user name and password fields are initialised to the values in the parent object. These will usually be blank for a new login, but may be preset should arcane circumstances require them to be so. The third argument specifies the login is from a handheld device. The handheld checkbox is preset, and the login form is customised for a small screen. The optional fourth argument specifies whether the "remember me" checkbox should be checked.

"HDiet/user.pm" 117 ≡

```
sub login_form {
    my $self = shift;
    my ($fh, $tzOff, $handheld, $remember) = @_;

    if (!(defined $fh)) {
        $fh = \*STDOUT;
    }

    my ($login_name, $password) = (
                    quoteHTML($self->{login_name}),
                    quoteHTML($self->{password})
                );

    my $ckhandheld = $handheld ? ' checked="checked"' : '';
    my $ckremember = $remember ? ' checked="checked"' : '';
    my $arghandheld = $handheld ? '&amp;HDiet_handheld=y' : '';
    print $fh <<"EOD";
<form id="Hdiet_login" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩

◇
```

File defined by 110, 112, 113, 114, 115, 116a, 117, 118, 119, 127ab, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138ab, 139.

The user name and password fields are enclosed in a table. Below this are the checkboxes for “Handheld” and “Remember me” mode. A link for password recovery and a button to create a new account.

"HDiet/user.pm" 118 ≡

```
<table border="border" class="login">
<tr><th><label for="HDiet_username"><span class="accesskey">U</span>ser Name:</label></th>
    <td><input accesskey="u" type="text" name="HDiet_username" id="HDiet_username" size="60"
               tabindex="1" maxlength="⟨ Maximum Text Input Field Length 9f ⟩" value="$login_name" /></td>
</tr>
<tr><th><label for="HDiet_password"><span class="accesskey">P</span>assword:</label></th>
    <td><input accesskey="p" type="password" name="HDiet_password" id="HDiet_password" size="60"
               tabindex="2" maxlength="⟨ Maximum Text Input Field Length 9f ⟩" value="$password" /></td>
</tr>
</table>
<p class="mlog_buttons">
<input type="hidden" name="q" value="validate_user" />
<input type="submit" tabindex="3" name="login" value=" Sign In " />
 
<input type="reset" value=" Reset " />
<br />
<input type="checkbox" name="HDiet_handheld" id="HDiet_handheld"
       value="y"$ckhandheld /> <label for="HDiet_handheld">Handheld device</label>
 
<input type="checkbox" name="HDiet_remember" id="HDiet_remember"
       value="y"$ckremember /> <label for="HDiet_remember">Remember me</label>
<br />
<a href="⟨ URL to invoke this program 12a ⟩?q=pwreset$arghandheld$tzOff">Forgotten
    your password?</a>
</p>
<p class="mlog_buttons">
<input type="submit" name="new" value=" Create a New Account " />
</p>
</form>
EOD

        if ((!$handheld) && (⟨ Beta test 3e ⟩)) {
            print $fh <<"EOD";
<h3 class="centred">Development log now online at:<br />
<a href="http://hdonline-dev.blogspot.com/"
   rel="Target:Fourmilab_Hdonline_Devlog">http://hdonline-dev.blogspot.com/</a>
</h3>
EOD
        }
    }
```

◇

File defined by 110, 112, 113, 114, 115, 116a, 117, 118, 119, 127ab, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138ab, 139.

8.8 New Account Form

The `new_account_form` method writes an HTML form to define a new user account on the file handle (default `STDOUT`) given by the argument. The fields in the form are initialised to the values in the user object. A form suitable for editing an existing account can be generated by calling this method with a second nonzero argument.

"HDiet/user.pm" 119 ≡

```
        sub new_account_form {
            my $self = shift;
            my ($fh, $edit_mode) = @_;

            if (!(defined $fh)) {
                $fh = \*STDOUT;
            }

            ⟨ Encode preset values for use in HTML 120 ⟩

            print $fh <<"EOD";
    <table border="border" class="login">

    EOD
            ⟨ User login name text field 121a ⟩

            if (⟨ Beta test 3e ⟩) {
                ⟨ Beta test invitation field 121b ⟩
            }

            my $ch_logunit = $edit_mode ? '' : ' onclick="set_logunit(this);"';
            my $ch_dispunit = $edit_mode ? '' : ' onclick="set_dispunit(this);"';

            print $fh <<"EOD";
    ⟨ Password and password confirmation fields 122 ⟩

    ⟨ E-mail address text field 123a ⟩

    ⟨ User's full name optional text fields 123b ⟩

    ⟨ Height (for body mass index) 124 ⟩

    ⟨ Weight and energy unit radio buttons 125 ⟩

    ⟨ Decimal character selection 126a ⟩
    EOD
            ⟨ Public name settings 126b ⟩

            print $fh <<"EOD";
    </table>
    EOD
        }
    ◇
```

File defined by 110, 112, 113, 114, 115, 116a, 117, 118, 119, 127ab, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138ab, 139.

8.8.1 Encode preset values for use in HTML

Convert the preset values from the user object to HTML encoding to be used in the form. This allows preserving values specified by the user when the form is re-issued due to an error.

⟨ Encode preset values for use in HTML 120 ⟩ ≡

```
my ($login_name, $first_name, $last_name, $middle_name,
    $e_mail) = (
                quoteHTML($self->{login_name}),
                quoteHTML($self->{first_name}),
                quoteHTML($self->{last_name}),
                quoteHTML($self->{middle_name}),
                quoteHTML($self->{e_mail})
            );

my %wunit = (0, '', 1, '', 2, '');
$wunit{$self->{log_unit}} = 'checked="checked"';

my %dunit = (0, '', 1, '', 2, '');
$dunit{$self->{display_unit}} = 'checked="checked"';

my %eunit = (0, '', 1, '');
$eunit{$self->{energy_unit}} = 'checked="checked"';

my %dchar = ('.', '', ',', '');
$dchar{$self->{decimal_character}} = 'checked="checked"';

my ($height_cm, $height_ft, $height_in) = ('', '', '');
if ($self->{height} > 0) {
    $height_cm = $self->localiseNumber($self->{height}, 1);
    $height_in = canonicalNumber($self->{height}, 1) / 2.54;
    $height_ft = int($height_in / 12);
    $height_in = $self->localiseNumber($height_in - ($height_ft * 12), 1);
}
```

◇

Macro referenced in 119.

8.8.2 User login name text field

The user login or account name must be unique system-wide. We derive the directory name in which all files for this user are kept from the login name. The login name can be any Unicode character string up to the maximum input field length; quoting and digests are used to deal with system file name character set and length restrictions.

⟨ User login name text field 121a ⟩ ≡

```
        if ($edit_mode) {
            my $llg = $login_name;
            if ($self->{administrator}) {
                $llg .= ' <span class="administrator">(Administrator)</span>';
            }
            print $fh <<"EOD";
    <tr><th>User Name:</th>
        <td><b>$llg</b></td>
    </tr>
    EOD
        } else {
            print $fh <<"EOD";
    <tr><th><span class="required">*</span> <label
            for="HDiet_username"><span class="accesskey">U</span>ser Name:</label></th>
        <td><input accesskey="u" type="text" name="HDiet_username" id="HDiet_username" size="60" tabindex="1"
                maxlength="⟨ Maximum Text Input Field Length 9f ⟩" value="$login_name" /></td>
    </tr>
    EOD
        }
    ◇
```

Macro referenced in 119.

8.8.3 Beta test invitation field

During the beta test period, new account creation requires the user to enter an invitation code. These codes are generated by the administrator and given out to the chosen few deemed sufficiently resilient to brave this software in its pre-release form.

⟨ Beta test invitation field 121b ⟩ ≡

```
        if (!$edit_mode) {
            print $fh <<"EOD";
    <tr><th><span class="required">*</span> <label for="HDiet_invitation"><span
            class="accesskey">B</span>eta test invitation:</label></th>
        <td><input accesskey="B" type="text" name="HDiet_invitation" id="HDiet_invitation" size="12"  tabindex="2"
                maxlength="⟨ Maximum Text Input Field Length 9f ⟩" value="" /></td>
    </tr>
    EOD
        }
    ◇
```

Macro referenced in 119.

8.8.4 Password and password confirmation fields

Two "password" type input fields are used to specify the password for the account. Because input typed in the field is hidden, the user is required to enter the password twice to guarantee it is the intended value. (Personally, I think the idea of hiding a password the user is entering for a new account or in a password change request is idiotic; many people type poorly, and even with confirmation there's no feedback to tell the user they didn't for example, accidentally transpose two characters, rendering the new account inaccessible. Still, the hidden field and confirmation scheme is what people have come to expect, and it's next to certain that if I don't follow this dumb convention, some moron will slam the process for being "insecure". My feeling is that somebody who is unable to arrange things so that nobody is looking over their shoulder when they create a new account [as opposed to a routine login]) is probably too stupid to use a computer for any confidential data whatsoever.)

There is, however, one advantage of designating a field a password. Most browsers will not remember the value of such fields and re-enter them if the user presses the "Back" button. This is a genuine improvement in security and another reason to bow to convention here.

⟨ Password and password confirmation fields 122 ⟩ ≡

```
<tr><th><span class="required">*</span> <label for="HDiet_password"><span
        class="accesskey">P</span>assword:</label></th>
    <td><input accesskey="p" type="password" name="HDiet_password"
               id="HDiet_password" size="48"  tabindex="2"
               onkeyup="showPasswordStrength(); checkPasswordMatch();"
               onchange="showPasswordStrength(); checkPasswordMatch();"
               maxlength="⟨ Maximum Text Input Field Length 9f ⟩" value="" />
               Strength: <input type="text" name="HDiet_password_strength" size="2"
               maxlength="3" readonly="readonly" tabindex="0" value="0" /></td>
</tr>

<tr><th><span class="required">*</span> <label for="HDiet_rpassword"><span
        class="accesskey">R</span>etype password:</label></th>
    <td><input accesskey="r" type="password" name="HDiet_rpassword"
               id="HDiet_rpassword" size="48"  tabindex="3"
               onkeyup="checkPasswordMatch();"
               onchange="checkPasswordMatch();"
               maxlength="⟨ Maximum Text Input Field Length 9f ⟩" value="" />
               Match? <input type="checkbox" name="HDiet_password_match"
               readonly="readonly" tabindex="0" checked="checked" /></td>
</tr>
◇
```

Macro referenced in 119.

8.8.5 E-mail address text field

We require the user to enter an E-mail address when creating the account. This address is primarily to facilitate password recovery. If the user loses their password, we can either E-mail the existing password to the designated address or else generate a new password and send that. We may also eventually require the user to receive and respond to a URL in an E-mail send to this address in order to activate the new account. This will verify that the E-mail address is valid and prevent attacks which create large numbers of bogus accounts with invalid E-mail addresses, or sign up unwitting third parties with E-mail addresses collected from junk mail databases.

⟨ E-mail address text field 123a ⟩ ≡

```
<tr><th><span class="required">*</span> <label for="HDiet_email"><span
    class="accesskey">E</span>-mail address:
     (for lost <br /> password recovery)</label></th>
    <td><input accesskey="e" type="text" name="HDiet_email" id="HDiet_email" size="60" tabindex="4"
               maxlength="⟨ Maximum Text Input Field Length 9f ⟩" value="$e_mail" /></td>
</tr>
◇
```

Macro referenced in 119.

8.8.6 User's full name optional text fields

The following fields, all of which are optional, allow the user to specify their first, last, and middle name. Since these are used simply to label output, their interpretation is entirely up to the user, who may specify them in whichever order they prefer their name to be written. These fields may contain any arbitrary Unicode characters.

⟨ User's full name optional text fields 123b ⟩ ≡

```
<tr><th><label for="HDiet_namef">First name:</label></th>
    <td><input type="text" name="HDiet_namef" id="HDiet_namef" size="60" tabindex="5"
               maxlength="⟨ Maximum Text Input Field Length 9f ⟩" value="$first_name" /></td>
</tr>

<tr><th><label for="HDiet_namel">Last name:</label></th>
    <td><input type="text" name="HDiet_namel" id="HDiet_namel" size="60" tabindex="6"
               maxlength="⟨ Maximum Text Input Field Length 9f ⟩" value="$last_name" /></td>
</tr>

<tr><th><label for="HDiet_namem">Middle name or initial:</label></th>
    <td><input type="text" name="HDiet_namem" id="HDiet_namem" size="60" tabindex="7"
               maxlength="⟨ Maximum Text Input Field Length 9f ⟩" value="$middle_name" /></td>
</tr>
◇
```

Macro referenced in 119.

8.8.7 Height (for body mass index)

If the user wishes to have their body mass index (BMI) displayed, they must specify their stature. We allow this to be specified in either centimetres or in feet and inches, although we always store the value internally in centimetres. When an entry is made in one unit field, a little JavaScript glue updates the other unit accordingly. This is purely for the user's convenience; no harm will be done if JavaScript is disabled.

Note that this field is optional. If not specified, the `height` value in the object will be zero, and no body mass index will be displayed. The BMI has a rather high coefficient of bogosity, and while some people find it useful, we don't want to go so far as to endorse it by requiring the user to disclose their height purely in order to compute it.

⟨ Height (for body mass index) 124 ⟩ ≡

```
<tr><th>Height:</th>
    <td>
        <input type="text" name="HDiet_height_cm" id="HDiet_height_cm" size="5" tabindex="8"
            maxlength="6" value="$height_cm" onchange="height_changed_cm();" />
                <label for="HDiet_height_cm">centimetres</label>
            <b>or</b>    
        <input type="text" name="HDiet_height_ft" id="HDiet_height_ft" size="2" tabindex="9"
            maxlength="2" value="$height_ft" onchange="height_changed_ft();" />
                <label for="HDiet_height_ft">feet</label>
        <input type="text" name="HDiet_height_in" id="HDiet_height_in" size="4" tabindex="10"
            maxlength="4" value="$height_in" onchange="height_changed_in();" />
                <label for="HDiet_height_in">inches</label>
    </td>
</tr>
◇
```

Macro referenced in 119.

8.8.8 Weight and energy unit radio buttons

Radio buttons are used to specify the weight and energy units the user prefers. These are preset based on the setting in the parent object.

⟨ Weight and energy unit radio buttons 125 ⟩ ≡

```
<tr><th>Weight unit:</th>
    <td>
    <table>
    <tr><td>
    <b>Log:</b></td><td>
    <input type="radio" name="HDiet_wunit" id="HDiet_wunit_kg" value="0"$wunit{0}
        tabindex="11"$ch_logunit /> <label for="HDiet_wunit_kg">kilogram</label>
    <input type="radio" name="HDiet_wunit" id="HDiet_wunit_lb" value="1"$wunit{1}
        tabindex="12"$ch_logunit /> <label for="HDiet_wunit_lb">pound</label>
    <input type="radio" name="HDiet_wunit" id="HDiet_wunit_st" value="2"$wunit{2}
        tabindex="13"$ch_logunit /> <label for="HDiet_wunit_st">stone</label>
    </td></tr>
    <tr><td>
    <b>Display:</b></td><td>
    <input type="radio" name="HDiet_dunit" id="HDiet_dunit_kg" value="0"$dunit{0}
        tabindex="14"$ch_dispunit /> <label for="HDiet_dunit_kg">kilogram</label>
    <input type="radio" name="HDiet_dunit" id="HDiet_dunit_lb" value="1"$dunit{1}
        tabindex="15"$ch_dispunit /> <label for="HDiet_dunit_lb">pound</label>
    <input type="radio" name="HDiet_dunit" id="HDiet_dunit_st" value="2"$dunit{2}
        tabindex="16"$ch_dispunit /> <label for="HDiet_dunit_st">stone</label>
    </td></tr>
    </table>
    </td>
</tr>

<tr><th>Energy unit:</th>
    <td>
    <input type="radio" name="HDiet_eunit" id="HDiet_eunit_cal" value="0"$eunit{0}
        tabindex="17" /> <label for="HDiet_eunit_cal">calorie</label>
    <input type="radio" name="HDiet_eunit" id="HDiet_eunit_kj" value="1"$eunit{1}
        tabindex="18" /> <label for="HDiet_eunit_kj">kilojoule</label>
    </td>
</tr>
◇
```

Macro referenced in 119.

8.8.9 Decimal character selection

Radio buttons are used to select whether a period or comma is used as the decimal character in numbers.

⟨ Decimal character selection 126a ⟩ ≡

```
<tr><th>Decimal character:</th>
    <td>
    <input type="radio" name="HDiet_dchar" id="HDiet_dchar_period" value="."$dchar{'.'}
        tabindex="19" /> <label for="HDiet_dchar_period">123.4</label>
    <input type="radio" name="HDiet_dchar" id="HDiet_dchar_comma" value=","$dchar{','}
        tabindex="20" /> <label for="HDiet_dchar_comma">123,4</label>
    </td>
</tr>
◇
```

Macro referenced in 119.

8.8.10 Public name settings

Radio buttons are used to specify the weight and energy units the user prefers. These are preset based on the setting in the parent object.

⟨ Public name settings 126b ⟩ ≡

```
        print $fh <<"EOD";
<tr><th>Public name:</th>
    <td>
EOD
        if ($self->{public}) {
            my $pub_name = quoteHTML($self->{public_name});
            print $fh <<"EOD";
<input type="checkbox" name="HDiet_public" checked="checked" tabindex="21" />
<b>Pseudonym:</b> $pub_name    
<input type="checkbox" name="HDiet_pubnew" id="HDiet_pubnew"
    tabindex="22" /> <label for="HDiet_pubnew">Assign new pseudonym?</label>
EOD
        } else {
            print $fh <<"EOD";
<input type="checkbox" name="HDiet_public" id="HDiet_public"
tabindex="21" /> <label for="HDiet_public">Check to make your
    logs visible to the public under a pseudonym.</label>
EOD
        }

        print $fh <<"EOD";
    </td>
</tr>
EOD
◇
```

Macro referenced in 119.

8.9 resetPassword

The `resetPassword` method generates a random password of the length given by the first argument with `generatePassword` and places it in the `password` field of the user object. The new password is returned to the caller. Note that this method does not alter the expiration date of the password; if you wish to do that, you'll have to change the `password_expires` field yourself.

"HDiet/user.pm" 127a ≡

```
sub resetPassword {
    my $self = shift;
    my ($nchars) = @_;

    my $npw = $self->generatePassword($nchars);
    $self->{password} = $npw;

    return $npw;
}
```

◇

File defined by 110, 112, 113, 114, 115, 116a, 117, 118, 119, 127ab, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138ab, 139.

8.10 generatePassword

The `generatePassword` method generates a random password of the length given by the first argument and returns it to the caller. You can optionally pass a second string argument which specifies the set of characters of which the password will be composed.

"HDiet/user.pm" 127b ≡

```
sub generatePassword {
    my $self = shift;
    my ($nchars, $pwchars) = @_;

    $pwchars = ("ABCDEFGHIJKLMNPQRSTUVWXYZ" .
                "abcdefghjkmnopqrstuvwxyz" .
                "23456789" .
                "-.") if !$pwchars;

    my $npw = '';
    for (my $i = 0; $i < $nchars; $i++) {
        $npw .= substr($pwchars, int(rand(length($pwchars))), 1);
    }
    return $npw;
}
```

◇

File defined by 110, 112, 113, 114, 115, 116a, 117, 118, 119, 127ab, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138ab, 139.

8.11 SendMail

The `sendMail` method sends an E-mail message to the designated address of the user with the subject given by the first argument and the message body passed as the second. It's up to the caller to ensure that the message body doesn't contain any lines consisting only of a single period, which would cause message truncation. (We could quote them here, but since this method is used to send standard messages over which we have formatting control, there's no reason to bother.)

Mail is sent using the system's `Sendmail` program, setting the "From" address to that configured (usually a no-reply bit bucket), which may be overridden by the optional third argument). Note that if you set the "From" address to one different than the user under which the program is executing, the mail sent will contain an "`X-Authentication-Warning`" header unless the original user account is listed among the "`trusted-users`" configured for Sendmail.

"HDiet/user.pm" 128 ≡

```
        sub sendMail {
            my $self = shift;
            my ($subject, $message, $from) = @_;

            $from = "⟨ From address for mail sent to users 10b ⟩" if !defined($from);

            open(MAIL, "|-:utf8", "⟨ Path to Invoke Sendmail 10a ⟩",
                    "-f$from",
                    $self->{e_mail}) ||
                die("Cannot create pipe to ⟨ Path to Invoke Sendmail 10a ⟩");
            print MAIL <<"EOD";
    From $from\r
    To: $self->{e_mail}\r
    Subject: [The Hacker's Diet Online] $subject\r
    Content-type: text/plain; charset=utf-8\r
    \r
    $message
    .\r
    EOD
            close(MAIL);
        }
    ◇
```

File defined by 110, 112, 113, 114, 115, 116a, 117, 118, 119, 127ab, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138ab, 139.

8.12 Export user information as XML

The `exportPreferencesXML` method writes the `user` XML element which specifies user identity information.

"HDiet/user.pm" 129 ≡

```
        sub exportUserInformationXML {
            my $self = shift;
            my ($fh) = @_;

            my $li = quoteXML($self->{login_name}, 1);
            my $fn = quoteXML($self->{first_name}, 1);
            my $mn = quoteXML($self->{middle_name}, 1);
            my $ln = quoteXML($self->{last_name}, 1);
            my $em = quoteXML($self->{e_mail}, 1);
            my $ac = timeXML($self->{account_created});

            print $fh <<"EOD";
            <user version="1.0">
                <login-name>$li</login-name>
                <first-name>$fn</first-name>
                <middle-name>$mn</middle-name>
                <last-name>$ln</last-name>
                <e-mail>$em</e-mail>
                <height>$self->{height}</height>
                <account-created>$ac</account-created>
            </user>
    EOD
        }
    ◇
```

File defined by 110, 112, 113, 114, 115, 116a, 117, 118, 119, 127ab, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138ab, 139.

8.13 Export preferences as XML

The `exportPreferencesXML` method writes the `preferences` XML element defining user preferences to the file handle passed as the argument.

"HDiet/user.pm" 130 ≡

```
        sub exportPreferencesXML {
            my $self = shift;
            my ($fh) = @_;

            my $lu = HDiet::monthlog::WEIGHT_UNITS->[$self->{log_unit}];
            my $du = HDiet::monthlog::WEIGHT_UNITS->[$self->{display_unit}];
            my $eu = HDiet::monthlog::ENERGY_UNITS->[$self->{energy_unit}];
            my $cr = $self->{current_rung};
            my $dc = quoteXML($self->{decimal_character});

            print $fh <<"EOD";
            <preferences version="1.0">
                <log-unit>$lu</log-unit>
                <display-unit>$du</display-unit>
                <energy-unit>$eu</energy-unit>
                <current-rung>$cr</current-rung>
                <decimal-character>$dc</decimal-character>
            </preferences>
    EOD
        }
    ◇
```

File defined by 110, 112, 113, 114, 115, 116a, 117, 118, 119, 127ab, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138ab, 139.

8.14 Export diet plan as XML

The `exportDietPlanXML` method writes the `diet-plan` XML element which contains the current diet plan from the diet calculator.

"HDiet/user.pm" 131 ≡

```
        sub exportDietPlanXML {
            my $self = shift;
            my ($fh) = @_;

            my $ac = timeXML($self->{calc_start_date});

            print $fh <<"EOD";
            <diet-plan version="1.0">
                <calorie-balance>$self->{calc_calorie_balance}</calorie-balance>
                <start-weight>$self->{calc_start_weight}</start-weight>
                <goal-weight>$self->{calc_goal_weight}</goal-weight>
                <start-date>$ac</start-date>
                <show-plan>$self->{plot_diet_plan}</show-plan>
            </diet-plan>
    EOD
        }
    ◇
```

File defined by 110, 112, 113, 114, 115, 116a, 117, 118, 119, 127ab, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138ab, 139.

8.15 enumerateMonths

The `enumerateMonths` method returns a list, sorted in chronological order, of all months in the database (in the form "*YYYY-MM*") if called with no argument, or just for the given year if called with the desired year as the argument.

"HDiet/user.pm" 132 ≡

```
sub enumerateMonths {
    my $self = shift;
    my ($year) = @_;

    my $user_file_name = quoteUserName($self->{login_name});
    my $selpat = $year ? $year : '\d+';

    opendir(CD, "⟨ Users Directory 6h ⟩/$user_file_name") ||
        die("Cannot open directory ⟨ Users Directory 6h ⟩/$user_file_name");
    my @logs;
    my $f;
    foreach $f (sort(grep(/^$selpat\-\d\d\.hdb/, readdir(CD)))) {
        $f =~ s/\.\w*$//;
        push(@logs, $f);
    }
    closedir(CD);

    return @logs;
}
```

◇

File defined by 110, 112, 113, 114, 115, 116a, 117, 118, 119, 127ab, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138ab, 139.

8.16 enumerateYears

The `enumerateYears` method returns a list, sorted in chronological order, of all years with one or more month entries in the database. The years are returned as numbers.

"HDiet/user.pm" 133 ≡

```
sub enumerateYears {
    my $self = shift;

    my $user_file_name = quoteUserName($self->{login_name});
    my $lyear = '';
    my @years;

    opendir(CD, "⟨ Users Directory 6h ⟩/$user_file_name") ||
        die("Cannot open directory ⟨ Users Directory 6h ⟩/$user_file_name");
    my $m;
    foreach $m (sort(grep(/^\d+\-\d\d\.hdb/, readdir(CD)))) {
        $m =~ m/^(\d+)\-/;
        if ($1 ne $lyear) {
            $lyear = $1;
            push(@years, $lyear);
        }
    }
    closedir(CD);

    return @years;
}
```

◇

File defined by 110, 112, 113, 114, 115, 116a, 117, 118, 119, 127ab, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138ab, 139.

8.17 dietPlanLimits

The `dietPlanLimits` method returns a list giving the start Julian day, start weight, end Julian day, and goal weight of the current diet plan. If no diet plan is defined, or the diet plan makes no sense (energy balance is opposite from the goal versus start weight) `undef` is returned.

"HDiet/user.pm" 134 ≡

```
        sub dietPlanLimits {
            my $self = shift;

            if (($self->{calc_start_weight} == 0) ||
                ($self->{calc_goal_weight} == 0) ||
                ($self->{calc_start_date} == 0) ||
                (::sgn($self->{calc_calorie_balance}) != ::sgn($self->{calc_goal_weight} - $self->{calc_start_weig
                return undef;
            }
            my $jdstart = unix_time_to_jd($self->{calc_start_date});
            my $jdend = $jdstart + (($self->{calc_goal_weight} - $self->{calc_start_weight}) /
                ($self->{calc_calorie_balance} /
                    HDiet::monthlog::CALORIES_PER_WEIGHT_UNIT->[HDiet::monthlog::WEIGHT_KILOGRAM]));
            return ($jdstart, $self->{calc_start_weight}, $jdend, $self->{calc_goal_weight});
        }
```

◇

File defined by 110, 112, 113, 114, 115, 116a, 117, 118, 119, 127ab, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138ab, 139.

8.18 generateEncryptedUserID

The `generateEncryptedUserID` method generates an opaque string encrypted with the master encryption key which we (but not others who lack the key) can decode to obtain the user file name. The encrypted user ID includes random data and internal consistency checks to thwart attempts to reverse engineer or probe for the key.

"HDiet/user.pm" 135 ≡

```
sub generateEncryptedUserID {
    my $self = shift;

    my $plain = '';
    for (my $i = 0; $i < 13; $i++) {
        $plain .= chr(int(rand(95)) + 32);
    }
    $plain .= quoteUserName($self->{login_name});
    for (my $i = 0; $i < 11; $i++) {
        $plain .= chr(int(rand(95)) + 32);
    }
    my $crc = new HDiet::Digest::Crc32();
    $plain .= sprintf("%08x", $crc->strcrc32($plain));

    my $crypto = Crypt::CBC->new(
            -key => ⟨ Master encryption key 4d ⟩,
            -cipher => "Crypt::OpenSSL::AES"
                                );
    my $encrypted = $crypto->encrypt($plain);
    my $ecrc = sprintf("%08x", $crc->strcrc32($encrypted));
    my $huid = unpack("H*", $encrypted);
    my $euid = substr($huid, 0, 17) . $ecrc . substr($huid, 17);
    $euid =~ tr/a-f/FGJKQW/;
    return $euid;
}
```

◇

File defined by 110, 112, 113, 114, 115, 116a, 117, 118, 119, 127ab, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138ab, 139.

8.19 decodeEncryptedUserID

The `decodeEncryptedUserID` function extracts the user file name from an opaque user identity string encoded with `generateEncryptedUserID` above. It validates both the external and internal CRCs and returns the user name if all validation tests pass; otherwise `undef` is returned. Note that this function is *not* a part of the `user` module—it is a macro which is included in the `HackDietBadge` lightweight program which retrieves and returns the current badge for a user. We define the macro here because it's easier to read the function in conjunction with `generateEncryptedUserID` above.

⟨ Decode encrypted user ID 136 ⟩ ≡

```
sub decodeEncryptedUserID {
    my ($crypt) = @_;

    $crypt =~ tr/FGJKQW/a-f/;
    my $cryptoSig = substr($crypt, 17, 8, "");
    $crypt = pack("H*", $crypt);

    my $crc = new HDiet::Digest::Crc32();
    my $outerSig = sprintf("%08x", $crc->strcrc32($crypt));

    if ($cryptoSig ne $outerSig) {
print(STDERR "user::decodeEncryptedUserID: Outer CRC bad: $cryptoSig $outerSig\n");
        return undef;
    }

    my $crypto = Crypt::CBC->new(
            -key => ⟨ Master encryption key 4d ⟩,
            -cipher => "Crypt::OpenSSL::AES"
                                );

    my $decrypted = $crypto->decrypt($crypt);

    my $rcrc = substr($decrypted, -8, 8, "");
    my $icrc = sprintf("%08x", $crc->strcrc32($decrypted));

    if ($rcrc ne $icrc) {
print(STDERR "user::decodeEncryptedUserID: Inner CRC bad:  RCRC = $rcrc  ICRC = $icrc\n");
        return undef;
    }

    return substr($decrypted, 13, -11);
 }
```

◇

Macro referenced in 431, 433.

8.20 QuoteUserName

The `quoteUserName` function takes an arbitrary UTF-8 string argument and returns an ASCII string suitable for use as a Unix file name. ASCII alphanumerics and underscores are left unchanged, spaces are replaced with plus signs, and all other characters are replaced with their hexadecimal codes enclosed in braces. If the quoted user name exceeds the system's maximum file name length, it is truncated to 40 characters less than that length and the 40 character hexadecimal SHA1 digest of the entire name is appended.

"HDiet/user.pm" 137 ≡

```
sub quoteUserName {
    my ($s) = @_;

    my $os = '';

    while ($s =~ s/^(.)//) {
        my $c = $1;

        if ((ord($c) < 256) && ($c =~ m/⟨ Characters Permissible in File Names 9b ⟩/)) {
            $os .= $c;
        } elsif ($c eq ' ') {
            $os .= '⟨ Encoding for Space in File Name Characters 9c ⟩';
        } else {
            $os .= sprintf('⟨ Left Delimiter for Quoted File Name Characters 9d ⟩' .
                           '%X' .
                           '⟨ Right Delimiter for Quoted File Name Characters 9e ⟩', ord($c));
        }
    }

    if (length($os) > ⟨ Maximum File Length 7e ⟩) {
        $os = substr($os, 0, ⟨ Maximum File Length 7e ⟩ - 40) .
                sha1_hex(encode_utf8($os));
    }

    return $os;
}
```

◇

File defined by 110, 112, 113, 114, 115, 116a, 117, 118, 119, 127ab, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138ab, 139.

8.21 Express number in canonical form

The `canonicalNumber` function converts a number to canonical form by rounding the number passed as the first argument to the number of decimal places given by the second, then eliding any trailing zeroes and the decimal point if the number is an integer.

"HDiet/user.pm" 138a ≡

```
sub canonicalNumber {
    my ($value, $places) = @_;

    $value = sprintf("%.${places}f", $value);

    $value =~ s/(\.[^0]*)0+$/$1/;
    $value =~ s/\.$//;

    return $value;
}
```

◇

File defined by 110, 112, 113, 114, 115, 116a, 117, 118, 119, 127ab, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138ab, 139.

8.22 Convert decimal number to localised form

The `localiseDecimal`function converts its argument, assumed to be a number with decimal places, to use the user's specified `decimal_character`.

"HDiet/user.pm" 138b ≡

```
sub localiseDecimal {
    my $self = shift;
    my ($value) = @_;

    $value =~ s/\./$self->{decimal_character}/;

    return $value;
}
```

◇

File defined by 110, 112, 113, 114, 115, 116a, 117, 118, 119, 127ab, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138ab, 139.

8.23 Express number in localised form

The `localiseNumber` method converts a number to canonical form by rounding the number passed as the first argument to the number of decimal places given by the second, then eliding any trailing zeroes and the decimal point if the number is an integer. If a decimal point remains in the value, the user's specified `decimal_character` separates the unit and decimal digits.

"HDiet/user.pm" 139 ≡

```
        sub localiseNumber {
            my $self = shift;
            my ($value, $places) = @_;

            return $self->localiseDecimal(canonicalNumber($value, $places));
        }
```

◇

File defined by 110, 112, 113, 114, 115, 116a, 117, 118, 119, 127ab, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138ab, 139.

Chapter 9

session.pm: Session Object

The session object represents an open session.

9.1 Package plumbing

"HDiet/session.pm" 140 ≡

```
#! ⟨ Perl directory 7d ⟩

    ⟨ Perl language modes 369b ⟩


    package HDiet::session;

    use Encode qw(encode_utf8);
    use Digest::SHA1  qw(sha1_hex);

    require Exporter;
    our @ISA = qw(Exporter);
    our @EXPORT = qw( load_active_session );
    1;

    use constant FILE_VERSION => 1;

◇
```

File defined by 140, 141, 142, 143, 144ac, 145ab.

9.2 Constructor

A new `session` object is created by calling the `new` constructor. The constructor has two optional arguments: the user login name and the UNIX date and time of the login. If omitted, the user name is set to blank and the login time to the current time.

login_name	User login name
session_id	Session identifier
login_time	Date and time of login
effective_name	Effective login name for administrator access
browse_name	Browse name for user accessing public accounts
read_only	Is session read-only ?
handheld	Is session on handheld device ?
cookie	Was session login via a cookie ?

"HDiet/session.pm" 141 ≡

```
sub new {
    my $self = {};
    my ($invocant, $login_name, $login_time) = @_;
    my $class = ref($invocant) || $invocant;

    $login_name = '' if !defined($login_name);
    $login_time = time() if !defined($login_time);

    bless($self, $class);

    $self->{version} = FILE_VERSION;

    #   Initialise instance variables
    $self->{login_name} = $login_name;
    if ($login_name ne '') {
        $self->{session_id} = generateSessionID($login_name);
    } else {
        $self->{session_id} = '';
    }
    $self->{login_time} = $login_time;

    $self->{effective_name} = $self->{browse_name} = '';
    $self->{read_only} = 0;
    $self->{handheld} = 0;
    $self->{cookie} = 0;

    return $self;
}
```

◇

File defined by 140, 141, 142, 143, 144ac, 145ab.

9.3 Describe

The `describe` method prints a primate-readable description of the session on the file handle (default `STDOUT`) given by the argument.

"HDiet/session.pm" 142 ≡

```
sub describe {
    my $self = shift;
    my ($outfile) = @_;

    if (!(defined $outfile)) {
        $outfile = \*STDOUT;
    }

    print($outfile "SESSION Version: $self->{version}\n");
    print($outfile "  User name:      '$self->{login_name}'\n");
    print($outfile "  Session ID:     '$self->{session_id}'\n");
    print($outfile "  Login time:      " . localtime($self->{login_time}) . "\n");
    print($outfile "  Effective name: '$self->{effective_name}'\n");
    print($outfile "  Browse name:    '$self->{browse_name}'\n");
    print($outfile "  Read only:      '$self->{read_only}'\n");
    print($outfile "  Handheld:       '$self->{handheld}'\n");
    print($outfile "  Cookie login:   '$self->{cookie}'\n");

}
```

◇

File defined by 140, 141, 142, 143, 144ac, 145ab.

9.4 Save

The `save` method writes the session item to the already-open file handle passed as the argument. Note that it's up to you to update the `last_access` time (if appropriate) to before saving the session.

"HDiet/session.pm" 143 ≡

```
sub save {
    my $self = shift;
    my ($outfile) = @_;

    #   File format version number
    print($outfile "$self->{version}\n");
    #   Login name
    print($outfile "$self->{login_name}\n");
    #   Session ID
    print($outfile "$self->{session_id}\n");
    #   Login time
    print($outfile "$self->{login_time}\n");
    #   Effective name
    print($outfile "$self->{effective_name}\n");
    #   Browse name
    print($outfile "$self->{browse_name}\n");
    #   Read only
    print($outfile "$self->{read_only}\n");
    #   Handheld device
    print($outfile "$self->{handheld}\n");
    #   Cookie login
    print($outfile "$self->{cookie}\n");
}
```

◇

File defined by 140, 141, 142, 143, 144ac, 145ab.

9.5 Load

The `load` method reads a session file from the argument file handle in the format produced by `save`.

"HDiet/session.pm" 144a ≡

```
sub load {
    my $self = shift;
    my ($infile) = @_;

    my $s = in($infile);

    if ($s != FILE_VERSION) {
        die("session::load: Incompatible file version $s");
    }

    $self->{login_name} = in($infile);
    $self->{session_id} = in($infile);
    $self->{login_time} = in($infile);
    $self->{effective_name} = in($infile);
    $self->{browse_name} = in($infile);
    $self->{read_only} = in($infile);
    $self->{handheld} = in($infile, 0);
    $self->{cookie} = in($infile, 0);
}

⟨ Read line from persistent object file (144b session ) 390b ⟩
```

◇

File defined by 140, 141, 142, 143, 144ac, 145ab.

9.6 Save Active Session

The `save_active_session` method writes the active session file, stored in the user directory, which provides a link to the currently open session for the user. The file simply contains a version number (same as the session file) and the session ID.

"HDiet/session.pm" 144c ≡

```
sub save_active_session {
    my $self = shift;
    my ($outfile) = @_;

    #   File format version number
    print($outfile "$self->{version}\n");
    #   Session ID
    print($outfile "$self->{session_id}\n");
}
```

◇

File defined by 140, 141, 142, 143, 144ac, 145ab.

9.7 Load Active Session

The `load_active_session` method reads an active session file from the argument file handle in the format produced by `save_active_session`. The session ID is returned, or the null string in case of error.

"HDiet/session.pm" 145a ≡

```
sub load_active_session {
    my ($infile) = @_;

    my $s = in($infile);

    if ($s != FILE_VERSION) {
        die("session::load_active_session: Incompatible file version $s");
        return '';
    }

    return in($infile);
}
```

◇

File defined by 140, 141, 142, 143, 144ac, 145ab.

9.8 GenerateSessionID

The `generateSessionID` function generates a pseudorandom session ID by hashing the login name and a pseudorandom sequence into a SHA1 hexadecimal signature. The session name is a unique "handle" used to identify the session in documents sent back and forth to the user as the session progresses.

"HDiet/session.pm" 145b ≡

```
sub generateSessionID {
    my ($login) = @_;

    $login = encode_utf8($login);
    for (my $i = 0; $i < 16; $i++) {
        $login .= chr(int(rand(256)));
    }
    my $si = sha1_hex($login);
    $si =~ tr/a-f/FGJKQW/;
    return $si;
}
```

◇

File defined by 140, 141, 142, 143, 144ac, 145ab.

Chapter 10

cookie.pm: Cookie Object

The cookie object represents a "cookie"—a token stored in the user's browser which allows automatic login to a designated account without a password.

10.1 Package plumbing

"HDiet/cookie.pm" 146 ≡

```
#! ⟨ Perl directory 7d ⟩

    ⟨ Perl language modes 369b ⟩

    package HDiet::cookie;

    use Encode qw(encode_utf8);
    use Digest::SHA1  qw(sha1_hex);

    use HDiet::Cluster;
    use HDiet::Julian;
    use HDiet::Digest::Crc32;

    require Exporter;
    our @ISA = qw(Exporter);
    our @EXPORT = qw( checkCookieSignature storeCookie testCookiePresent );
    1;

    use constant FILE_VERSION => 1;

◇
```

File defined by 146, 147, 148ab, 149ac, 150ab, 151, 152, 153ab.

10.2 Constructor

A new `cookie` object is created by calling the `new` constructor. The constructor has three optional arguments: the user login name, the UNIX date and time of the login, and the date and time the cookie will expire. If omitted, the user name is set to blank, the login time is set to the current time, and the expiration is set to that time plus the default cookie retention time.

login_name	User login name
login_time	Date and time of login (cookie creation)
expiry_time	Date and time cookie expires
cookie_id	Cookie ID

"HDiet/cookie.pm" 147 ≡

```
sub new {
    my $self = {};
    my ($invocant, $login_name, $login_time, $expiry_time) = @_;
    my $class = ref($invocant) || $invocant;

    $login_name = '' if !defined($login_name);
    $login_time = time() if !defined($login_time);
    $expiry_time = $login_time + ⟨ Default cookie retention time 11b ⟩ if !defined($expiry_time);

    bless($self, $class);

    $self->{version} = FILE_VERSION;

    #   Initialise instance variables
    $self->{login_name} = $login_name;
    if ($login_name ne '') {
        $self->{cookie_id} = generateCookieID($login_name);
    } else {
        $self->{cookie_id} = '';
    }
    $self->{login_time} = $login_time;
    $self->{expiry_time} = $expiry_time;

    return $self;
}
```

◇

File defined by 146, 147, 148ab, 149ac, 150ab, 151, 152, 153ab.

10.3 Describe

The `describe` method prints a primate-readable description of the cookie on the file handle (default `STDOUT`) given by the argument.

"HDiet/cookie.pm" 148a ≡

```
sub describe {
    my $self = shift;
    my ($outfile) = @_;

    if (!(defined $outfile)) {
        $outfile = \*STDOUT;
    }

    print($outfile "COOKIE Version: $self->{version}\n");
    print($outfile "  User name:      '$self->{login_name}'\n");
    print($outfile "  Cookie  ID:     '$self->{cookie_id}'\n");
    print($outfile "  Creation time:   " . localtime($self->{login_time}) . "\n");
    print($outfile "  Expiration time: " . localtime($self->{expiry_time}) . "\n");

}
```

◇

File defined by 146, 147, 148ab, 149ac, 150ab, 151, 152, 153ab.

10.4 Save

The `save` method writes the cookie item to the already-open file handle passed as the argument.

"HDiet/cookie.pm" 148b ≡

```
sub save {
    my $self = shift;
    my ($outfile) = @_;

    #   File format version number
    print($outfile "$self->{version}\n");
    #   Login name
    print($outfile "$self->{login_name}\n");
    #   Cookie ID
    print($outfile "$self->{cookie_id}\n");
    #   Creation time
    print($outfile "$self->{login_time}\n");
    #   Expiration time
    print($outfile "$self->{expiry_time}\n");
}
```

◇

File defined by 146, 147, 148ab, 149ac, 150ab, 151, 152, 153ab.

10.5 Load

The `load` method reads a cookie file from the argument file handle in the format produced by `save`.

"HDiet/cookie.pm" 149a ≡

```
sub load {
    my $self = shift;
    my ($infile) = @_;

    my $s = in($infile);

    if ($s != FILE_VERSION) {
        die("cookie::load: Incompatible file version $s");
    }

    $self->{login_name} = in($infile);
    $self->{cookie_id} = in($infile);
    $self->{login_time} = in($infile);
    $self->{expiry_time} = in($infile);
}
```

⟨ Read line from persistent object file (149b `cookie`) 390b ⟩

◇

File defined by 146, 147, 148ab, 149ac, 150ab, 151, 152, 153ab.

10.6 signCookie

The `signCookie` method returns the cookie ID with a salted CRC-32 signature appended, encoded as the hexadecimal cookie ID is.

"HDiet/cookie.pm" 149c ≡

```
sub signCookie {
    my $self = shift;

    my $crc = new HDiet::Digest::Crc32();
    my $cookSig = sprintf("%08x", $crc->strcrc32(⟨ Confirmation signature encoding suffix 4c ⟩ .
        $self->{cookie_id}));
    $cookSig =~ tr/a-f/FGJKQW/;

    return substr($self->{cookie_id}, 0, 23) . $cookSig . substr($self->{cookie_id}, 23);
}
```

◇

File defined by 146, 147, 148ab, 149ac, 150ab, 151, 152, 153ab.

10.7 generateCookie

The `generateCookie` method returns the HTTP cookie definition as used in the "`Set-Cookie`" header item.

"HDiet/cookie.pm" 150a ≡

```
        sub generateCookie {
            my $self = shift;
            my ($name) = @_;

            return "$name=" . $self->signCookie() . "; " .
                   "Domain=⟨ Domain for cookies 11c ⟩; " .
                   "Path=⟨ Path for cookies 11d ⟩; " .
                   "Expires=" .
                    jd_to_old_cookie_date(unix_time_to_jd($self->{expiry_time}));
        }
```

◇

File defined by 146, 147, 148ab, 149ac, 150ab, 151, 152, 153ab.

10.8 expireCookie

The `expireCookie` method returns an HTTP cookie definition which marks the cookie as expired.

"HDiet/cookie.pm" 150b ≡

```
        sub expireCookie {
            my $self = shift;
            my ($name) = @_;

            return "$name=EXPIRED; " .
                   "Domain=⟨ Domain for cookies 11c ⟩; " .
                   "Path=⟨ Path for cookies 11d ⟩; " .
                   "Expires=" .
                    jd_to_old_cookie_date(gregorian_to_jd(1990, 1, 1));
        }
```

◇

File defined by 146, 147, 148ab, 149ac, 150ab, 151, 152, 153ab.

10.9 storeCookie

The `storeCookie` function is called with the `user` object for which the cookie is being created. It creates a cookie with the default expiration time, saves it in the "Remember me" directory, and returns the body of the cookie to be included in the HTTP header.

"HDiet/cookie.pm" 151 ≡

```
sub storeCookie {
    my ($ui) = @_;

    my $cook = HDiet::cookie->new($ui->{login_name}, time());
     open(CO, ">:utf8", "⟨ Remember Me Directory 11e ⟩/$cook->{cookie_id}.hdr") ||
        die("Cannot create persistent login file ⟨ Remember Me Directory 11e ⟩/$cook->{cookie_id}.hdr");
    $cook->save(\*CO);
    close(CO);
    clusterCopy("⟨ Remember Me Directory 11e ⟩/$cook->{cookie_id}.hdr");

    return $cook->generateCookie(⟨ Cookie name 11a ⟩);
}
```

◇

File defined by 146, 147, 148ab, 149ac, 150ab, 151, 152, 153ab.

10.10 testCookiePresent

The `testCookiePresent` function checks the environment for the presence of a cookie with the name given by the argument. If present (and not expired), the signature is checked and, if valid, the cookie is looked up in the "`RememberMe`" directory. If the cookie is found in the directory, it is loaded into memory and the expiration time checked. If the cookie has not expired, it is deleted from the directory and and the user name returned to proceed with the login. If the cookie is absent, invalid, or expired, `undef` is returned.

"HDiet/cookie.pm" 152 ≡

```
sub testCookiePresent {
    my ($name) = @_;

    my $cuser;

    if (defined($ENV{HTTP_COOKIE}) &&
        ($ENV{HTTP_COOKIE} =~ m/$name=([0-9FGJKQW]{48})/)) {
        my $csig = $1;
        my $cid = checkCookieSignature($csig);
        if (defined($cid)) {
            if (-f "⟨ Remember Me Directory 11e ⟩/$cid.hdr") {
                if (open(CI, "<:utf8", "⟨ Remember Me Directory 11e ⟩/$cid.hdr")) {
                    my $cook = HDiet::cookie->new();
                    $cook->load(\*CI);
                    close(CI);
                    if (($cook->{cookie_id} eq $cid) &&
                        ($cook->{expiry_time} >= time())) {
                        $cuser = $cook->{login_name};
                    }
                }
                unlink("⟨ Remember Me Directory 11e ⟩/$cid.hdr");
                clusterDelete("⟨ Remember Me Directory 11e ⟩/$cid.hdr");
            }
        }
    }
    return $cuser;
}
```

◇

File defined by 146, 147, 148ab, 149ac, 150ab, 151, 152, 153ab.

10.11 checkCookieSignature

The `checkCookieSignature` function is called with the opaque signed cookie received from a client. We validate the signature and, if valid, return the cookie ID. If the syntax of the signed cookie is invalid or the signature fails to validate, `undef` is returned.

"HDiet/cookie.pm" 153a ≡

```
        sub checkCookieSignature {
            my ($signedCookie) = @_;

            if ($signedCookie !~ m/^[0-9FGJKQW]{48}$/) {
    #print("Cookie syntax bad signedCookie ($signedCookie)\n");
                return undef;
            }

            my $crc = new HDiet::Digest::Crc32();

            my $cookieSig = substr($signedCookie, 23, 8, "");
            $cookieSig =~ tr/FGJKQW/a-f/;
            my $cookSig = sprintf("%08x", $crc->strcrc32(⟨ Confirmation signature encoding suffix 4c ⟩ .
                $signedCookie));
    #print("cookSig ($cookSig)  cookieSig ($cookieSig)  signedCookie ($signedCookie)\n");
            if ($cookSig eq $cookieSig) {
                return $signedCookie;
            }
            return undef;
        }
    ◇
```

File defined by 146, 147, 148ab, 149ac, 150ab, 151, 152, 153ab.

10.12 generateCookieID

The `generateCookieID` function generates a pseudorandom cookie ID by hashing the login name and a pseudorandom sequence into a SHA1 hexadecimal signature.

"HDiet/cookie.pm" 153b ≡

```
        sub generateCookieID {
            my ($login) = @_;

            $login = encode_utf8($login);
            for (my $i = 0; $i < 16; $i++) {
                $login .= chr(int(rand(256)));
            }
            my $si = sha1_hex($login);
            $si =~ tr/a-f/FGJKQW/;
            return $si;
        }
    ◇
```

File defined by 146, 147, 148ab, 149ac, 150ab, 151, 152, 153ab.

Chapter 11

pubname.pm: Public Name Manager Object

The pubname object provides utilities for managing randomly assigned public names for users who wish to disclose their logs without revealing their actual identity.

11.1 Package plumbing

"HDiet/pubname.pm" 154 ≡

```
#! ⟨ Perl directory 7d ⟩

    ⟨ Perl language modes 369b ⟩

    package HDiet::pubname;

    use Fcntl;

    use HDiet::Cluster;

    require Exporter;
    our @ISA = qw(Exporter);
    our @EXPORT = ( );

    my @nameSources = ( 'firstnames.txt', 'lastnames.txt' );
    #   Names per nameSources entry.  Zero means use seek technique
    my @nameLength = ( 24, 0 );

    1;

    use constant FILE_VERSION => 1;
```

◇

File defined by 154, 155, 156, 157, 158, 159ab, 160b, 161ab.

11.2 Constructor

A new `pubname` object is created by calling the `new` constructor.

"HDiet/pubname.pm" 155 ≡

```
sub new {
    my $self = {};
    my ($invocant) = @_;
    my $class = ref($invocant) || $invocant;

    bless($self, $class);

    $self->{version} = FILE_VERSION;
    $self->{public_name} = $self->{true_name} = '';
    $self->{true_create_time} = $self->{public_create_time} = 0;

    return $self;
}
```

◇

File defined by 154, 155, 156, 157, 158, 159ab, 160b, 161ab.

11.3 Generate random name

The `generateRandomName` method returns a randomly chosen name. There is no guarantee that the name is not already in use; see `generateUniqueName` for a function which verifies the name unique at the time of its creation.

"HDiet/pubname.pm" 156 ≡

```
sub generateRandomName {
    my $self = shift;

    my $name = '';

    for (my $i = 0; $i <= $#nameSources; $i++) {
        my $filename = "⟨Public Name Directory 6i⟩/$nameSources[$i]";
        open(NF, "<:utf8", $filename) ||
            die("pubname::generateUniqueName:  Cannot open $filename");
        my $s;
        if ($nameLength[$i] > 0) {
            my $line = int(rand() * $nameLength[$i]) + 1;
            while ($line-- > 0) {
                $s = <NF> ||
                    die("pubname::generateUniqueName:  Unexpected EOF on $filename");
            }
        } else {
            while (1) {
                seek(NF, int(rand() * (-s $filename)) - 64, 0);

                $s = <NF>;          # Burn characters to align with next line
                my $n = int(rand() * 7) + 1;
                while ($n-- > 0) {
                    if (eof(NF)) {
                        next;
                    }
                    $s = <NF>;
                }
                last;
            }

        }
        close(NF);

        $s =~ s/\s+$//;
        $name .= $s . ' ';
    }

    $name =~ s/\s$//;
    return $name;
}
```

◇

File defined by 154, 155, 156, 157, 158, 159ab, 160b, 161ab.

11.4 Generate unique name

The `generateUniqueName` method returns a randomly chosen name which is guaranteed, at the time it is returned, not to be used as a public name. Unless the caller has set some kind of lock, however, there is no assurance some other process may not grab it before you actually create such a name.

"HDiet/pubname.pm" 157 ≡

```
sub generateUniqueName {
    my $self = shift;

    my $name;
    while (1) {
        $name = $self->generateRandomName();
        my $ufn = HDiet::user::quoteUserName($name);
        if (!(-f "⟨ Public Name Directory 6i ⟩/$ufn.hdp")) {
            last;
        }
    }

    return $name;
}
```

◇

File defined by 154, 155, 156, 157, 158, 159ab, 160b, 161ab.

11.5 Assign public name

A public name is assigned to the user object passed as the argument. If the user already has a public name assigned, it is discarded and replaced with the new one. If no public name was previously assigned, the date the user went public is set to the present time; if this is an assignment of a new name to a user who is already public, the `public_since` time is not changed.

The public name fields in the user object are updated, but it is the responsibility of the caller to write the user object back to the user database.

The guarantee of uniqueness of the public name depends upon the correct operation of the "`O_EXCL`" flag in the `sysopen` function. If this does not always fail for a pre-existing file (as might be the case on cached cluster systems or tacky network file storage architectures), then it is possible for two users to be assigned the same public name. While this is confusing, it will not damage the data of either of the users.

"HDiet/pubname.pm" 158 ≡

```
sub assignPublicName {
    my $self = shift;

    my ($ui) = @_;

    if ($ui->{public}) {
        ⟨ Delete existing public name 160a ⟩
    }

    my ($name, $pfn);
    while (1) {
        $name = $self->generateRandomName();
        $pfn = HDiet::user::quoteUserName($name);
        if (sysopen(PF, "⟨ Public Name Directory 6i ⟩/$pfn.hdp", O_CREAT | O_EXCL | O_WRONLY)) {
            binmode(PF, ":utf8");
            last;
        }
    }

    if (!($ui->{public})) {
        $ui->{public} = 1;
        $ui->{public_since} = time();
    }

    $self->{public_name} = $ui->{public_name} = $name;
    $self->{true_name} = $ui->{login_name};
    $self->{true_create_time} = $ui->{account_created};
    $self->{public_create_time} = $ui->{public_since};

    $self->save(\*PF);
    close(PF);
    clusterCopy("⟨ Public Name Directory 6i ⟩/$pfn.hdp");

    return $name;
}
```

◇

File defined by 154, 155, 156, 157, 158, 159ab, 160b, 161ab.

11.6 Find public name

The `findPublicName` method is called with the public name requested. If such a name exists, its properties are loaded into the object and the true name of the user is returned to the caller. If no such name exists `undef` is returned.

"HDiet/pubname.pm" 159a ≡

```
sub findPublicName {
    my $self = shift;

    my ($pname) = @_;

    #   Clear out object to avoid confusion in case of no find
    $self->{public_name} = $self->{true_name} = '';
    $self->{true_create_time} = $self->{public_create_time} = 0;

    my $pfn = HDiet::user::quoteUserName($pname);
    if (open(PF, "<:utf8", "⟨ Public Name Directory 6i ⟩/$pfn.hdp")) {
        $self->load(\*PF);
        close(PF);
        return $self->{true_name};
    }

    return undef;
}
```

◇

File defined by 154, 155, 156, 157, 158, 159ab, 160b, 161ab.

11.7 Delete public name

This method is called with the user object whose public name is to be deleted. If the user has no public name, the call is ignored. Otherwise, the public name file is deleted and the public name flag, name string, and date assigned are cleared, rendering the user private once again.

"HDiet/pubname.pm" 159b ≡

```
sub deletePublicName {
    my $self = shift;

    my ($ui) = @_;

    if ($ui->{public}) {
        ⟨ Delete existing public name 160a ⟩
        $ui->{public} = 0;
        $self->{public_name} = $ui->{public_name} = '';
        $self->{public_create_time} = $self->{true_create_time} = $ui->{public_since} = 0;
    }
}
```

◇

File defined by 154, 155, 156, 157, 158, 159ab, 160b, 161ab.

11.7.1 Delete existing public name

The existing public name for a user is deleted. The public name field is set to the null string, but the public name status and time of creation of the public name are not changed.

⟨ Delete existing public name 160a ⟩ ≡

```
my $pfn = HDiet::user::quoteUserName($ui->{public_name});
if (!unlink("⟨ Public Name Directory 6i ⟩/$pfn.hdp")) {
    die("Unable to delete old public name: ⟨ Public Name Directory 6i ⟩/$pfn.hdp");
}
clusterDelete("⟨ Public Name Directory 6i ⟩/$pfn.hdp");
$ui->{public_name} = '';
```

◇

Macro referenced in 158, 159b.

11.8 Describe

The `describe` method prints a primate-readable description of the public name on the file handle (default `STDOUT`) given by the argument.

"HDiet/pubname.pm" 160b ≡

```
sub describe {
    my $self = shift;
    my ($outfile) = @_;

    if (!(defined $outfile)) {
        $outfile = \*STDOUT;
    }

    print($outfile "PUBNAME Version: $self->{version}\n");
    print($outfile "  Public name:  '$self->{public_name}'\n");
    print($outfile "  True name:    '$self->{true_name}'\n");
    print($outfile "  First login:  " . localtime($self->{true_create_time}) . "\n");
    print($outfile "  Public since: " . localtime($self->{public_create_time}) . "\n");
}
```

◇

File defined by 154, 155, 156, 157, 158, 159ab, 160b, 161ab.

11.9 Save

The `save` method writes the public name item to the already-open file handle passed as the argument.

"HDiet/pubname.pm" 161a ≡

```
        sub save {
            my $self = shift;
            my ($outfile) = @_;

            print $outfile <<"EOD";
    $self->{version}
    $self->{public_name}
    $self->{true_name}
    $self->{true_create_time}
    $self->{public_create_time}
    EOD
        }
    ◇
```

File defined by 154, 155, 156, 157, 158, 159ab, 160b, 161ab.

11.10 Load

The `load` method reads a public name file from the argument file handle in the format produced by `save`.

"HDiet/pubname.pm" 161b ≡

```
        sub load {
            my $self = shift;
            my ($infile) = @_;

            my $s = in($infile);

            if ($s != FILE_VERSION) {
                die("user::load: Incompatible file version $s");
            }

            $self->{public_name} = in($infile);
            $self->{true_name} = in($infile);
            $self->{true_create_time} = in($infile);
            $self->{public_create_time} = in($infile);
        }

        ⟨ Read line from persistent object file (161c pubname ) 390b ⟩
    ◇
```

File defined by 154, 155, 156, 157, 158, 159ab, 160b, 161ab.

Chapter 12

`HackDiet.pl`: Main CGI Application

`HackDiet.pl` is the main CGI application program. It is invoked when one of its request forms is submitted by a user. Apart from the modules defined elsewhere in this document, the libraries included in the distribution, and the required Perl library modules, it is entirely self-contained.

12.1 Main program

"HackDiet.pl" 163 ≡

```
#! ⟨ Perl directory 7d ⟩

    ⟨ Documentation in POD format 427 ⟩

    ⟨ Global declarations 369a ⟩

    ⟨ Set handler for termination signals 164a ⟩

    #   Override site address in otherwise relative URLs
    my $homeBase = "⟨ Web Document Home 5a ⟩";

    my $dataBase = "⟨ Database Directory 6f ⟩";

    ⟨ Process command line options 371a ⟩
    ⟨ Validate option specifications 371b ⟩

    if ($#ARGV != -1) {
        &print_command_line_help;
        exit(0);
    }

    binmode(STDIN, ":utf8");

    my $fh = \*STDOUT;

    my %CGIargs = &parse_cgi_arguments;

    my $inHTML = 0;
    my $readOnly = 0;
    my $cookieLogin = 0;
    my $cookieUser;
    our @HTTP_header;

    ⟨ Define requests permissible whilst browsing public account 164b ⟩
    ⟨ Extract brain-dead Internet Exploder request field 166 ⟩
    ⟨ Estimate local time at user site 167 ⟩
    ⟨ Dispatch requests which return non-HTML results 168 ⟩

requeue:
    binmode(STDOUT, ":utf8");

    #   Emit Content-type if we were invoked as a CGI program
    if ($ENV{'REQUEST_METHOD'}) {
        ⟨ MIME Content-type specification 372a ⟩
    }

    $inHTML = 1;

    while (1) {
        ⟨ Dispatch requests which return HTML result documents 169 ⟩
        exit(0);
    }

    ⟨ Utility functions 373 ⟩
◇
```

12.1.1 Set handler for termination signals

To allow debugging of CPU loop and other "hung program" conditions which lead to CGI timeouts and 500 errors, we catch the `INT` signal, which can be sent to a CGI process observed to be hung. Upon receiving the signal, we print a stack trace to `STDERR`, which will be placed in the HTTP server's error log and then terminate.

⟨ Set handler for termination signals 164a ⟩ ≡

```
$SIG{INT} =
    sub {
        my $i = 0;
        my ($pkg, $file, $line);
        print(STDERR "Termination by INT signal.  Stack trace:\n");
        while (($pkg, $file, $line) = caller($i)) {
            print(STDERR "    $i:  Package $pkg  File $file  Line $line\n");
            $i++;
        }
        die("INT received");
    };
```

◇

Macro referenced in 163.

12.1.2 Define requests permissible whilst browsing public account

Users browsing public accounts are permitted access only to the strictly limited subset of transactions explicitly enumerated below. A browsing user should never see a link which leads to a transaction not on this list, but if a malicious user should cobble up a URL specifying one, it will be thwarted when it is discovered that the user is browsing and the request is not listed in the following hash.

⟨ Define requests permissible whilst browsing public account 164b ⟩ ≡

```
my %browsing_user_requests = (
    account => 1,
    browsepub => 1,
    calendar => 1,
    chart => 1,
    dietcalc => 1,
    do_public_browseacct => 1,
    histchart => 1,
    histreq => 1,
    log => 1,
    logout => 1,
    quitbrowse => 1,
    trendan => 1
);
```

◇

Macro referenced in 163.

12.1.3 Extract brain-dead Internet Exploder request field

Microsoft's lame attempt at a Web browser, "Internet Exploder", seems to take every possible opportunity to misinterpret and mal-implement even the simplest of Web standards. Consider the humble "`<button>`" tag, introduced in HTML 4.0 and part of the current HTML 4.01 and XHTML 1.0 standards. Unlike the original "`<input type="submit">`" button, in which both the label of the button and the text submitted with the form when it is clicked are identical (and hence the button's label cannot contain HTML mark-up or style specifications—just the plain text permissible in an attribute value), the "`<button>`" control decoupled these; the value submitted with the form remains specified by the `value=` attribute, but the label on the button is given by the content of the tag, for example:

```
<button type="submit" name="action" value="detonate">
<b>Ka-<em>BOOM!</em></b>
</button>
```

Even Web designers uninterested in fancy text in buttons find the "`<button>`" tag attractive, because it provides a simple way to include multiple buttons in a single form which send different codes when pressed, independent of their labels. A common example is a form which lists a number of items as rows in a table and, for each item, includes action buttons which operate upon it, for example "Edit" and "Delete". The buttons in each row have the same label, but are distinguished by their `value` fields, for example:

```
<button type="submit" name="edit" value="row12">Edit</button>
<button type="submit" name="delete" value="row12">Delete</button>
```

When one of these buttons is clicked, it informs the application of both the operation requested and the row of the table operated upon. The "`<button>`" tag is useful in any situation where you wish to send different text to the server than is displayed as the label on the button, which is why the keepers of the HTML standard incorporated it back in 1998.

Well, it *would* be useful, if the idiots at Microsoft, who retain a dismaying large share of the Web browser market, had implemented it correctly. Unlike every other competently-implemented browser, which sends the `name` and `value` fields to the server as CGI arguments, Internet Exploder, even the much-vaunted version 7, sends the `name` with the *content* of the button tag instead of the `value`.

In the first example, the server would see an argument named "`action`" with a value of "`<b>Ka-<em>BOOM!</em></b>`", and in the second, if the user pressed the "Edit" button, the server would receive "`edit=Edit`", providing no indication whatsoever of the table row upon which the user wished to operate. As a final kick in the face of the developer trying to build an application on top of that rickety platform, Exploder renders buttons defined with the `<button>` and `<input type="submit">` tags at different vertical positions, so if you combine a number of them on the same line, it looks like they weren't securely glued to the page and shook loose as they travelled over the Internet.

There are several discussions of this outrage on various Web sites, and many suggestions of work-arounds, the vast majority of which use JavaScript. Now, I'm a fan of JavaScript, which, used appropriately, can make pages more responsive and interactive, but I dislike making pages that depend upon it for correct operation; some users block JavaScript as a matter of security (or are behind firewalls which do so), and many text-only browsers and screen reader programs used by blind users do not support JavaScript. Disenfranchising these individuals from using a Web application just to work around Microsoft incompetence is unacceptable.

Here is the solution I have settled on. This is implemented in the context of a Perl CGI application which uses the Perl `CGI` package with extensions of my own devising to parse form arguments into a hash named `%CGIargs`. Rather than use a button declared as:

```
<button type="submit" name="edit" value="row12">Edit</button>
```

I declare it as:

```
<input type="submit" name="edit=row12" value="Edit" />
```

and then use the following Perl code to parse any CGI arguments with name fields containing an equal sign into name and value pairs, which are assigned to new values in the CGI arguments hash. (If for some bizarre reason you require CGI argument names containing equal signs for some other purpose, simply pick a different delimiter.)

This solves the problem of multiple buttons per form with the same value. If you require buttons with HTML content, you can use the same trick in the `name=` field of a `<button>` tag. However, if your users are using Exploder 6, you *still* cannot use multuple `<button>` tags in a single form because it moronically sends *all* buttons in the form, *not just the one which was pressed!* (Note that this doesn't happen with multiple `<input type="submit">` buttons—that would be too consistent for Microsoft.) This has been fixed in Exploder 7, but unless you don't care about users who have yet to "upgrade" to that piece of... software, you're going to have to stay away from `<button>` entirely.

⟨ Extract brain-dead Internet Exploder request field 166 ⟩ ≡

```
if (!defined $CGIargs{q}) {
    for my $qk (keys(%CGIargs)) {
        if ($qk =~ m/^(\w+)=(.*)$/) {
            $CGIargs{$1} = $2;
        }
    }
}
```

◇

Macro referenced in 163.

12.1.4 Estimate local time at user site

On various occasions, for example when deciding which monthly log to display when a user logs in, or deciding if log entries are "in the future", we would like to know the local wall clock time at the user's location. Unfortunately, the HTTP request does not provide this information, so we fall back on a scheme in which a `hidden` input field called `HDiet_tzoffset` is set by JavaScript code when each of our pages is loaded and then submitted with each transaction the user performs. This code checks for the presence of this field in the CGI request. If present, and set to a valid number, it will be used to set variables which represent the local time in seconds since the epoch and the local civil time. If the field has the default value of "`unknown`", indicating JavaScript is not enabled, or the field is out of range (more than 25 hours from UTC), the specification is ignored and UTC is used as the local time.

⟨ Estimate local time at user site 167 ⟩ ≡

```
    my ($timeZoneOffset, $userTime) = ('unknown', time());
    if (defined($CGIargs{HDiet_tzoffset})) {
        if ($CGIargs{HDiet_tzoffset} =~ m/^\-?\d+$/) {
            $timeZoneOffset = $CGIargs{HDiet_tzoffset};
            if (abs($timeZoneOffset) > (25 * 60)) {
                $timeZoneOffset = 'unknown';
            } else {
                $userTime -= $timeZoneOffset * 60;
            }
        }
    }
    my $tzOff = "&amp;HDiet_tzoffset=$timeZoneOffset";
    my ($userYear, $userMon, $userMday, $userHour, $userMin, $userSec) =
        unix_time_to_civil_date_time($userTime);
#if ($CGIargs{HDiet_tzoffset}) {
#print(STDERR "Local time($CGIargs{HDiet_tzoffset}:$timeZoneOffset): $userYear-$userMon-$userMday $userHour:$u
#}
◇
```

Macro referenced in 163.

12.1.5 Dispatch requests which return non-HTML results

Requests which return results with types other than HTML (for example, the images for chart requests, zipped backup archives, CSV files, etc.) are dispatched by the following code. It is up to the request handler to generate a "`Content-type`" header appropriate for the result it returns. An error in one of the handlers can flip the request back to one of the HTML handlers by performing a "`goto requeue`" before the `Content-type` is specified.

⟨ Dispatch requests which return non-HTML results 168 ⟩ ≡

```
if (defined $CGIargs{q}) {
    if ($CGIargs{q} eq 'chart') {
        ⟨ Generate monthly chart 250a ⟩
    } elsif ($CGIargs{q} eq 'histchart') {
        ⟨ Generate historical chart 287 ⟩
    } elsif ($CGIargs{q} eq 'csvout') {
        ⟨ Download monthly log as CSV file 235b ⟩
    } elsif ($CGIargs{q} eq 'xmlout') {
        ⟨ Download monthly log as XML file 236 ⟩
    } elsif ($CGIargs{q} eq 'backup') {
        ⟨ Download backup copy of all logs for user 249 ⟩
    } elsif ($CGIargs{q} eq 'do_exportdb') {
        ⟨ Process database export 239 ⟩
    }
}
```

◇

Macro referenced in 163.

12.1.6 Dispatch requests which return HTML result documents

This is the master request dispatcher for requests which result in the return of an HTML document to the user. By the time we get here the "`Content-type`" of HTML has already been specified. This dispatcher is wrapped in a `while` loop and has an `exit` at the bottom, so request handlers can hand off requests to one another by modifying the CGI request arguments and performing a `next`.

⟨ Dispatch requests which return HTML result documents 169 ⟩ ≡

```
⟨ Login-related transactions 170a ⟩
⟨ Account management transactions 170b ⟩
} elsif ($CGIargs{q} eq 'log') {
    ⟨ Display monthly log 196 ⟩
} elsif ($CGIargs{q} eq 'update_log') {
    ⟨ Update monthly log 206 ⟩
} elsif ($CGIargs{q} eq 'calendar') {
    ⟨ Display calendar navigation page 208 ⟩
} elsif ($CGIargs{q} eq 'trendan') {
    ⟨ Trend analysis 251 ⟩
} elsif (($CGIargs{q} eq 'dietcalc') || ($CGIargs{q} eq 'update_dietcalc')) {
    ⟨ Diet calculator 261 ⟩
} elsif ($CGIargs{q} eq 'save_dietcalc') {
    ⟨ Save diet calculator settings 278 ⟩
} elsif ($CGIargs{q} eq 'histreq') {
    ⟨ Request historical chart 279 ⟩
} elsif ($CGIargs{q} eq 'importcsv') {
    ⟨ Display CSV import request form 211 ⟩
} elsif ($CGIargs{q} eq 'csv_import_data') {
    ⟨ Import uploaded CSV log entries 214, ... ⟩
} elsif ($CGIargs{q} eq 'exportdb') {
    ⟨ Export log database 237 ⟩
} elsif ($CGIargs{q} eq 'browsepub') {
    ⟨ List publicly-visible accounts 301 ⟩
} elsif ($CGIargs{q} eq 'do_public_browseacct') {
    ⟨ Provide browse access to public account 304 ⟩
} elsif ($CGIargs{q} eq 'quitbrowse') {
    ⟨ Quit browsing another account 235a ⟩
} elsif ($CGIargs{q} eq 'configure_badge') {
    ⟨ Configure Web page status badge 228, ... ⟩
} elsif ($CGIargs{q} eq 'paper_logs') {
    ⟨ Request paper log forms 245 ⟩
} elsif ($CGIargs{q} eq 'do_paper_logs') {
    ⟨ Generate paper log forms 247 ⟩
} elsif ($CGIargs{q} eq 'update_badge') {
    ⟨ Update Web page status badge 231, ... ⟩
} elsif ($CGIargs{q} eq 'update_trend') {
    ⟨ Recalculate trend carry-forward for all logs for a user 234 ⟩
} elsif ($CGIargs{q} eq 'feedback') {
    ⟨ Send a feedback message 348 ⟩
} elsif ($CGIargs{q} eq 'send_feedback') {
    ⟨ Send a message from the feedback form 353 ⟩
} elsif ($CGIargs{q} eq 'test') {
    ⟨ Generate test output page 367b ⟩

⟨ Dispatch administrator requests 171 ⟩

} else {
    ⟨ Emit diagnostic for undefined query 367c ⟩
}
```

◇

Macro referenced in 163.

12.1.6.1 Login-related transactions

The following transactions are associated with logging into or out of an existing account.

⟨ Login-related transactions 170a ⟩ ≡

```
if ((!defined $CGIargs{q}) ||
    ($CGIargs{q} eq 'login') ||
    ($CGIargs{q} eq 'newlogin')) {
    ⟨ Return login request form 172 ⟩
} elsif ($CGIargs{q} eq 'validate_user') {
    ⟨ Validate user login request 173 ⟩
} elsif ($CGIargs{q} eq 'relogin') {
    ⟨ Force re-login if session terminated or invalid 186 ⟩
} elsif ($CGIargs{q} eq 'logout') {
    ⟨ Log out user: end session 192 ⟩
} elsif ($CGIargs{q} eq 'wipedb') {
    ⟨ Delete entire log database 357 ⟩
} elsif ($CGIargs{q} eq 'do_wipedb') {
    ⟨ Process database delete 360 ⟩
} elsif ($CGIargs{q} eq 'closeaccount') {
    ⟨ Close this user account 363 ⟩
} elsif ($CGIargs{q} eq 'do_closeaccount') {
    ⟨ Process user account close 365 ⟩
```

◇

Macro referenced in 169.

12.1.6.2 Account management transactions

The following transactions perform various functions on a user account.

⟨ Account management transactions 170b ⟩ ≡

```
} elsif ($CGIargs{q} eq 'account') {
    ⟨ Main account dispatch page 179 ⟩
} elsif ($CGIargs{q} eq 'new_account') {
    ⟨ Process new user account request 289 ⟩
} elsif ($CGIargs{q} eq 'modacct') {
    ⟨ Modify user account request 295 ⟩
} elsif ($CGIargs{q} eq 'clearcookies') {
    ⟨ Forget all persistent logins 300 ⟩
} elsif ($CGIargs{q} eq 'edit_account') {
    ⟨ Process user account modification 297 ⟩
} elsif ($CGIargs{q} eq 'pwreset') {
    ⟨ Display password reset request form 187 ⟩
} elsif ($CGIargs{q} eq 'new_password') {
    ⟨ Reset a user's password 188 ⟩
```

◇

Macro referenced in 169.

12.1.6.3 Dispatch administrator requests

Here we dispatch requests which are permitted only for users with administrative privilege. Each of these request handlers is responsible for verifying that the user is so endowed; we can't do it here because we haven't yet retrieved the session and user information, which does not exist for all transactions.

⟨ Dispatch administrator requests 171 ⟩ ≡

```
} elsif ($CGIargs{q} eq 'acctmgr') {
    ⟨ Display administrator account manager 309, ... ⟩
} elsif ($CGIargs{q} eq 'do_admin_browseacct') {
    ⟨ Provide administrator access to user account 314 ⟩
} elsif ($CGIargs{q} eq 'do_admin_purgeacct') {
    ⟨ Process administrator database purge 316, ... ⟩
} elsif ($CGIargs{q} eq 'do_admin_delacct') {
    ⟨ Process administrator account delete 318, ... ⟩
} elsif ($CGIargs{q} eq 'sessmgr') {
    ⟨ Display administrator session manager 320 ⟩
} elsif ($CGIargs{q} eq 'cookiemgr') {
    ⟨ Display administrator persistent login manager 327 ⟩
} elsif ($CGIargs{q} eq 'do_admin_delcookie') {
    ⟨ Delete a persistent login token 329 ⟩
} elsif ($CGIargs{q} eq 'globalstats') {
    ⟨ Display administrator global statistics 331 ⟩
} elsif ($CGIargs{q} eq 'synthdata') {
    ⟨ Generate synthetic data for user account 341 ⟩
} elsif ($CGIargs{q} eq 'do_admin_forceclose') {
    ⟨ Force termination of user session 323 ⟩
} elsif (⟨ Beta test 3e ⟩ && ($CGIargs{q} eq 'invite')) {
    ⟨ Request invitation codes 306 ⟩
} elsif (⟨ Beta test 3e ⟩ && ($CGIargs{q} eq 'generate_invitations')) {
    ⟨ Generate invitation codes 307, ... ⟩
```

◇

Macro referenced in 169.

12.1.7 Return login request form

The login request form is the point of entry when no session is active (and hence when the program is invoked with no CGI arguments). The user is invited to enter their user name and password. A "`newlogin`" transaction indicates the user has explicitly logged out. In this case we ignore the presence of a cookie, which allows the user to explictly log back in without "Remember me" in order to perform operations which are forbidden in a cookie login.

⟨ Return login request form 172 ⟩ ≡

```
    $CGIargs{HDiet_handheld} = 'y' if $CGIargs{handheld};

    if ((!defined($CGIargs{q})) || ($CGIargs{q} ne 'newlogin')) {
        $cookieUser = testCookiePresent(⟨ Cookie name 11a ⟩);
        if (defined($cookieUser)) {
#print(STDERR "A cookie was present for ($cookieUser)\n");
            $cookieLogin = 1;
            $CGIargs{q} = 'validate_user';
            next;
        }
    }

    write_XHTML_prologue($fh, $homeBase, "Please Sign In", " checkSecure();", $CGIargs{HDiet_handheld});
    print $fh <<"EOD";
<h1 class="c">Please Sign In</h1>
EOD
    $CGIargs{HDiet_username} = '' if !defined($CGIargs{HDiet_username});
    my $u = HDiet::user->new($CGIargs{HDiet_username});
    $u->login_form($fh, $tzOff, $CGIargs{HDiet_handheld}, $CGIargs{HDiet_remember});

    write_XHTML_epilogue($fh, $homeBase);
```

◇

Macro referenced in 170a.

12.1.8 Validate user login request

This code handles the submission of the login request form. We receive the user ID and password as form arguments, and verify them. We also arrive here when the user requests to create a new account, but that is distinguished by the presence of the "`new`" item among the arguments and immediately dispatched to the new account form.

⟨ Validate user login request 173 ⟩ ≡

```
if (defined($CGIargs{new})) {
    ⟨ Create new user account request 288a ⟩
} else {

    #   If no user name given, re-issue login form
    if ((!$cookieLogin) && ((!defined($CGIargs{HDiet_username})) ||
        ($CGIargs{HDiet_username} eq '''))) {
        $CGIargs{q} = 'login';
        next;
    }

    my ($user_file_name, $ui);

    if ($cookieLogin) {
        $user_file_name = quoteUserName($cookieUser);
        if (!(-f "⟨ Users Directory 6h ⟩/$user_file_name/UserAccount.hdu")
            || (!open(FU, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/UserAccount.hdu"))) {
            ⟨ Reject login: Unknown user name 174b ⟩
        }
        $ui = HDiet::user->new();
        $ui->load(\*FU);
        close(FU);
        $CGIargs{HDiet_username} = $ui->{login_name};
        $CGIargs{HDiet_remember} = 'y';
    } else {
        ⟨ Validate user name and password 174a ⟩
    }

    ⟨ Close previous session if still open 176a ⟩

    ⟨ Open new session and link to user directory 176b ⟩

    ⟨ Update last login and transaction time 177a ⟩

    ⟨ Add login to history database 177b ⟩

    ⟨ Update persistent login state 178 ⟩

    #   Queue the transaction to display the current month's log for this user

    %CGIargs = (
        q => "log",
        s => $s->{session_id},
        m => "now",
        HDiet_tzoffset => $timeZoneOffset
    );
    next;
}
```

◇

Macro referenced in 170a.

12.1.8.1 Validate user name and password

Validate the user name and password by first determining that a user directory exists for the file name encoded user name supplied. If so, read the account information file (which should exist, unless this login arrived while we're in the middle of creating a new account which, of course, can happen) and validate the password. If either the user name or password is incorrect, reject the login attempt.

At the moment, we don't do anything to deter high-speed blasts of login attempts to try to crack passwords by brute force. This is one of the many refinements to add in the future.

⟨ Validate user name and password 174a ⟩ ≡

```
    #   Verify user account directory exists and contains
    #   valid user information file.
    $user_file_name = quoteUserName($CGIargs{HDiet_username});
    if (!(-f "⟨ Users Directory 6h ⟩/$user_file_name/UserAccount.hdu")
        || (!open(FU, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/UserAccount.hdu"))) {
        ⟨ Reject login: Unknown user name 174b ⟩
    }

    #   Read user account information and check password
    $ui = HDiet::user->new();
    $ui->load(\*FU);
    close(FU);
    if (($CGIargs{HDiet_password} eq '') && $ui->{read_only}) {
        $readOnly = 1;
    } elsif ($CGIargs{HDiet_password} ne $ui->{password}) {
        ⟨ Reject login: Incorrect password 175b ⟩
    }
```

◇

Macro referenced in 173.

12.1.8.1.1 Reject login: Unknown user name Reject the login if a user name was specified for which we have no account. To increase security, for failed logins, we do not indicate whether the user name or password was incorrect, but we process these errors in separate code to allow diagnostics to be inserted for debugging.

⟨ Reject login: Unknown user name 174b ⟩ ≡

```
    ⟨ Log failed login attempt in system log (174c 0 ) 175a ⟩
    $CGIargs{HDiet_handheld} = 'y' if $CGIargs{handheld};
    write_XHTML_prologue($fh, $homeBase, "Please Sign In", " checkSecure();", $CGIargs{HDiet_handheld});
    print $fh <<"EOD";
<h1 class="c">Sign In Invalid: Incorrect User Name or Password</h1>
<h1 class="c">Please Sign In</h1>
EOD
    my $u = HDiet::user->new();
    $u->login_form($fh, $tzOff, $CGIargs{HDiet_handheld}, $CGIargs{HDiet_remember});
    write_XHTML_epilogue($fh, $homeBase);
    last;
```

◇

Macro referenced in 173, 174a, 188.

12.1.8.1.1.1 Log failed login attempt in system log When a login attempt fails, we record it in the system log in the same format used by PAM authentication failures. At Fourmilab, these records are parsed by the Gardol attack mitigation package which, after a specified number of successive login failures, will banish the IP address until it stops trying and goes silent for a specified period of time. The specified user name is given in the "`ruser`" field in the encoded form from `quoteUserName`, surrounded by single quotes. The "`uid`" field is zero if the user name was incorrect and 1 if the user name was valid but an incorrect password was given.

⟨ Log failed login attempt in system log 175a ⟩ ≡

```
    my @lt = localtime(time());
    my $ct = sprintf("%s %d %02d:%02d:%02d",
        MONTH_ABBREVIATIONS->[$lt[4]], $lt[3], $lt[2], $lt[1], $lt[0]);

    openlog("HackDiet", "pid", "LOG_AUTH");
    syslog("info",
        "$ENV{REMOTE_ADDR}: (1) $ct $ENV{SERVER_NAME} HackDiet(pam_unix)[$$]: " .
        "authentication failure; logname= uid=@1 euid=0 tty=http " .
        "ruser='$user_file_name' rhost=$ENV{REMOTE_ADDR}");
    closelog();
```

◇

Macro referenced in 174b, 175b.

12.1.8.1.2 Reject login: Incorrect password The user name specifies a valid account but the password doesn't match. As noted above, in production we don't inform the user that the user name was actually valid, but we handle a bad password in separate code so diagnostics can be added in case of problems.

⟨ Reject login: Incorrect password 175b ⟩ ≡

```
    ⟨ Log failed login attempt in system log (175c 1 ) 175a ⟩
    $CGIargs{HDiet_handheld} = 'y' if $CGIargs{handheld};
    write_XHTML_prologue($fh, $homeBase, "Please Sign In", " checkSecure();", $CGIargs{HDiet_handheld});
    print $fh <<"EOD";
<h1 class="c">Sign In Invalid: Incorrect User Name or Password</h1>
<h1 class="c">Please Sign In</h1>
EOD
    my $u = HDiet::user->new();
    $u->login_form($fh, $tzOff, $CGIargs{HDiet_handheld}, $CGIargs{HDiet_remember});
    write_XHTML_epilogue($fh, $homeBase);
    append_history($user_file_name, 10);
    last;
```

◇

Macro referenced in 174a.

12.1.8.2 Close previous session if still open

If a session is still open for this user (presumably because they didn't bother to log off after the last session), there will be an `ActiveSession.hda` file in the user directory. Use it to obtain the session ID, then delete the session file from the session directory and the back-link to it in the user directory. The forced closing of an active session is recorded in the `History.hdh` database in the user directory as a type 3 record containing the time the session was closed and the IP address from which the login was made which caused it to be closed.

⟨ Close previous session if still open 176a ⟩ ≡

```
if ((!$readOnly) && (-f "⟨ Users Directory 6h ⟩/$user_file_name/ActiveSession.hda")
    && open(FS, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/ActiveSession.hda")) {
    my $asn = load_active_session(\*FS);
    close(FS);
    unlink("⟨ Users Directory 6h ⟩/$user_file_name/ActiveSession.hda");
    clusterDelete("⟨ Users Directory 6h ⟩/$user_file_name/ActiveSession.hda");
    unlink("⟨ Session Directory 6g ⟩/$asn.hds");
    clusterDelete("⟨ Session Directory 6g ⟩/$asn.hds");
    append_history($user_file_name, 3);
}
```

◇

Macro referenced in 173, 188, 323.

12.1.8.3 Open new session and link to user directory

Open a new session for this user's login, and add it to the session directory. We create a back-link from the user directory to the currently open session in the form of an "active session file" named `ActiveSession.hda` in the user directory; this simply contains the session ID.

⟨ Open new session and link to user directory 176b ⟩ ≡

```
#   Create new session and add file to session directory
my $s = HDiet::session->new($CGIargs{HDiet_username});
$s->{read_only} = $readOnly;
$s->{handheld} = 1 if $CGIargs{HDiet_handheld};
$s->{cookie} = $cookieLogin;
open(FS, ">:utf8", "⟨ Session Directory 6g ⟩/$s->{session_id}.hds") ||
    die("Cannot create session file ⟨ Session Directory 6g ⟩/$s->{session_id}.hds");
$s->save(\*FS);
close(FS);
clusterCopy("⟨ Session Directory 6g ⟩/$s->{session_id}.hds");

#   Add the ActiveSession.hda back-link to the user directory
if (!$readOnly) {
    open(FS, ">:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/ActiveSession.hda") ||
        die("Cannot create active session file ⟨ Users Directory 6h ⟩/$user_file_name/ActiveSession.hda");
    $s->save_active_session(\*FS);
    close(FS);
    clusterCopy("⟨ Users Directory 6h ⟩/$user_file_name/ActiveSession.hda");
}
```

◇

Macro referenced in 173.

12.1.8.4 Update last login and transaction time

We keep track of the time and date of the last login by a user and the last transaction processed for an open session in the `LastLogin.hdl` and `LastTransaction.hdl` files in the user directory. These are simple text files, with the first line a version identifier ("`1`" currently), and the second the UNIX `time()` of the event. These files permit timing out inactive sessions.

⟨ Update last login and transaction time 177a ⟩ ≡

```
        #   Update the date and time of the last login by this user
        if ($readOnly) {
            open(FL, ">:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/LastLogin.hdl") ||
               die("Cannot create last login file ⟨ Users Directory 6h ⟩/$user_file_name/LastLogin.hdl");
            print FL <<"EOD";
1
$s->{login_time}
EOD
            close(FL);
            clusterCopy("⟨ Users Directory 6h ⟩/$user_file_name/LastLogin.hdl");

            update_last_transaction($user_file_name);
        }
```

◇

Macro referenced in 173.

12.1.8.5 Add login to history database

The new login is recorded in the `History.hdh` database in the user directory as a record of type 1 containing the UNIX time of the login and the IP address from which the user logged in.

⟨ Add login to history database 177b ⟩ ≡

```
        append_history($user_file_name, 1, "$s->{handheld},$s->{cookie}") if !$readOnly;
```

◇

Macro referenced in 173.

12.1.8.6 Update persistent login state

If the user has checked the "Remember me' box, create and store a persistent login cookie which will be used in subsequent automatic logins. If the box is not checked and a cookie (valid or not) is present, issue a header item which causes it to be revoked; this allows a user to cancel "Remember me" mode by explicitly logging out, then logging back in with the box unchecked.

Another small subtlety if that the user may have a login cookie present, but opt to explicitly log out and then log back with with "Remember me" set. In this case, we want to get rid of the persistent cookie on the server to avoid them accumulating. (The periodic expired cookie sweep will clean them up, but it's better to avoid the clutter in the most common case which can lead to orphaned cookies.)

⟨ Update persistent login state 178 ⟩ ≡

```
if (!$ui->{read_only}) {
    if ($CGIargs{HDiet_remember}) {
        testCookiePresent(⟨ Cookie name 11a ⟩);
        push(@HTTP_header, "Set-Cookie: " . storeCookie($ui));
    } else {
        my $cname = ⟨ Cookie name 11a ⟩;
        if (defined(testCookiePresent($cname)) ||
            (defined($ENV{HTTP_COOKIE}) &&
            ($ENV{HTTP_COOKIE} =~ m/$cname=([0-9FGJKQW]{48})/))) {
#print(STDERR "Revoking cookie $ENV{HTTP_COOKIE}\n");
            my $excook = HDiet::cookie->new();
            push(@HTTP_header, "Set-Cookie: " . $excook->expireCookie($cname));
        }
    }
}
```

◇

Macro referenced in 173.

12.1.9 Main account dispatch page

This page is displayed after a user successfully logs in. It contains links to all of the things the user can do once logged in.

⟨ Main account dispatch page 179 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    my $qun = quoteHTML($user_name);
    write_XHTML_prologue($fh, $homeBase, $qun, undef, $session->{handheld});
    generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, "Utilities", undef, $browse_public,
    ⟨ Generate assumed identity notification 185 ⟩

    ⟨ Show user name and account being browsed 180a ⟩

    ⟨ Standard navigation bar functions 180b ⟩

    if ($browse_public) {
        ⟨ Browsing public account functions 182 ⟩
    } else {
        ⟨ Utility functions for regular session 181 ⟩

#       if (⟨ Beta test 3e ⟩) {
           print $fh <<"EOD";
    <li class="skip"><a href="⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=feedback$tzOff">Se
EOD
#       }
    }

    print $fh <<"EOD";
    <li class="skip"><a href="⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=logout$tzOff">Sign
</ul>
EOD

    if ($ui->{administrator} || $assumed_identity) {
        ⟨ Administrator-only functions 183 ⟩
    }

    ⟨ Show build number and date 184 ⟩

    write_XHTML_epilogue($fh, $homeBase);
◇
```

Macro referenced in 170b.

12.1.9.1 Show user name and account being browsed

If the user is browsing a public account, show its public name. Otherwise, generate the "Welcome" message for a user logged into their own account.

⟨ Show user name and account being browsed 180a ⟩ ≡

```
        if ($browse_public) {
            my $qrn = quoteHTML($real_user_name);
            print $fh <<"EOD";
    <h2 class="c">$qrn browsing<br /> public $qun account</h2>
    EOD
        } else {
            print $fh <<"EOD";
    <h1 class="c">Welcome, $qun</h1>
    EOD
        }
    ◇
```

Macro referenced in 179.

12.1.9.2 Standard navigation bar functions

These menu items provide links to functions which, with the exception of the diet calculator, also appear in the navigation bar. They're here in the interest of completeness. The last item is, however, the only way to get to the diet calculator.

⟨ Standard navigation bar functions 180b ⟩ ≡

```
        print $fh <<"EOD";
    <ul>
        <li><a href="⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=log&amp;m=now$tzOff">Current mo
        <li><a href="⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=calendar$tzOff">Historical logs
        <li><a href="⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=histreq$tzOff">Historical chart
        <li><a href="⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=trendan$tzOff">Trend analysis</
        <li><a href="⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=dietcalc$tzOff">Diet calculator
    EOD
    ◇
```

Macro referenced in 179.

12.1.9.3 Utility functions for regular session

The following menu items are the grab-bag of functions a user can perform when logged in to their account in the normal fashion.

⟨ Utility functions for regular session 181 ⟩ ≡

```
    print $fh <<"EOD";
    <li class="skip"><a href="⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=modacct$tzOff">Edi
    <li><a href="⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=configure_badge$tzOff">Configur
    <li><a href="⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=paper_logs$tzOff">Print paper l
    <li><a href="⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=update_trend&amp;m=0000-00&amp;
    <li><a href="⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=clearcookies$tzOff">Forget pers

    <li class="skip"><a href="⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=exportdb$tzOff">Ex
    <li><a href="⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=importcsv$tzOff">Import CSV  or
    <li><a href="⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=backup$tzOff">Download native d

    <li class="skip"><a href="⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=wipedb$tzOff">Dele
    <li><a href="⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=closeaccount$tzOff">Close this
EOD

    if (!$readOnly) {
        print $fh <<"EOD";

    <li class="skip">

        <form id="Hdiet_pubacct" ⟨ Form processing action and method 12b ⟩>
            <p style="margin-top: 0px; margin-bottom: 4px;">
            <input type="hidden" name="s" value="$session->{session_id}" />
            Browse public user accounts:
            <select name="acct_category" size="1">
                <option value="active" selected="selected">Active accounts</option>
                <option value="inactive">Inactive accounts</option>
                <option value="all">All accounts</option>
            </select>
            <input type="submit" name="q=browsepub" value=" View " />
            </p>
        </form>

        <form id="Hdiet_acctmgr" ⟨ Form processing action and method 12b ⟩>
            <p style="margin-top: 0px;">
            Access public account name:
            <input type="text" name="pubacct" maxlength="80" size="21" />
            <input type="hidden" name="s" value="$session->{session_id}" />
            <input type="submit" name="q=do_public_browseacct" value=" View " />
            </p>
        </form>
    </li>
EOD
    }
```

◇

Macro referenced in 179.

12.1.9.4 Browsing public account functions

The following menu items are shown only when the user is browsing a public account. They allow the user to quit browsing or choose a different account to browse.

⟨ Browsing public account functions 182 ⟩ ≡

```
print $fh <<"EOD";

<li class="skip"><a href="⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=quitbrowse$tzOff">
<li><a href="⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=browsepub$tzOff">Browse a diffe
    <form id="Hdiet_acctmgr" ⟨ Form processing action and method 12b ⟩>
        ⟨ Local time zone offset field 372b ⟩
        <p style="margin-top: 0px;">
        Access public account name:
        <input type="text" name="pubacct" maxlength="80" size="21" />
        <input type="hidden" name="s" value="$session->{session_id}" />
        <input type="submit" name="q=do_public_browseacct" value=" View " />
        </p>
    </form>
</li>
EOD
◇
```

Macro referenced in 179.

12.1.9.5 Administrator-only functions

The following functions are available only if the user has administrator privilege. The transactions which perform the following functions individually check for privilege—they are not just protected by being hidden on this page.

⟨ Administrator-only functions 183 ⟩ ≡

```
        print $fh <<"EOD";
    <h2 class="c">Administrator Functions</h2>

    <ul>
        <li class="skip">
            <form id="Hdiet_admacct" ⟨ Form processing action and method 12b ⟩>
                <p style="margin-top: 0px; margin-bottom: 4px;">
                <input type="hidden" name="s" value="$session->{session_id}" />
                Manage user accounts:
                <select name="acct_category" size="1">
                    <option value="active" selected="selected">Active accounts</option>
                    <option value="inactive">Inactive accounts</option>
                    <option value="all">All accounts</option>
                </select>
                <input type="submit" name="q=acctmgr" value=" View " />
                </p>
            </form>

            <form id="Hdiet_acctadm" ⟨ Form processing action and method 12b ⟩>
                <p style="margin-top: 0px;">
                User account name:
                <input type="text" name="useracct" maxlength="80" size="21" />
                <input type="hidden" name="s" value="$session->{session_id}" />
                <input type="submit" name="q=do_admin_browseacct" value=" View " />
                 
                <input type="submit" name="q=do_admin_delacct" value=" Delete " />
                <input type="submit" name="q=do_admin_purgeacct" value=" Purge Logs " />
                <input type="password" name="HDiet_password" size="20" maxlength="4096" value="" />
                </p>
            </form>
    </li>

        <li><a href="⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=sessmgr$tzOff">Manage sessions<
        <li><a href="⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=cookiemgr$tzOff">Manage persist
        <li><a href="⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=globalstats$tzOff">Display glob
        <li><a href="⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=synthdata$tzOff">Generate synth
    EOD

        if (⟨ Beta test 3e ⟩) {
            print $fh <<"EOD";
        <li><a href="⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=invite$tzOff">Create invitation
    EOD
        }

        print $fh <<"EOD";
    </ul>
    EOD
    ◇
```

Macro referenced in 179.

12.1.9.6 Show build number and date

On beta test builds, show the build number (updated automatically by the `Makefile`) and the time and date of the build at the bottom right of the page.

⟨ Show build number and date 184 ⟩ ≡

```
#    if (⟨ Beta test 3e ⟩) {
        my $bn = <<"EOD";
⟨ Build Number 3c ⟩
EOD
        $bn =~ s/\s+$/:/;
        print $fh <<"EOD";
<p class="build">
Build $bn ⟨ Build Time 3d ⟩</p>
EOD
#    }
```

◇

Macro referenced in 179.

12.1.9.7 Generate assumed identity notification

If the user is operating under an assumed identity (which can be the administrator performing maintenance in the name of a user, or a regular user browsing an account whose owner has granted public read access), generate a message to remind the user the data presented is not their own.

If the user is logged into a demonstration account in read-only mode, a legend indicating this is displayed.

⟨ Generate assumed identity notification 185 ⟩ ≡

```
    if ($readOnly) {
        print $fh <<"EOD";
<h3 class="browsing">Read-only access: Changes are not saved.</h3>
EOD
    }

    if ($assumed_identity) {
        my $eu = quoteHTML($effective_user_name);
        print $fh <<"EOD";
<form id="Hdiet_quitadm" ⟨ Form processing action and method 12b ⟩>
<div>
<input type="hidden" name="q" value="quitbrowse" />
<input type="hidden" name="s" value="$session->{session_id}" />
</div>

<h3 class="browsing">Administrator accessing account of $eu
     
<input type="submit"
    title="End browsing this account" value="Exit" />
</h3>
</form>
EOD
    } elsif ($browse_public) {
        my $eu = quoteHTML($effective_user_name);
        print $fh <<"EOD";
<form id="Hdiet_quitbrowse" ⟨ Form processing action and method 12b ⟩>
<div>
<input type="hidden" name="q" value="quitbrowse" />
<input type="hidden" name="s" value="$session->{session_id}" />
</div>

<h3 class="browsing">Browsing public account $eu
     
<input type="submit"
    title="End browsing this public account" value="Exit" />
</h3>
</form>
EOD
    }
```

◇

Macro referenced in 179, 196, 208, 211, 214, 228, 231, 234, 237, 239, 245, 247, 249, 251, 261, 279, 295, 296, 297, 299, 300, 301, 304, 305, 306, 307, 309, 314, 315, 316, 318, 320, 324, 325, 326, 327, 330, 331, 341, 348, 353, 357, 360, 363, 365.

12.1.10 Force re-login if session terminated or invalid

If the request contains a session ID which corresponds to no active session, it's probably because the session has timed out or because the user is submitting a form from one browser window after having logged out from another. In this case, we have no alternative but to discard the request and ask the user to log back in.

⟨ Force re-login if session terminated or invalid 186 ⟩ ≡

```
        $CGIargs{HDiet_handheld} = 'y' if $CGIargs{handheld};
        write_XHTML_prologue($fh, $homeBase, "Please Sign In", " checkSecure();", $CGIargs{HDiet_handheld});
        print $fh <<"EOD";
    <h1 class="c">Your session has timed out or has been ended.</h1>
    <h1 class="c">Please Sign In Again</h1>
    EOD
        my $u = HDiet::user->new();
        $u->login_form($fh, $tzOff, $CGIargs{HDiet_handheld}, $CGIargs{HDiet_remember});
        write_XHTML_epilogue($fh, $homeBase);
    ◇
```

Macro referenced in 170a.

12.1.11 Display password reset request form

Users who forget their password may request the password be reset to a random value which is sent to their registered E-mail address. To guard against malicious reset requests by those who guess common user names, we require the user to specify the registered E-mail address for the account when resetting the password.

⟨ Display password reset request form 187 ⟩ ≡

```
    my $qun = '';
    $qun = quoteHTML($CGIargs{HDiet_username}) if defined($CGIargs{HDiet_username});


    write_XHTML_prologue($fh, $homeBase, "Reset Password", undef, $CGIargs{HDiet_handheld});
    print $fh <<"EOD";
<h1 class="c">Reset Password</h1>
<form id="Hdiet_reset_password" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩
EOD

    ⟨ Propagate handheld setting to subsequent forms 288b ⟩

    print $fh <<"EOD";
<p class="justified">
To reset your password to a new value, which will be sent via E-mail
to the registered E-mail address for your account, enter your User
Name and E-mail address in the boxes below and press the &ldquo;Reset
Password&rdquo; button.
</p>

<table border="border" class="login">
<tr><th><span class="accesskey">U</span>ser Name:</th>
    <td><input accesskey="u" type="text" name="HDiet_username" size="60"
               maxlength="⟨ Maximum Text Input Field Length 9f ⟩" value="$qun" /></td>
</tr>
<tr><th><span class="accesskey">E</span>-mail address:</th>
    <td><input accesskey="e" type="text" name="HDiet_email" size="60"
               maxlength="⟨ Maximum Text Input Field Length 9f ⟩" value="" /></td>
</tr>
</table>
EOD
    my $u = HDiet::user->new($CGIargs{HDiet_username});
    print $fh <<"EOD";
<p class="mlog_buttons">
<input type="hidden" name="q" value="new_password" />
<input type="submit" name="reset" value=" Reset Password " />
 
<input type="submit" name="cancel" value=" Cancel " />
</p>
</form>
EOD
    write_XHTML_epilogue($fh, $homeBase);
◇
```

Macro referenced in 170b.

12.1.12 Reset a user's password

When a user requests a password reset, we first validate the user name and confirm that the user has an E-mail address on file and that it agrees with the E-mail address specified in the password reset request. Then we close any session which may be active, call the `resetPassword` method of the `user` object to generate the new password, and send an E-mail to notify the user of the new password.

⟨ Reset a user's password 188 ⟩ ≡

```
#   If no user name given or the user clicked "Cancel" re-issue login form
if (($CGIargs{HDiet_username} eq '') || $CGIargs{cancel}) {
    $CGIargs{q} = 'login';
    next;
}

#   Verify user account directory exists and contains
#   valid user information file.
my $user_file_name = quoteUserName($CGIargs{HDiet_username});
if (!(-f "⟨ Users Directory 6h ⟩/$user_file_name/UserAccount.hdu")
    || (!open(FU, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/UserAccount.hdu"))) {
    ⟨ Reject login: Unknown user name 174b ⟩
}

#   Read user account information
my $ui = HDiet::user->new();
$ui->load(\*FU);
close(FU);

⟨ Validate E-mail address agrees with specification in reset request 189 ⟩

⟨ Prohibit password reset on read-only account 190a ⟩

⟨ Close previous session if still open 176a ⟩

$ui->resetPassword(⟨ Length of automatically generated passwords 10g ⟩);

⟨ Update user account information 293 ⟩

⟨ Send E-mail confirming password reset 190b ⟩

⟨ Return password reset confirmation page 191 ⟩

append_history($user_file_name, 6);
```

◇

Macro referenced in 170b.

12.1.12.1 Validate E-mail address agrees with specification in reset request

To prevent vandals from guessing common names and maliciously resetting their passwords just to create irritation, we require the user to enter their E-mail address in the password reset request and require it to match the registered E-mail address to which the new password will be sent.

⟨ Validate E-mail address agrees with specification in reset request 189 ⟩ ≡

```
        if ($CGIargs{HDiet_email} ne $ui->{e_mail}) {
            write_XHTML_prologue($fh, $homeBase, "Incorrect E-mail Address", undef, $CGIargs{HDiet_handheld});
            my $arghandheld = $CGIargs{HDiet_handheld} ? '&amp;HDiet_handheld=y' : '';
            my $qun = quoteHTML($CGIargs{HDiet_username});

            print $fh <<"EOD";
    <h1 class="c">Incorrect E-mail Address</h1>

    <p class="justified">
    Your password reset request specified a different E-mail
    address than the one registered for your account to which
    the new password will be sent.  To avoid abuse, you must specify
    the registered E-mail address to confirm your identity before
    the password will be reset.
    </p>

    <h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=login$arghandheld$tzOff">Sign In</a></h4>
    <h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=pwreset$arghandheld&amp;HDiet_username=$qun$tzOff">P
    EOD
            write_XHTML_epilogue($fh, $homeBase);
            append_history($user_file_name, 9);
            last;
       }
    ◇
```

Macro referenced in 188.

12.1.12.2 Prohibit password reset on read-only account

Users are not allowed to reset the password of read-only demonstration accounts even if they know the E-mail address (which they can determine from the account settings page).

⟨ Prohibit password reset on read-only account 190a ⟩ ≡

```
        if ($ui->{read_only}) {
            write_XHTML_prologue($fh, $homeBase, "Password Reset Rejected", undef, $CGIargs{HDiet_handheld});
            my $arghandheld = $CGIargs{HDiet_handheld} ? '&amp;HDiet_handheld=y' : '';
            my $qun = quoteHTML($CGIargs{HDiet_username});

            print $fh <<"EOD";
    <h1 class="c">Password Reset Rejected</h1>

    <p class="justified">
    This is a read-only demonstration account.  You are not permitted
    to request a password reset.  You can sign in to this account in
    read-only mode using a blank password.
    </p>

    <h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=login$arghandheld$tzOff">Sign In</a></h4>
    <h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=pwreset$arghandheld&amp;HDiet_username=$qun$tzOff">P
    EOD
            write_XHTML_epilogue($fh, $homeBase);
            last;
       }
    ◇
```

Macro referenced in 188.

12.1.12.3 Send E-mail confirming password reset

Send an E-mail message to the user's registered account to confirm that the password has been reset to the string given in the message.

⟨ Send E-mail confirming password reset 190b ⟩ ≡

```
        $ui->sendMail("Password reset",
    "Your password for The Hacker's Diet Online:

        ⟨ Site home URL 11f ⟩⟨ URL to invoke this program 12a ⟩

    has been reset at your request.  The new password is:

        $ui->{password}

    You must enter the password exactly as given above; upper and
    lower case letters are not the same.  After logging into your
    account with this new password, you are encouraged to change
    your password to something easier to remember, but difficult
    for a stranger to guess.
    \n");
    ◇
```

Macro referenced in 188.

12.1.12.4 Return password reset confirmation page

Generate an HTML page which confirms that the new password has been sent via E-mail. The confirmation includes a button which returns the user to the login page.

⟨ Return password reset confirmation page 191 ⟩ ≡

```
    write_XHTML_prologue($fh, $homeBase, "Password Reset and Mailed", undef, $CGIargs{HDiet_handheld});
    my ($qun, $qem) = (quoteHTML($ui->{login_name}), quoteHTML($ui->{e_mail}));
    print $fh <<"EOD";
<h1 class="c">Password Reset and Mailed</h1>
<form id="Hdiet_password_reset_confirmation" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩
EOD

    ⟨ Propagate handheld setting to subsequent forms 288b ⟩

    print $fh <<"EOD";
<p class="justified">
The password for your Hacker's Diet Online account:
</p>

<blockquote>
    <p><b>$qun</b></p>
</blockquote>

<p class="justified">
has been reset to a randomly-generated value which has
been sent to your E-mail address:
</p>

<blockquote>
    <p><b>$qem</b></p>
</blockquote>

<p class="justified">
Once you receive this E-mail, return to the login page
and enter the new password to access your account.
</p>

<p class="mlog_buttons">
<input type="hidden" name="q" value="login" />
<input type="hidden" name="HDiet_username" value="$ui->{login_name}" />
<input type="submit" name="reset" value=" Return to Login Page " />
</p>
</form>
EOD
    write_XHTML_epilogue($fh, $homeBase);
◇
```

Macro referenced in 188.

12.1.13 Log out user: end session

When a log out request is received from a user, close the active session, deleting the session file and its back-link in the user directory, and make a history log item (type 2) for the log out.

⟨ Log out user: end session 192 ⟩ ≡

```
⟨ Retrieve active session information 193 ⟩

#   Delete active session file
unlink("⟨ Session Directory 6g ⟩/$CGIargs{s}.hds");
clusterDelete("⟨ Session Directory 6g ⟩/$CGIargs{s}.hds");

if (!$readOnly) {
    unlink("⟨ Users Directory 6h ⟩/$user_file_name/ActiveSession.hda");
    clusterDelete("⟨ Users Directory 6h ⟩/$user_file_name/ActiveSession.hda");
    append_history($user_file_name, 2);
}

#   Return user to login screen
%CGIargs = (
    q => "newlogin",
);
$CGIargs{HDiet_handheld} = 'y' if $session->{handheld};
next;
```

◇

Macro referenced in 170a.

12.1.13.1 Retrieve active session information

The session ID from the CGI request is used to locate the active session file, from which the user name is extracted and quoted into the name of the directory containing the information for that user. If the session ID is invalid, a "relogin" request is queued to permit the user to log back in.

⟨ Retrieve active session information 193 ⟩ ≡

```
$CGIargs{s} = '' if !defined($CGIargs{s});
if ($CGIargs{s} !~ m/^[0-9FGJKQW]{40}$/) {
    die("Invalid (probably spoofed) session identifier ($CGIargs{s})");
}
my $session = HDiet::session->new();
if (!open(FS, "<:utf8", "⟨ Session Directory 6g ⟩/$CGIargs{s}.hds")) {
    %CGIargs = (
        q => "relogin",
    );
    if (!$inHTML) {
        goto requeue;
    }
    next;
}
$session->load(\*FS);
close(FS);
my $user_name = $session->{login_name};
my $real_user_name = $user_name;
my $effective_user_name = '';
my $assumed_identity = 0;
my $browse_public = 0;
$readOnly = $session->{read_only};
if ($readOnly) {
    delete $browsing_user_requests{browsepub};
    delete $browsing_user_requests{do_public_browseacct};
}
if ($session->{effective_name} ne '') {
    $assumed_identity = 1;
    $effective_user_name = $session->{effective_name};
} elsif ($session->{browse_name} ne '') {
    $browse_public = 1;
    $effective_user_name = $session->{browse_name};
    if (!$browsing_user_requests{$CGIargs{q}}) {
        my $qun = quoteUserName($real_user_name);
        my $qpn = quoteUserName($effective_user_name);
        die("Invalid \"$CGIargs{q}\" transaction attempted by $qun while browsing public account $qpn");
    }
}
my $user_file_name = quoteUserName($user_name);
```

◇

Macro referenced in 179, 192, 196, 206, 208, 211, 214, 228, 231, 234, 235ab, 236, 237, 239, 241, 242, 243, 244, 245, 247, 249, 250a, 251, 261, 278, 279, 287, 295, 297, 300, 301, 304, 306, 307, 309, 314, 316, 318, 320, 323, 327, 329, 331, 341, 348, 353, 357, 360, 363, 365, 367c.

12.1.13.2 Retrieve user account information

Load the account information for the user submitting this request (obtained from the active session identifier, above) into a `user` object named `$ui`.

⟨ Retrieve user account information 194 ⟩ ≡

```
open(FU, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/UserAccount.hdu") ||
    die("Cannot open user account file ⟨ Users Directory 6h ⟩/$user_file_name/UserAccount.hdu");
my $ui = HDiet::user->new();
$ui->load(\*FU);
close(FU);

if ($assumed_identity) {
    if (!$ui->{administrator}) {
        die("Attempt by non-administrator $user_file_name to assume identity");
    }
    $user_name = $effective_user_name;
    $user_file_name = quoteUserName($user_name);
    open(FU, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/UserAccount.hdu") ||
        die("Cannot open effective user account file ⟨ Users Directory 6h ⟩/$user_file_name/UserAccount.hdu")
    $ui->load(\*FU);
    close(FU);
} elsif ($browse_public) {
    my $pn = HDiet::pubname->new();
    if (defined($pn->findPublicName($effective_user_name))) {
        $user_name = $pn->{public_name};
        $user_file_name = quoteUserName($pn->{true_name});
        open(FU, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/UserAccount.hdu") ||
            die("Cannot open effective user account file ⟨ Users Directory 6h ⟩/$user_file_name/UserAccount.hd
        $ui->load(\*FU);
        close(FU);
    } else {
        $browse_public = 0;
    }
}
```

◇

Macro referenced in 179, 196, 206, 208, 211, 214, 228, 231, 234, 235ab, 236, 237, 239, 241, 242, 243, 244, 245, 247, 249, 250a, 251, 261, 278, 279, 287, 295, 297, 300, 301, 304, 306, 307, 309, 314, 316, 318, 320, 323, 327, 329, 331, 341, 348, 353, 357, 360, 363, 365.

12.1.13.3 Sanity check year and month specification

Many transactions include a "`$CGIargs{m}`" argument which specifies the year and month of the log bring processed in ISO-8601 (*YYYY-MM*) format. This specification is turned directly into the name of the file containing the monthly log, so we must be very careful not to allow an abusive specification which might escape the desired directory. The following code, which should be invoked after expansion of a "`now`" specification into the current year and month, if any, validates strict compliance with the syntax and reasonableness for the numerical values. It is permissible to make this check from transactions which do not return HTML results, but it must be made before they write their `Content-type` specification to the result stream.

⟨ Sanity check year and month specification 195 ⟩ ≡

```
        if (!(($CGIargs{m} =~ m/^(\d\d\d\d)\-(\d\d)$/) &&
            ($1 >= 1980) && ($1 <= ((unix_time_to_civil_date_time($userTime))[0] + 1)) &&
            ($2 >= 1) && ($2 <= 12))) {
            if (!$inHTML) {
                if ($ENV{'REQUEST_METHOD'}) {
                    ⟨ MIME Content-type specification 372a ⟩
                }
                $inHTML = 1;
            }
            write_XHTML_prologue($fh, $homeBase, "Create New User Account", undef, $session->{handheld});
            my $qm = quoteHTML($CGIargs{m});
            print $fh <<"EOD";
    <h1 class="c">Invalid Log Date Specification</h1>

    <p class="justified">
    Your request specified an invalid date:
    </p>

    <p class="centred">
    <tt>$qm</tt>
    </p>

    <p class="justified">
    for a monthly log.  Dates must be specified as &ldquo;<i>YYYY</i><tt>-</tt><i>MM</i>&rdquo;.
    </p>


    <h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=account&amp;s=$session->{session_id}$tzOff">Return t
    EOD
            write_XHTML_epilogue($fh, $homeBase);
            last;
        }
    ◇
```

Macro referenced in 197b, 234, 250a.

12.1.14 Display monthly log

The monthly log view is the user's main point of interaction with the application. The current month's log is displayed when the user logs in, and the log page contains links to other components of the application.

⟨ Display monthly log 196 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    ⟨ Determine which monthly log to display 197a ⟩

    ⟨ Read log if in database or create blank log if it's not 197b ⟩

    write_XHTML_prologue($fh, $homeBase,
        "Monthly log for " . $monthNames[$mlog->{month}] . " " . $mlog->{year},
        "setResizeEventHandle();", $session->{handheld});
    generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id},
        (($CGIargs{m} eq $nowmonth) ? "Log" : undef),
        'onclick="return leaveDocument();"', $browse_public, $timeZoneOffset);
    ⟨ Generate assumed identity notification 185 ⟩

    print $fh <<"EOD";
<form id="monthlog" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩
EOD

    my $printFriendly = defined($CGIargs{print}) && $CGIargs{print};
    my $monochrome = defined($CGIargs{mono}) && $CGIargs{mono};
    my $printfix = ($printFriendly ? 'pr_' : '') . ($monochrome ? 'mo_' : '');

    ⟨ Monthly log title and navigation buttons 198 ⟩

    ⟨ Set monthly log property variables 199a ⟩

    my $mlw = $session->{handheld} ? 320 : 640;
    my $mlh = $session->{handheld} ? 240 : 480;

    print $fh <<"EOD";
<div id="canvas" class="canvas"></div>
<p class="trendan">
<script type="text/javascript" src="⟨ Web Document Home 5a ⟩/wz_jsgraphics.js"></script>
<img id="chart" src="⟨ URL to invoke this program 12a ⟩?q=chart&amp;s=$session->{session_id}&amp;m=$CGIargs{m}$mod
     width="$mlw" height="$mlh" alt="Chart for $monthyear" />
<br />
⟨ Generate hidden monthly log property fields 200 ⟩
EOD

    ⟨ Display trend summary below monthly chart 202 ⟩

    print($fh "</p>\n");

    $mlog->toHTML($fh, 1, 31,
        $ui->{display_unit}, $ui->{decimal_character}, $browse_public,
        $printFriendly, $monochrome);

    ⟨ Monthly log control panel 203 ⟩
    ⟨ Dump objects if requested by administrator 205 ⟩
    write_XHTML_epilogue($fh, $homeBase);
◇
```

Macro referenced in 169.

12.1.14.1 Determine which monthly log to display

The choice of the year and month can be made in a variety of ways. In normal navigation, the year and month are passed as "m" and "y" CGI arguments. The initial display of the current log after a user logs in is performed with an argument of "now". Arguments of "new_m" and "new_y" are used when creating a new log from the selection boxes on the Calendar page.

⟨ Determine which monthly log to display 197a ⟩ ≡

```
if (defined($CGIargs{new_y}) && defined($CGIargs{new_m}) &&
    (!defined($CGIargs{m}))) {
    $CGIargs{m} = sprintf("%04d-%02d", $CGIargs{new_y}, $CGIargs{new_m});
}

#   If the date argument is "now", fill in the current year and month
$CGIargs{m} = "now" if !defined($CGIargs{m});
my ($year, $mon, $mday, $hour, $min, $sec) =
    unix_time_to_civil_date_time($userTime);
my $nowmonth = sprintf("%04d-%02d", $year, $mon);
if ($CGIargs{m} eq "now") {
    $CGIargs{m} = $nowmonth;
}
```

◇

Macro referenced in 196.

12.1.14.2 Read log if in database or create blank log if it's not

If a log exists for this month, load it into a monthlog structure. Should no log exist (the user wishes to enter data for a new month), we create an empty log on the fly, plugging in the log weight unit preference from the user object.

⟨ Read log if in database or create blank log if it's not 197b ⟩ ≡

```
⟨ Sanity check year and month specification 195 ⟩

my $mlog = HDiet::monthlog->new();
if (-f "⟨ Users Directory 6h ⟩/$user_file_name/$CGIargs{m}.hdb") {
    open(FL, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/$CGIargs{m}.hdb") ||
        die("Cannot open monthly log file ⟨ Users Directory 6h ⟩/$user_file_name/$CGIargs{m}.hdb");
    $mlog->load(\*FL);
    close(FL);
} else {
    $mlog->{login_name} = $user_name;
    $CGIargs{m} =~ m/(^\d+)\-(\d+)$/;
    my ($yy, $mm) = ($1, $2);
    $mlog->{year} = $yy + 0;
    $mlog->{month} = $mm + 0;
    $mlog->{log_unit} = $ui->{log_unit};
    $mlog->{last_modification_time} = 0;
    $mlog->{trend_carry_forward} = 0;
}
⟨ Fill in trend carry-forward from most recent previous log, if required 201 ⟩
```

◇

Macro referenced in 196, 206, 235b, 236, 250a.

12.1.14.3 Monthly log title and navigation buttons

Above the monthly chart, a title identifying the year and month is displayed with two navigation buttons on either side which select the previous and next month.

⟨ Monthly log title and navigation buttons 198 ⟩ ≡

```
my $monthyear = $monthNames[$mlog->{month}] . " " . $mlog->{year};

my ($lasty, $lastm) = $mlog->previousMonth();
my $slast = sprintf("%04d-%02d", $lasty, $lastm);
my ($slast_link, $slast_button);
my $modeArgs = '';
$modeArgs .= '&amp;print=y' if $CGIargs{print};
$modeArgs .= '&amp;mono=y' if $CGIargs{mono};
if ($slast ne '') {
    $slast_link = "<a class=\"i\" href=\"⟨ URL to invoke this program 12a ⟩?q=log&amp;" .
        "HDiet_tzoffset=$timeZoneOffset&amp;" .
        "s=$session->{session_id}&amp;m=$slast$modeArgs\" onclick=\"return leaveDocument();\">";
    if ($session->{handheld}) {
        $slast_button = "$slast_link<b>&lt;</b></a>";
    } else {
        $slast_button = "$slast_link<img src=\"$homeBase/figures/prev.png\" class=\"b0\" width=\"32\" heig
    }
} else {
    if ($session->{handheld}) {
        $slast_button = "<b>&lt;</b>";
    } else {
        $slast_button = "<img src=\"$homeBase/figures/prev_gr.png\" class=\"b0\" width=\"32\" height=\"32\
    }
}

my ($nexty, $nextm) = $mlog->nextMonth();
my $snext = sprintf("%04d-%02d", $nexty, $nextm);
my ($snext_link, $snext_button);
if ($snext ne '') {
    $snext_link = "<a class=\"i\" href=\"⟨ URL to invoke this program 12a ⟩?q=log&amp;" .
        "HDiet_tzoffset=$timeZoneOffset&amp;" .
        "s=$session->{session_id}&amp;m=$snext$modeArgs\" onclick=\"return leaveDocument();\">";
    if ($session->{handheld}) {
        $snext_button = "$snext_link<b>&gt;</b></a>";
    } else {
        $snext_button = "$snext_link<img src=\"$homeBase/figures/next.png\" class=\"b0\" width=\"32\" heig
    }
} else {
    if ($session->{handheld}) {
        $snext_button = "<b>&gt;</b>";
    } else {
        $snext_button = "<img src=\"$homeBase/figures/next_gr.png\" class=\"b0\" width=\"32\" height=\"32\
    }
}

print($fh "<h1 class=\"${printfix}monthyear\">" .
          $slast_button .
          '   <span>' .
          $monthyear .
          "</span>   $snext_button</h1>\n");
```

◇

Macro referenced in 196.

12.1.14.4 Set monthly log property variables

The following variables are to set to properties of the monthly log and user which will appear in the HTML form we're about to generate. Many of these will be embedded in the hidden variables below which serve to pass them to the JavaScript live update code.

⟨ Set monthly log property variables 199a ⟩ ≡

```
my $mdays = $mlog->monthdays();
my $fracf = $mlog->fractionFlagged();
my $mbmi = $mlog->bodyMassIndex($ui->{height});
my $lbmi = $mlog->bodyMassIndex($ui->{height}, -1);
my $qun = quoteHTML($user_name);
my $t0 = $mlog->{trend_carry_forward} * HDiet::monthlog::WEIGHT_CONVERSION->[$mlog->{log_unit}][$ui->{disp
my @dcalc;
if ($ui->{plot_diet_plan}) {
    @dcalc = $ui->dietPlanLimits();
}

my $iscale = $mlog->computeChartScale(640, 480, $ui->{display_unit}, \@dcalc);

⟨ Define "cachebuster" argument 199b ⟩
```

◇

Macro referenced in 196.

12.1.14.4.1 Define "cachebuster" argument Some browsers and HTTP proxy servers ignore our "`Cache-control`" header item and improperly cache images embedded in pages if their CGI arguments are the same, even though the image may have changed (for example, by adding items to a log). To keep this from happening, we include a "cachebuster" argument named "`qx`" (courtesy of Doc Smith), which is set to a pseudorandom value which will differ on every request. This suffices to keep the miscreants in the middle from caching the image.

⟨ Define "cachebuster" argument 199b ⟩ ≡

```
my $cachebuster = sprintf("%x", (int(rand(65536))) & 0xFFFF);
$cachebuster =~ tr/a-f/FGJKQW/;
```

◇

Macro referenced in 199a, 281b.

12.1.14.5 Generate hidden monthly log property fields

The following hidden input fields serve as fixed arguments to the form and to provide the JaavScript live update code access to quantities it needs to update the log and chart.

⟨ Generate hidden monthly log property fields 200 ⟩ ≡

```
<input type="hidden" name="q" value="update_log" />
<input type="hidden" name="s" value="$session->{session_id}" />
<input type="hidden" name="m" value="$CGIargs{m}" />
<input type="hidden" name="md" id="md" value="$mdays" />
<input type="hidden" name="t0" id="t0" value="$t0" />
<input type="hidden" name="du" id="du" value="$ui->{display_unit}" />
<input type="hidden" name="hgt" id="hgt" value="$ui->{height}" />
<input type="hidden" name="dc" id="dc" value="$ui->{decimal_character}" />
<input type="hidden" name="sc" id ="sc" value="$iscale" />
◇
```

Macro referenced in 196.

12.1.14.6 Fill in trend carry-forward from most recent previous log, if required

No log exists in the database for this month, so we've created a blank log for this month (which won't be added to the database until until and unless the user saves it). Now we need to find the most recent log in the database and fill in its final trend value as the trend carry-forward for this probationary new log. We need to do this now, as opposed to propagating the trend from the previous log when the log is saved, because the trend carry=forward is needed by the JavaScript code to perform live trend updates when the user enters weights.

This code also handles supplying trend carry-forwards for logs in the database which somehow happened to end up with a zero carry-forward. This shouldn't happen, but should some obscure import circumstance or other operation result in such a log making it to the database, this will keep it from confusing the user adding entries to it, and the problem will be corrected when the log is saved.

Note that we must cope with the possibility that the weight unit in the newly created log may differ from that of the most recent log. This happens when the user changes the log unit setting to one different from that of the existing log. In this circumstance, we must convert the trend value to the units of the new log.

⟨ Fill in trend carry-forward from most recent previous log, if required 201 ⟩ ≡

```
        if ($mlog->{trend_carry_forward} == 0) {
            my $cmon = sprintf("%04d-%02d", $mlog->{year}, $mlog->{month});
            my @logs = $ui->enumerateMonths();
            for (my $m = $#logs; $m >= 0; $m--) {
                if ($logs[$m] lt $cmon) {
                    my $llog = HDiet::monthlog->new();
                    open(LL, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/$logs[$m].hdb") ||
                        die("Cannot open previous monthly log file ⟨ Users Directory 6h ⟩/$user_file_name/$logs[$m].ho
                    $llog->load(\*LL);
                    close(LL);
                    for (my $d = $llog->monthdays(); $d >= 1; $d--) {
                        if ($llog->{trend}[$d]) {
                            $mlog->{trend_carry_forward} = $llog->{trend}[$d] *
                                HDiet::monthlog::WEIGHT_CONVERSION->[$llog->{log_unit}][$mlog->{log_unit}];;
                            last;
                        }
                    }
                    last;
                }
            }
        }
```

◇

Macro referenced in 197b.

12.1.14.7 Display trend summary below monthly chart

Below the monthly chart we display a trend analysis and, if the user has specified a height, the body mass index value. The HTML code for this section is complicated by the need to include "`id=` attributes so that the JavaScript live update code can modify these values as log entries are modified.

If we're displaying an historical chart (one older than the most recent in the database), we fit the trend slope based on the data plotted in the chart: what you see is what you get. When plotting the most recent chart (usually the current month's), we create a `history` object and use it to fit a trend for the last week's data (even if this requires retrieving data for the end of the previous month). This avoids discontinuous jumps in the trend analysis at the start of a month, where there are only a few data points to fit, and makes the data plotted in the current chart always agree with the "Last Week" analysis in the Trend Analysis page.

⟨ Display trend summary below monthly chart 202 ⟩ ≡

```
    my $tslope = 0;
    my $hist = HDiet::history->new($ui, $user_file_name);
    my ($ly, $lm, $ld, $ldu, $lw, $lt) = $hist->lastDay();

#print(STDERR "Last day: $ly-$lm-$ld  ($mlog->{year}-$mlog->{month})   Lw $lw   Lt $lt\n");
    if (defined($lw) &&
        ($mlog->{year} == $ly) &&
        ($mlog->{month} == $lm)) {
#print(STDERR "Computed trend the hard way for $ly-$lm-$ld\n");
        my $l_jd = gregorian_to_jd($ly, $lm, $ld);
        my ($s_y, $s_m, $s_d) = $hist->firstDay();
        my $s_jd = gregorian_to_jd($s_y, $s_m, $s_d);

        my (@intervals, @slopes);

        if (($l_jd - $s_jd) > 1) {
            my ($f_y, $f_m, $f_d) = $hist->firstDayOfInterval($ly, $lm, $ld, 7);
            my $f_jd = gregorian_to_jd($f_y, $f_m, $f_d);
            push(@intervals, sprintf("%04d-%02d-%02d", $f_y, $f_m, $f_d),
                             sprintf("%04d-%02d-%02d", $ly, $lm, $ld));
            @slopes = $hist->analyseTrend(@intervals);
            $tslope = $slopes[0];
        }
    } else {
#print(STDERR "Computed trend the easy way for $ly-$lm-$ld\n");
        $tslope = $mlog->computeTrend();
        $tslope *= HDiet::monthlog::WEIGHT_CONVERSION->[$mlog->{log_unit}][$ui->{display_unit}];
    }
    my $sweekly = $ui->localiseDecimal(sprintf("%.2f", abs($tslope) * 7));
    print($fh 'Weekly <span id="delta_sign">' .
            (($tslope > 0) ? "gain" : "loss") .
            "</span> <span id=\"weekly_delta\">$sweekly</span> " .
            $mlog->DELTA_WEIGHT_UNITS->[$ui->{display_unit}] .
            "s.  Daily <span id=\"calorie_sign\">" .
            (($tslope > 0) ? "excess" : "deficit") .
            sprintf("</span>: <span id=\"daily_calories\">%.0f</span> ", abs($tslope) *
                ($mlog->CALORIES_PER_WEIGHT_UNIT->[$ui->{display_unit}] /
                $mlog->CALORIES_PER_ENERGY_UNIT->[$ui->{energy_unit}])) .
            $mlog->ENERGY_UNITS->[$ui->{energy_unit}] . "s" .
            "." .
            (($fracf > 0) ? sprintf("  <span id=\"fracf\" " .
                "style=\"display: inline;\"><span id=\"percent_flagged\">" .
                "%.0f%%</span> flagged.</span>", $fracf * 100) :
                            sprintf("  <span id=\"fracf\" " .
                "style=\"display: none;\"><span id=\"percent_flagged\">" .
                "%.0f%%</span> flagged.</span>", $fracf * 100)));

    if ($mbmi > 0) {
        my ($lmbmi, $llbmi) = ($ui->localiseDecimal($mbmi), $ui->localiseDecimal($lbmi));
        print($fh "\n<span id=\"bmi\" style=\"display: inline;\">" .
```

12.1.14.8 Monthly log control panel

For regular logins, the the monthly log control panel consists of just the "Update" and "Reset" buttons. For administrator logins or access to another account, three checkboxes are displayed which allow the `monthlog`, `user`, and/or `session` objects to be displayed.

⟨ Monthly log control panel 203 ⟩ ≡

```
        if ($browse_public) {
            print $fh <<"EOD";
    </form>
    EOD
        } else {
            my $ckprint = $CGIargs{print} ? ' checked="checked"' : '';
            my $ckmono = $CGIargs{mono} ? ' checked="checked"' : '';

            print $fh <<"EOD";
    <p class="mlog_buttons">
    <input type="submit" value=" Update " />
     
    <input type="reset" onclick="unsavedChanges = 0;" value=" Reset " />
    <br />
    <label><input type="checkbox" name="print" value="y"$ckprint  /> Printer friendly</label>
     
    <label><input type="checkbox" name="mono" value="y"$ckmono  /> Monochrome</label>
    EOD

            ⟨ Administrator object dump selection 204 ⟩

            print $fh <<"EOD";
    </p>
    </form>
    EOD
        }
    ◇
```

Macro referenced in 196.

12.1.14.8.1 Administrator object dump selection By checking one or more of the following checkboxes, the administrator can request a dump of the `monthlog`, `user`, and/or `session` objects.

⟨ Administrator object dump selection 204 ⟩ ≡

```
    if ($ui->{administrator} || $assumed_identity) {
        my $ckdl = $CGIargs{dumplog} ? ' checked="checked"' : '';
        my $ckdu = $CGIargs{dumpuser} ? ' checked="checked"' : '';
        my $ckds = $CGIargs{dumpsession} ? ' checked="checked"' : '';
        my $ckde = $CGIargs{dumpenvironment} ? ' checked="checked"' : '';
        print $fh <<"EOD";
<br />
Dump: <label><input type="checkbox" name="dumplog" value="y"$ckdl  /> Log</label>
      <label><input type="checkbox" name="dumpuser" value="y"$ckdu  /> User</label>
      <label><input type="checkbox" name="dumpsession" value="y"$ckds  /> Session</label>
      <label><input type="checkbox" name="dumpenvironment" value="y"$ckde  /> Environment</label>
EOD
    }
```

◇

Macro referenced in 203.

12.1.14.9 Dump objects if requested by administrator

If the administrator has checked one or more of the object dump boxes at the bottom of the monthly log form (either in the administrator's own account or when accessing another user's account), perform the dump. Note that since the object dump methods write to a file, we need to capture the output in a temporary file and then run it through `quoteHTMLFile` to quote the output for inclusion in an HTML `<pre>` section.

⟨ Dump objects if requested by administrator 205 ⟩ ≡

```
if ($ui->{administrator} || $assumed_identity) {

    sub describeHTML {
        my ($object, $fh, $title) = @_;

        print($fh "<h4>$title</h4>\n") if $title;
        print($fh "<pre style=\"unicode-bidi: bidi-override;\">\n");
        use File::Temp qw(tempfile);
        my $tfh = tempfile();
        binmode($tfh, ":utf8");
        $object->describe($tfh);
        seek($tfh, 0, 0);
        quoteHTMLFile($tfh, $fh);
        close($tfh);
        print($fh "</pre>\n");
    }

    if ($CGIargs{dumplog}) {
        describeHTML($mlog, $fh, "Log");
    }
    if ($CGIargs{dumpuser}) {
        describeHTML($ui, $fh, "User");
    }
    if ($CGIargs{dumpsession}) {
        describeHTML($session, $fh, "Session");
    }
    if ($CGIargs{dumpenvironment}) {
        use Data::Dumper;
        my $denv = Data::Dumper->Dump([\%CGIargs, \%ENV], ['*CGIargs', '*ENV']);
        $denv = quoteHTML($denv);
        print($fh "<h4>Environment</h4>\n");
        print($fh "<pre style=\"unicode-bidi: bidi-override;\">\n");
        print($fh $denv);
        print($fh "</pre>\n");
    }
}
```

◇

Macro referenced in 196.

12.1.15 Update monthly log

The changes made by the user in a monthly log form are applied to the log item in the database, which is written back if any changes were made. Note that this code must handle the case where there is no existing item for this month in the database, as the user may be making the first entries for this month.

An update is logged in the history file as a type 5 transaction, with the extra fields indicating the month updated, and the total number of changes and changes to weight, rung, flag, and comment fields separately. If the month updated is not the most recent in the database (which we detect by comparing its date against the current date), and any weights were changed, then the resulting change to the final trend for the month must ne propagated to subsequent months in the database.

⟨ Update monthly log 206 ⟩ ≡

⟨ Retrieve active session information 193 ⟩

⟨ Retrieve user account information 194 ⟩

⟨ Read log if in database or create blank log if it's not 197b ⟩

```
my ($changes, $change_weight, $change_rung,
    $change_flag, $change_comment) = $mlog->updateFromCGI(\%CGIargs);
```

⟨ Write updated log item back to database 207a ⟩

```
    #   Enqueue a transaction to display the updated log
    %CGIargs = (
        q => "log",
        s => $session->{session_id},
        m => $CGIargs{m},
        dumplog => $CGIargs{dumplog},
        dumpuser => $CGIargs{dumpuser},
        dumpsession => $CGIargs{dumpsession},
        dumpenvironment => $CGIargs{dumpenvironment},
        print => $CGIargs{print},
        mono => $CGIargs{mono},
    );
    next;
```

◇

Macro referenced in 169.

12.1.15.1 Write updated log item back to database

If any changes were made in the log, write it back to the database, note the modification in the user's transaction history log, and update the time of the last transaction from this user.

⟨ Write updated log item back to database 207a ⟩ ≡

```
if (($changes > 0) && (!$readOnly)) {
    $mlog->{last_modification_time} = time();
    open(FL, ">:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/$CGIargs{m}.hdb") ||
        die("Cannot update monthly log file ⟨ Users Directory 6h ⟩/$user_file_name/$CGIargs{m}.hdb");
    $mlog->save(\*FL);
    close(FL);
    clusterCopy("⟨ Users Directory 6h ⟩/$user_file_name/$CGIargs{m}.hdb");

    if ($ui->{badge_trend} != 0) {
        ⟨ Update Web page badge 207b ⟩
    }

    append_history($user_file_name, 5,
        "$CGIargs{m},$changes,$change_weight,$change_rung,$change_flag,$change_comment");

    update_last_transaction($user_file_name);

    if ($change_weight > 0) {
#print("Propagating trend starting at $CGIargs{m}<br />\n");
        propagate_trend($ui, $CGIargs{m}, 0);
    }
}
```

◇

Macro referenced in 206.

12.1.15.1.1 Update Web page badge

When a change is made to a monthly log or to the badge configuration, we re-generate the badge image served to requesters by the `HackDietBadge` program (page 430). The badge is generated in a temporary file and then renamed to the badge file name to avoid race conditions when the badge is being updated and served to a requester at the same time.

⟨ Update Web page badge 207b ⟩ ≡

```
open(FB, ">⟨ Users Directory 6h ⟩/$user_file_name/BadgeImageNew.png") ||
    die("Cannot update monthly log file ⟨ Users Directory 6h ⟩/$user_file_name/BadgeImageNew.png");
my $hist = HDiet::history->new($ui, $user_file_name);
$hist->drawBadgeImage(\*FB, $ui->{badge_trend});
close(FB);
do_command("mv ⟨ Users Directory 6h ⟩/$user_file_name/BadgeImageNew.png ⟨ Users Directory 6h ⟩/$user_file_name/Ba
clusterCopy("⟨ Users Directory 6h ⟩/$user_file_name/BadgeImage.png");
```

◇

Macro referenced in 207a, 231.

12.1.16 Display calendar navigation page

The calendar page provides the primary means of access to the database of monthly logs. For each year with a log in the database, a calendar is generated with links for months in the database which display the log.

⟨ Display calendar navigation page 208 ⟩ ≡

```
    my $calPerLine = 4;             # Calendars per line

    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    my $qun = quoteHTML($user_name);
    write_XHTML_prologue($fh, $homeBase, "Choose Monthly Log", undef, $session->{handheld});
    generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, "History", undef, $browse_public, $t
    ⟨ Generate assumed identity notification 185 ⟩

    $calPerLine = 1 if $session->{handheld};

    print $fh <<"EOD";
<h1 class="c">Choose Monthly Log</h1>
<table class="list_of_calendars">
EOD

    my ($intr, $calsline) = (0, 0);

    my @years = $ui->enumerateYears();

    ⟨ Generate table of yearly calendars 209 ⟩

    if ($intr) {
        print($fh "</tr>\n");
    }

    print $fh <<"EOD";
</table>
EOD

    if (!$browse_public) {
        ⟨ New monthly log creation form 210 ⟩
    }

    write_XHTML_epilogue($fh, $homeBase);

    update_last_transaction($user_file_name);
◇
```

Macro referenced in 169.

12.1.16.1 Generate table of yearly calendars

Having obtained a list of all the years for which logs exist in the database, we iterate over the years and generate tables for all years which have one or more monthly logs in the database. This is all wrapped into an XHTML table with `$calPerLine` yearly calendars per row.

⟨ Generate table of yearly calendars 209 ⟩ ≡

```
        for (my $y = 0; $y <= $#years; $y++) {

            if (!$intr) {
                print($fh "<tr>\n");
                $intr = 1;
                $calsline = 0;
            }

            if ($calsline >= $calPerLine) {
                print($fh "</tr><tr>\n");
                $calsline = 0;
            }

            print $fh <<"EOD";
    <td><table class="calendar" border="border">
    <tr><th colspan="3">$years[$y]</th></tr>
    EOD
            my @months = $ui->enumerateMonths($years[$y]);
            my $m = 0;
            for (my $i = 0; $i < 4; $i++) {
            print $fh <<"EOD";
        <tr>
    EOD
                for (my $j = 0; $j < 3; $j++) {
                    $m++;
                    print($fh "        <td>");
                    my $ym = sprintf("%04d-%02d", $years[$y], $m);
                    my $havemonth = 0;
                    for (my $k = 0; $k <= $#months; $k++) {
                        if ($months[$k] eq $ym) {
                            print($fh "<a href=\"⟨ URL to invoke this program 12a ⟩?s=$session->{session_id}&amp;q=log
                            $havemonth = 1;
                            last;
                        }
                    }
                    print($fh substr($monthNames[$m], 0, 3));
                    if ($havemonth) {
                        print($fh "</a>");
                    }
                    print($fh "</td>\n");
                }
            print $fh <<"EOD";
        </tr>
    EOD
            }
            print $fh <<"EOD";
    </table></td>
    EOD
            $calsline++;
        }
    ◇
```

Macro referenced in 208.

12.1.16.2 New monthly log creation form

Generate an HTML form which allows the user to create a new monthly log. All this actually does is invoke the monthly log display transaction with the specified year and month; the monthly log form already knows how to create blank logs for months not present in the database.

⟨ New monthly log creation form 210 ⟩ ≡

```
    print $fh <<"EOD";
<form id="Hdiet_create_new_monthly_log" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩
<p>
<input type="hidden" name="q" value="log" />
<input type="hidden" name="s" value="$session->{session_id}" />
<b>Create/display log for:</b>
<select name="new_m" id="new_m">
EOD

    my ($year, $mon, $mday, $hour, $min, $sec) =
        unix_time_to_civil_date_time($userTime);
    for (my $i = 1; $i <= 12; $i++) {
        my $sel = ($i == $mon) ? ' selected="selected"' : '';
        print($fh "<option value=\"$i\"$sel>$monthNames[$i]</option>\n")
    }

    print $fh <<"EOD";
</select>

<select name="new_y" id="new_y">
EOD

    for (my $y = $year; $y >= 1985; $y--) {
        print($fh "<option>$y</option>\n")
    }

    print $fh <<"EOD";
</select>

<label title="Create/display log for the specified month and year"><input type="submit" value=" Create Log " /
</p>
</form>
EOD
◇
```

Macro referenced in 208.

12.1.17 Display CSV import request form

A user can import log items into a database from CSV files in either the form created by saving an Excel weight log or those written by `hdread` from Palm database backups. The CSV file can either be uploaded from a file on the local computer or pasted into a text box on the import request form itself.

⟨ Display CSV import request form 211 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    write_XHTML_prologue($fh, $homeBase, "Import CSV or XML Database", undef, $session->{handheld});
    generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $timeZ
    ⟨ Generate assumed identity notification 185 ⟩

    print $fh <<"EOD";
<h1 class="c">Import CSV or XML Database</h1>

<p class="justified">
You can import log entries from CSV files either saved from Excel
logs or exported from a backup of a Palm database with the
<tt>hdread</tt> program.  You can also import XML database backups
from this application.  Logs in any format can either be uploaded from a
file on your computer or simply pasted into the text box below.
</p>

<p class="justified">
Normally, log entries from files you import will not overwrite
existing entries in the online database; if one or more fields in
a daily entry are nonblank, they will not be replaced by the
contents of a record for the same day in the imported file.  If you wish
to have records imported from the file override existing records,
check the &ldquo;Allow overwrite&rdquo; box in the import request.
</p>

<div>

    ⟨ CSV file upload import form 212 ⟩

<br />

    ⟨ CSV direct upload import form 213a ⟩
<br />
</div>

<h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=account&amp;s=$session->{session_id}$tzOff">Back to
EOD

    write_XHTML_epilogue($fh, $homeBase);

    update_last_transaction($user_file_name);
◇
```

Macro referenced in 169.

12.1.17.1 CSV file upload import form

The fields in this fieldset allow the user to specify a local file containing CSV or XML to be imported.

⟨ CSV file upload import form 212 ⟩ ≡

```
<fieldset id="Hdiet_CSV_upload_fs"><legend>Import CSV or XML by File Upload</legend>
<form id="Hdiet_CSV_upload" enctype="multipart/form-data"
    method="post" action="⟨ URL to invoke this program 12a ⟩?enc=raw$tzOff">
⟨ Local time zone offset field 372b ⟩
  <p>
    <label title="Choose a Local CSV or XML File to Upload and Import" for="uploaded_file">Local File:</label>
    <input type="file" id="uploaded_file" name="uploaded_file" size="30" />
    <input type="hidden" name="q" value="csv_import_data" />
    <input type="hidden" name="s" value="$session->{session_id}" />
    <label title="Upload and import CSV or XML file"><input type="submit" value=" Import " /></label>
    <br />
    <label><input type="checkbox" name="overwrite" value="y" /> Allow overwrite</label>
    <label><input type="checkbox" name="listimp" value="y" /> List imported records</label>
  </p>
</form>

<p>
Select the file you wish to upload and import.
</p>

</fieldset>
◇
```

Macro referenced in 211.

12.1.17.2 CSV direct upload import form

The fields in this fieldset allow the user to directly paste CSV or XML into a `textarea` box, which is sent with the form to be imported.

⟨ CSV direct upload import form 213a ⟩ ≡

```
<fieldset class="front" id="Hdiet_CSV_submit_fs"><legend>Import Pasted CSV or XML Log Entries</legend>
<form id="Hdiet_CSV_submit" enctype="multipart/form-data"
    method="post" action="⟨ URL to invoke this program 12a ⟩">
⟨ Local time zone offset field (213b 1 ) 372b ⟩
<p>Paste the CSV or XML you wish to import in the text area below:</p>
<p>
    <label title="Paste the CSV or XML log entries here" for="file">
    <textarea cols="75" rows="12" name="file" id="file"></textarea></label><br />
    <input type="hidden" name="q" value="csv_import_data" />
    <input type="hidden" name="s" value="$session->{session_id}" />
    <label title="Import CSV or XML log entries"><input type="submit" value=" Import " /></label>
    <label title="Clear the entry field"><input type="reset" value=" Clear " /></label>
    <label><input type="checkbox" name="overwrite" value="y" /> Allow overwrite</label>
    <label><input type="checkbox" name="listimp" value="y" /> List imported records</label>
</p>
</form>
</fieldset>
◇
```

Macro referenced in 211.

12.1.18 Import uploaded CSV log entries

This task is invoked when the user uploads log entries in CSV or XML format. We automatically determine whether they were in XML, exported from an Excel log, written by `hdread`, or exported in CSV from this application and parse them accordingly. Note that for this determination to be correct, it is essential that the user include all the header lines in the CSV. We make no assumptions about the order of log items imported, but the header lines must be at the top.

⟨ Import uploaded CSV log entries 214 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    write_XHTML_prologue($fh, $homeBase, "Database Imported", undef, $session->{handheld});
    generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $timeZ
    ⟨ Generate assumed identity notification 185 ⟩

    print $fh <<"EOD";
<h1 class="c">Database Imported</h1>
<form id="Hdiet_CSV_import_confirmation" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩

<p class="justified">
The submitted log items have been processed as follows.
</p>

<p class="mlog_buttons">
<input type="hidden" name="q" value="account" />
<input type="hidden" name="s" value="$session->{session_id}" />
<input type="submit" name="account" value=" Return to Account Page " />
</p>
</form>

EOD
◇
```

Macro defined by 214, 215ab, 216.
Macro referenced in 169.

Initialise variables for the database import process.

⟨ Import uploaded CSV log entries 215a ⟩ ≡

```
        my $listStyle;
        my ($imp, $over);

        my $csv = HDiet::Text::CSV->new();
        my ($n, $imported, $noparse, $already, $notentry) = (0, 0, 0, 0, 0);
        my (%mondb, %monchanges);

        my $overwrite = defined($CGIargs{overwrite});
        my $listCSV = defined($CGIargs{listimp});

        #   Set log format and weight unit unknown
        my ($logFormat, $csvUnit, $hdOnlineLog) = ('Unknown', -1, 0);
    ◇
```

Macro defined by 214, 215ab, 216.
Macro referenced in 169.

Import the records, producing an annotated list of items imported if requested.

⟨ Import uploaded CSV log entries 215b ⟩ ≡

```
        if ($listCSV) {
            print $fh <<"EOD";
    <pre>
    EOD
        }

        if ($CGIargs{file} =~ m/\s*<\?xml\s+/) {
            ⟨ Import log items from XML database file 217 ⟩
        } else {
            ⟨ Import log items from CSV database file 220 ⟩
        }

        ⟨ Write back logs modified by database import 226b ⟩

        if ($listCSV) {
            print $fh <<"EOD";
    </pre>
    EOD
        }

        ⟨ Append summary of records imported 227 ⟩
        write_XHTML_epilogue($fh, $homeBase);

    ◇
```

Macro defined by 214, 215ab, 216.
Macro referenced in 169.

Add a history record summarising the records imported and update the user's last transaction time.

⟨ Import uploaded CSV log entries 216 ⟩ ≡

```
        if (!$readOnly) {
            my $histrec = "$logFormat,$overwrite,$imported,$notentry,$already,$noparse";
            foreach $md (sort(keys(%mondb))) {
                if ($monchanges{$md} > 0) {
                    $histrec .= ",$md,$monchanges{$md}";
                }
            }
            append_history($user_file_name, 7, "$histrec");

            update_last_transaction($user_file_name);
        }
```

◇

Macro defined by 214, 215ab, 216.
Macro referenced in 169.

12.1.18.1 Import log items from XML database file

This appears to be an XML database export file. Parse it with the XML parser, then walk through the document tree and import the log entries it contains.

⟨ Import log items from XML database file 217 ⟩ ≡

```
my $parser = XML::LibXML->new();
my $doc = $parser->parse_string($CGIargs{file});
my $root = $doc->getDocumentElement();

my $indent = '';

my %logItem;
my ($logYear, $logMonth);

$logFormat = 'XML';
parseDOMTree($root, '');

#   For node name mnemonics see:
#       /usr/lib/perl5/vendor_perl/5.8.8/i386-linux-thread-multi/XML/LibXML/Common.pm
sub parseDOMTree {
    my ($elem, $parent) = @_;

    if ($elem->nodeType() == TEXT_NODE) {
        my $v = $elem->nodeValue();
        if ($v !~ m/^\s*$/) {
            if (($parent eq 'log-unit') || ($parent eq 'weight-unit')) {
                $csvUnit = WEIGHT_KILOGRAM if ($elem->nodeValue() eq 'kilogram');
                $csvUnit = WEIGHT_POUND if ($elem->nodeValue() eq 'pound');
                $csvUnit = WEIGHT_STONE if ($elem->nodeValue() eq 'stone');
            } elsif ($parent eq 'year') {
                $logYear = $elem->nodeValue();
            } elsif ($parent eq 'month') {
                $logMonth = $elem->nodeValue();
            } elsif ($parent =~ m/(date|weight|rung|flag|comment)/) {
                $logItem{$parent} = $elem->nodeValue();
            }
        }
    } elsif ($elem->nodeType() == ELEMENT_NODE) {
        if ($elem->nodeName() eq 'day') {
            %logItem = ();                  # Clear log item fields
        }
    }
    if ($elem->hasChildNodes()) {
        my @kids = $elem->getChildNodes();
        for my $kid (@kids) {
            $indent .= '  ';
            parseDOMTree($kid, $elem->nodeName());
            $indent =~ s/  //;
        }
    }

    if (($elem->nodeType() == ELEMENT_NODE) &&
        ($elem->nodeName() eq 'day')) {
        ⟨ Process XML daily log entry 218 ⟩
    }
}
```

◇

Macro referenced in 215b.

12.1.18.1.1 Process XML daily log entry All of the fields for a log entry are children of the `day` element, which we process here. If we're making a listing, we generate a pseudo-record solely to be listed.

⟨ Process XML daily log entry 218 ⟩ ≡

```
        #   Sanity check date before proceeding
        if (($logYear >= 1980) &&  ($logYear <= ((unix_time_to_civil_date_time($userTime))[0]))) {

            my $monkey = sprintf("%04d-%02d", $logYear, $logMonth);
            my ($yy, $mm) = ($logYear, $logMonth);
            ⟨ Load or create monthly log containing imported record 223 ⟩

            #   Test whether an entry already exists for this day

            if ((!$overwrite) && ($mlog->{weight}[$logItem{date}] || $mlog->{rung}[$logItem{date}] ||
                    $mlog->{flag}[$logItem{date}] || $mlog->{comment}[$logItem{date}])) {
                $listStyle = 'conflict';
                $already++;
            } else {
                $mlog->{weight}[$logItem{date}] = ($logItem{weight} *
                        WEIGHT_CONVERSION->[$csvUnit][$mlog->{log_unit}])
                    if defined($logItem{weight});
                $mlog->{rung}[$logItem{date}] = $logItem{rung}
                    if defined($logItem{rung});
                $mlog->{flag}[$logItem{date}] = $logItem{flag}
                    if defined($logItem{flag});
                $mlog->{comment}[$logItem{date}] = $logItem{comment};

                $monchanges{$monkey}++;
                $listStyle = 'imported';
                $imported++;
           }
        } else {
            $listStyle = 'noparse';
            $noparse++;
        }

        $n++;
        if ($listCSV) {
            my $listline =  quoteHTML(sprintf("%04d-%02d-%02d  %4.1f  %3d  %1d  %s",
                $logYear, $logMonth, $logItem{date},
                defined($logItem{weight}) ? $logItem{weight} : 0,
                defined($logItem{rung}) ? $logItem{rung} : 0,
                defined($logItem{flag}) ? $logItem{flag} : 0,
                defined($logItem{comment}) ? $logItem{comment} : ''));
            printf("<span class=\"$listStyle\">%4d.  %s</span>\n", $n, $listline);
        }
```

◇

Macro referenced in 217.

12.1.18.2 Dump XML database file

Output an interpreted dump of the uploaded XML database file to standard output. This is used for debugging XML parsing and interpretation.

⟨ Dump XML database file 219 ⟩ ≡

```
my $parser = XML::LibXML->new();
my $doc = $parser->parse_string($CGIargs{file});
my $root = $doc->getDocumentElement();

my $indent = '';

guzz($root);

#   For node name mnemonics see:
#       /usr/lib/perl5/vendor_perl/5.8.8/i386-linux-thread-multi/XML/LibXML/Common.pm
sub guzz {
    my $elem = shift();
    if ($elem->nodeType() == TEXT_NODE) {
        my $v = $elem->nodeValue();
        if ($v !~ m/^\s*$/) {
            print($indent . "Txt: " . $elem->nodeName() . "  " . $elem->nodeValue(), "\n");
        }
    } elsif ($elem->nodeType() == ELEMENT_NODE) {
        print($indent . "Elt: " . $elem->nodeName() . "\n");
        if ($elem->hasAttributes()) {
            my @attrs = $elem->attributes();
            for my $a (@attrs) {
                print($indent . "Atr: " . $a->nodeName() . "  " . $a->nodeValue(), "\n");
            }
        }
    }
    if ($elem->hasChildNodes()) {
        my @kids = $elem->getChildNodes();
        for my $kid (@kids) {
            $indent .= '  ';
            guzz($kid);
            $indent =~ s/  //;
        }
    }
}
```

◇

Macro never referenced.

12.1.18.3 Import log items from CSV database file

The file did not begin with an XML header, so we conclude it must be in CSV format. Attempt to parse in the assorted CSV formats we support.

⟨ Import log items from CSV database file 220 ⟩ ≡

```
while ($CGIargs{file} =~ s/^([^\n]*\r?\n)//s) {
    $n++;

    my $l = $1;
    my $listline = $l;
    $listline =~ s/\s+$//;
    $listStyle = 'noparse';
    if (($listline ne '') && $csv->parse($l)) {
        $listStyle = 'notentry';
        my @f = $csv->fields();
        $imp = 0;
        $over = 0;

        ⟨ Check for Excel CSV record 221 ⟩

        if (!($imp || $over)) {
            ⟨ Check for Palm/HDREAD CSV record 225 ⟩
        }

        if ($imp) {
            $listStyle = 'imported';
            $imported++;
        }

        if ($listStyle eq 'notentry') {
            $notentry++;
        }
    } else {
        $listStyle = 'noparse';
        $noparse++;
    }

    if ($listCSV) {
        $listline = quoteHTML($listline);
        printf("<span class=\"$listStyle\">%4d.  %s</span>\n", $n, $listline);
    }
}
```

◇

Macro referenced in 215b.

12.1.18.4 Check for Excel CSV record

Test if this record is in the format of the CSV export of an Excel yearly log. If so, determine whether it is a the header line specifying the log units and set `$csvUnit` accordingly. If this is a non-void daily log entry, extract the fields and import them into the database.

⟨ Check for Excel CSV record 221 ⟩ ≡

```
    my $excelCSVdebug = 0;
    if ($listline =~ m/^Date,,Weight,Trend,Variance,,Rung,Flag$/) {
        $logFormat = 'Excel';
#print("Set format Excel\n") if $excelCSVdebug;
    } elsif (($logFormat eq 'Excel') && ($csvUnit < 0) &&
             ($listline =~ m/^,,(\w+),,\w+\s+\d+,,,$/)) {
        my $wunit = $1;

        $csvUnit = WEIGHT_KILOGRAM if ($wunit =~ m/^Kilograms/i) || ($wunit eq 0);
        $csvUnit = WEIGHT_POUND if ($wunit =~ m/^Pounds/i) || ($wunit eq 1);
        $csvUnit = WEIGHT_STONE if ($wunit =~ m/^Stones/i) || ($wunit eq -1);
#print("Setunit $csvUnit\n") if $excelCSVdebug;
    } else {
#print("       Parsed($#f) 0($f[0])  1($f[1])   2($f[2])   3($f[3])   4($f[4])   5($f[5])   6($f[6])  7($f[7]
        if (($#f >= 5) &&
            ($f[0] =~ m/^\d+[\/\-\.]\d+[\/\-\.]\d+$/) &&  # Date
            ($f[1] =~ m/^[a-z]+$/i) &&                    # Day of week
            ($f[2] =~ m/^[\p{IsWord}\s\.]+$/) &&          # Weight
            ($f[3] =~ m/^[\d\.]+$/) &&                    # Trend
            ($f[4] =~ m/^\-?[\d\.]*$/) &&                 # Variance
            ($f[5] =~ m/^[\d\.]*$/) &&                    # Hidden carry-forward
            ($f[6] =~ m/^\s*\d*$/)) {                     # Exercise rung
#print("        Import ($f[0])  ($f[2])   ($f[6])  ($f[7])\n") if $excelCSVdebug;
            my ($date, $weight, $rung) = ($f[0], $f[2], $f[6]);
            $rung =~ s/\s//g;
            my $flag = $f[7] ? 1 : 0;
            my $comment = defined($f[8]) ? $f[8] : '';

            #   See if the first field is something we can interpret plausibly as a date
            $f[0] =~ m/^(\d+)([\/\-\.])(\d+)([\/\-\.])(\d+)$/;
            if ($2 eq $4) {
                ⟨ Parse Excel CSV date field 222 ⟩

                #   Sanity check date before proceeding
                if (($yy >= 1980) &&  ($yy <= ((unix_time_to_civil_date_time($userTime))[0]))) {
                    my $monkey = sprintf("%04d-%02d", $yy, $mm);

                    ⟨ Load or create monthly log containing imported record 223 ⟩

                    #   Test whether an entry already exists for this day

                    if ((!$overwrite) && ($mlog->{weight}[$dd] || $mlog->{rung}[$dd] ||
                            $mlog->{flag}[$dd] || $mlog->{comment}[$dd])) {
                        $listStyle = 'conflict';
                        $already++;
                        $over = 1;
                    } else {
                        ⟨ Set monthly log entry from Excel CSV record 224 ⟩
                    }
                } else { print ("ExcelBarfel: $yy-$mm-$dd\n") if $excelCSVdebug; }
            } else { print ("ExcelGarfel: $1 $2 $3 $4 $5 $6\n") if $excelCSVdebug; }
        }
    }
```

◇

Macro referenced in 220.

12.1.18.4.1 Parse Excel CSV date field We accept three different date formats in Excel CSV: *YYYY-MM-DD*, *MM/DD/YYYY*, and *DD.MM.YYYY*. In any of these, the year may be given as a two digit quantity to which 1900 is added if 89 or greater and 2000 otherwise.

⟨ Parse Excel CSV date field 222 ⟩ ≡

```
my ($yy, $mm, $dd);
if ($2 eq '-') {            # YYYY-MM-DD
    $yy = $1 + 0;
    $mm = $3 + 0;
    $dd = $5 + 0;
} elsif ($2 eq '/') {       # MM/DD/YYYY
    $yy = $5 + 0;
    $mm = $1 + 0;
    $dd = $3 + 0;
} elsif ($2 eq '.') {       # DD.MM.YYYY
    $yy = $5 + 0;
    $mm = $3 + 0;
    $dd = $1 + 0;
}

#   Kludge for two digit years
if ($yy < 100) {
    if ($yy > 88) {
        $yy += 1900;
    } else {
        $yy += 2000;
    }
}
```

◇

Macro referenced in 221.

12.1.18.4.2 Load or create monthly log containing Excel CSV record If the log containing the record is already in the `%mondb` cache, return it. Otherwise, load it from the database or create a new blank log if this is a month not present in the database.

⟨ Load or create monthly log containing imported record 223 ⟩ ≡

```
my $mlog;

if (defined($mondb{$monkey})) {
    $mlog = $mondb{$monkey};
} else {
    $mlog = HDiet::monthlog->new();
    $mondb{$monkey} = $mlog;
    $monchanges{$monkey} = 0;
    if (-f "⟨ Users Directory 6h ⟩/$user_file_name/$monkey.hdb") {
        open(FL, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/$monkey.hdb") ||
            die("Cannot open monthly log file ⟨ Users Directory 6h ⟩/$user_file_name/$monkey.hdb");
        $mlog->load(\*FL);
        close(FL);
    } else {
        $mlog->{year} = $yy;
        $mlog->{month} = $mm;
        $mlog->{log_unit} = $ui->{log_unit};
        $mlog->{trend_carry_forward} = 0;
        $mlog->{last_modification_time} = 0;
    }
}
```

◇

Macro referenced in 218, 221, 225.

12.1.18.4.3 Set monthly log entry from Excel CSV record Store the parsed fields from the Excel CSV record into the indicated date in the monthly log. Note that in Excel logs a comment can appear either in the weight field or in column 8 along with a valid weight field.

⟨ Set monthly log entry from Excel CSV record 224 ⟩ ≡

```
my $cmt = '';
if ($f[2] !~ m/^[\d\.]+$/) {
    $cmt = $f[2];
    $f[2] = '';
}
$cmt = $f[8] if defined($f[8]) && $f[8] ne '';

$mlog->{weight}[$dd] = ($f[2] * WEIGHT_CONVERSION->[$csvUnit][$mlog->{log_unit}]) if $f[2] ne '';
$mlog->{rung}[$dd] = $f[6] if $f[6] ne '';
$mlog->{flag}[$dd] = '1' if $f[7] ne '';
$mlog->{flag}[$dd] = undef if (!$f[7]) || ($f[7] eq '0') ||
    ($f[7] eq '') || ($f[7] =~ m/^\s*$/);
$mlog->{comment}[$dd] = $cmt if $cmt ne '';
if ($cmt eq '') {
    undef $mlog->{comment}[$dd];
}

$monchanges{$monkey}++;
$imp = 1;
```

◇

Macro referenced in 221.

12.1.18.5 Check for Palm/HDREAD CSV record

Records which appear to be in HDREAD format may have come from HDREAD or have been exported from this application. The only difference occurs in the comments field, which may contain escaped UTF-8 characters exported from an online monthly log. HDREAD-generated files contain only ISO 8859-1 characters and use the Excel quoting convention. We detect records we've exported by the presence of a CSV version number on the "`StartTrend`" record and set the `$hdOnlineLog` flag, which causes records to be imported with the `importCSV` method of the monthly log.

⟨ Check for Palm/HDREAD CSV record 225 ⟩ ≡

```
if ($listline =~ m/^Date,Weight,Rung,Flag,Comment$/) {
    $logFormat = 'HDRead';
} elsif (($logFormat eq 'HDRead') &&
         ($#f >= 4) &&
         ($f[0] eq 'StartTrend') &&
         ($f[2] >= WEIGHT_KILOGRAM) && ($f[2] <= WEIGHT_STONE)) {

    $csvUnit = $f[2];
    $hdOnlineLog = 0;
    if (($#f >= 5) && ($f[5] =~ m/^[\d\.]+$/)) {
        $hdOnlineLog = 1;
    }
} else {
    $f[0] =~ s/\s//g if defined($f[0]);
    $f[1] =~ s/\s//g if defined($f[1]);
    $f[2] =~ s/\s//g if defined($f[2]);
    $f[3] =~ s/\s//g if defined($f[3]);
    if (($#f >= 4) &&
        ($f[0] =~ m/^\d+\-\d+\-\d+$/) &&        # Date
        ($f[1] =~ m/^[\d\.]*$/) &&              # Weight
        ($f[2] =~ m/^\d*$/)) {                  # Exercise rung
        my ($date, $weight, $rung) = ($f[0], $f[1], $f[2]);
        my $flag = $f[3] ? 1 : 0;
        my $comment = defined($f[4]) ? $f[4] : '';

        #   See if the first field is an ISO 8601 date
        $f[0] =~ m/^(\d+)\-(\d+)\-(\d+)$/;
        my ($yy, $mm, $dd) = ($1, $2, $3);

        #   Sanity check date before proceeding
        if (($yy >= 1980) &&  ($yy <= ((unix_time_to_civil_date_time($userTime))[0]))) {

            my $monkey = sprintf("%04d-%02d", $yy, $mm);

            ⟨ Load or create monthly log containing imported record 223 ⟩

            #   Test whether an entry already exists for this day

            if ((!$overwrite) && ($mlog->{weight}[$dd] || $mlog->{rung}[$dd] ||
                    $mlog->{flag}[$dd] || $mlog->{comment}[$dd])) {
                $listStyle = 'conflict';
                $already++;
            } else {
                ⟨ Set monthly log entry from Palm CSV record 226a ⟩
            }
        }# else { print ("PalmBarfel: $yy-$mm-$dd\n"); }
     }# else { print ("PalmGarfel: ($f[0])  ($f[1])   ($f[2])  ($f[3]) ($f[4])\n"); }
 }
```

◇

Macro referenced in 220.

12.1.18.5.1 Set monthly log entry from Palm CSV record The CSV record has passed all the tests, so now nothing remains but to actually import it. If this is a record we have exported (see the above section for details on how that is detected), we simply pass it to the `importCSV` method of the monthly log which contains the record. Otherwise, we set the fields in the log entry directly from the values we've parsed from the record.

⟨ Set monthly log entry from Palm CSV record 226a ⟩ ≡

```
        if ($hdOnlineLog) {
            if ($mlog->importCSV($listline)) {
                $monchanges{$monkey}++;
                $imp = 1;
           }
        } else {
            $mlog->{weight}[$dd] = ($f[1] *
                WEIGHT_CONVERSION->[$csvUnit][$mlog->{log_unit}]) if $f[1] ne '';
            $mlog->{rung}[$dd] = $f[2] if $f[2] ne '';
            $mlog->{flag}[$dd] = $flag;
            $mlog->{comment}[$dd] = $comment;
            if ($comment eq '') {
                undef $mlog->{comment}[$dd];
            }
        }

        $monchanges{$monkey}++;
        $imp = 1;
    ◇
```

Macro referenced in 225.

12.1.18.6 Write back logs modified by database import

Any logs which have been modified as a result of the database import are written back to the database. The last modification time in each log is sent to the current time.

⟨ Write back logs modified by database import 226b ⟩ ≡

```
        my $md;
        if (!$readOnly) {
            foreach $md (sort(keys(%mondb))) {
                if ($monchanges{$md} > 0) {
                    $mondb{$md}->{last_modification_time} = time();
                    open(FL, ">:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/$md.hdb") ||
                        die("Cannot open monthly log file ⟨ Users Directory 6h ⟩/$user_file_name/$md.hdb");
                    $mondb{$md}->save(\*FL);
                    close(FL);
                    clusterCopy("⟨ Users Directory 6h ⟩/$user_file_name/$md.hdb");
                }
            }
        }
    ◇
```

Macro referenced in 215b.

12.1.18.7 Append summary of records imported

Generate a legend in the results page which shows the disposition of the lines imported: added to the database, ignored to avoid overwriting existing data, and skipped due to syntax errors or not being a daily log item.

⟨ Append summary of records imported 227 ⟩ ≡

```
        print($fh "<p>\n
    Records submitted: $n.<br />\n
    Log items imported: $imported.<br />\n");

        print($fh "<span class=\"notentry\">Records ignored as not daily log entries: $notentry.</span><br />\n")
            if $notentry > 0;
        print($fh "<span class=\"conflict\">Records skipped to avoid overwriting existing entries: $already.</spar
            if $already > 0;
        print($fh "<span class=\"noparse\">Records discarded due to parsing errors: $noparse.</span><br />\n")
            if $noparse > 0;
        print($fh "</p>\n");
    ◇
```

Macro referenced in 215b.

12.1.19 Configure Web page status badge

This form allows the user to enable the generation of a Web page "badge" image which shows the most recent weight entry and the energy balance and weight gain/loss trend for a user-specified interval. When the form is submitted a `update_badge` transaction is submitted to apply the changes.

⟨ Configure Web page status badge 228 ⟩ ≡

```
⟨ Retrieve active session information 193 ⟩

⟨ Retrieve user account information 194 ⟩

write_XHTML_prologue($fh, $homeBase, "Configure Web Page Status Badge", undef, $session->{handheld});
generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $timeZ
⟨ Generate assumed identity notification 185 ⟩

my @cterm;
$cterm[0] = $cterm[7] = $cterm[14] = $cterm[1] = $cterm[3] = $cterm[6] = $cterm[12] = '';
$cterm[abs($ui->{badge_trend})] = ' selected="selected"';

print $fh <<"EOD";
<h1 class="c">Configure Web Page Status Badge</h1>

<p class="centred">
<img src="$homeBase/figures/badge_sample.png"
    width="200" height="78"
    alt="Sample Web status badge" />
</p>

<p class="justified">
A Web badge is a small image like the example above which you can add to
your personal Web page or Web log which shows, as of the
most recent log entry, your weight, daily energy (calorie or kilojoule)
balance, and your present weekly rate of weight loss or gain based
upon fitting a linear trend to the trend values for the interval
chosen below.
</p>

<form id="Hdiet_badgeconf" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩
EOD
◇
```

Macro defined by 228, 229, 230.
Macro referenced in 169.

12.1.19.1 Interval selection

A `select` box allows the user to specify the interval over which the trend shown in the badge is to be computed.

⟨ Configure Web page status badge 229 ⟩ ≡

```
        print $fh <<"EOD";
    <p class="mlog_buttons">
    <select name="badge_term" id="badge_term"
        onchange="change_badge_term();">
        <option value="0"$cterm[0]>Disable badge</option>
        <option value="7"$cterm[7]>Week</option>
        <option value="14"$cterm[14]>Fortnight</option>
        <option value="-1"$cterm[1]>Month</option>
        <option value="-3"$cterm[3]>Quarter</option>
        <option value="-6"$cterm[6]>Six months</option>
        <option value="-12"$cterm[12]>Year</option>
    </select>
    </p>

    <p class="justified">
    After choosing the interval over which you wish the trend to be
    computed, press the &ldquo;Apply&rdquo; button below.  You'll
    be taken to a confirmation page which includes HTML/XHTML code
    you can cut and paste into your Web page to display the badge.
    If you select &ldquo;Disable badge&rdquo;, badge generation will
    be disabled and any existing badge image deleted; if you disable
    badge generation, be sure to remove the badge image from your Web
    page, as otherwise visitors will see an &ldquo;Invalid request&rdquo;
    icon instead of the badge.
    </p>

    EOD
    ◇
```

Macro defined by 228, 229, 230.
Macro referenced in 169.

12.1.19.2 Form action buttons

The following buttons appear at the bottom of the badge configuration form.

⟨ Configure Web page status badge 230 ⟩ ≡

```
    print $fh <<"EOD";
<p class="mlog_buttons">
<input type="hidden" name="s" value="$session->{session_id}" />
<input type="submit" name="q=update_badge" value=" Apply " />
 
<input type="reset" value=" Reset " />
 
<input type="submit" name="q=account" value=" Cancel " />
</p>
</form>
EOD

    write_XHTML_epilogue($fh, $homeBase);
◇
```

Macro defined by 228, 229, 230.
Macro referenced in 169.

12.1.20 Update Web page status badge

When an `update_badge` transaction is received, this code updates the `user` object with the new badge trend interval (zero if badge generation is disabled) and updates the badge (deleting it if generation has been disabled). A `textbox` containing the XHTML to be copied into a page to display the badge is supplied to the user if badge generation is enabled.

⟨ Update Web page status badge 231 ⟩ ≡

```
⟨ Retrieve active session information 193 ⟩

⟨ Retrieve user account information 194 ⟩

write_XHTML_prologue($fh, $homeBase, "Web Page Status Badge Configuration Changed", undef, $session->{hand
generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $timeZ
⟨ Generate assumed identity notification 185 ⟩

$CGIargs{badge_term} = '0' if !defined($CGIargs{badge_term});

$ui->{badge_trend} = $CGIargs{badge_term};

my %valid_term = ( 0, 1, 7, 1, 14, 1, -1, 1, -3, 1, -6, 1, -12, 1 );
if (!defined($valid_term{$ui->{badge_trend}})) {
    $ui->{badge_trend} = 0;
}

⟨ Update user account information 293 ⟩

if ($ui->{badge_trend} != 0) {
    ⟨ Update Web page badge 207b ⟩
} else {
    if (-f "⟨ Users Directory 6h ⟩/$user_file_name/BadgeImage.png") {
        unlink("⟨ Users Directory 6h ⟩/$user_file_name/BadgeImage.png");
        clusterDelete("⟨ Users Directory 6h ⟩/$user_file_name/BadgeImage.png");
    }
}

    print $fh <<"EOD";
<h1 class="c">Web Page Status Badge<br />
Configuration Changed</h1>
EOD
◇
```

Macro defined by 231, 232, 233.
Macro referenced in 169.

12.1.20.1 Show updated badge configuration

Display the current badge configuration in the result page returned when the badge trend interval is changed.

⟨ Update Web page status badge 232 ⟩ ≡

```
        if ($ui->{badge_trend} == 0) {
            print $fh <<"EOD";
    <p class="justified">
    You have disabled generation of a Web page status badge.  Please be
    sure to remove the HTML/XHTML code from your Web page which displays
    the badge, otherwise you'll see an &ldquo;Invalid request&rdquo; icon
    where the badge used to appear.
    </p>
    EOD
        } else {
            my @cterm;

            $cterm[7]  = 'week';
            $cterm[14] = 'fortnight';
            $cterm[1]  = 'month';
            $cterm[3]  = 'quarter';
            $cterm[6]  = 'six months';
            $cterm[12] = 'year';
            my $ct = $cterm[abs($ui->{badge_trend})];

            my $uec = $ui->generateEncryptedUserID();

            print $fh <<"EOD";
    <p class="justified">
    You have enabled Web page status badge generation with the
    trend for the last
    </p>

    <p class="centred">
    <b>$ct</b>
    </p>

    <p class="justified">
    displayed in the badge.  To display the badge on your Web page,
    copy and paste the following HTML/XHTML code into the page
    where you'd like to badge to appear.  Be sure to select the
    <em>entire</em> text in the box: the URL for the image is
    very long and must not be truncated.
    </p>

    EOD
    ◇
```

Macro defined by 231, 232, 233.
Macro referenced in 169.

12.1.20.2 Show prototype XHTML code to display badge

The XHTML code which the user can copy and paste into their Web page to display the badge is shown in a read-only `textbox`, ready to be selected and copied.

⟨ Update Web page status badge 233 ⟩ ≡

```
    print $fh <<"EOD";
<form id="Hdiet_badgeproto" action="#" onsubmit="return false;">
<p class="centred">
<textarea cols="80" rows="4" name="protocode" readonly="readonly"
    style="background-color: #FFFFA0; color: inherit;">
&lt;a href="⟨ Site home URL 11f ⟩⟨ Web Document Home 5a ⟩/"&gt;&lt;img style="border: 0px;"
src="⟨ Site home URL 11f ⟩⟨ URL to invoke this program 12a ⟩Badge?t=1&amp;amp;b=$uec"
alt="The Hacker's Diet Online" /&gt;&lt;/a&gt;
</textarea>
</p>
</form>

<h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=account&amp;s=$session->{session_id}$tzOff">Return t
EOD
    }

    write_XHTML_epilogue($fh, $homeBase);

    if (!$readOnly) {
        append_history($user_file_name, 19, $ui->{badge_trend});
        update_last_transaction($user_file_name);
    }
◇
```

Macro defined by 231, 232, 233.
Macro referenced in 169.

12.1.21 Recalculate trend carry-forward for all logs for a user

An "update_trend" request forces a complete recalculation and re-propagation of the trend throughout all monthly logs in a user's database. This can be used to recover from mess-ups, but can also be handy after bulk database changes such as importing large amounts of CSV data. A record of type 4 is appended to the history database to log the forced recalculation.

Optional CGI arguments allow specifying the first month in the recalculation (for example, "`m=1998-09`") and forced canonicalisation of all weight entries in the logs processed ("`canon=1`").

⟨ Recalculate trend carry-forward for all logs for a user 234 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩

    ⟨ Retrieve user account information 194 ⟩

    $CGIargs{m} = '0000-00' if !defined($CGIargs{m});
    $CGIargs{canon} = 0 if !defined($CGIargs{canon});
    if ($CGIargs{canon} ne 0) {
        $CGIargs{canon} = 1;
    }

    if ($CGIargs{m} ne '0000-00') {
        ⟨ Sanity check year and month specification 195 ⟩
    }

write_XHTML_prologue($fh, $homeBase, "Recompute trend carry-forward", undef, $session->{handheld});
    generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $timeZ
    ⟨ Generate assumed identity notification 185 ⟩

    print $fh <<"EOD";
<h1 class="c">Trend Recalculation Complete</h1>
EOD
    propagate_trend($ui, $CGIargs{m}, $CGIargs{canon}) if !$readOnly;

    print $fh <<"EOD";
<h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=account&amp;s=$session->{session_id}$tzOff">Back to
EOD
write_XHTML_epilogue($fh, $homeBase);

    if (!$readOnly) {
        append_history($user_file_name, 4, "$CGIargs{m},$CGIargs{canon}");
        update_last_transaction($user_file_name);
    }
◇
```

Macro referenced in 169.

12.1.22 Quit browsing another account

The "quitbrowse" transaction terminates browsing of another user account by the administrator or read-only browsing of a public account by a user. If somebody fakes up this transaction in a session which is not browsing, no harm will be done—it will simply be ignored.

⟨ Quit browsing another account 235a ⟩ ≡

```
⟨ Retrieve active session information 193 ⟩
⟨ Retrieve user account information 194 ⟩

if ($assumed_identity || $browse_public) {
    $session->{effective_name} = $session->{browse_name} = '';
    open(FS, ">:utf8", "⟨ Session Directory 6g ⟩/$session->{session_id}.hds") ||
        die("Cannot create session file ⟨ Session Directory 6g ⟩/$session->{session_id}.hds");
    $session->save(\*FS);
    close(FS);
    clusterCopy("⟨ Session Directory 6g ⟩/$session->{session_id}.hds");
}
$CGIargs{q} = 'account';
next;
```

◇

Macro referenced in 169.

12.1.23 Download monthly log as CSV file

Download the current-displayed monthly log as a CSV file in our extended format.

⟨ Download monthly log as CSV file 235b ⟩ ≡

```
⟨ Retrieve active session information 193 ⟩
⟨ Retrieve user account information 194 ⟩

⟨ Read log if in database or create blank log if it's not 197b ⟩

print($fh "Content-type: text/csv; charset=iso-8859-1\r\n");
print($fh "Content-disposition: attachment; filename=\"$CGIargs{m}.csv\"\r\n");
print($fh "\r\n");

$mlog->exportCSV($fh);

exit(0);
```

◇

Macro referenced in 168.

12.1.24 Download monthly log as XML file

Download the current-displayed monthly log as an XML in our format. We simply output a "`hackersdiet`" DTD XML file with a single `monthlog`" element containing the present log.

⟨ Download monthly log as XML file 236 ⟩ ≡

```
⟨ Retrieve active session information 193 ⟩
⟨ Retrieve user account information 194 ⟩

⟨ Read log if in database or create blank log if it's not 197b ⟩

binmode($fh, ":utf8");
print($fh "Content-type: application/xml; charset=utf-8\r\n");
print($fh "Content-disposition: attachment; filename=\"$CGIargs{m}.xml\"\r\n");
print($fh "\r\n");

generateXMLprologue($fh);
$mlog->exportXML($fh, 1);
generateXMLepilogue($fh);

exit(0);
```

◇

Macro referenced in 168.

12.1.25 Export log database

Export the entire database in a user-selected format selected from the following request form.

⟨ Export log database 237 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    my @years = $ui->enumerateYears();

    ⟨ Set variables to default to previous request settings 283, ... ⟩

    write_XHTML_prologue($fh, $homeBase, "Export Log Database", undef, $session->{handheld});
    generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $timeZ
    ⟨ Generate assumed identity notification 185 ⟩

    print $fh <<"EOD";
<h1 class="c">Export Log Database</h1>
EOD

    print $fh <<"EOD";
<form id="Hdiet_exportdb" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩

<p class="mlog_buttons">
<b>Format:</b><br />
    <label><input type="radio" name="format" value="xml" checked="checked" /> Hacker's Diet <em>Online</e
    <label><input type="radio" name="format" value="csv" /> Hacker's Diet <em>Online</em> CSV</label><br
    <label><input type="radio" name="format" value="palm" /> Palm Eat Watch CSV</label><br />
    <label><input type="radio" name="format" value="excel" /> Legacy Excel CSV</label><br />
</p>

⟨ Selection of months to export from database 238a ⟩

<input type="hidden" name="s" value="$session->{session_id}" />
<input type="submit" name="q=do_exportdb" value=" Export " />
 
<input type="reset" value=" Reset " />
 
<input type="submit" name="q=account" value=" Cancel " />
</p>
</form>
EOD

    write_XHTML_epilogue($fh, $homeBase);
◇
```

Macro referenced in 169.

12.1.25.1 Selection of months to export from database

The following controls allow the user to select either all months in the database or a range of months. The starting and ending months are preset to the first and last months present in the database.

⟨ Selection of months to export from database 238a ⟩ ≡

```
<p class="mlog_buttons">
<label><input type="radio" name="period" value="a" checked="checked" /> <b>Export all months</b></label>
<br />
<label><input type="radio" name="period" value="c" /> <b>Export months</b></label>
EOD

    $fy_selected[0] = ' selected="selected"';
    my @f_mon = $ui->enumerateMonths($years[0]);
    $f_mon[0] =~ m/^\d+\-(\d+)$/;
    my $fmon = $1 + 0;
    $fm_selected[$fmon] = ' selected="selected"';

    print($fh "From\n");
    ⟨ Custom trend start date (238b 0 ) 259 ⟩

    print $fh <<"EOD";
<br />
EOD

    $ty_selected[$#years] = ' selected="selected"';
    @f_mon = $ui->enumerateMonths($years[$#years]);
    $f_mon[$#f_mon] =~ m/^\d+\-(\d+)$/;
    $fmon = $1 + 0;
    $tm_selected[$fmon] = ' selected="selected"';

    print($fh "To\n");
    ⟨ Custom trend end date (238c 0 ) 260 ⟩

    print $fh <<"EOD";
<br />
◇
```

Macro referenced in 237.

12.1.26 Process database export

Perform the database export operation requested by the above form.

⟨ Process database export 239 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    ⟨ Determine first and last days in database 240 ⟩

    $CGIargs{from_d} = 1;
    $CGIargs{to_d} = 31;
    ⟨ Process custom interval specification, if any 253 ⟩

    my ($start_ym, $end_ym) = ("0000-00", "9999-99");

    if ($custom) {
        $start_ym = sprintf("%04d-%02d", $cust_start_y, $cust_start_m);
        $end_ym = sprintf("%04d-%02d", $cust_end_y, $cust_end_m);
    }

    $CGIargs{format} = '?' if !$CGIargs{format};

    if ($CGIargs{format} eq 'xml') {
        ⟨ Export database as XML 241 ⟩
    } elsif ($CGIargs{format} eq 'csv') {
        ⟨ Export database as Hacker's Diet Online CSV 242 ⟩
    } elsif ($CGIargs{format} eq 'palm') {
        ⟨ Export database as Palm Eat Watch CSV 243 ⟩
    } elsif ($CGIargs{format} eq 'excel') {
        ⟨ Export database as Legacy Excel Eat Watch CSV 244 ⟩
    } else {
    print $fh <<"EOD";
<h1>Invalid format specified for database export.</h1>
EOD
    }

    ⟨ MIME Content-type specification 372a ⟩
    write_XHTML_prologue($fh, $homeBase, "Export Log Database", undef, $session->{handheld});
    generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $timeZ
    ⟨ Generate assumed identity notification 185 ⟩

    print $fh <<"EOD";

<h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=account&amp;s=$session->{session_id}$tzOff">Back to
EOD
    write_XHTML_epilogue($fh, $homeBase);
◇
```

Macro referenced in 168.

12.1.26.1 Determine first and last days in database

Find the first and last dates in the database. These are used as the default settings for a custom request and to limit out of range requests to data actually present in the database.

⟨ Determine first and last days in database 240 ⟩ ≡

```
my $hist = HDiet::history->new($ui, $user_file_name);
my ($s_y, $s_m, $s_d) = $hist->firstDay();
my $s_jd = gregorian_to_jd($s_y, $s_m, $s_d);
my ($l_y, $l_m, $l_d) = $hist->lastDay();
my $l_jd = gregorian_to_jd($l_y, $l_m, $l_d);
```

◇

Macro referenced in 239, 252a, 280.

12.1.26.2 Export database in XML format

Export the database as XML. The user identification, preferences, and diet plan precede the monthly logs selected to be exported.

⟨ Export database as XML 241 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    binmode($fh, ":utf8");

#my $oldfh = select $fh; $| = 1; select $oldfh;
    print($fh "Content-type: application/xml; charset=utf-8\r\n");
    print($fh "Content-disposition: attachment; filename=\"hackdiet_db.xml\"\r\n");
    print($fh "\r\n");

    generateXMLprologue($fh);

    my $ep = timeXML(time());

    print $fh <<"EOD";
    <epoch>$ep</epoch>
    <account version="1.0">
EOD
    $ui->exportUserInformationXML($fh);
    $ui->exportPreferencesXML($fh);
    $ui->exportDietPlanXML($fh);
    print $fh <<"EOD";
    </account>
EOD

    my @logs = $ui->enumerateMonths();

    print $fh <<"EOD";
    <monthlogs version="1.0">
EOD
    for (my $i = 0; $i <= $#logs; $i++) {
        if (($logs[$i] ge $start_ym) && ($logs[$i] le $end_ym)) {
            my $mlog = HDiet::monthlog->new();
            open(FL, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/$logs[$i].hdb") ||
                die("Cannot open monthly log file ⟨ Users Directory 6h ⟩/$user_file_name/$logs[$i].hdb");
            $mlog->load(\*FL);
            close(FL);

            $mlog->exportXML($fh, 1);

            undef($mlog);
        }
    }
    print $fh <<"EOD";
    </monthlogs>
EOD

    generateXMLepilogue($fh);
    exit(0);
◇
```

Macro referenced in 239.

12.1.26.3 Export database as Hacker's Diet Online CSV

Export the database in our extended CSV format, which allows encoding of embedded Unicode characters. All of the selected months are simply concatenated into one big CSV file, repeating the header lines for each month. Our CSV import code handles this with no difficulty, and it's the only way we can inform it if, for example, the weight unit changes from one monthly log to another.

⟨ Export database as Hacker's Diet Online CSV 242 ⟩ ≡

```
⟨ Retrieve active session information 193 ⟩
⟨ Retrieve user account information 194 ⟩

print($fh "Content-type: text/csv; charset=iso-8859-1\r\n");
print($fh "Content-disposition: attachment; filename=\"hackdiet_db.csv\"\r\n");
print($fh "\r\n");

print($fh encodeCSV("Epoch", timeXML(time())), "\r\n");

print($fh encodeCSV("User", "1.0", $ui->{login_name},
    $ui->{first_name}, $ui->{middle_name}, $ui->{last_name},
    $ui->{e_mail}, timeXML($ui->{account_created})), "\r\n");

print($fh encodeCSV("Preferences", "1.0",
    HDiet::monthlog::WEIGHT_UNITS->[$ui->{log_unit}],
    HDiet::monthlog::WEIGHT_UNITS->[$ui->{display_unit}],
    HDiet::monthlog::ENERGY_UNITS->[$ui->{energy_unit}],
    $ui->{current_rung},
    $ui->{decimal_character}), "\r\n");

my $at = timeXML($ui->{calc_start_date});

print($fh encodeCSV("Diet-Plan", "1.0",
    $ui->{calc_calorie_balance},
    $ui->{calc_start_weight},
    $ui->{calc_goal_weight},
    $at,
    $ui->{plot_diet_plan}), "\r\n");

my @logs = $ui->enumerateMonths();

for (my $i = 0; $i <= $#logs; $i++) {
    if (($logs[$i] ge $start_ym) && ($logs[$i] le $end_ym)) {
        my $mlog = HDiet::monthlog->new();
        open(FL, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/$logs[$i].hdb") ||
            die("Cannot open monthly log file ⟨ Users Directory 6h ⟩/$user_file_name/$logs[$i].hdb");
        $mlog->load(\*FL);
        close(FL);

        $mlog->exportCSV($fh);

        undef($mlog);
    }
}
exit(0);
```

◇

Macro referenced in 239.

12.1.26.4 Export database as Palm Eat Watch CSV

The database is exported in the format used by the `hdread` desktop utility used with the Palm Eat Watch. Exporting in this format allows logs created with this application to be imported onto handheld devices running the Palm software.

⟨ Export database as Palm Eat Watch CSV 243 ⟩ ≡

```
⟨ Retrieve active session information 193 ⟩
⟨ Retrieve user account information 194 ⟩

print($fh "Content-type: text/csv; charset=iso-8859-1\r\n");
print($fh "Content-disposition: attachment; filename=\"hackdiet_db.csv\"\r\n");
print($fh "\r\n");

my @logs = $ui->enumerateMonths();

for (my $i = 0; $i <= $#logs; $i++) {
    if (($logs[$i] ge $start_ym) && ($logs[$i] le $end_ym)) {
        my $mlog = HDiet::monthlog->new();
        open(FL, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/$logs[$i].hdb") ||
            die("Cannot open monthly log file ⟨ Users Directory 6h ⟩/$user_file_name/$logs[$i].hdb");
        $mlog->load(\*FL);
        close(FL);

        $mlog->exportHDReadCSV($fh);

        undef($mlog);
    }
}
exit(0);
```

◇

Macro referenced in 239.

12.1.26.5 Export database as Legacy Excel Eat Watch CSV

The database is exported in the legacy format used by the Excel Eat Watch spreadsheet. The data from this export must be pasted into a spreadsheet created for the year from the master template. Exporting in this format loses comments which appear on days for which a weight is entered.

⟨ Export database as Legacy Excel Eat Watch CSV 244 ⟩ ≡

```
⟨ Retrieve active session information 193 ⟩
⟨ Retrieve user account information 194 ⟩

print($fh "Content-type: text/csv; charset=iso-8859-1\r\n");
print($fh "Content-disposition: attachment; filename=\"hackdiet_db.csv\"\r\n");
print($fh "\r\n");

my @logs = $ui->enumerateMonths();

for (my $i = 0; $i <= $#logs; $i++) {
    if (($logs[$i] ge $start_ym) && ($logs[$i] le $end_ym)) {
        my $mlog = HDiet::monthlog->new();
        open(FL, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/$logs[$i].hdb") ||
            die("Cannot open monthly log file ⟨ Users Directory 6h ⟩/$user_file_name/$logs[$i].hdb");
        $mlog->load(\*FL);
        close(FL);

        $mlog->exportExcelCSV($fh);

        undef($mlog);
    }
}
exit(0);
```

◇

Macro referenced in 239.

12.1.27 Request paper log forms

Request a document the user can print to use to log weight and exercise for later entry to the application. We set the `target` property of the form so that the paper log document opens in a new window. If JavaScript is not available, it will open in the current window and the user will have to return to the request page with the "Back" button.

⟨ Request paper log forms 245 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    my @years;

    ⟨ Set variables to default to previous request settings 283, ... ⟩

    write_XHTML_prologue($fh, $homeBase, "Generate Log Forms", undef, $session->{handheld});
    generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $timeZ
    ⟨ Generate assumed identity notification 185 ⟩

    print $fh <<"EOD";
<h1 class="c">Generate Paper Log Forms</h1>
EOD

    print $fh <<"EOD";
<form id="Hdiet_plog" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩

⟨ Selection of months for paper logs 246a ⟩

<input type="hidden" name="s" value="$session->{session_id}" />
<input type="submit" name="q=do_paper_logs" value=" Generate " />
 
<input type="reset" value=" Reset " />
 
<input type="submit" name="q=account" value=" Cancel " />
</p>
</form>

<script type="text/javascript" defer="defer">
/* <![CDATA[ */
    if (document.getElementById && document.getElementById("Hdiet_plog")) {
        document.getElementById("Hdiet_plog").target = "_blank";
    }
/* ]]> */
</script>

EOD

    write_XHTML_epilogue($fh, $homeBase);
◇
```

Macro referenced in 169.

12.1.27.1 Selection of months for paper logs

The following controls allow the user to select which months will be included in the paper log document.

⟨ Selection of months for paper logs 246a ⟩ ≡

```
<p class="mlog_buttons">
EOD

    my $jdnow = unix_time_to_jd(time());
    my ($enowy, $enowm, $enowd) = jd_to_gregorian($jdnow);
    my @f_mon;

    for (my $y = $enowy - 1; $y <= $enowy + 1; $y++) {
        $fy_selected[$y - ($enowy - 1)] = $ty_selected[$y - ($enowy - 1)] = '';
        $years[$y - ($enowy - 1)] = $y;
        for (my $m = 1; $m <= 12; $m++) {
            $f_mon[$m] = sprintf("%4d-%02d", $y, $m);
        }
    }
    $fy_selected[1] = ' selected="selected"';
    $fm_selected[1] = ' selected="selected"';

    print($fh "From\n");
    ⟨ Custom trend start date (246b 0 ) 259 ⟩

    print $fh <<"EOD";
<br />
EOD

    $ty_selected[1] = ' selected="selected"';
    $tm_selected[12] = ' selected="selected"';

    print($fh "To\n");
    ⟨ Custom trend end date (246c 0 ) 260 ⟩

    print $fh <<"EOD";
<br />
◇
```

Macro referenced in 245.

12.1.28 Generate paper log forms

Generate a document the user can print to use to log weight and exercise for later entry to the application. This transaction is invoked from the request form above, and passed the starting and ending year and month for the logs to be printed. The paper log document is opened it a new window and, a second later, a print operation is queued (the latter two operations require JavaScript).

⟨ Generate paper log forms 247 ⟩ ≡

```
⟨ Retrieve active session information 193 ⟩
⟨ Retrieve user account information 194 ⟩

my @years;

⟨ Set variables to default to previous request settings 283, ... ⟩

write_XHTML_prologue($fh, $homeBase, "Weight and Exercise Log",
    " if (window.print) { setTimeout('window.print()', 1000); }", $session->{handheld}, 1);
⟨ Generate assumed identity notification 185 ⟩

#   If start and end dates are reversed, sliently exchange them.
if (gregorian_to_jd($CGIargs{from_y}, $CGIargs{from_m}, 1) >
    gregorian_to_jd($CGIargs{to_y}, $CGIargs{to_m}, 1)) {
    my ($fy, $fm) = ($CGIargs{from_y}, $CGIargs{from_m});
    ($CGIargs{from_y}, $CGIargs{from_m}) = ($CGIargs{to_y}, $CGIargs{to_m});
    ($CGIargs{to_y}, $CGIargs{to_m}) = ($fy, $fm);
}

my $firstpage = 1;
for (my $y = $CGIargs{from_y}; $y <= $CGIargs{to_y}; $y++) {
    for (my $m = $CGIargs{from_m}; $m <= $CGIargs{to_m}; ) {
        ⟨ Generate paper log form for month 248 ⟩
        $m++;
        if ($m > 12) {
            $m = 1;
            last;
        }
    }
}

write_XHTML_epilogue($fh, $homeBase);
```

◇

Macro referenced in 169.

12.1.28.1 Generate paper log form for month

A paper log page is generated for the current year and month. This page is styled for "paged media" (dead trees), and is designed to print on ISO A4 or U.S. "A" size paper with reasonable margins.

⟨ Generate paper log form for month 248 ⟩ ≡

```
        my $plc = $firstpage ? 'plog_first' : 'plog_subsequent';
        $firstpage = 0;
        my $mdays = HDiet::monthlog::monthdays($y, $m);

        print $fh <<"EOD";
    <div class="$plc">
    <h1 class="plog">Weight and Exercise Log</h1>
    <h2 class="plog">$monthNames[$m] $y</h2>
    <table class="plog">
        <tr class="heading">
            <th class="h1" colspan="3">Date</th>
            <td class="s2"></td>
            <th class="h4">Weight</th>
            <td class="s4"></td>
            <th class="h5">Rung</th>
            <td class="s5"></td>
            <th class="h6">Flag</th>
            <td class="s6"></td>
            <th class="h7">Comments</th>
        </tr>
    EOD

        my $wday = jd_to_weekday(gregorian_to_jd($y, $m, 1));

        for (my $d = 1; $d <= $mdays; $d++) {
            my $wdn = substr(HDiet::Julian::WEEKDAY_NAMES->[$wday], 0, 3);
            $wday = ($wday + 1) % 7;
            #   The " "s in this table are courtesy of crap-bag
            #   Internet Explorer, which doesn't draw a border below
            #   a table cell if it's empty.
            print $fh <<"EOD";
        <tr>
            <th class="c1">$d</th>
            <td class="s1"></td>
            <td class="c2">$wdn</td>
            <td class="s2"></td>
            <td class="c3"> </td>
            <td class="s3"></td>
            <td class="c4"> </td>
            <td class="s4"></td>
            <td class="c5"> </td>
            <td class="s5"></td>
            <td class="c6"> </td>
        </tr>
    EOD
        }

        print $fh <<"EOD";
    </table>
    </div>
    EOD
```

◇

Macro referenced in 247.

12.1.29 Download backup copy of all logs for user

A user may download a Zipped archive containing logs for all months in the database by submitting a "backup" request. The backup is returned as an in-line download, created on the fly by invoking the "`zip`" program with its output directed to standard output, which is returned from the CGI application. The default file name is "`hackdiet_log_backup_`*YYYY*`-`*MM*`-`*DD*`.zip`", based on the current date in UTC.

⟨ Download backup copy of all logs for user 249 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩

    ⟨ Retrieve user account information 194 ⟩

    my $nlogs = `ls -1 ⟨ Users Directory 6h ⟩/$user_file_name/????-??.hdb 2>/dev/null | wc -l`;
    chomp($nlogs);

    if ($nlogs > 0) {
        my ($year, $mon, $mday, $hour, $min, $sec) =
            unix_time_to_civil_date_time($userTime);
        my $date = sprintf("%04d-%02d-%02d", $year, $mon, $mday);

        print($fh "Content-type: application/zip\r\n");
        print($fh "Content-disposition: attachment; filename=\"hackdiet_log_backup_$date.zip\"\r\n");
        print($fh "\r\n");

        system("zip -q -j - ⟨ Users Directory 6h ⟩/$user_file_name/????-??.hdb");
        exit(0);
    }

⟨ MIME Content-type specification 372a ⟩
write_XHTML_prologue($fh, $homeBase, "Download backup copy", undef, $session->{handheld});
generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $timeZone(
⟨ Generate assumed identity notification 185 ⟩

print $fh <<"EOD";
<h1 class="c">You have no logs to back up!</h1>

<h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=log&amp;s=$session->{session_id}$tzOff">Back to mont
EOD
write_XHTML_epilogue($fh, $homeBase);

    update_last_transaction($user_file_name) if !$readOnly;
    exit(0);
◇
```

Macro referenced in 168.

12.1.30 Generate monthly chart

A monthly chart is generated and returned as an in-line PNG image by the "chart" query, where, as for the monthly log documents in which such charts are usually embedded, the "m" argument specifies the year and month to be charted. It is perfectly valid to request a chart for a month for which no log exists in the database; an empty log will be synthesised and a blank chart generated. Optional "width" and "height" arguments allow specifying the chart size. If omitted, the chart defaults to 640×480 pixels.

⟨ Generate monthly chart 250a ⟩ ≡

```
⟨ Retrieve active session information 193 ⟩
⟨ Retrieve user account information 194 ⟩

⟨ Sanity check year and month specification 195 ⟩

⟨ Read log if in database or create blank log if it's not 197b ⟩

⟨ Specify Content-type for PNG image 250b ⟩

$CGIargs{width} = ($session->{handheld} ? 320 : 640) if !defined $CGIargs{width};
$CGIargs{height} = ($session->{handheld} ? 240 : 480) if !defined $CGIargs{height};

my @dcalc;
if ($ui->{plot_diet_plan}) {
    @dcalc = $ui->dietPlanLimits();
}

$mlog->plotChart($fh, $CGIargs{width}, $CGIargs{height},
    $ui->{display_unit}, $ui->{decimal_character}, \@dcalc,
    $CGIargs{print}, $CGIargs{mono});

update_last_transaction($user_file_name) if !$readOnly;
exit(0);
```

◇

Macro referenced in 168.

12.1.30.1 Specify Content-type for PNG image

Embedded images for charts are included in HTML documents as query URLs which provide all of the parameters required to generate the image. When the browser renders the document, it will send a request to the server, which prepares the image on the fly. This approach makes generation of the image "stateless"—it can be performed by any server in the cluster so long as it has access to the current database from which the image is to be generated.

⟨ Specify Content-type for PNG image 250b ⟩ ≡

```
print($fh "Content-type: image/png\r\n\r\n");
```

◇

Macro referenced in 250a, 287.

12.1.31 Trend analysis

Output the trend analysis page. This page contains the default trend analyses for periods ending at the present, and includes a form to request custom trend analyses.

⟨ Trend analysis 251 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    my @years = $ui->enumerateYears();

    write_XHTML_prologue($fh, $homeBase, "Trend Analysis", undef, $session->{handheld});
    generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, "Trend", undef, $browse_public, $tim
    ⟨ Generate assumed identity notification 185 ⟩

    if ($#years >= 0) {
        ⟨ Emit trend anlysis page 252a ⟩
    } else {
        print $fh <<"EOD";
        <h2>You have no log entries!  You must enter weight logs
            before you can perform trend analysis.</h2>
EOD
    }

    print $fh <<"EOD";

<h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=account&amp;s=$session->{session_id}$tzOff">Back to
EOD

    write_XHTML_epilogue($fh, $homeBase);

    update_last_transaction($user_file_name) if !$readOnly;
◇
```

Macro referenced in 169.

12.1.31.1 Emit trend anlysis page

Generate the page containing the trend analyses for the various intervals.

⟨ Emit trend anlysis page 252a ⟩ ≡

```
    print $fh <<"EOD";
<h1 class="c">Trend Analysis</h1>
EOD

    ⟨ Determine first and last days in database 240 ⟩

    ⟨ Add standard intervals to analysis list 252b ⟩

    ⟨ Process custom interval specification, if any 253 ⟩

    if ($custom) {
        push(@intervals, sprintf("%04d-%02d-%02d", $cust_start_y, $cust_start_m, $cust_start_d),
                         sprintf("%04d-%02d-%02d", $cust_end_y, $cust_end_m, $cust_end_d));
        push(@dayspan, ($cust_end_jd - $cust_start_jd) + 1);
    }

    ⟨ Output trend analysis report for intervals evaluated 254 ⟩

    ⟨ Set variables to default to previous request settings 283, ... ⟩

    ⟨ Generate form fields for custom trend interval 257 ⟩
◇
```

Macro referenced in 251.

12.1.31.2 Add standard intervals to analysis list

Each of the standard trend analysis intervals (week, fortnight, month, etc.) are evaluated back from the last log entry in the database. Each interval which begins on or after the first entry in the database is added to the list of intervals for which the trend will be evaluated.

⟨ Add standard intervals to analysis list 252b ⟩ ≡

```
    my (@intervals, @dayspan);
    for my $interval (7, 14, -1, -3, -6, -12) {
        my ($f_y, $f_m, $f_d) = $hist->firstDayOfInterval($l_y, $l_m, $l_d, $interval);
        my $f_jd = gregorian_to_jd($f_y, $f_m, $f_d);
        if ($f_jd < $s_jd) {
            last;
        }
        push(@intervals, sprintf("%04d-%02d-%02d", $f_y, $f_m, $f_d),
                         sprintf("%04d-%02d-%02d", $l_y, $l_m, $l_d));
        push(@dayspan, ($l_jd - $f_jd) + 1);
    }
◇
```

Macro referenced in 252a.

12.1.31.3 Process custom interval specification, if any

If a custom interval has been requested, validate it and add it to the end of the list of intervals to be computed. If the custom interval is void, custom interval generation is disabled. If the start and end of the interval are reversed, put them in the correct order.

⟨ Process custom interval specification, if any 253 ⟩ ≡

```
my $custom = $CGIargs{period} && ($CGIargs{period} eq 'c');
my ($cust_start_y, $cust_start_m, $cust_start_d, $cust_start_jd,
    $cust_end_y, $cust_end_m, $cust_end_d,$cust_end_jd);
if ($custom) {
    ($cust_start_y, $cust_start_m, $cust_start_d) = ($CGIargs{from_y}, $CGIargs{from_m}, $CGIargs{from_d})
    $cust_start_jd = gregorian_to_jd($cust_start_y, $cust_start_m, $cust_start_d);
    ($cust_end_y, $cust_end_m, $cust_end_d) = ($CGIargs{to_y}, $CGIargs{to_m}, $CGIargs{to_d});
    $cust_end_jd = gregorian_to_jd($cust_end_y, $cust_end_m, $cust_end_d);

    if ($cust_end_jd != $cust_start_jd) {
        #   If start or end of interval is outside the database,
        #   constrain it to the  first or last entry.
        if (($cust_start_jd < $s_jd) || ($cust_start_jd > $l_jd)) {
            ($cust_start_y, $cust_start_m, $cust_start_d, $cust_start_jd) =
                ($s_y, $s_m, $s_d, $s_jd);
           ($CGIargs{from_y}, $CGIargs{from_m}, $CGIargs{from_d}) =
                ($cust_start_y, $cust_start_m, $cust_start_d);
        }
        if (($cust_end_jd < $s_jd) || ($cust_end_jd > $l_jd)) {
            ($cust_end_y, $cust_end_m, $cust_end_d, $cust_end_jd) =
                ($l_y, $l_m, $l_d, $l_jd);
            ($CGIargs{to_y}, $CGIargs{to_m}, $CGIargs{to_d}) =
                ($cust_end_y, $cust_end_m, $cust_end_d);
        }

        #   If end of interval is before start, reverse them
        if ($cust_end_jd < $cust_start_jd) {
            my @temp = ($cust_start_y, $cust_start_m, $cust_start_d, $cust_start_jd);
            ($cust_start_y, $cust_start_m, $cust_start_d, $cust_start_jd) =
                ($cust_end_y, $cust_end_m, $cust_end_d, $cust_end_jd);
            ($CGIargs{from_y}, $CGIargs{from_m}, $CGIargs{from_d}) =
                ($cust_start_y, $cust_start_m, $cust_start_d);
            ($cust_end_y, $cust_end_m, $cust_end_d, $cust_end_jd) = @temp;
            ($CGIargs{to_y}, $CGIargs{to_m}, $CGIargs{to_d}) =
                ($cust_end_y, $cust_end_m, $cust_end_d);
        }
    } else {
        $custom = 0;                # Void interval disables custom display
        $CGIargs{period} = '';
    }
}
```

◇

Macro referenced in 239, 252a, 280.

12.1.31.4 Output trend analysis report for intervals evaluated

Generate a table with rows for each of the intervals for which we we computed a trend. If there were so few days that we couldn't compute even the shortest interval, a message is output explaining the absence of a trend report.

⟨ Output trend analysis report for intervals evaluated 254 ⟩ ≡

```
        if ($#intervals >= 0) {
            my $wu = HDiet::monthlog::DELTA_WEIGHT_UNITS->[$ui->{display_unit}];
            my $eu = HDiet::monthlog::ENERGY_UNITS->[$ui->{energy_unit}];

            print $fh <<"EOD";
    <table class="trendan" border="border">
    <tr>
        <th class="custitle" colspan="6">Intervals ending $intervals[1]</th>
    </tr>
    <tr>
        <th rowspan="2">Last&hellip;</th>
        <th rowspan="2"><span class="r">Gain</span>/<span class="g">Loss</span><br /> ${wu}s/week</th>
        <th rowspan="2"><span class="r">Excess</span>/<span class="g">Deficit</span><br />${eu}s/day</th>
        <th colspan="3" style="border-bottom: none;">Weight Trend</th>
    </tr>
    <tr>
        <th style="border-top: none; border-right: none;">Min.</th>
        <th style="border-top: none; border-left: none; border-right: none;">Mean</th>
        <th style="border-top: none; border-left: none;">Max.</th>
    </tr>
    EOD
            my @slopes = $hist->analyseTrend(@intervals);
            my @inames = ( 'Week', 'Fortnight', 'Month', 'Quarter', 'Six months', 'Year' );

            ⟨ Output table rows for each interval analysed 255 ⟩
            print $fh <<"EOD";
    </table>
    EOD
        } else {
            print $fh <<"EOD";
    <h2>There are insufficient log entries to perform
    trend analysis.  You need at least a week's data
    to compute a trend.</h2>
    EOD
        }
```

◇

Macro referenced in 252a.

12.1.31.4.1 Output table rows for each interval analysed For each interval for which we computed a trend, output a row in the analysis table listing the interval, weight gain/loss, and energy balance for that interval.

⟨ Output table rows for each interval analysed 255 ⟩ ≡

```
        for (my $i = 0; $i < (($#slopes + 1) / 4); $i++) {
            my $tslope = $slopes[$i * 4];
            my $deltaW = sprintf("%.2f", $tslope * 7);
            $deltaW =~ s/\./$ui->{decimal_character}/;
            my $deltaE = sprintf("%.0f", $tslope *
                (HDiet::monthlog::CALORIES_PER_WEIGHT_UNIT->[$ui->{display_unit}] /
                 HDiet::monthlog::CALORIES_PER_ENERGY_UNIT->[$ui->{energy_unit}]));
            my $colour = $tslope > 0 ? 'r' : 'g';
            my $ecolour = $colour;
            if ($deltaW =~ m/^\-?0[\.,]00$/) {
                $colour = 'bk';
                $deltaW =~ s/^\-//;
            } else {
                $deltaW =~ s/^(\d)/\+$1/;
            }
            if ($deltaE =~ m/^\-?0$/) {
                $ecolour = 'bk';
                $deltaE =~ s/^\-//;
            } else {
                $deltaE =~ s/^(\d)/\+$1/;
            }
            $deltaW =~ s/\-/&minus;/;
            $deltaE =~ s/\-/&minus;/;

            my $eMinWeight = HDiet::monthlog::editWeight($slopes[($i * 4) + 1],
                $ui->{display_unit}, $ui->{decimal_character});
            my $eMaxWeight = HDiet::monthlog::editWeight($slopes[($i * 4) + 2],
                $ui->{display_unit}, $ui->{decimal_character});
            my $eMeanWeight = HDiet::monthlog::editWeight($slopes[($i * 4) + 3],
                $ui->{display_unit}, $ui->{decimal_character});

            ⟨ Label trend report for custom interval 256 ⟩

            print $fh <<"EOD";
    <tr>
        <td>$inames[$i]</td>
        <td class="w"><span class="$colour">$deltaW</span></td>
        <td class="e"><span class="$ecolour">$deltaE</span></td>
        <td class="e">$eMinWeight</td>
        <td class="e">$eMeanWeight</td>
        <td class="e">$eMaxWeight</td>
    </tr>
    EOD
            }
```

◇

Macro referenced in 254.

12.1.31.4.2 Label trend report for custom interval If this is the row reporting a custom interval, synthesise a label giving its duration in years, months, and days. A separator row sets off the custom interval from the standard intervals.

⟨ Label trend report for custom interval 256 ⟩ ≡

```
#print(STDERR "Custom $custom $i $#slopes\n");
    if ($custom && ($i == (($#slopes + 1) / 4) - 1)) {
        print $fh <<"EOD";
<tr>
    <th class="custitle" colspan="6">$intervals[$i * 2] &ndash; $intervals[($i * 2) + 1]</th>
</tr>
EOD
        my ($cd_y, $cd_m, $cd_d) = (0, 0, 0);
        my $cd_lastm = $cust_end_jd;
        while (1) {
            my ($ly, $lm, $ld) = $hist->firstDayOfInterval($cust_end_y, $cust_end_m, $cust_end_d, -($cd_m + 1)
            my $mjd = gregorian_to_jd($ly, $lm, $ld);
            if ($mjd < $cust_start_jd) {
                last;
            }
            $cd_m++;
            $cd_lastm = $mjd;
        }
        $cd_d = $cd_lastm - $cust_start_jd;
        $cd_y = int($cd_m / 12);
        $cd_m %= 12;
        my $custdur = (($cd_y > 0) ? "$cd_y y " : '') .
                      (($cd_m > 0) ? "$cd_m m " : '') .
                      (($cd_d > 0) ? "$cd_d d " : '');
        $inames[$i] = $custdur;
    }
◇
```

Macro referenced in 255.

12.1.31.5 Generate form fields for custom trend interval

Following the trend report we output a form, initialise to the parameters of the current report (or defaults if this is the first trend report generated), which allows the user to request a trend report for a custom interval.

⟨ Generate form fields for custom trend interval 257 ⟩ ≡

```
    print $fh <<"EOD";
<form id="Hdiet_histchart" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩
<p class="mlog_buttons">
<label><input type="checkbox" name="period" value="c"$percheck{c} /> <b>Custom</b></label>
EOD

    ⟨ Custom start and end date selection boxes 258a ⟩

    print $fh <<"EOD";
<br />

<input type="hidden" name="s" value="$session->{session_id}" />
<input type="submit" name="q=trendan" value=" Update " />
 
<input type="reset" value=" Reset " />
</p>
</form>
EOD
◇
```

Macro referenced in 252a.

12.1.31.5.1 Custom start and end date selection boxes The start and end dates of a custom interval are chosen with selection boxes populated with the range of dates present in the database. If no previous selection was made, these fields are initially set to the first and last days in the entire database.

⟨ Custom start and end date selection boxes 258a ⟩ ≡

```
        my @f_mon;
        my $fmon;
        if (!$CGIargs{from_y}) {
            $fy_selected[0] = ' selected="selected"';
            @f_mon = $ui->enumerateMonths($years[0]);
            $f_mon[0] =~ m/^\d+\-(\d+)$/;
            $fmon = $1 + 0;
            $fm_selected[$fmon] = ' selected="selected"';
            $fd_selected[$s_d] = ' selected="selected"';
        }

        print($fh "From\n");
        ⟨ Custom trend start date (258b 1 ) 259 ⟩

        print $fh <<"EOD";
    <br />
    EOD

        if (!$CGIargs{to_y}) {
            $ty_selected[$#years] = ' selected="selected"';
            @f_mon = $ui->enumerateMonths($years[$#years]);
            $f_mon[$#f_mon] =~ m/^\d+\-(\d+)$/;
            $fmon = $1 + 0;
            $tm_selected[$fmon] = ' selected="selected"';
            $td_selected[$l_d] = ' selected="selected"';
        }

        print($fh "To\n");
        ⟨ Custom trend end date (258c 1 ) 260 ⟩
    ◇
```

Macro referenced in 257, 281a.

12.1.31.5.1.1 Custom trend start date The following selection fields specify the starting date of the custom interval. A macro argument controls whether a selection field for days is generated (nonzero) or omitted (zero). A second macro argument controls whether an `onchange` attribute is included in the selection fields.

⟨ Custom trend start date 259 ⟩ ≡

```
    my ($ysel, $msel, $dsel) = ("") x 3;
    if ("@2") {
        $ysel = ' onchange="change_from_y();"';
        $msel = ' onchange="change_from_m();"';
        $dsel = ' onchange="change_from_d();"';
    }

    print $fh <<"EOD";
    <select name="from_y" id="from_y"$ysel>
EOD

    ⟨ Generate option items for years in database 285b ⟩

    print $fh <<"EOD";
    </select> <select name="from_m" id="from_m"$msel>
EOD

    my $mid = "fm_";
    ⟨ Generate option items for months 285c ⟩

        print $fh <<"EOD";
    </select>
EOD

    if (@1) {
        print $fh <<"EOD";
    <select name="from_d" id="from_d"$dsel>
EOD
    }

    my $did;

    if (@1) {
        $did = "fd_";
        ⟨ Generate option items for days 286a ⟩

        print $fh <<"EOD";
        </select>
EOD
    }
```

◇

Macro referenced in 238a, 246a, 258a, 269a.

12.1.31.5.1.2 Custom trend end date The following selection fields specify the ending date of the custom interval. A macro argument controls whether a selection field for days is generated (nonzero) or omitted (zero). A second macro argument controls whether an `onchange` attribute is included in the selection fields.

⟨ Custom trend end date 260 ⟩ ≡

```
    ($ysel, $msel, $dsel) = ("") x 3;
    if ("@2") {
        $ysel = ' onchange="change_to_y();"';
        $msel = ' onchange="change_to_m();"';
        $dsel = ' onchange="change_to_d();"';
    }
    print $fh <<"EOD";
    <select name="to_y" id="to_y"$ysel>
EOD

    @fy_selected = @ty_selected;
    ⟨ Generate option items for years in database 285b ⟩

    print $fh <<"EOD";
    </select> <select name="to_m" id="to_m"$msel>
EOD

    $mid = "tm_";
    @fm_selected = @tm_selected;
    ⟨ Generate option items for months 285c ⟩

    print $fh <<"EOD";
    </select>
EOD

    if (@1) {
        print $fh <<"EOD";
    <select name="to_d" id="to_d"$dsel>
EOD
    }

    if (@1) {
        $did = "td_";
        @fd_selected = @td_selected;
        ⟨ Generate option items for days 286a ⟩

        print $fh <<"EOD";
    </select>
EOD
    }
◇
```

Macro referenced in 238a, 246a, 258a, 269a.

12.1.32 Diet calculator

Generate the diet calculator page. The fields are initially filled in from the values saved in the `user` object.

⟨ Diet calculator 261 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    write_XHTML_prologue($fh, $homeBase, "Diet Calculator", "loadDietCalcFields();", $session->{handheld});
    generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef,
        'onclick="return leaveDocument();"', $browse_public, $timeZoneOffset);
    ⟨ Generate assumed identity notification 185 ⟩

    ⟨ Set primary diet calculator fields from user object 262 ⟩
    ⟨ Calculate dependent variables from primary variables 270b ⟩
    my @years;
    ⟨ Generate array of years for diet calculator selection 264a ⟩

    my @goofs;
    if ($CGIargs{q} eq 'update_dietcalc') {
        ⟨ Perform static update of diet calculator 272 ⟩
    }

    ⟨ Preset diet calculator start and end dates 264b ⟩
    ⟨ Set variables to default to previous request settings 283, ... ⟩

    my $e_sw = HDiet::monthlog::editWeight($calc_start_weight, $calc_weight_unit, $ui->{decimal_character});
    my $e_gw = HDiet::monthlog::editWeight($calc_goal_weight, $calc_weight_unit, $ui->{decimal_character});
    my $e_dw = HDiet::monthlog::editWeight($calc_weight_change, $calc_weight_unit, $ui->{decimal_character});
    my $e_ww = HDiet::monthlog::editWeight($calc_weight_week, $calc_weight_unit, $ui->{decimal_character});

    print $fh <<"EOD";
<h1 class="c">Diet Calculator</h1>

⟨ Generate warning if JavaScript disabled in diet calculator form 263a ⟩
⟨ Report warnings from static diet calculator update 263b ⟩

    print $fh <<"EOD";
<form id="Hdiet_newacct" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩

⟨ Generate diet calculator form 265a, ... ⟩
EOD

    if (!$browse_public) {
        print $fh <<"EOD";
⟨ Diet calculator form action buttons 270a ⟩
EOD
    }

    print $fh <<"EOD";
</form>
EOD

    write_XHTML_epilogue($fh, $homeBase);

    update_last_transaction($user_file_name) if !$readOnly;
```

◇

Macro referenced in 169.

12.1.32.1 Set primary diet calculator fields from user object

We somewhat arbitrarily divide the diet calculator fields into "primary" and "dependent" quantities. The primary quantities are those which are stored in the `user` object, and the dependent quantities are computed from them when the form is displayed. Of course, the user may change the dependent quantities, with the changes propagated back to the primary quantities in the defined fashion.

⟨ Set primary diet calculator fields from user object 262 ⟩ ≡

```
my ($calc_calorie_balance, $calc_start_weight, $calc_goal_weight,
    $calc_weight_change, $calc_weight_week, $calc_weeks, $calc_months,, $calc_end_date,
    $calc_start_date, $plot_diet_plan, $calc_weight_unit, $calc_energy_unit) =
   (round($ui->{calc_calorie_balance} * ENERGY_CONVERSION->[ENERGY_CALORIE][$ui->{energy_unit}]),
    $ui->{calc_start_weight} * WEIGHT_CONVERSION->[WEIGHT_KILOGRAM][$ui->{display_unit}],
    $ui->{calc_goal_weight} * WEIGHT_CONVERSION->[WEIGHT_KILOGRAM][$ui->{display_unit}],
    0, 0, 0, 0, 0,
    $ui->{calc_start_date}, $ui->{plot_diet_plan},
    $ui->{display_unit}, $ui->{energy_unit}
   );
```

⟨ Override diet calculator primary fields from form fields 271 ⟩

```
my $ckplan = $ui->{plot_diet_plan} ? ' checked="checked"' : '';
my @eunit = ('', '');
$eunit[$calc_energy_unit] = ' selected="selected"';
my @wunit = ('', '', '');
$wunit[$calc_weight_unit] = ' selected="selected"';

if ($calc_start_date == 0) {
    $calc_start_date = time();
}

#   If no start weight specified, use last trend value from the
#   most recent log or a default if no logs exist.
if ($calc_start_weight == 0) {
    my $hist = HDiet::history->new($ui, $user_file_name);
    my ($ly, $lm, $ld, $ldu, $lw, $lt) = $hist->lastDay();
    if (defined($lt)) {
        $calc_start_weight = sprintf("%.1f", $lt * WEIGHT_CONVERSION->[$ldu][$ui->{display_unit}]);
    } else {
        $calc_start_weight = ($ui->{display_unit} == WEIGHT_KILOGRAM) ? 80 : 175;
    }
}

#   If no goal weight specified, assume 5 kilos or 10 pounds loss
if ($calc_goal_weight == 0) {
    $calc_goal_weight = $calc_start_weight - (($ui->{display_unit} == WEIGHT_KILOGRAM) ? 5 : 10);
}
```

◇

Macro referenced in 261.

12.1.32.2 Generate warning if JavaScript disabled in diet calculator form

If JavaScript is not enabled, a warning is omitted which reminds the user to press the "Update" button after each single-field change. This is the only way the static CGI update code can know the order in which to apply changes.

⟨ Generate warning if JavaScript disabled in diet calculator form 263a ⟩ ≡

```
<p class="justified" id="noJS"
   style="margin-left: auto; margin-right: auto; width: 75%; display: block; font-family: sans-serif; font-wei
<script type="text/javascript">
/* <![CDATA[ */
    document.getElementById("noJS").style.display = "none";
/* ]]> */
</script>
Your browser does not support JavaScript (or it is disabled).  Please
click the &ldquo;Update&rdquo; button after each change to a form field
to update the rest of the form.
</p>
EOD
◇
```

Macro referenced in 261.

12.1.32.3 Report warnings from static diet calculator update

Any errors detected in a static update to the diet calculator are pushed onto the `@goofs` array. If the array is not non-void, we emit the errors in a list before the diet calculator form.

⟨ Report warnings from static diet calculator update 263b ⟩ ≡

```
    if ($#goofs >= 0) {
   print $fh <<"EOD";
<h3 class="warning">Warning:  The following errors were found in
your changes to the diet calculator</h3>
<ul class="goofs">
EOD
    for (my $i = 0; $i <= $#goofs; $i++) {
        print($fh "<li>$goofs[$i].</li>\n");
    }
    print $fh <<"EOD";
</ul>
EOD
   }
◇
```

Macro referenced in 261.

12.1.32.4 Generate array of years for diet calculator selection

The list of years in the diet calculator selection box must encompass the range including the starting and ending dates of the diet plan and any year the user is likely to select. We include all years for which we have logs in the database, the start and end years of the diet plan, and the previous and next years.

⟨ Generate array of years for diet calculator selection 264a ⟩ ≡

```
my $cyear = (jd_to_gregorian(unix_time_to_jd($userTime)))[0];
@years = $ui->enumerateYears();
if ($#years < 0) {          # If no years in database, include current year
    push(@years, $cyear);
}
my $lyear = max($cyear, (jd_to_gregorian(unix_time_to_jd($calc_start_date)))[0],
    (jd_to_gregorian(unix_time_to_jd($calc_end_date)))[0]);
while ($years[$#years] < ($lyear + 1)) {
    push(@years, $years[$#years] + 1);
}
while ($years[0] > ($cyear - 1)) {
    unshift(@years, $years[0] - 1);
}
```

◇

Macro referenced in 261, 272.

12.1.32.5 Preset diet calculator start and end dates

The diet calculator start and end dates from the `user` object are converted from UNIX time to CGI arguments which will be used when generating the form to preset the date selection boxes. We also convert the dates into Julian dates for use in the actual diet calculator computations.

⟨ Preset diet calculator start and end dates 264b ⟩ ≡

```
#   Preset start date selection fields
my $s_jd = unix_time_to_jd($calc_start_date);
($CGIargs{from_y}, $CGIargs{from_m}, $CGIargs{from_d}) = jd_to_gregorian($s_jd);

#   Preset end date selection fields
my $e_jd = unix_time_to_jd($calc_end_date);
($CGIargs{to_y}, $CGIargs{to_m}, $CGIargs{to_d}) = jd_to_gregorian($e_jd);
```

◇

Macro referenced in 261.

12.1.32.6 Generate diet calculator form

The diet calculator is embedded in an XHTML `<table>` generated by the following. The editable fields are complicated by the need for them to activate JavaScript functions upon change. For every input field the user can modify, there is a companion "`r_`" field which contains the initial value displayed in the form. Users who do not have JavaScript may change a form value, then submit the form with the "Update" button; when the server receives the form, it compares the "`r_`" fields with the input fields to determine which field the user changed and applies the change, returning an updated form to the user.

⟨ Generate diet calculator form 265a ⟩ ≡

```
<table border="border" class="login">
◇
```

Macro defined by 265ab, 266ab, 267ab, 268, 269a.
Macro referenced in 261.

12.1.32.6.1 Diet calculator: calorie balance

⟨ Generate diet calculator form 265b ⟩ ≡

```
    <tr>
        <th><label for="calc_calorie_balance">Daily balance</label></th>
        <td><input type="text" name="calc_calorie_balance"  id="calc_calorie_balance"
                onchange="change_calc_calorie_balance();"
                size="5" maxlength="5"
                value="$calc_calorie_balance" />
            <input type="hidden" name="r_calc_calorie_balance"
                value="$calc_calorie_balance" />
            <select name="calc_energy_unit" id="calc_energy_unit"
                    onchange="change_calc_energy_unit();">
                <option value="0"$eunit[0]>cal</option>
                <option value="1"$eunit[1]>kJ</option>
            </select>
            <input type="hidden" name="r_calc_energy_unit"
                value="$calc_energy_unit" />
        </td>
    </tr>
◇
```

Macro defined by 265ab, 266ab, 267ab, 268, 269a.
Macro referenced in 261.

12.1.32.6.2 Diet calculator: initial weight

⟨ Generate diet calculator form 266a ⟩ ≡

```
        <tr>
            <th><label for="calc_start_weight">Initial weight</label></th>
            <td><input type="text" name="calc_start_weight" id="calc_start_weight"
                    onchange="change_calc_start_weight();"
                    size="7" maxlength="7"
                    value="$e_sw" />
                <input type="hidden" name="r_calc_start_weight"
                    value="$e_sw" />
                <select name="calc_weight_unit" id="calc_weight_unit"
                    onchange="change_calc_weight_unit();">
                    <option value="0"$wunit[0]>kilograms</option>
                    <option value="1"$wunit[1]>pounds</option>
                    <option value="2"$wunit[2]>stones</option>
                </select>
                <input type="hidden" name="r_calc_weight_unit"
                    value="$calc_weight_unit" />
            </td>
        </tr>
    ◇
```

Macro defined by 265ab, 266ab, 267ab, 268, 269a.
Macro referenced in 261.

12.1.32.6.3 Diet calculator: goal weight

⟨ Generate diet calculator form 266b ⟩ ≡

```
        <tr>
            <th><label for="calc_goal_weight">Goal weight</label></th>
            <td><input type="text" name="calc_goal_weight" id="calc_goal_weight"
                    onchange="change_calc_goal_weight();"
                    size="7" maxlength="7"
                    value="$e_gw" />
                <input type="hidden" name="r_calc_goal_weight"
                    value="$e_gw" />
            </td>
        </tr>
    ◇
```

Macro defined by 265ab, 266ab, 267ab, 268, 269a.
Macro referenced in 261.

12.1.32.6.4 Diet calculator: desired weight change

⟨ Generate diet calculator form 267a ⟩ ≡

```
<tr>
    <th><label for="calc_weight_change">Desired weight change</label></th>
    <td><input type="text" name="calc_weight_change" id="calc_weight_change"
            onchange="change_calc_weight_change();"
            size="7" maxlength="7"
            value="$e_dw" />
        <input type="hidden" name="r_calc_weight_change"
            value="$e_dw" />
    </td>
</tr>
```

◇

Macro defined by 265ab, 266ab, 267ab, 268, 269a.
Macro referenced in 261.

12.1.32.6.5 Diet calculator: weight change per week

⟨ Generate diet calculator form 267b ⟩ ≡

```
<tr>
    <th><label for="calc_weight_week">Weight change per week</label></th>
    <td><input type="text" name="calc_weight_week" id="calc_weight_week"
            onchange="change_calc_weight_week();"
            size="7" maxlength="7"
            value="$e_ww" />
        <input type="hidden" name="r_calc_weight_week"
            value="$e_ww" />
    </td>
</tr>
```

◇

Macro defined by 265ab, 266ab, 267ab, 268, 269a.
Macro referenced in 261.

12.1.32.6.6 Diet calculator: diet duration

⟨ Generate diet calculator form 268 ⟩ ≡

```
        <tr>
            <th><label for="calc_weeks">Diet duration</label></th>
            <td><input type="text" name="calc_weeks" id="calc_weeks"
                    onchange="change_calc_weeks();"
                    size="5" maxlength="5"
                    value="$calc_weeks" /> weeks,
                <input type="hidden" name="r_calc_weeks"
                    value="$calc_weeks" />
                <input type="text" name="calc_months" id="calc_months"
                    onchange="change_calc_months();"
                    size="5" maxlength="5"
                    value="$calc_months" /> months
                <input type="hidden" name="r_calc_months"
                    value="$calc_months" />
            </td>
        </tr>
    ◇
```

Macro defined by 265ab, 266ab, 267ab, 268, 269a.
Macro referenced in 261.

12.1.32.6.7 Diet calculator: start and end dates

⟨ Generate diet calculator form 269a ⟩ ≡

```
        <tr>
            <th><label for="from_y">Start date</label></th>
            <td>
    EOD

        ⟨ Custom trend start date (269b 1,269c 1 ) 259 ⟩

        my $disped = ($e_jd >= $s_jd) ? 'inline' : 'none';
        print $fh <<"EOD";
                <input type="hidden" name="r_calc_start_date"
                    value="$s_jd" />
            </td>
        </tr>
        <tr>
            <th><label for="to_y">End date</label></th>
            <td>
                <span id="end_date" style="display: $disped;">
    EOD

        ⟨ Custom trend end date (269d 1,269e 1 ) 260 ⟩

        $disped = ($e_jd < $s_jd) ? 'inline' : 'none';
        print $fh <<"EOD";
            </span>
                <input type="hidden" name="r_calc_end_date"
                    value="$e_jd" />
                <span id="endless_date" style="display: $disped;"
                    onmouseover="document.getElementById('end_date').style.display = 'inline'; document.getElement
                <i>Never</i>
                </span>
            </td>
        </tr>

    </table>
    ◇
```

Macro defined by 265ab, 266ab, 267ab, 268, 269a.
Macro referenced in 261.

12.1.32.7 Diet calculator form action buttons

In addition to the usual "Save", "Reset", and "Cancel" buttons, if the user's browser lacks JavaScript, an "Update" button will be displayed which, when pressed, submits the form with a transaction which discovers the change made to a field and updates the calculator in a page returned to the user.

⟨ Diet calculator form action buttons 270a ⟩ ≡

```
<p class="centred" id="noJS1" style="display: block;">
<script type="text/javascript">
/* <![CDATA[ */
    document.getElementById("noJS1").style.display = "none";
/* ]]> */
</script>
<input type="submit" name="q=update_dietcalc" value="     Update     " />
</p>

<p class="mlog_buttons">
<label><input type="checkbox" name="plot_plan"
    value="y"$ckplan onchange="change_calc_plot_plan();"
    /> Plot plan in chart</label>
<br />
<input type="hidden" name="du" id="du" value="$ui->{display_unit}" />
<input type="hidden" name="dc" id="dc" value="$ui->{decimal_character}" />
<input type="hidden" name="eu" id="eu" value="$ui->{energy_unit}" />
<input type="hidden" name="s" value="$session->{session_id}" />
<input type="submit" name="q=save_dietcalc" value=" Save " />
 
<input type="reset" onclick="unsavedChanges = 0;" value=" Reset " />
 
<input type="submit" name="q=account" value=" Cancel " />
</p>
◇
```

Macro referenced in 261.

12.1.32.8 Calculate dependent variables from primary variables

We take the initial weight, weight goal, and daily calorie balance as the primary variables; changes made by the user in other quantities are always reflected in them. Starting from these variables, then, we compute all the dependent variables for the form.

⟨ Calculate dependent variables from primary variables 270b ⟩ ≡

```
$calc_calorie_balance = (-500 * ENERGY_CONVERSION->[ENERGY_CALORIE][$calc_energy_unit]) if $calc_calorie_balar

    $calc_weight_change = $calc_goal_weight - $calc_start_weight;
    $calc_weight_week = (($calc_calorie_balance * ENERGY_CONVERSION->[$calc_energy_unit][ENERGY_CALORIE]) * 7)
    $calc_weeks = round($calc_weight_change / $calc_weight_week);
    $calc_months = round((($calc_weight_change / $calc_weight_week) * 7.0) / 30.44));
    $calc_end_date = $calc_start_date + ($calc_weeks * 7.0 * 24.0 * 60.0 * 60.0);
◇
```

Macro referenced in 261, 272.

12.1.32.9 Override diet calculator primary fields from form fields

If JavaScript is not enabled, we may arrive here because the user has pressed the "Update" button without having changed any fields. In this circumstance we need to override the primary fields from the user object with those in the form fields.

⟨ Override diet calculator primary fields from form fields 271 ⟩ ≡

```
        if ($CGIargs{q} eq 'update_dietcalc') {
            if (defined($CGIargs{r_calc_energy_unit})) {
                $calc_energy_unit = $CGIargs{r_calc_energy_unit};
            }
            if (defined($CGIargs{r_calc_calorie_balance})) {
                $calc_calorie_balance = $CGIargs{r_calc_calorie_balance};
                $calc_calorie_balance =~ s/,/./g;
            }

            if (defined($CGIargs{r_calc_weight_unit})) {
                $calc_weight_unit = $CGIargs{r_calc_weight_unit};
            }
            if (defined($CGIargs{r_calc_start_weight})) {
                my $w = $CGIargs{r_calc_start_weight};
                $w =~ s/,/./g;
                #   If specification is stones and pounds, convert to pounds
                if (($w ne '') && ($calc_weight_unit == WEIGHT_STONE)) {
                    if ($w =~ m/\s*(\d+)\s+([\d\.]+)/) {
                        $w = ($1 * 14) + $2;
                    }
                }
                $calc_start_weight = $w *
                    HDiet::monthlog::WEIGHT_CONVERSION->[$CGIargs{r_calc_weight_unit}][$calc_weight_unit];
            }

            if (defined($CGIargs{r_calc_goal_weight})) {
                my $w = $CGIargs{r_calc_goal_weight};
                $w =~ s/,/./g;
                #   If specification is stones and pounds, convert to pounds
                if (($w ne '') && ($calc_weight_unit == WEIGHT_STONE)) {
                    if ($w =~ m/\s*(\d+)\s+([\d\.]+)/) {
                        $w = ($1 * 14) + $2;
                    }
                }
                $calc_goal_weight = $w *
                    HDiet::monthlog::WEIGHT_CONVERSION->[$CGIargs{r_calc_weight_unit}][$calc_weight_unit];
            }

            if (defined($CGIargs{r_calc_start_date})) {
                $calc_start_date = jd_to_unix_time($CGIargs{r_calc_start_date});
            }

            if (defined($CGIargs{r_plot_plan})) {
                $plot_diet_plan = defined($CGIargs{r_plot_plan}) ? 1 : 0;
            }
        }
```

◇

Macro referenced in 262.

12.1.32.10 Perform static update of diet calculator

This code is invoked when the diet calculator form is invoked by an "`update_dietcalc`" transaction. This occurs when a user deprived of JavaScript changes a field and then presses the "Update" button to propagate the changes through the diet calculator form. We compare the fields with their initial values saved in the hidden "`r_`" fields and process the first one found to have changed.

⟨ Perform static update of diet calculator 272 ⟩ ≡

```
my $nschanges = 0;

⟨ Static change to energy unit 273a ⟩
⟨ Static change to daily balance 273b ⟩

⟨ Static change to weight unit 273c ⟩
⟨ Static change to initial weight 274a ⟩

⟨ Static change to goal weight 274b ⟩

⟨ Static change to desired weight change 274c ⟩

⟨ Static change to weight change per week 275a ⟩

⟨ Static change to diet duration in weeks 275b ⟩
⟨ Static change to diet duration in months 276a ⟩

⟨ Static change to start date 276b ⟩

⟨ Static change to end date 277 ⟩

if ($nschanges > 0) {
    ⟨ Calculate dependent variables from primary variables 270b ⟩
    ⟨ Generate array of years for diet calculator selection 264a ⟩
    if ($nschanges > 1) {
        push(@goofs, "Warning: you have changed more than one field in the
Diet Calculator before pressing the &ldquo;Update&rdquo; button.
This may result in unintended changes.  Please press the
&ldquo;Update&rdquo; button after each change to a field");
    }
}
```

◇

Macro referenced in 261.

12.1.32.10.1 Static change to energy unit When the user changes the energy unit, the daily energy balance is converted to the equivalent value in the newly chosen unit.

⟨ Static change to energy unit 273a ⟩ ≡

```
if ($CGIargs{calc_energy_unit} ne $CGIargs{r_calc_energy_unit}) {
    $calc_energy_unit = $CGIargs{calc_energy_unit};
    $calc_calorie_balance = round($calc_calorie_balance *
        ENERGY_CONVERSION->[$CGIargs{r_calc_energy_unit}][$calc_energy_unit]);
    @eunit = ('', '');
    $eunit[$calc_energy_unit] = ' selected="selected"';
    $nschanges++;
}
```

◇

Macro referenced in 272.

12.1.32.10.2 Static change to daily balance The daily energy balance is a primary quantity; changes to it are handled by the computation of derived quantities.

⟨ Static change to daily balance 273b ⟩ ≡

```
if ($CGIargs{calc_calorie_balance} ne $CGIargs{r_calc_calorie_balance}) {
    if ($CGIargs{calc_calorie_balance} =~ m/^\s*([\+\-]?\d+([\.,]\d*)?)\s*$/) {
        $calc_calorie_balance = $1;
        $calc_calorie_balance =~ s/,/./g;
        $calc_calorie_balance = round($calc_calorie_balance);
        $nschanges++;
    } else {
        push(@goofs, "Invalid daily balance");
    }
}
```

◇

Macro referenced in 272.

12.1.32.10.3 Static change to weight unit If the user changes the weight unit, the initial and goal weight are converted to the equivalent in the new unit.

⟨ Static change to weight unit 273c ⟩ ≡

```
if ($CGIargs{calc_weight_unit} ne $CGIargs{r_calc_weight_unit}) {
    $calc_weight_unit = $CGIargs{calc_weight_unit};
    $calc_start_weight *= HDiet::monthlog::WEIGHT_CONVERSION->[$CGIargs{r_calc_weight_unit}][$CGIargs{calc
    $calc_goal_weight *= HDiet::monthlog::WEIGHT_CONVERSION->[$CGIargs{r_calc_weight_unit}][$CGIargs{calc_
    @wunit = ('', '', '');
    $wunit[$calc_weight_unit] = ' selected="selected"';
    $nschanges++;
}
```

◇

Macro referenced in 272.

12.1.32.10.4 Static change to initial weight The initial weight is a primary quantity; changes to it are handled by the regular recalculation of derived quantities.

⟨ Static change to initial weight 274a ⟩ ≡

```
        if ($CGIargs{calc_start_weight} ne $CGIargs{r_calc_start_weight}) {
            my $w = parseWeight($CGIargs{calc_start_weight}, $calc_weight_unit);
            if (defined($w)) {
                $calc_start_weight = $w;
            } else {
                push(@goofs, "Invalid initial weight");
            }
            $nschanges++;
        }
```

◇

Macro referenced in 272.

12.1.32.10.5 Static change to goal weight The goal weight is a primary quantity; changes to it are handled by the regular recalculation of derived quantities.

⟨ Static change to goal weight 274b ⟩ ≡

```
        if ($CGIargs{calc_goal_weight} ne $CGIargs{r_calc_goal_weight}) {
            my $w = parseWeight($CGIargs{calc_goal_weight}, $calc_weight_unit);
            if (defined($w)) {
                $calc_goal_weight = $w;
            } else {
                push(@goofs, "Invalid goal weight");
            }
            $nschanges++;
        }
```

◇

Macro referenced in 272.

12.1.32.10.6 Static change to desired weight change Changing the desired weight change adjusts the goal weight to be the specified difference from the initial weight.

⟨ Static change to desired weight change 274c ⟩ ≡

```
        if ($CGIargs{calc_weight_change} ne $CGIargs{r_calc_weight_change}) {
            my $w = parseSignedWeight($CGIargs{calc_weight_change}, $calc_weight_unit);
            if (defined($w)) {
                $calc_goal_weight = $calc_start_weight + $w;
            } else {
                push(@goofs, "Invalid desired weight change");
            }
            $nschanges++;
        }
```

◇

Macro referenced in 272.

12.1.32.10.7 Static change to weight change per week Changing the desired weight change per week adjusts the energy balance to achieve the requested gain or loss.

⟨ Static change to weight change per week 275a ⟩ ≡

```
        if ($CGIargs{calc_weight_week} ne $CGIargs{r_calc_weight_week}) {
            my $w = parseSignedWeight($CGIargs{calc_weight_week}, $calc_weight_unit);
            if (defined($w)) {
                $calc_calorie_balance = round(($w * CALORIES_PER_WEIGHT_UNIT->[$calc_weight_unit]) /
                    ((ENERGY_CONVERSION->[$calc_energy_unit][ENERGY_CALORIE]) * 7));
            } else {
                push(@goofs, "Invalid weight change per week");
            }
            $nschanges++;
        }
    ◇
```

Macro referenced in 272.

12.1.32.10.8 Static change to diet duration in weeks Changing the diet duration in weeks adjusts the energy balance to achieve the specified duration.

⟨ Static change to diet duration in weeks 275b ⟩ ≡

```
        if ($CGIargs{calc_weeks} ne $CGIargs{r_calc_weeks}) {
            my $ddw = -1;
            if ($CGIargs{calc_weeks} =~ m/^\s*(\d+)\s*$/) {
                if ($1 > 0) {
                    $ddw = $1;
                }
            }
            if ($ddw > 0) {
                $calc_calorie_balance = round((($calc_weight_change / $ddw) *
                    (CALORIES_PER_WEIGHT_UNIT->[$calc_weight_unit] / 7)));
            } else {
                push(@goofs, "Invalid diet duration in weeks");
            }
            $nschanges++;
        }
    ◇
```

Macro referenced in 272.

12.1.32.10.9 Static change to diet duration in months Changing the diet duration in months adjusts the energy balance to achieve the specified duration. We use the mean length of months in the Gregorian calendar.

⟨ Static change to diet duration in months 276a ⟩ ≡

```
        if ($CGIargs{calc_months} ne $CGIargs{r_calc_months}) {
            my $ddm = -1;
            if ($CGIargs{calc_months} =~ m/^\s*(\d+)\s*$/) {
                if ($1 > 0) {
                    $ddm = $1;
                }
            }
            if ($ddm > 0) {
                $calc_calorie_balance = int((($calc_weight_change / $ddm) *
                    (CALORIES_PER_WEIGHT_UNIT->[$calc_weight_unit] / 30.44)));
            } else {
                push(@goofs, "Invalid diet duration in months");
            }
            $nschanges++;
        }
    ◇
```

Macro referenced in 272.

12.1.32.10.10 Static change to start date The start date is a primary quantity; changing it simply causes the end date to reflect the diet duration.

⟨ Static change to start date 276b ⟩ ≡

```
        if (gregorian_to_jd($CGIargs{from_y}, $CGIargs{from_m}, $CGIargs{from_d}) !=
                $CGIargs{r_calc_start_date}) {
            $calc_start_date = jd_to_unix_time(gregorian_to_jd($CGIargs{from_y}, $CGIargs{from_m}, $CGIargs{from_c
            $nschanges++;
        }
    ◇
```

Macro referenced in 272.

12.1.32.10.11 Static change to end date When the user changes the end date, we adjust the daily energy balance to achieve the desired diet duration.

⟨ Static change to end date 277 ⟩ ≡

```
        if (gregorian_to_jd($CGIargs{to_y}, $CGIargs{to_m}, $CGIargs{to_d}) !=
                $CGIargs{r_calc_end_date}) {
            my $ed = jd_to_unix_time(gregorian_to_jd($CGIargs{to_y}, $CGIargs{to_m}, $CGIargs{to_d}));
            if ($ed > $calc_start_date) {
                $calc_calorie_balance = round((($calc_weight_change /
                    (($ed - $calc_start_date) / (24 * 60 * 60))) *
                        CALORIES_PER_WEIGHT_UNIT->[$calc_weight_unit]) /
                        ENERGY_CONVERSION->[$calc_energy_unit][ENERGY_CALORIE]);
            } else {
                push(@goofs, "End date must be after start date");
            }
            $nschanges++;
        }
```

◇

Macro referenced in 272.

12.1.33 Save diet calculator settings

We save the settings from the diet calculator form in the `user` object and update the user database, then re-display the diet calculator with the new settings.

⟨ Save diet calculator settings 278 ⟩ ≡

```
⟨ Retrieve active session information 193 ⟩
⟨ Retrieve user account information 194 ⟩

$CGIargs{calc_calorie_balance} =~ s/,/./g;
$ui->{calc_calorie_balance} = $CGIargs{calc_calorie_balance} *
    ENERGY_CONVERSION->[$CGIargs{calc_energy_unit}][ENERGY_CALORIE];

$CGIargs{calc_start_weight} =~ s/,/./g;
my $w = $CGIargs{calc_start_weight};
#   If specification is stones and pounds, convert to pounds
if (($w ne '') && ($CGIargs{calc_weight_unit} == WEIGHT_STONE)) {
    if ($w =~ m/\s*(\d+)\s+([\d\.]+)/) {
        $w = ($1 * 14) + $2;
    }
}
$ui->{calc_start_weight} = $w *
    HDiet::monthlog::WEIGHT_CONVERSION->[$CGIargs{calc_weight_unit}][WEIGHT_KILOGRAM];

$CGIargs{calc_goal_weight} =~ s/,/./g;
$w = $CGIargs{calc_goal_weight};
#   If specification is stones and pounds, convert to pounds
if (($w ne '') && ($CGIargs{calc_weight_unit} == WEIGHT_STONE)) {
    if ($w =~ m/\s*(\d+)\s+([\d\.]+)/) {
        $w = ($1 * 14) + $2;
    }
}
$ui->{calc_goal_weight} = $w *
    HDiet::monthlog::WEIGHT_CONVERSION->[$CGIargs{calc_weight_unit}][WEIGHT_KILOGRAM];
$ui->{calc_start_date} = jd_to_unix_time(gregorian_to_jd($CGIargs{from_y}, $CGIargs{from_m}, $CGIargs{fro
$ui->{plot_diet_plan} = defined($CGIargs{plot_plan}) ? 1 : 0;

if (!$readOnly) {
    ⟨ Update user account information 293 ⟩
    append_history($user_file_name, 15);
    update_last_transaction($user_file_name);
}
$CGIargs{q} = 'dietcalc';
next;
```

◇

Macro referenced in 169.

12.1.34 Request historical chart

The "histreq" query displays the Historical Chart Request form, which allows the user to specify the date range to be charted in a variety of ways. The result is a page with much the same format which includes a stateless request to generate the chart, which is embedded in the page.

⟨ Request historical chart 279 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    my @years = $ui->enumerateYears();

    write_XHTML_prologue($fh, $homeBase, "Chart Workshop", undef, $session->{handheld});
    generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, "Chart", undef, $browse_public, $tim
    ⟨ Generate assumed identity notification 185 ⟩

    if ($#years >= 0) {
        ⟨ Emit historical chart request form 280 ⟩
    } else {
        print $fh <<"EOD";
<h2>You have no log entries!  You must enter weight logs
    before you can request historical charts.</h2>
EOD
    }

    write_XHTML_epilogue($fh, $homeBase);
    update_last_transaction($user_file_name);
◇
```

Macro referenced in 169.

12.1.34.1 Emit historical chart request form

The HTML form used to request a historical chart is written to the output stream. If the request was invoked with arguments specified by a previous invocation, the fields in the request are preset to those of the last request.

⟨ Emit historical chart request form 280 ⟩ ≡

```
    print $fh <<"EOD";
<h1 class="c">Chart Workshop</h1>
EOD

    ⟨ Determine first and last days in database 240 ⟩

    ⟨ Process custom interval specification, if any 253 ⟩

    ⟨ Embed historical chart image in request/result page 281b ⟩

    ⟨ Set variables to default to previous request settings 283, ... ⟩

    print $fh <<"EOD";
<form id="Hdiet_histchart" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩
<p class="mlog_buttons">
<b>Last:</b>
    <label><input type="radio" name="period" value="m"$percheck{m} /> Month</label>
    <label><input type="radio" name="period" value="q"$percheck{q} /> Quarter</label>
    <label><input type="radio" name="period" value="h"$percheck{h} /> Six months</label>
    <label><input type="radio" name="period" value="y"$percheck{y} /> Year</label>

    <br />
EOD

    ⟨ Generate form fields for custom chart interval 281a ⟩

    print $fh <<"EOD";
<br />

<b><label for="size">Chart size:</label></b> <select name="size" id="size">
EOD

    ⟨ Generate option items for chart sizes 286b ⟩

    print $fh <<"EOD";
</select>
<br />
<label><input type="checkbox" name="print" value="y"$ckprint  /> Printer friendly</label>
 
<label><input type="checkbox" name="mono" value="y"$ckmono  /> Monochrome</label>
<br />

<input type="hidden" name="s" value="$session->{session_id}" />
<input type="submit" name="q=histreq" value=" Update " />
 
<input type="reset" value=" Reset " />
</p>
</form>
EOD
◇
```

Macro referenced in 279.

12.1.34.2 Generate form fields for custom chart interval

The following fields allow the user to request a chart covering any interval in the database. The selection fields are initialised to produce a chart of all days in the database.

⟨ Generate form fields for custom chart interval 281a ⟩ ≡

```
    print $fh <<"EOD";
<label><input type="radio" name="period" value="c"$percheck{c} /> <b>Custom</b></label>
EOD

    ⟨ Custom start and end date selection boxes 258a ⟩
◇
```

Macro referenced in 280.

12.1.34.2.1 Embed historical chart image in request/result page The historical chart is generated in a "stateless" fashion by embedding an image which invokes this application with a "histchart" transaction, specifying all of the arguments with which we were invoked.

⟨ Embed historical chart image in request/result page 281b ⟩ ≡

```
    my ($chart_w, $chart_h) = (800, 600);

    ($chart_w, $chart_h) = (320, 240) if $session->{handheld};

    if (defined($CGIargs{size})) {
        if ($CGIargs{size} =~ m/^(\d+)x(\d+)$/) {
            ($chart_w, $chart_h) = ($1, $2);
            $chart_w = min(1600, max($chart_w, 320));
            $chart_h = min(1600, max($chart_h, 200));
        }
    }

    my $chart_args = "width=$chart_w&amp;height=$chart_h";

    ⟨ Determine range of dates to plot in historical chart 282 ⟩

#    my ($start_y, $start_m, $start_d) = ($CGIargs{from_y}, $CGIargs{from_m}, $CGIargs{from_d});
#    my ($end_y, $end_m, $end_d) = ($CGIargs{to_y}, $CGIargs{to_m}, $CGIargs{to_d});
    $chart_args .= "&amp;start=$start_y-$start_m-$start_d&amp;end=$end_y-$end_m-$end_d";

    my $modeArgs = '';
    $modeArgs .= '&amp;print=y' if $CGIargs{print};
    $modeArgs .= '&amp;mono=y' if $CGIargs{mono};

    ⟨ Define "cachebuster" argument 199b ⟩

    print $fh <<"EOD";
<p class="centred">
<img src="⟨ URL to invoke this program 12a ⟩?q=histchart&amp;s=$session->{session_id}&amp;$chart_args$modeArgs&amp
     width="$chart_w" height="$chart_h" alt="Historical chart" />
</p>
EOD
◇
```

Macro referenced in 280.

12.1.34.2.1.1 Determine range of dates to plot in historical chart Use the setting of the "`period`" radio button to select the start and end dates for the historical chart. In no circumstances will the date span we determine exceed that of log entries in the database.

⟨ Determine range of dates to plot in historical chart 282 ⟩ ≡

```
my ($start_y, $start_m, $start_d, $end_y, $end_m, $end_d);

my $period = $CGIargs{period};
$period = 'q' if !$period;

if ($custom) {
    ($start_y, $start_m, $start_d) = ($cust_start_y, $cust_start_m, $cust_start_d);
    ($end_y, $end_m, $end_d) = ($cust_end_y, $cust_end_m, $cust_end_d);
} else {
    my %periodIntervals = (
                            'm' => -1,
                            'q' => -3,
                            'h' => -6,
                            'y' => -12
                          );
    my $pint = $periodIntervals{$period};
    if (!$pint) {
        $period = $CGIargs{period} = 'q';
        $pint = $periodIntervals{$period};
    }

    my ($f_y, $f_m, $f_d) = $hist->firstDayOfInterval($l_y, $l_m, $l_d, $pint);
    my $f_jd = gregorian_to_jd($f_y, $f_m, $f_d);

    if ($f_jd < $s_jd) {
        ($f_y, $f_m, $f_d, $f_jd) = ($s_y, $s_m, $s_d, $s_jd);
    }

    ($start_y, $start_m, $start_d) = ($f_y, $f_m, $f_d);
    ($end_y, $end_m, $end_d) = ($l_y, $l_m, $l_d);
}
```

◇

Macro referenced in 281b.

12.1.34.2.2 Set variables to default to previous request settings If we have been invoked from a previous historical chart request, examing the CGI query arguments and preset the variables which will initialise the fields in our request form to those of the last request. The user perceived this as the request parameters being persistent, with the ability to adjust them and refine the request.

This code is used by several forms; fields which aren't defined by a form will be ignored.

We have a variety of standard periods as well as "Custom", which is defined by the fields which follow.

⟨ Set variables to default to previous request settings 283 ⟩ ≡

```
my %percheck = ( 'm', '', 'q', '', 'h', '', 'y', '', 'c', '' );

if (defined($CGIargs{period})) {
    $percheck{$CGIargs{period}} = ' checked="checked"';
} else {
    $percheck{q} = ' checked="checked"';
}
```

◇

Macro defined by 283, 284, 285a.
Macro referenced in 237, 245, 247, 252a, 261, 280.

The following selection boxes specify the starting and ending dates of the custom interval.

⟨ Set variables to default to previous request settings 284 ⟩ ≡

```
my (@fy_selected, @ty_selected);
for (my $i = 0; $i <= $#years; $i++) {
    if (defined($CGIargs{from_y}) && ($CGIargs{from_y} eq $years[$i])) {
        $fy_selected[$i] = ' selected="selected"';
    } else {
        $fy_selected[$i] = '';
    }
    if (defined($CGIargs{to_y}) && ($CGIargs{to_y} eq $years[$i])) {
        $ty_selected[$i] = ' selected="selected"';
    } else {
        $ty_selected[$i] = '';
    }
}

my (@fm_selected, @tm_selected);
for (my $i = 1; $i <= 12; $i++) {
    if (defined($CGIargs{from_m}) && ($CGIargs{from_m} == $i)) {
        $fm_selected[$i] = ' selected="selected"';
    } else {
        $fm_selected[$i] = '';
    }
    if (defined($CGIargs{to_m}) && ($CGIargs{to_m} == $i)) {
        $tm_selected[$i] = ' selected="selected"';
    } else {
        $tm_selected[$i] = '';
    }
}

my (@fd_selected, @td_selected);
for (my $i = 1; $i <= 31; $i++) {
    if (defined($CGIargs{from_d}) && ($CGIargs{from_d} == $i)) {
        $fd_selected[$i] = ' selected="selected"';
    } else {
        $fd_selected[$i] = '';
    }
    if (defined($CGIargs{to_d}) && ($CGIargs{to_d} == $i)) {
        $td_selected[$i] = ' selected="selected"';
    } else {
        $td_selected[$i] = '';
    }
}
```

◇

Macro defined by 283, 284, 285a.
Macro referenced in 237, 245, 247, 252a, 261, 280.

The chart size is set by a selection box, which defaults fo 800×600 for screen presentation and 320×240 for handheld devices. In fact, chart generation can handle arbitrary size specifications, but for the moment we only allow the choices in the selection field.

⟨ Set variables to default to previous request settings 285a ⟩ ≡

```
        my @cs_selected;
        $CGIargs{size} = '800x600' if !defined($CGIargs{size});
        $CGIargs{size} = '320x240' if $session->{handheld};
        for (my $i = 0; $i <= $#chartSizes; $i++) {
            if ($CGIargs{size} eq $chartSizes[$i]) {
                $cs_selected[$i] = ' selected="selected"';
            } else {
                $cs_selected[$i] = '';
            }
        }

        my $ckprint = $CGIargs{print} ? ' checked="checked"' : '';
        my $ckmono = $CGIargs{mono} ? ' checked="checked"' : '';
    ◇
```

Macro defined by 283, 284, 285a.
Macro referenced in 237, 245, 247, 252a, 261, 280.

12.1.34.2.3 Generate option items for years in database Crank out an XHTML `<option>` item for each year present in the database. This is used to supply the values for the from and to `<select>` fields in the request form.

⟨ Generate option items for years in database 285b ⟩ ≡

```
        for (my $i = 0; $i <= $#years; $i++) {
            print $fh <<"EOD";
            <option$fy_selected[$i]>$years[$i]</option>
    EOD
        }
    ◇
```

Macro referenced in 259, 260.

12.1.34.2.4 Generate option items for months Output XHTML `<option>` item for each month name. These may be twiddle with JavaScript to hide months for which no log exists in the specified year.

⟨ Generate option items for months 285c ⟩ ≡

```
        for (my $i = 1; $i <= $#monthNames; $i++) {
            print $fh <<"EOD";
            <option id="$mid$i" value="$i"$fm_selected[$i]>$monthNames[$i]</option>
    EOD
        }
    ◇
```

Macro referenced in 259, 260, 344, 345.

12.1.34.2.5 Generate option items for days Output XHTML `<option>` item for each day of the month. These may be twiddled with JavaScript to hide days which are not present in the given month of the specified year.

⟨ Generate option items for days 286a ⟩ ≡

```
        for (my $i = 1; $i <= 31; $i++) {
            print $fh <<"EOD";
            <option id="$did$i"$fd_selected[$i]>$i</option>
    EOD
        }
    ◇
```

Macro referenced in 259, 260, 344, 345.

12.1.34.2.6 Generate option items for chart sizes Output XHTML `<option>` item for each day of the month. These may be twiddle with JavaScript to hide days which are not present in the given month of the specified year.

⟨ Generate option items for chart sizes 286b ⟩ ≡

```
        for (my $i = 0; $i <= $#chartSizes; $i++) {
            my $cs = $chartSizes[$i];
            $cs =~ s/x/&times;/;
            print $fh <<"EOD";
            <option id="cs$chartSizes[$i]" value="$chartSizes[$i]"$cs_selected[$i]>$cs</option>
    EOD
        }
    ◇
```

Macro referenced in 280.

12.1.35 Generate historical chart

A historical chart is generated and returned as an in-line PNG image by the "histchart" query. Optional "width" and "height" arguments allow specifying the chart size. If omitted, the chart defaults to 640×480 pixels.

⟨ Generate historical chart 287 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    my $hc = HDiet::history->new($ui, $user_file_name);

    ⟨ Specify Content-type for PNG image 250b ⟩

    $CGIargs{width} = 640 if !defined $CGIargs{width};
    $CGIargs{height} = 480 if !defined $CGIargs{height};

    my ($start_date, $end_date);

###### FIXME:  Sanity check arguments and default if not specified.
    $start_date = $CGIargs{start} if defined($CGIargs{start});
    $end_date = $CGIargs{end} if defined($CGIargs{end});

    my @dcalc;
    if ($ui->{plot_diet_plan}) {
        @dcalc = $ui->dietPlanLimits();
    }

    $hc->drawChart($fh, $start_date, $end_date, $CGIargs{width}, $CGIargs{height}, \@dcalc,
        $CGIargs{print}, $CGIargs{mono});

    update_last_transaction($user_file_name);
    exit(0);
◇
```

Macro referenced in 168.

12.1.36 Create new user account request

Return the form a user fills in to request the creation of a new account. If this request is made from the login page, any name already entered in the login form is filled into the corresponding field on the new account form.

⟨ Create new user account request 288a ⟩ ≡

```
    write_XHTML_prologue($fh, $homeBase, "Create New Account", undef, $CGIargs{HDiet_handheld});

    print $fh <<"EOD";
<h1 class="c">Create New Account</h1>
<form id="Hdiet_newacct" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩
EOD

    ⟨ Propagate handheld setting to subsequent forms 288b ⟩

    my $u = HDiet::user->new($CGIargs{HDiet_username});
    $u->new_account_form($fh);
    print $fh <<"EOD";
<p class="mlog_buttons">
<input type="hidden" name="q" value="new_account" />
<input type="submit" name="login" value=" Create Account " tabindex="19" />
 
<input type="reset" value=" Clear Form " tabindex="20" />
</p>
</form>
EOD
    write_XHTML_epilogue($fh, $homeBase);
◇
```

Macro referenced in 173.

12.1.36.1 Propagate handheld setting to subsequent forms

Since we are not logged in at this point, we don't have a session in which to remember whether the user is accessing us from a handheld device; we'll have been passed the `HDiet_handheld` setting from the main login form checkbox, but we must explicitly propagate this setting to forms we invoke. This is done by including a hidden form field which sets `HDiet_handheld` if that variable is set in our own arguments.

⟨ Propagate handheld setting to subsequent forms 288b ⟩ ≡

```
    if ($CGIargs{HDiet_handheld}) {
        print $fh <<"EOD";
<div><input type="hidden" name="HDiet_handheld" value="y" /></div>
EOD
    }
◇
```

Macro referenced in 187, 191, 288a, 294.

12.1.37 Process new user account request

Return the form a user fills in to request the creation of a new account. If this request is made from the login page, any name already entered in the login form is filled into the corresponding field on the new account form.

⟨ Process new user account request 289 ⟩ ≡

```
        my @goofs;

    if (0) {        # Set to 1 to investigate reports of account creation problems
        open(NOF, ">/tmp/hdiet_newacct_$$.txt");
        use Data::Dumper;
        print(NOF Data::Dumper->Dump([\%CGIargs, \%ENV], ['*CGIargs', '*ENV']));
        close(NOF);

    }

        ⟨ Validate user name for new account 290a ⟩

        ⟨ Validate beta test invitation code 290b ⟩

        if ($CGIargs{HDiet_password} ne $CGIargs{HDiet_rpassword}) {
            push(@goofs, "Password does not match password confirmation");
        } else {
            if (length($CGIargs{HDiet_password}) < 6) {
                push(@goofs, "Password must be at least six characters");
            }
        }
        if ($CGIargs{HDiet_email} eq '') {
            push(@goofs, "E-mail address is blank");
        } else {
            if ($CGIargs{HDiet_email} !~ m/@/) {
                push(@goofs, "E-mail address contains no '\@' sign");
            } else {
                $CGIargs{HDiet_email} =~ m/@(.*)$/;
                if (!validMailDomain(encodeDomainName($1))) {
                    my $dn = quoteHTML($1);
                    push(@goofs, "Domain name <tt>$dn</tt> in your E-mail address is invalid");
                }
            }
        }

        if ($#goofs >= 0) {
            ⟨ Report errors in new account request and re-issue form 294 ⟩
        }

        ⟨ Create the new user account 291b ⟩

        ⟨ Cancel used beta test invitation code 291a ⟩

        next;
    ◇
```

Macro referenced in 170b.

12.1.37.1 Validate user name for new account

Verify that the user name is non-blank and doesn't duplicate that of an existing account. We discard trailing white space in user names but permit leading spaces for the kinky crowd who gets off on such esoterica.

⟨ Validate user name for new account 290a ⟩ ≡

```
my $user_file_name;
$CGIargs{HDiet_username} =~ s/\s+$//;
if ($CGIargs{HDiet_username} eq '') {
    push(@goofs, "User name is blank");
} else {
    $user_file_name = quoteUserName($CGIargs{HDiet_username});
    if (-d "⟨ Users Directory 6h ⟩/$user_file_name") {
        push(@goofs, "User name is already taken: please choose another");
    }
}
```

◇

Macro referenced in 289.

12.1.37.2 Validate beta test invitation code

In beta test mode, we require an "invitation code" to create a new account. These invitation codes are generated by an admnistrator task and stored as files in a directory within the database tree. We have the ability to define a "backdoor" code which allows creation of accounts without an invitation code. This allows testing new account creation without using up invitation codes at an excessive rate. Obviously, you shouldn't set the backdoor to something too easy to guess. The best choice is to set the backdoor to the null string; that disables the backdoor entirely.

⟨ Validate beta test invitation code 290b ⟩ ≡

```
my $betaInvitation = '';
if (⟨ Beta test 3e ⟩) {
    if ((⟨ Beta test backdoor 4a ⟩ eq '') ||
        ($CGIargs{HDiet_invitation} ne ⟨ Beta test backdoor 4a ⟩)) {
        if ($CGIargs{HDiet_invitation} eq '') {
            push(@goofs, "Beta test invitation is blank");
        } else {
            $betaInvitation = $CGIargs{HDiet_invitation};
            $betaInvitation =~ s/\W//g;
            if (!(-f "⟨ Beta Test Invitations Directory 7a ⟩/$betaInvitation.hdi")) {
                push(@goofs, "Beta test invitation is invalid or already used");
            }
        }
    }
}
```

◇

Macro referenced in 289.

12.1.37.3 Cancel used beta test invitation code

Once a beta test invitation code has been used to create an account, we cancel it by deleting it from the invitations directory. If the backdoor was used, `$betaInvitation` will remain the null string and this code will be skipped.

⟨ Cancel used beta test invitation code 291a ⟩ ≡

```
if (⟨ Beta test 3e ⟩) {
    if ($betaInvitation ne '') {
        if (!unlink("⟨ Beta Test Invitations Directory 7a ⟩/$betaInvitation.hdi")) {
            print(STDERR "Unable to unlink ⟨ Beta Test Invitations Directory 7a ⟩/$betaInvitation.hdi\n");
        }
        clusterDelete("⟨ Beta Test Invitations Directory 7a ⟩/$betaInvitation.hdi");
    }
}
```

◇

Macro referenced in 289.

12.1.37.4 Create the new user account

Everything appears to be on the up and up. Create the directory for the new user account, write the account information into it, and return the user to the login page so they can access the new account.

⟨ Create the new user account 291b ⟩ ≡

```
if (mkdir("⟨ Users Directory 6h ⟩/$user_file_name")) {
    clusterMkdir("⟨ Users Directory 6h ⟩/$user_file_name");
    my $ui = HDiet::user->new($CGIargs{HDiet_username});
    ⟨ Store settings for user account 292 ⟩
    ⟨ Update user account information 293 ⟩
    $CGIargs{q} = 'login';
} else {
     push(@goofs, "Sorry, somebody else just took that user name: please choose another");
     ⟨ Report errors in new account request and re-issue form 294 ⟩
}
```

◇

Macro referenced in 289.

12.1.37.4.1 Store settings for user account Store the settings from the fields of the account settings form into the user object. The test for a blank password allows this code to be used both for new account creation (where we have already verified the password is non-blank) and for account modification (where a blank password indicates no change).

⟨ Store settings for user account 292 ⟩ ≡

```
if (defined($CGIargs{HDiet_height_cm})) {
    $CGIargs{HDiet_height_cm} =~ s/,/./g;
}
if (defined($CGIargs{HDiet_height_in})) {
    $CGIargs{HDiet_height_in} =~ s/,/./g;
}

if ($CGIargs{HDiet_height_cm} eq '') {
    $CGIargs{HDiet_height_cm} = 0;
    if (($CGIargs{HDiet_height_ft} ne '') || ($CGIargs{HDiet_height_in} ne '')) {
        $CGIargs{HDiet_height_cm} = 2.54 *
            ((($CGIargs{HDiet_height_in} ne '') ? $CGIargs{HDiet_height_in} : 0) +
             ((($CGIargs{HDiet_height_ft} ne '') ? $CGIargs{HDiet_height_ft} * 12 : 0)));
    }
}
if ($CGIargs{HDiet_password} ne '') {
    $ui->{password} = $CGIargs{HDiet_password};
}
$CGIargs{HDiet_dchar} = '.' if ($CGIargs{HDiet_dchar} !~ m/^[\.,]$/);

$ui->{first_name} = $CGIargs{HDiet_namef};
$ui->{last_name} = $CGIargs{HDiet_namel};
$ui->{middle_name} = $CGIargs{HDiet_namem};
$ui->{e_mail} = $CGIargs{HDiet_email};
$ui->{log_unit} = $CGIargs{HDiet_wunit};
$ui->{display_unit} = $CGIargs{HDiet_dunit};
$ui->{energy_unit} = $CGIargs{HDiet_eunit};
$ui->{decimal_character} = $CGIargs{HDiet_dchar};
$ui->{height} = $CGIargs{HDiet_height_cm};
$ui->{account_created} = time() if $ui->{account_created} == 0;
$ui->{last_modification_time} = time();

if ($CGIargs{HDiet_public}) {
    if ((!$ui->{public}) || $CGIargs{HDiet_pubnew}) {
        my $pn = HDiet::pubname->new();
        $pn->assignPublicName($ui);
    }
} else {
    if ($ui->{public}) {
        my $pn = HDiet::pubname->new();
        $pn->deletePublicName($ui);
    }
}
```

◇

Macro referenced in 291b, 297.

12.1.37.4.2 Update user account information Save the user account information in the `UserAccount.hdu` file in the user directory.

⟨ Update user account information 293 ⟩ ≡

```
open(FU, ">:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/UserAccount.hdu") ||
    die("Cannot open user account file ⟨ Users Directory 6h ⟩/$user_file_name/UserAccount.hdu");
$ui->save(\*FU);
close(FU);
clusterCopy("⟨ Users Directory 6h ⟩/$user_file_name/UserAccount.hdu");
```

◇

Macro referenced in 188, 231, 278, 291b, 297.

12.1.37.5 Report errors in new account request and re-issue form

One or more errors were detected in the user's request to create a new account. List the errors, then re-issue the form with the previous values already filled in.

⟨ Report errors in new account request and re-issue form 294 ⟩ ≡

```
    write_XHTML_prologue($fh, $homeBase, "Create New User Account", undef, $CGIargs{HDiet_handheld});
    print $fh <<"EOD";
<h1 class="c">Errors in New Account Request</h1>
EOD
    print $fh <<"EOD";
<p>
The following errors were found in your request to create
a new account.  Please remedy them and try again.
</p>

<ol>
EOD

    for (my $i = 0; $i <= $#goofs; $i++) {
        print($fh "<li>$goofs[$i].</li>\n");
    }
    print $fh <<"EOD";
</ol>
<form id="Hdiet_newacct" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩
EOD

    ⟨ Propagate handheld setting to subsequent forms 288b ⟩

    my $u = HDiet::user->new($CGIargs{HDiet_username});
    $u->{e_mail} = $CGIargs{HDiet_email};
    $u->{first_name} = $CGIargs{HDiet_namef};
    $u->{last_name} = $CGIargs{HDiet_namel};
    $u->{middle_name} = $CGIargs{HDiet_namem};
    $u->{log_unit} = $CGIargs{HDiet_wunit};
    $u->{display_unit} = $CGIargs{HDiet_dunit};
    $u->{energy_unit} = $CGIargs{HDiet_eunit};
    $CGIargs{HDiet_dchar} = '.' if ($CGIargs{HDiet_dchar} !~ m/^[\.,]$/);
    $u->{decimal_character} = $CGIargs{HDiet_dchar};

    $u->new_account_form($fh);
    print $fh <<"EOD";
<p class="mlog_buttons">
<input type="hidden" name="q" value="new_account" />
<input type="submit" name="login" value=" Create Account " tabindex="19" />
 
<input type="reset" value=" Clear Form " tabindex="20" />
</p>
</form>
EOD

    write_XHTML_epilogue($fh, $homeBase);
    last;
◇
```

Macro referenced in 289, 291b.

12.1.38 Modify user account request

This form is returned when a logged in user wishes to modify the settings for their account.

⟨ Modify user account request 295 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    if ($session->{cookie}) {
        ⟨ Reject setting query or change from cookie-based login 296 ⟩
    } else {

        write_XHTML_prologue($fh, $homeBase, "Modify Account Settings", undef, $session->{handheld});
        generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, "Settings", undef, $browse_publi
        ⟨ Generate assumed identity notification 185 ⟩

        print $fh <<"EOD";
<h1 class="c">Modify Account Settings</h1>
<p class="justified">
To change your password, enter the new value in the &ldquo;Password&rdquo;
and &ldquo;Retype password&rdquo; fields; if these fields are left blank,
your password will be unchanged.
</p>
<form id="Hdiet_newacct" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩
EOD
        $ui->new_account_form($fh, 1);
        print $fh <<"EOD";
<p class="mlog_buttons">
<input type="hidden" name="s" value="$session->{session_id}" />
<input type="hidden" name="decimal_character" value="$ui->{decimal_character}" />
<input type="submit" name="q=edit_account" value=" Apply " />
 
<input type="reset" value=" Reset " />
 
<input type="submit" name="q=account" value=" Cancel " />
</p>
</form>
EOD
        write_XHTML_epilogue($fh, $homeBase);

        update_last_transaction($user_file_name);
    }
```

◇

Macro referenced in 170b.

12.1.39 Reject setting query or change from cookie-based login

A user who has logged in with a "Remember me" cookie is not permitted to view or modify the account settings. This reduces the potential damage in the case of a cookie compromise, as the attacker cannot view or change the user's personal information, or change the password and thus render the account inaccessible to its owner. The legitimate user can access the settings by logging out, logging back in with a password, and then navigating to the settings page.

⟨ Reject setting query or change from cookie-based login 296 ⟩ ≡

```
    write_XHTML_prologue($fh, $homeBase, "Settings Inaccessible", undef, $session->{handheld});
    generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, "Settings", undef, $browse_public, $
    ⟨ Generate assumed identity notification 185 ⟩

        print $fh <<"EOD";
<h1 class="c">Settings Inaccessible</h1>
<p class="justified">
You signed into this session with &ldquo;Remember me&rdquo;.
In the interest of security, the private information in the Settings
page cannot be displayed or changed in such a session.  To access the
Settings page, please sign out and then sign back in with your
user name and password.
</p>
EOD
    write_XHTML_epilogue($fh, $homeBase);

    update_last_transaction($user_file_name);
◇
```

Macro referenced in 295.

12.1.40 Process user account modification

This transaction runs when the user submits the user account modification form. It validates the fields and if all is well updates the user account properties.

⟨ Process user account modification 297 ⟩ ≡

```
my @goofs;

⟨ Retrieve active session information 193 ⟩
⟨ Retrieve user account information 194 ⟩

if ($CGIargs{HDiet_password} ne $CGIargs{HDiet_rpassword}) {
    push(@goofs, "Password does not match password confirmation");
} else {
    if (($CGIargs{HDiet_password} ne '') && (length($CGIargs{HDiet_password})) < 6) {
        push(@goofs, "New password must be at least six characters");
    }
}
if ($CGIargs{HDiet_email} eq '') {
    push(@goofs, "E-mail address is blank");
} else {
    if ($CGIargs{HDiet_email} !~ m/@/) {
        push(@goofs, "E-mail address contains no '\@' sign");
    } else {
        $CGIargs{HDiet_email} =~ m/@(.*)$/;
        if (!validMailDomain(encodeDomainName($1))) {
            my $dn = quoteHTML($1);
            push(@goofs, "Domain name <tt>$dn</tt> in your E-mail address is invalid");
        }
    }
}

if ($#goofs >= 0) {
    ⟨ Report errors in account modification request and re-issue form 299 ⟩
}

if (!$readOnly) {
    ⟨ Log changes to account settings 298 ⟩
    ⟨ Store settings for user account 292 ⟩
    ⟨ Update user account information 293 ⟩
}

write_XHTML_prologue($fh, $homeBase, "Account Settings Changed", undef, $session->{handheld});
generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $timeZ
⟨ Generate assumed identity notification 185 ⟩

print $fh <<"EOD";
<h1 class="c">Account Settings Changed</tt></h1>

<p class="justified">
The settings for your account have been changed per your request.
Please click the link below to return to your account's home page.
</p>

<h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=account&amp;s=$session->{session_id}$tzOff">Return t
EOD
write_XHTML_epilogue($fh, $homeBase);

update_last_transaction($user_file_name);
```

◇

Macro referenced in 170b.

12.1.40.1 Log changes to account settings

Compare the new account settings to those presently in effect and compose a string indicating which have changed, which is appended to the history log.

⟨ Log changes to account settings 298 ⟩ ≡

```
my $settings_changed = '';

$CGIargs{HDiet_height_cm} =~ s/,/./;
$CGIargs{HDiet_height_in} =~ s/,/./;
my $heightcm = $CGIargs{HDiet_height_cm};
if ($heightcm eq '') {
    $heightcm = 0;
    if (($CGIargs{HDiet_height_ft} ne '') || ($CGIargs{HDiet_height_in} ne '')) {
        $heightcm = 2.54 *
            ((($CGIargs{HDiet_height_in} ne '') ? $CGIargs{HDiet_height_in} : 0) +
             ((($CGIargs{HDiet_height_ft} ne '') ? $CGIargs{HDiet_height_ft} * 12 : 0)));
    }
}
if ($CGIargs{HDiet_password} ne '') {
    $settings_changed .= ',Password' if $ui->{password} ne $CGIargs{HDiet_password};
}
$settings_changed .= ',FirstName' if $ui->{first_name} ne $CGIargs{HDiet_namef};
$settings_changed .= ',LastName' if $ui->{last_name} ne $CGIargs{HDiet_namel};
$settings_changed .= ',MiddleName' if $ui->{middle_name} ne $CGIargs{HDiet_namem};
$settings_changed .= ',E-Mail' if $ui->{e_mail} ne $CGIargs{HDiet_email};
$settings_changed .= ',LogUnit' if $ui->{log_unit} ne $CGIargs{HDiet_wunit};
$settings_changed .= ',DisplayUnit' if $ui->{display_unit} ne $CGIargs{HDiet_dunit};
$settings_changed .= ',EnergyUnit' if $ui->{energy_unit} ne $CGIargs{HDiet_eunit};
$settings_changed .= ',DecimalCharacter' if $ui->{decimal_character} ne $CGIargs{HDiet_dchar};
$settings_changed .= ',Height' if $ui->{height} ne $heightcm;
$settings_changed .= ',Public' if $ui->{public} != ($CGIargs{HDiet_public} ? 1 : 0);
$settings_changed .= ',Pubname' if $CGIargs{HDiet_pubnew};

$settings_changed =~ s/^,//;
append_history($user_file_name, 8, $settings_changed);
```

◇

Macro referenced in 297.

12.1.40.2 Report errors in account modification request and re-issue form

One or more errors were detected in the user's request to change account settings. List the errors, then re-issue the form with the previous values already filled in.

⟨ Report errors in account modification request and re-issue form 299 ⟩ ≡

```
    write_XHTML_prologue($fh, $homeBase, "Modify Account Settings", undef, $session->{handheld});
    generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, "Settings", undef, $browse_public, $
    ⟨ Generate assumed identity notification 185 ⟩

    print $fh <<"EOD";
<h1 class="c">Errors in Account Settings</h1>
EOD
    print $fh <<"EOD";
<p>
The following errors were found in your request to change
your account settings.  Please remedy them and try again.
</p>

<ol>
EOD

    for (my $i = 0; $i <= $#goofs; $i++) {
        print($fh "<li>$goofs[$i].</li>\n");
    }
    print $fh <<"EOD";
</ol>
<form id="Hdiet_newacct" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩
EOD

    $ui->new_account_form($fh, 1);
    print $fh <<"EOD";
<p class="mlog_buttons">
<input type="hidden" name="s" value="$session->{session_id}" />
<input type="hidden" name="decimal_character" value="$ui->{decimal_character}" />
<input type="submit" name="q=edit_account" value=" Apply " />
 
<input type="reset" value=" Reset " />
 
<input type="submit" name="q=account" value=" Cancel " />
</p>
</form>

<h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=account&amp;s=$session->{session_id}$tzOff">Back to
EOD

    write_XHTML_epilogue($fh, $homeBase);
    last;
◇
```

Macro referenced in 297.

12.1.41 Forget all persistent logins

Scan the "Remember me" directory and delete all tokens belonging to this user. This has the effect of invalidating all persistent login cookies stored in browsers for this user. The browser will retain the cookie and send it, but since there is no token corresponding to its value, it will be ignored.

⟨ Forget all persistent logins 300 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    ⟨ Obtain list of persistent login tokens 328a ⟩

    write_XHTML_prologue($fh, $homeBase, "Forget Persistent Logins", undef, $session->{handheld});
    generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, "Settings", undef, $browse_public, $
    ⟨ Generate assumed identity notification 185 ⟩

    my $ndel = 0;
#print($fh "<pre>\n");
    for my $f (keys(%cookies)) {
        my $cook = $cookies{$f};
        if ($cook->{login_name} eq $ui->{login_name}) {
#            $cook->describe($fh);
            $ndel += unlink("⟨ Remember Me Directory 11e ⟩/$f.hdr");
            clusterDelete("⟨ Remember Me Directory 11e ⟩/$f.hdr");
        }
    }
#print($fh "</pre>\n");


    print $fh <<"EOD";
<h1 class="c">Forget Persistent Logins</h1>
<p class="justified">
EOD

    if ($ndel > 0) {
        print $fh <<"EOD";
All persistent logins (a total of $ndel) have been forgotten.  You
will have to log in with your name and password on the next session
from all browsers.
EOD
    } else {
        print $fh <<"EOD";
You had no persistent logins.
EOD
    }

    print $fh <<"EOD";
</p>
EOD
    write_XHTML_epilogue($fh, $homeBase);

    update_last_transaction($user_file_name);
    append_history($user_file_name, 18, $ndel);
◇
```

Macro referenced in 170b.

12.1.42 List publicly-visible accounts

Show a list of accounts whose owners have granted read-only browse access to the general public. The form includes buttons which permit visitors to browse the public account of their choice.

⟨ List publicly-visible accounts 301 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    if ($readOnly) {
        my $qun = quoteUserName($real_user_name);
        die("Invalid \"$CGIargs{q}\" transaction attempted by read-only account $qun");
    }

    write_XHTML_prologue($fh, $homeBase, "Browse Public Accounts", undef, $session->{handheld});
    generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $timeZ
    ⟨ Generate assumed identity notification 185 ⟩

    my $acct_category = $CGIargs{acct_category};

    ⟨ Obtain list of public accounts 302 ⟩

    print $fh <<"EOD";
<h1 class="c" style="margin-bottom: 0px;">Browse Public Accounts</h1>

EOD

    my $acct_qual;
    my ($chk_all, $chk_act, $chk_inact) = ('', '', '');
    if (!defined($acct_category) || ($acct_category eq 'all')) {
        print($fh "<h3 class=\"acct_category\">All Public Accounts</h3>\n");
        $acct_qual = '';
        $chk_all = ' selected="selected"';
    } elsif ($acct_category eq 'active') {
        print($fh "<h3 class=\"acct_category\">Active Public Accounts (Updated in the last 30 days)</h3>\n");
        $acct_qual = 'active ';
        $chk_act = ' selected="selected"';
    } elsif ($acct_category eq 'inactive') {
        print($fh "<h3 class=\"acct_category\">Inactive Public Accounts (No update in the last 30 days)</h3>\n
        $acct_qual = 'inactive ';
        $chk_inact = ' selected="selected"';
    }

    print $fh <<"EOD";
<form id="Hdiet_pubacct" ⟨ Form processing action and method 12b ⟩>
    <p class="centred" style="margin-top: 0px; margin-bottom: 4px;">
    <input type="hidden" name="s" value="$session->{session_id}" />
    <select name="acct_category" size="1">
        <option value="active"$chk_act>Active accounts</option>
        <option value="inactive"$chk_inact>Inactive accounts</option>
        <option value="all"$chk_all>All accounts</option>
    </select>
    <input type="submit" name="q=browsepub" value=" View " />
    </p>
</form>

<form id="Hdiet_acctmgr" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩

<p class="mlog_buttons">
<input type="submit" name="q=do_public_browseacct" value=" Access " />
</p>

<table border="border" class="mlog">
```

12.1.42.1 Obtain list of public accounts

Walk through the `Pubname` directory and build a hash of all public accounts. The key to the hash is the name of the account, and the value is the real user name.

⟨ Obtain list of public accounts 302 ⟩ ≡

```
my %accounts;

opendir(CD, "⟨ Public Name Directory 6i ⟩") ||
    die("Cannot open directory ⟨ Public Name Directory 6i ⟩");
for my $f (grep(/.*\.hdp$/, readdir(CD))) {
    open(FU, "<:utf8", "⟨ Public Name Directory 6i ⟩/$f") ||
        die("Cannot open user account directory ⟨ Public Name Directory 6i ⟩/$f");
    my $pn = HDiet::pubname->new();
    $pn->load(\*FU);
    close(FU);
    my $sortcode = $pn->{public_name};
    $accounts{$sortcode} = $pn->{true_name};
}
closedir(CD);
```

◇

Macro referenced in 301.

12.1.42.2 Generate table of public accounts

Iterate over the hash containing the public accounts and generate the XHTML table from which the user can choose the one to access. Each row in the table has a radio button which allows that account to be selected. The table is generated in alphabetical order of public name. The `%accounts` hash uses the public name as the key and contains the true name as the value; the properties of the account are retrieved for each account from the user's account directory.

⟨ Generate table of public accounts 303 ⟩ ≡

```
        if (!defined($acct_category)) {
            $acct_category = 'active';
        }

        for my $n (sort(keys(%accounts))) {
            my $qn = quoteHTML($n);
            my $qun = quoteUserName($accounts{$n});

            if ($acct_category ne 'all') {
                my $lti = time() - last_transaction_time($qun);
                my $month = 30 * 24 * 60 * 60;
                if ((($acct_category eq 'active') && ($lti > $month)) ||
                    (($acct_category eq 'inactive') && ($lti < $month))) {
                    next;
                }
            }

            open(FU, "<:utf8", "⟨ Users Directory 6h ⟩/$qun/UserAccount.hdu") ||
                next;
            my $ui = HDiet::user->new();
            $ui->load(\*FU);
            close(FU);
            my $alink = quoteHTML($n);
            my @acreate = gmtime($ui->{account_created});
            my $acr = sprintf("%04d-%02d-%02d", $acreate[5] + 1900, $acreate[4] + 1, $acreate[3]);
            my @apsince = gmtime($ui->{public_since});
            my $aps = sprintf("%04d-%02d-%02d", $apsince[5] + 1900, $apsince[4] + 1, $apsince[3]);
            my ($wu, $eu) = (HDiet::monthlog::WEIGHT_ABBREVIATIONS->[$ui->{display_unit}],
                             HDiet::monthlog::ENERGY_ABBREVIATIONS->[$ui->{energy_unit}]);
            my @months = $ui->enumerateMonths();
            my $nmonths = $#months + 1;
            $months[0] = '' if $nmonths == 0;

            $accts_displayed++;

            print $fh <<"EOD";
    <tr>
        <td><input type="radio" name="pubacct" value="$alink" /></td>
        <td>$n</td>
        <td>$acr</td>
        <td>$aps</td>
        <td>$wu</td>
        <td>$eu</td>
        <td>$nmonths</td>
        <td>$months[0]</td>
        <td>$months[$#months]</td>
    </tr>
    EOD
        }
    ◇
```

Macro referenced in 301.

12.1.43 Provide browse access to public account

The user has selected a public account to browse. After verifying that there is, in fact, a public account with the name requested, the `browse_name` of the user's session is set to the requested public name. On subsequent transactions, the user will be granted read-only access to the designated public account.

⟨ Provide browse access to public account 304 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    if ($readOnly) {
        my $qun = quoteUserName($real_user_name);
        die("Invalid \"$CGIargs{q}\" transaction attempted by read-only account $qun");
    }

    if (!defined($CGIargs{pubacct})) {
        write_XHTML_prologue($fh, $homeBase, "Invalid Access Request", undef, $session->{handheld});
        generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $t
        ⟨ Generate assumed identity notification 185 ⟩

        print $fh <<"EOD";
<h1 class="c">Invalid Access Request</h1>

<p class="justified">
You entered a request to access a public account, but did not specify which
account you wished to access.
</p>

<h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=browsepub&amp;s=$session->{session_id}$tzOff">Return
EOD
        write_XHTML_epilogue($fh, $homeBase);
        exit(0);
    }

    ⟨ Look up public account and verify it exists 305 ⟩

    $session->{effective_name} = '';
    $session->{browse_name} = $pn->{public_name};
    open(FS, ">:utf8", "⟨ Session Directory 6g ⟩/$session->{session_id}.hds") ||
        die("Cannot create session file ⟨ Session Directory 6g ⟩/$session->{session_id}.hds");
    $session->save(\*FS);
    close(FS);
    clusterCopy("⟨ Session Directory 6g ⟩/$session->{session_id}.hds");
    $CGIargs{q} = 'account';
    next;
```

◇

Macro referenced in 169.

12.1.43.1 Look up public account and verify it exists

The specified public account name is looked up using the `pubname::findPublicName` method. If no such account is found, an XHTML reply page is returned to the user.

⟨ Look up public account and verify it exists 305 ⟩ ≡

```
        my $pn = HDiet::pubname->new();
        if (!defined($pn->findPublicName($CGIargs{pubacct}))) {
            my $qn = quoteHTML($CGIargs{pubacct});
            write_XHTML_prologue($fh, $homeBase, "Invalid Access Request", undef, $session->{handheld});
            generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $t
            ⟨ Generate assumed identity notification 185 ⟩

            print $fh <<"EOD";
    <h1 class="c">Invalid Access Request</h1>

    <p class="justified">
    You requested to access a public account
    &ldquo;<b>$qn</b>&rdquo;, but no such public
    account exists.
    </p>

    <h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=browsepub&amp;s=$session->{session_id}$tzOff">Return
    EOD
            write_XHTML_epilogue($fh, $homeBase);
            exit(0);
        }
    ◇
```

Macro referenced in 304.

12.1.44 Request invitation codes

Request one or more codes used to invite users during the beta test phase. The request form allows the administrator to specify how many codes are to be generated. The codes are returned in a text box on a result form.

⟨ Request invitation codes 306 ⟩ ≡

```
        if (⟨ Beta test 3e ⟩) {
            ⟨ Retrieve active session information 193 ⟩
            ⟨ Retrieve user account information 194 ⟩

            ⟨ Verify that user has administrator privilege 315 ⟩

            write_XHTML_prologue($fh, $homeBase, "Request Invitation Codes", undef, $session->{handheld});
            generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $t
            ⟨ Generate assumed identity notification 185 ⟩

            print $fh <<"EOD";
    <h1 class="c">Request Invitation Codes</h1>

    <form id="Hdiet_invite" ⟨ Form processing action and method 12b ⟩>
    ⟨ Local time zone offset field 372b ⟩

    <p class="mlog_buttons">
    Number of invitations to generate:
    <input type="text" name="ninvite" size="4" maxlength="4" value="1" />
    </p>

    <p class="mlog_buttons">
    <input type="hidden" name="s" value="$session->{session_id}" />
    <input type="submit" name="q=generate_invitations" value=" Generate " />
     
    <input type="reset" value=" Reset " />
    </p>
    </form>
    EOD
            write_XHTML_epilogue($fh, $homeBase);
        }
    ◇
```

Macro referenced in 171.

12.1.45 Generate invitation codes

Generate the requested beta test invitation codes and report them in a text box whence they can be copied and saved for issuance to authorised testers.

⟨ Generate invitation codes 307 ⟩ ≡

```
        if (⟨ Beta test 3e ⟩) {
            ⟨ Retrieve active session information 193 ⟩
            ⟨ Retrieve user account information 194 ⟩

            ⟨ Verify that user has administrator privilege 315 ⟩

            write_XHTML_prologue($fh, $homeBase, "Invitation Codes Generated", undef, $session->{handheld});
            generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $t
            ⟨ Generate assumed identity notification 185 ⟩

            my $ninvite = $CGIargs{ninvite};
            $ninvite = 1 if !$ninvite;
            $ninvite = max(1, min($ninvite, 20));

            print $fh <<"EOD";
    <h1 class="c">Invitation Codes Generated</h1>

    <form id="Hdiet_invgen" ⟨ Form processing action and method 12b ⟩>
    ⟨ Local time zone offset field 372b ⟩
    <input type="hidden" name="ninvite" value="$ninvite" />

    <p class="mlog_buttons">
    <textarea cols="20" rows="$ninvite" name="invitations">
    EOD
    ◇
```

Macro defined by 307, 308.
Macro referenced in 171.

Now we actually generate the requested number of invitation codes and plug them into the results text box.

⟨ Generate invitation codes 308 ⟩ ≡

```
            for (my $i = 0; $i < $ninvite; $i++) {
                my $pw;

                while (1) {
                    #   Generate invitations until we find a unique one
                    $pw = $ui->generatePassword(8,
                            "ABCDEFGHIJKLMNPQRSTUVWXYZ" .
                            "abcdefghjkmnopqrstuvwxyz" .
                            "23456789");
                    if (!(-f "⟨ Beta Test Invitations Directory 7a ⟩/$pw.hdi")) {
                        last;
                    }
                }
                open(FO, ">:utf8", "⟨ Beta Test Invitations Directory 7a ⟩/$pw.hdi") ||
                    die("Cannot create invitation file ⟨ Beta Test Invitations Directory 7a ⟩/$pw.hdi");
                print(FO time() . "\n");
                close(FO);
                clusterCopy("⟨ Beta Test Invitations Directory 7a ⟩/$pw.hdi");
                print($fh quoteHTML($pw), "\n");
            }

            print $fh <<"EOD";
    </textarea>
    </p>

    <p class="mlog_buttons">
    <input type="hidden" name="s" value="$session->{session_id}" />
    <input type="submit" name="q=generate_invitations" value=" Generate More " />
     
    <input type="submit" name="q=account" value=" Done " />
    </p>
    </form>
    EOD
            write_XHTML_epilogue($fh, $homeBase);
        }
```

◇

Macro defined by 307, 308.
Macro referenced in 171.

12.1.46 Display administrator account manager

Display the account manager page, which summaries existing accounts and allows the administrator to perform various functions upon them.

⟨ Display administrator account manager 309 ⟩ ≡

```
⟨ Retrieve active session information 193 ⟩
⟨ Retrieve user account information 194 ⟩

⟨ Verify that user has administrator privilege 315 ⟩

write_XHTML_prologue($fh, $homeBase, "Account Manager", undef, $session->{handheld});
generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $timeZ
⟨ Generate assumed identity notification 185 ⟩

print $fh <<"EOD";
<h1 class="c" style="margin-bottom: 0px;">Account Manager</h1>
EOD

my $acct_qual;
my ($chk_all, $chk_act, $chk_inact) = ('', '', '');

my $acct_category = $CGIargs{acct_category};
if (!defined($acct_category) || ($acct_category eq 'all')) {
    print($fh "<h3 class=\"acct_category\">All Accounts</h3>\n");
    $acct_qual = '';
    $chk_all = ' selected="selected"';
} elsif ($acct_category eq 'active') {
    print($fh "<h3 class=\"acct_category\">Active Accounts (Updated in the last 30 days)</h3>\n");
    $acct_qual = 'active ';
    $chk_act = ' selected="selected"';
} elsif ($acct_category eq 'inactive') {
    print($fh "<h3 class=\"acct_category\">Inactive Accounts (No update in the last 30 days)</h3>\n");
    $acct_qual = 'inactive ';
    $chk_inact = ' selected="selected"';
}

print $fh <<"EOD";
<form id="Hdiet_pubacct" ⟨ Form processing action and method 12b ⟩>
    <p class="centred" style="margin-top: 0px; margin-bottom: 4px;">
    <input type="hidden" name="s" value="$session->{session_id}" />
    <select name="acct_category" size="1">
        <option value="active"$chk_act>Active accounts</option>
        <option value="inactive"$chk_inact>Inactive accounts</option>
        <option value="all"$chk_all>All accounts</option>
    </select>
    <input type="submit" name="q=acctmgr" value=" View " />
    </p>
</form>


<form id="Hdiet_acctmgr" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩
◇
```

Macro defined by 309, 310, 311, 312.
Macro referenced in 171.

The following table lists the accounts. The column headings are defined in the first row.

⟨ Display administrator account manager 310 ⟩ ≡

```
<table border="border" class="mlog">
<tr>
    <th>Sel</th>
    <th>Login</th>
    <th>First</th>
    <th>Mid</th>
    <th>Last</th>
    <th>E-mail</th>
    <th>Created</th>
    <th>Weight</th>
    <th>Energy</th>
    <th>Adm</th>
    <th>Pub</th>
    <th>R/O</th>
    <th>Pubname</th>
    <th>Months</th>
    <th>Start</th>
    <th>Latest</th>
</tr>
EOD
◇
```

Macro defined by 309, 310, 311, 312.
Macro referenced in 171.

Loop through the accounts, sorted in alphabetical order (case-insensitive), and output one account per line.

⟨ Display administrator account manager 311 ⟩ ≡

```
    ⟨ Obtain list of open accounts 313 ⟩

    my ($naccts, $npub) = (0, 0);
    for my $n (sort({ lc($a) cmp lc($b)} keys(%accounts))) {
        my $qn = quoteHTML($n);
        open(FU, "<:utf8", "⟨ Users Directory 6h ⟩/$accounts{$n}/UserAccount.hdu") ||
            die("Cannot open user account directory ⟨ Users Directory 6h ⟩/$accounts{$n}/UserAccount.hdu");
        my $ui = HDiet::user->new();
        $ui->load(\*FU);
        close(FU);
        my $alink = quoteHTML($n);
        my @acreate = gmtime($ui->{account_created});
        my $acr = sprintf("%04d-%02d-%02d", $acreate[5] + 1900, $acreate[4] + 1, $acreate[3]);
        my $qem = quoteHTML($ui->{e_mail});
        my $adm = $ui->{administrator} ? '&#10004;' : '';
        my $pub = $ui->{public} ? '&#10004;' : '';
        my $ronly = $ui->{read_only} ? '&#10004;' : '';
        my @name = (quoteHTML($ui->{first_name}), quoteHTML($ui->{middle_name}),
                    quoteHTML($ui->{last_name}), quoteHTML($ui->{public_name}));
#if ($ui->{log_unit} eq '' || $ui->{display_unit} eq '' || $ui->{energy_unit} eq '') { print(STDERR "Gronk!  $
        my ($wu, $eu) = ((HDiet::monthlog::WEIGHT_ABBREVIATIONS->[$ui->{log_unit}] .
                          "/" . HDiet::monthlog::WEIGHT_ABBREVIATIONS->[$ui->{display_unit}]),
                           HDiet::monthlog::ENERGY_ABBREVIATIONS->[$ui->{energy_unit}]);
        my @months = $ui->enumerateMonths();
        my $nmonths = $#months + 1;
        $months[0] = '' if $nmonths == 0;

        $naccts++;
        $npub++ if $ui->{public};

        print $fh <<"EOD";
<tr>
    <td><input type="radio" name="useracct" value="$alink" /></td>
    <td>$n</td>
    <td>$name[0]</td>
    <td>$name[1]</td>
    <td>$name[2]</td>
    <td>$qem</td>
    <td>$acr</td>
    <td>$wu</td>
    <td>$eu</td>
    <td>$adm</td>
    <td>$pub</td>
    <td>$ronly</td>
    <td>$name[3]</td>
    <td>$nmonths</td>
    <td>$months[0]</td>
    <td>$months[$#months]</td>
</tr>
EOD
    }
```

◇

Macro defined by 309, 310, 311, 312.
Macro referenced in 171.

Generate the controls at the bottom of the account manager form. The password field is used to confirm account deletion or database purge operations.

⟨ Display administrator account manager 312 ⟩ ≡

```
        my $percentPub = int(($npub * 100) / $naccts);

        print $fh <<"EOD";
    </table>

    <p class="acct_summary">
    $naccts accounts, $npub of which ($percentPub%) grant public access.
    </p>

    <p class="mlog_buttons">
    <input type="hidden" name="s" value="$session->{session_id}" />
    <input type="submit" name="q=do_admin_browseacct" value=" Access " />
     
    <input type="submit" name="q=do_admin_delacct" value=" Delete " />
     
    <input type="submit" name="q=do_admin_purgeacct" value=" Purge Logs " />
    </p>

    <p class="mlog_buttons">
    Administrator password:
        <input type="password" name="HDiet_password" size="20"
                   maxlength="⟨ Maximum Text Input Field Length 9f ⟩" value="" />
    </p>

    </form>
    EOD
        write_XHTML_epilogue($fh, $homeBase);
    ◇
```

Macro defined by 309, 310, 311, 312.
Macro referenced in 171.

12.1.46.1 Obtain list of open accounts

Walk through the `Users` directory and build a hash of all open accounts. The key to the hash is the sort code for the account, and the value is the user directory name. If the account category is set to active or inactive, we filter the list of accounts based on whether the last transaction time is less than or greater than 30 days from the present, respectively.

⟨ Obtain list of open accounts 313 ⟩ ≡

```
my %accounts;

if (!defined($acct_category)) {
    $acct_category = 'active';
}

opendir(CD, "⟨ Users Directory 6h ⟩") ||
    die("Cannot open directory ⟨ Users Directory 6h ⟩");
for my $f (grep(!/\.\.?\z/, readdir(CD))) {

    if ($acct_category ne 'all') {
        my $lti = time() - last_transaction_time($f);
        my $month = 30 * 24 * 60 * 60;
        if ((($acct_category eq 'active') && ($lti > $month)) ||
            (($acct_category eq 'inactive') && ($lti < $month))) {
            next;
        }
    }

    open(FU, "<:utf8", "⟨ Users Directory 6h ⟩/$f/UserAccount.hdu") ||
        die("Cannot open user account directory ⟨ Users Directory 6h ⟩/$f/UserAccount.hdu");
    my $ui = HDiet::user->new();
    $ui->load(\*FU);
    close(FU);
    my $sortcode = $ui->{login_name};
    $accounts{$sortcode} = $f;
}
closedir(CD);
```

◇

Macro referenced in 311.

12.1.47 Provide administrator access to user account

The administrator has requested access to a user account. After confirming that this account, indeed, has administrator privileges, we update the session to assume the identity of the requested user account. This provides full read/write access to the account—the administrator can do anything the user can do.

⟨ Provide administrator access to user account 314 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    ⟨ Verify that user has administrator privilege 315 ⟩

    if (!defined($CGIargs{useracct})) {
        write_XHTML_prologue($fh, $homeBase, "Invalid Access Request", undef, $session->{handheld});
        generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $t
        ⟨ Generate assumed identity notification 185 ⟩

        print $fh <<"EOD";
<h1 class="c">Invalid Access Request</h1>

<p class="justified">
You entered a request to access a user account, but did not specify which
account you wished to access.
</p>

<h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=acctmgr&amp;s=$session->{session_id}$tzOff">Return t
EOD
        write_XHTML_epilogue($fh, $homeBase);
        exit(0);
    }

    $user_file_name = quoteUserName($CGIargs{useracct});

    if (!(-d "⟨ Users Directory 6h ⟩/$user_file_name")) {
        write_XHTML_prologue($fh, $homeBase, "Invalid Access Request", undef, $session->{handheld});
        generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $t
        ⟨ Generate assumed identity notification 185 ⟩

        my $qun = quoteHTML($CGIargs{useracct});

        print $fh <<"EOD";
<h1 class="c">Invalid Access Request</h1>

<p class="justified">
You requested access to account <b>$qun</b>, but no such account exists.
</p>

<h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=acctmgr&amp;s=$session->{session_id}$tzOff">Return t
EOD
        write_XHTML_epilogue($fh, $homeBase);
        exit(0);
    }

    $session->{effective_name} = $CGIargs{useracct};
    $session->{browse_name} = '';
    open(FS, ">:utf8", "⟨ Session Directory 6g ⟩/$session->{session_id}.hds") ||
        die("Cannot create session file ⟨ Session Directory 6g ⟩/$session->{session_id}.hds");
    $session->save(\*FS);
    close(FS);
    clusterCopy("⟨ Session Directory 6g ⟩/$session->{session_id}.hds");
    $CGIargs{q} = 'account';
    next;
```

◇

Macro referenced in 171

12.1.48 Verify that user has administrator privilege

The user has requested a transaction which requires administrative privilege. If he doesn't have it, bounce the transaction with a curt reply an send the user scurrying back into the permitted area.

⟨ Verify that user has administrator privilege 315 ⟩ ≡

```
        if (!$ui->{administrator}) {
            write_XHTML_prologue($fh, $homeBase, "Administrator Privilege Required", undef, $session->{handheld});
            generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $t
            ⟨ Generate assumed identity notification 185 ⟩

            print $fh <<"EOD";
    <h1 class="c">Administrator Privilege Required</h1>

    <p class="justified">
    This operation requires administrator privilege, which you do not
    have.  This request from IP address $ENV{REMOTE_ADDR} has been
    logged.
    </p>

    <h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=account&amp;s=$session->{session_id}$tzOff">Return t
    EOD
            write_XHTML_epilogue($fh, $homeBase);

            append_history($user_file_name, 11, $CGIargs{q});
            exit(0);
        }
```

◇

Macro referenced in 306, 307, 309, 314, 316, 318, 320, 323, 327, 329, 331, 341.

12.1.49 Process administrator database purge

The administrator has requested to purge all the logs in a user's account. Verify the administrator password and then delete the logs after backing them up "just in case".

⟨ Process administrator database purge 316 ⟩ ≡

```
        ⟨ Retrieve active session information 193 ⟩
        ⟨ Retrieve user account information 194 ⟩

        ⟨ Verify that user has administrator privilege 315 ⟩

        if (!defined($CGIargs{useracct})) {
            write_XHTML_prologue($fh, $homeBase, "Invalid Access Request", undef, $session->{handheld});
            generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $t
            ⟨ Generate assumed identity notification 185 ⟩

            print $fh <<"EOD";
    <h1 class="c">Invalid Access Request</h1>

    <p class="justified">
    You entered a request to purge a user account's logs, but did not
    specify which account you wished to purge.
    </p>

    <h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=acctmgr&amp;s=$session->{session_id}$tzOff">Return t
    EOD
            write_XHTML_epilogue($fh, $homeBase);
            exit(0);
        }

        ⟨ Validate administrator password 324 ⟩

        write_XHTML_prologue($fh, $homeBase, "Delete User Account", undef, $session->{handheld});
        generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $timeZ
        ⟨ Generate assumed identity notification 185 ⟩
    ◇
```

Macro defined by 316, 317.
Macro referenced in 171.

Validate that a user directory exists and, if so, back up all of the `.hdb` files in it and delete them.

⟨ Process administrator database purge 317 ⟩ ≡

```
    my $qun = quoteHTML($CGIargs{useracct});
    my $aufn = $user_file_name;     # Save administrator's user file name
    $user_file_name = quoteUserName($CGIargs{useracct});

    if (!(-d "⟨ Users Directory 6h ⟩/$user_file_name")) {
        print $fh <<"EOD";
<h3>There is no user account named <b>$qun</b>.</h3>

<h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=acctmgr&amp;s=$session->{session_id}$tzOff">Return t
EOD
    } elsif (is_user_session_open($CGIargs{useracct})) {
       print $fh <<"EOD";
<h3>User <b>$qun</b> has an active session.  You must terminate
it before the database can be purged.</h3>

<h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=sessmgr&amp;s=$session->{session_id}$tzOff">Go to se
<h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=acctmgr&amp;s=$session->{session_id}$tzOff">Return t
EOD
    } else {
        open(FU, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/UserAccount.hdu") ||
            die("Administrator purge logs: cannot open user account file ⟨ Users Directory 6h ⟩/$user_file_name/Us
        my $di = HDiet::user->new();
        $di->load(\*FU);
        close(FU);

        my @months = $di->enumerateMonths();
        my $nmonths = $#months + 1;
        my $mont = 'month' . (($nmonths != 1) ? 's' : '');

        ⟨ Backup user account before destructive operation 362a ⟩

        for my $m (@months) {
            unlink("⟨ Users Directory 6h ⟩/$user_file_name/$m.hdb") ||
               die("Cannot delete log file ⟨ Users Directory 6h ⟩/$user_file_name/$m.hdb");
            clusterDelete("⟨ Users Directory 6h ⟩/$user_file_name/$m.hdb");
#print($fh "<pre>unlink ⟨ Users Directory 6h ⟩/$user_file_name/$m.hdb</pre>\n");
        }

        append_history($user_file_name, 14, $nmonths);

        print $fh <<"EOD";
<h1 class="c">Logs Purged</h1>

<p class="justified">
Purged all $nmonths $mont of logs from user account <b>$qun</b>.
</p>

<h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=acctmgr&amp;s=$session->{session_id}$tzOff">Return t
EOD
    }
    write_XHTML_epilogue($fh, $homeBase);
◇
```

Macro defined by 316, 317.
Macro referenced in 171.

12.1.50 Process administrator account delete

The administrator has requested to delete a user account. Verify that there are logs in the database (if there are, the administrator must first perform a "Purge Logs" operation to delete tiem), and if the administrator password entered to confirm the operation is correct, delete the account.

⟨ Process administrator account delete 318 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    ⟨ Verify that user has administrator privilege 315 ⟩

    if (!defined($CGIargs{useracct})) {
        write_XHTML_prologue($fh, $homeBase, "Invalid Access Request", undef, $session->{handheld});
        generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $t
        ⟨ Generate assumed identity notification 185 ⟩

        print $fh <<"EOD";
<h1 class="c">Invalid Access Request</h1>

<p class="justified">
You entered a request to delete a user account, but did not specify which
account you wished to delete.
</p>

<h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=acctmgr&amp;s=$session->{session_id}$tzOff">Return t
EOD
        write_XHTML_epilogue($fh, $homeBase);
        exit(0);
    }

    ⟨ Validate administrator password 324 ⟩

    write_XHTML_prologue($fh, $homeBase, "Delete User Account", undef, $session->{handheld});
    generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $timeZ
    ⟨ Generate assumed identity notification 185 ⟩
```

◇

Macro defined by 318, 319.
Macro referenced in 171.

Validate that the user directory exists and that no session for this user is open, and that there are no monthly logs in the database for this user. If all these conditions obtain, we make a backup of the user directory, then delete it and all of its contents.

⟨ Process administrator account delete 319 ⟩ ≡

```
        my $qun = quoteHTML($CGIargs{useracct});
        my $aufn = $user_file_name;     # Save administrator's user file name
        $user_file_name = quoteUserName($CGIargs{useracct});

        if (!(-d "⟨ Users Directory 6h ⟩/$user_file_name")) {
            print $fh <<"EOD";
    <h3>There is no user account named <b>$qun</b>.</h3>
    EOD
        } elsif (is_user_session_open($CGIargs{useracct})) {
            print $fh <<"EOD";
    <h3>User <b>$qun</b> has an active session.  You must terminate
    it before the account can be deleted.</h3>

    <h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=sessmgr&amp;s=$session->{session_id}$tzOff">Go to se
    <h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=acctmgr&amp;s=$session->{session_id}$tzOff">Return t
    EOD
        } else {
            open(FU, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/UserAccount.hdu") ||
                die("Administrator delete account: cannot open user account file ⟨ Users Directory 6h ⟩/$user_file_nam
            my $di = HDiet::user->new();
            $di->load(\*FU);
            close(FU);
            my @months = $di->enumerateMonths();
            my $nmonths = $#months + 1;
            my $mont = 'month' . (($nmonths != 1) ? 's' : '');

            if ($nmonths > 0) {
                print $fh <<"EOD";
    <h3>User <b>$qun</b> has $nmonths $mont of logs in the database.
    Before you can delete this account, you must first purge the logs from
    the database.  Return here after the logs have been purged.</h3>

    <h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=acctmgr&amp;s=$session->{session_id}$tzOff">Return t
    EOD
            } else {
                ⟨ Backup user account before destructive operation 362a ⟩
                do_command("rm -rf ⟨ Users Directory 6h ⟩/$user_file_name");
                clusterRecursiveDelete("⟨ Users Directory 6h ⟩/$user_file_name");

                print $fh <<"EOD";
    <h1 class="c">Account Deleted</h1>

    <p class="justified">
    User account <b>$qun</b> has been deleted.
    </p>

    <h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=acctmgr&amp;s=$session->{session_id}$tzOff">Return t
    EOD
            }
        }
        write_XHTML_epilogue($fh, $homeBase);
    ◇
```

Macro defined by 318, 319.
Macro referenced in 171.

12.1.51 Display administrator session manager

Display the session manager page, which summaries open sessions and allows the administrator to perform various functions upon them.

⟨ Display administrator session manager 320 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    ⟨ Verify that user has administrator privilege 315 ⟩

    write_XHTML_prologue($fh, $homeBase, "Session Manager", undef, $session->{handheld});
    generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $timeZ
    ⟨ Generate assumed identity notification 185 ⟩

    ⟨ Obtain list of open sessions 321 ⟩

    print $fh <<"EOD";
<h1 class="c">Session Manager</h1>

<form id="Hdiet_sessmgr" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩

<table border="border" class="mlog">
<tr>
    <th>Sel</th>
    <th>User</th>
    <th>Session Start</th>
    <th>Administering</th>
    <th>Browsing</th>
    <th>R/O</th>
    <th>Handheld</th>
    <th>Cookie</th>
</tr>
EOD

    ⟨ Generate table of open sessions 322 ⟩

    print $fh <<"EOD";
</table>

<p class="mlog_buttons">
<input type="hidden" name="s" value="$session->{session_id}" />
<input type="submit" name="q=do_admin_forceclose" value=" Terminate " />
</p>

<p class="mlog_buttons">
Administrator password:
    <input type="password" name="HDiet_password" size="20"
               maxlength="⟨ Maximum Text Input Field Length 9f ⟩" value="" />
</p>

</form>
EOD
    write_XHTML_epilogue($fh, $homeBase);
◇
```

Macro referenced in 171.

12.1.51.1 Obtain list of open sessions

Walk through the `Sessions` directory and build a hash of all open sessions. The key to the hash is the user name, and the value is the session ID.

⟨ Obtain list of open sessions 321 ⟩ ≡

```
my %sessions;

opendir(CD, "⟨ Session Directory 6g ⟩") ||
    die("Cannot open directory ⟨ Session Directory 6g ⟩");
for my $f (grep(/\w+\.hds/, readdir(CD))) {
    open(FU, "<:utf8", "⟨ Session Directory 6g ⟩/$f") ||
        die("Cannot open session ⟨ Session Directory 6g ⟩/$f");
    my $session = HDiet::session->new();
    $session->load(\*FU);
    close(FU);
    $sessions{$session->{login_name}} = $session->{session_id};
}
closedir(CD);
```

◇

Macro referenced in 320, 323.

12.1.51.2 Generate table of open sessions

Generate a table of active sessions, sorted by the user name (case-insensitive). Each item contains a radio button which can be used to select the session for termination.

⟨ Generate table of open sessions 322 ⟩ ≡

```
        for my $f (sort({ lc($a) cmp lc($b)} keys(%sessions))) {
            open(FU, "<:utf8", "⟨ Session Directory 6g ⟩/$sessions{$f}.hds") ||
                die("Cannot open session ⟨ Session Directory 6g ⟩/$sessions{$f}.hds");
            my $session = HDiet::session->new();
            $session->load(\*FU);
            close(FU);
            my $qun = quoteHTML($f);
            my $alink = quoteHTML($sessions{$f});
            my @sopen = gmtime($session->{login_time});
            my $acr = sprintf("%04d-%02d-%02d %02d:%02d", $sopen[5] + 1900, $sopen[4] + 1, $sopen[3], $sopen[2], $
            my $qef = quoteHTML($session->{effective_name});
            my $qbr = quoteHTML($session->{browse_name});
            my $rocheck = $session->{read_only} ? '&#10004;' : '';
            my $hhcheck = $session->{handheld} ? '&#10004;' : '';
            my $cookiecheck = $session->{cookie} ? '&#10004;' : '';

            print $fh <<"EOD";
    <tr>
        <td><input type="radio" name="sessionid" value="$alink" /></td>
        <td>$qun</td>
        <td>$acr</td>
        <td>$qef</td>
        <td>$qbr</td>
        <td class="centred">$rocheck</td>
        <td class="centred">$hhcheck</td>
        <td class="centred">$cookiecheck</td>
    </tr>
    EOD
        }
    ◇
```

Macro referenced in 320.

12.1.52 Force termination of user session

Force termination of an open user session. The administrator password must have been specified in the request form in order to perform the termination. As with all administrative requests, we "trust no one" and re-verify all aspects of the request. If for some screwball reason there is an open session file in the `Sessions` for a user, but the user's `ActiveSession.hda` points back to a different session, the act of closing the session will clean this up—both the bogus session and the one the account points back to will be deleted.

⟨ Force termination of user session 323 ⟩ ≡

```
⟨ Retrieve active session information 193 ⟩
⟨ Retrieve user account information 194 ⟩

⟨ Verify that user has administrator privilege 315 ⟩

⟨ Confirm a session is selected 325 ⟩

⟨ Validate specified session 326 ⟩

⟨ Obtain list of open sessions 321 ⟩

my $user = '';
for my $f (sort(keys(%sessions))) {
    if ($sessions{$f} eq $CGIargs{sessionid}) {
        $user = $f;
        last;
    }
}

⟨ Validate administrator password 324 ⟩

my $qun = quoteHTML($user);
my $aufn = $user_file_name;     # Save administrator's user file name
$user_file_name = quoteUserName($user);

⟨ Close previous session if still open 176a ⟩

#   On the off possibility that there is a discrepancy between the
#   session pointer in the Sessions directory and the back pointer
#   in the Users directory, if the session close above did not
#   delete the open session file, manually delete it now.

if (-f "⟨ Session Directory 6g ⟩/$CGIargs{sessionid}.hds") {
    unlink("⟨ Session Directory 6g ⟩/$CGIargs{sessionid}.hds");
    clusterDelete("⟨ Session Directory 6g ⟩/$CGIargs{sessionid}.hds");
print(STDERR "Deleting bogus open session $CGIargs{sessionid} for user $user_file_name\n");
    }

    append_history($aufn, 13, $user_file_name);

    $CGIargs{q} = 'sessmgr';
    undef($CGIargs{sessionid});
    undef($CGIargs{password});
    next;
```

◇

Macro referenced in 171.

12.1.52.1 Validate administrator password

Potentially destructive administrative operations must be confirmed by manually entering the administrator's password on the request form. Validate the password entered agrees with that of the account under which the administrator is logged in and, if it doesn't, reject the request and log it in the administrator's history file.

⟨ Validate administrator password 324 ⟩ ≡

```
        if ($CGIargs{HDiet_password} ne $ui->{password}) {
            write_XHTML_prologue($fh, $homeBase, "Administrator Password Required", undef, $session->{handheld});
            generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $t
            ⟨ Generate assumed identity notification 185 ⟩

            print $fh <<"EOD";
    <h1 class="c">Administrator Password Required</h1>

    <p class="justified">
    This operation requires confirmation by entering your password.  You
    either failed to enter a password, or the password you entered is
    incorrect.  This request from IP address $ENV{REMOTE_ADDR} has been
    logged.
    </p>

    <h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=account&amp;s=$session->{session_id}$tzOff">Return t
    EOD
            write_XHTML_epilogue($fh, $homeBase);

            append_history($user_file_name, 11, $CGIargs{q});
            exit(0);
        }
    ◇
```

Macro referenced in 316, 318, 323, 329.

12.1.52.2 Confirm a session is selected

Make sure the administrator actually selected a session. If not, reject the request.

⟨ Confirm a session is selected 325 ⟩ ≡

```
        if (!defined($CGIargs{sessionid})) {
            write_XHTML_prologue($fh, $homeBase, "No Session Selected", undef, $session->{handheld});
            generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $t
            ⟨ Generate assumed identity notification 185 ⟩

            print $fh <<"EOD";
    <h1 class="c">No Session Selected</h1>

    <p class="justified">
    You requested to terminate a selection, but failed to specify which
    session you wish to terminate.
    </p>

    <h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=account&amp;s=$session->{session_id}$tzOff">Return t
    EOD
            write_XHTML_epilogue($fh, $homeBase);
            exit(0);
        }
    ◇
```

Macro referenced in 323.

12.1.52.3 Validate specified session

Validate the syntax of the session name specified and confirm that a session by that name exists in the session directory. Syntax checking is essential to prevent malicious traversal of parent directories. We do not give the attacker the satisfaction of knowing we spotted the attempt, but simply report it as an invalid session ID.

⟨ Validate specified session 326 ⟩ ≡

```
        if (($CGIargs{sessionid} !~ m/^[0-9FGJKQW]{40}$/) ||
            (!-f "⟨ Session Directory 6g ⟩/$CGIargs{sessionid}.hds")) {
            write_XHTML_prologue($fh, $homeBase, "No Such Session", undef, $session->{handheld});
            generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $t
            ⟨ Generate assumed identity notification 185 ⟩

            print $fh <<"EOD";
    <h1 class="c">No Such Session</h1>

    <p class="justified">
    You requested to terminate session ID <tt>$CGIargs{sessionid}</tt>, but
    no such session is open.
    </p>

    <h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=account&amp;s=$session->{session_id}$tzOff">Return t
    EOD
            write_XHTML_epilogue($fh, $homeBase);
            exit(0);
        }
    ◇
```

Macro referenced in 323.

12.1.53 Display administrator persistent login manager

Display the persistent manager page, which summaries persistent login cookies and allows the administrator to perform various functions upon them.

⟨ Display administrator persistent login manager 327 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    ⟨ Verify that user has administrator privilege 315 ⟩

    write_XHTML_prologue($fh, $homeBase, "Persistent Login Manager", undef, $session->{handheld});
    generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $timeZ
    ⟨ Generate assumed identity notification 185 ⟩

    ⟨ Obtain list of persistent login tokens 328a ⟩

    print $fh <<"EOD";
<h1 class="c">Persistent Login Manager</h1>

<form id="Hdiet_cookiemgr" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩

<table border="border" class="mlog">
<tr>
    <th>Sel</th>
    <th>User</th>
    <th>Token</th>
    <th>Created</th>
    <th>Expiration</th>
</tr>
EOD

    ⟨ Generate table of persistent logins 328b ⟩

    print $fh <<"EOD";
</table>

<p class="mlog_buttons">
<input type="hidden" name="s" value="$session->{session_id}" />
<input type="submit" name="q=do_admin_delcookie" value=" Delete " />
 
<input type="submit" name="q=cookiemgr" value=" Update " />
</p>

<p class="mlog_buttons">
Administrator password:
    <input type="password" name="HDiet_password" size="20"
               maxlength="⟨ Maximum Text Input Field Length 9f ⟩" value="" />
</p>

</form>
EOD
    write_XHTML_epilogue($fh, $homeBase);
◇
```

Macro referenced in 171.

12.1.53.1 Obtain list of persistent login tokens

Walk through the `RememberMe` directory and build a hash of all persistent login tokens. The key to the hash is the token ID, and the value is the token object.

⟨ Obtain list of persistent login tokens 328a ⟩ ≡

```
    my %cookies;

    opendir(CD, "⟨ Remember Me Directory 11e ⟩") ||
        die("Cannot open directory ⟨ Remember Me Directory 11e ⟩");
    for my $f (grep(/\w+\.hdr/, readdir(CD))) {
        open(FU, "<:utf8", "⟨ Remember Me Directory 11e ⟩/$f") ||
#        open(FU, "<", "⟨ Remember Me Directory 11e ⟩/$f") ||                #### Poison cookie search
           die("Cannot open persistent login ⟨ Remember Me Directory 11e ⟩/$f");
        my $cookie = HDiet::cookie->new();
        $cookie->load(\*FU);
#if ($cookie->{login_name} =~ m/^[ -~]*$/) { next; }                    #### Poison cookie search
        close(FU);
        $cookies{$cookie->{cookie_id}} = $cookie;
    }
    closedir(CD);
◇
```

Macro referenced in 300, 327, 329.

12.1.53.2 Generate table of persistent logins

A table row is generated for each existing persistent logins, sorted by the user name to whom the login belongs, case-insensitive. Each item contains a radio button which can be used to select it for deletion.

⟨ Generate table of persistent logins 328b ⟩ ≡

```
    for my $f (sort({ lc($cookies{$a}->{login_name}) cmp lc($cookies{$b}->{login_name})} keys(%cookies))) {
        my $cook = $cookies{$f};
        my $qtok = quoteHTML($f);
        my $qun = quoteHTML($cook->{login_name});
        my @sopen = gmtime($cook->{login_time});
        my $acr = sprintf("%04d-%02d-%02d %02d:%02d", $sopen[5] + 1900, $sopen[4] + 1, $sopen[3], $sopen[2], $
        @sopen = gmtime($cook->{expiry_time});
        my $aex = sprintf("%04d-%02d-%02d %02d:%02d", $sopen[5] + 1900, $sopen[4] + 1, $sopen[3], $sopen[2], $

        print $fh <<"EOD";
<tr>
    <td><input type="radio" name="cookieid" value="$qtok" /></td>
    <td>$qun</td>
    <td class="monospace">$qtok</td>
    <td>$acr</td>
    <td>$aex</td>
</tr>
EOD
    }
◇
```

Macro referenced in 327.

12.1.54 Delete a persistent login token

Delete a persistent login token. The administrator password must have been specified in the request form in order to perform the termination. As with all administrative requests, we "trust no one" and re-verify all aspects of the request.

⟨ Delete a persistent login token 329 ⟩ ≡

```
⟨ Retrieve active session information 193 ⟩
⟨ Retrieve user account information 194 ⟩

⟨ Verify that user has administrator privilege 315 ⟩

⟨ Confirm a persistent login is selected 330 ⟩

⟨ Obtain list of persistent login tokens 328a ⟩

if (defined($cookies{$CGIargs{cookieid}})) {
    my $cook = $cookies{$CGIargs{cookieid}};

    ⟨ Validate administrator password 324 ⟩

    my $qun = quoteUserName($cook->{login_name});

    if (-f "⟨ Remember Me Directory 11e ⟩/$CGIargs{cookieid}.hdr") {
        unlink("⟨ Remember Me Directory 11e ⟩/$CGIargs{cookieid}.hdr");
        clusterDelete("⟨ Remember Me Directory 11e ⟩/$CGIargs{cookieid}.hdr");
    }

    append_history($user_file_name, 17, "$qun,$cook->{cookie_id}");
} else {
print(STDERR "Bogus delete cookie request for $CGIargs{cookieid}\n");
}

$CGIargs{q} = 'cookiemgr';
undef($CGIargs{cookieid});
undef($CGIargs{password});
next;
```

◇

Macro referenced in 171.

12.1.54.1 Confirm a persistent login is selected

Make sure the administrator actually selected a persistent login token. If not, reject the request.

⟨ Confirm a persistent login is selected 330 ⟩ ≡

```
        if (!defined($CGIargs{cookieid})) {
            write_XHTML_prologue($fh, $homeBase, "No Persistent Login Selected", undef, $session->{handheld});
            generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $t
            ⟨ Generate assumed identity notification 185 ⟩

            print $fh <<"EOD";
    <h1 class="c">No Persistent Login Selected</h1>

    <p class="justified">
    You requested to delete a persistent login, but failed to specify which
    login you wish to terminate.
    </p>

    <h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=account&amp;s=$session->{session_id}$tzOff">Return t
    EOD
            write_XHTML_epilogue($fh, $homeBase);
            exit(0);
        }
    ◇
```

Macro referenced in 329.

12.1.55 Display administrator global statistics

Display the global statistics. This invokes the aggregator to retrieve recent information across all accounts and generates a report of activity, including a histogram of how current user log entries are, aggregate trend values for active account and active public accounts, and the fastest weight gain and loss for all and public accounts. This is an administrator-only page, and hence is free to disclose information for non-public accounts.

⟨ Display administrator global statistics 331 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    ⟨ Verify that user has administrator privilege 315 ⟩

    write_XHTML_prologue($fh, $homeBase, "Global Statistics", undef, $session->{handheld});
    generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $timeZ
    ⟨ Generate assumed identity notification 185 ⟩

    print $fh <<"EOD";
<h1 class="c">Global Statistics</h1>

<form id="Hdiet_globalstats" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩

EOD

    my $hndays = 30;            # Number of days to analyse
    my $mincov = 80;            # Minimum coverage in percent to rank gain/loss

    ⟨ Request log records from the aggregator and compute global statistics 332 ⟩

    ⟨ Generate global statistics for open accounts 333 ⟩

    ⟨ Display global statistics mean trend change 334 ⟩

    ⟨ Compute global statistics gain and loss extrema 335 ⟩
    ⟨ Display global statistics gain and loss extrema 336 ⟩

    ⟨ Display global statistics log update frequency 337 ⟩

    write_XHTML_epilogue($fh, $homeBase);

    ⟨ Receive log records from the aggregator for global statistics 338 ⟩
◇
```

Macro referenced in 171.

12.1.55.1 Request log records from the aggregator and compute global statistics

Most of the actual computation of statistics is actually done in the `receive_aggregated_statistics_records` subroutine below (page 338), which is called by the aggregator for each log item returned. Here we define the variables which will be used in computing the statistics and start the aggregator running. Note that we call `receive_aggregated_statistics_records` with a dummy user account at the end to flush out the last user's statistics.

⟨ Request log records from the aggregator and compute global statistics 332 ⟩ ≡

```
my (@acchist, @pacchist);
my ($acctotal, $pacctotal, $badgetotal) = (0, 0, 0);
my (@ttrend, @pttrend);
my (@ntrend, @nptrend);
my ($minslope, $maxslope) = (1E100, -1E100);
my ($pminslope, $pmaxslope) = (1E100, -1E100);
my ($minslopeuser, $maxslopeuser, $pminslopeuser, $pmaxslopeuser);
my ($minslopecov, $maxslopecov, $pminslopecov, $pmaxslopecov);
my $jdnow = unix_time_to_jd(time());
my ($enowy, $enowm, $enowd) = jd_to_gregorian($jdnow);
$jdnow = gregorian_to_jd($enowy, $enowm, $enowd);
my $jdthen = $jdnow - ($hndays + 1);
my ($lastuser, $lastpubname) = ('', '');
my $lastacc = -1;
my $totuser = 0;
my $agg = HDiet::Aggregator->new(\&receive_aggregated_statistics_records, $ui->{display_unit});
my ($naccts, $npaccts) = $agg->retrieve($jdthen, $jdnow, 0);
my %lu = ( "login_name", $lastuser . "xxx" );
receive_aggregated_statistics_records(\%lu, $jdnow, undef);
```

◇

Macro referenced in 331.

12.1.55.2 Generate global statistics for open accounts

The open account statistics are presented in a table which shows the number of active and inactive accounts for all accounts and just public accounts.

⟨ Generate global statistics for open accounts 333 ⟩ ≡

```
        my ($cumaccts, $pcumaccts) = (0, 0);
        for (my $i = 0; $i <= $hndays; $i++) {
            $acchist[$i] = 0 if !defined($acchist[$i]);
            $pacchist[$i] = 0 if !defined($pacchist[$i]);
            $cumaccts += $acchist[$i];
            $pcumaccts += $pacchist[$i];
        }
        my ($inacccts, $pinaccts) = ($naccts - $cumaccts, $npaccts - $pcumaccts);

        print $fh <<"EOD";
    <h2>Open Accounts</h2>

    <table class="global_stats">
        <tr>
            <th class="v"></th>
            <th>All</th>
            <th>Public</th>
        </tr>

        <tr>
            <th class="l">Active</th>
            <td>$cumaccts</td>
            <td>$pcumaccts</td>
        </tr>

        <tr>
            <th class="l">Inactive</th>
            <td>$inacccts</td>
            <td>$pinaccts</td>
        </tr>

        <tr>
            <th class="l">Total</th>
            <td>$naccts</td>
            <td>$npaccts</td>
        </tr>
    </table>

    <p>
    &ldquo;Active&rdquo; accounts are those with a weight log
    entry in the last $hndays days.  A total of $badgetotal accounts
    have badge generation enabled.
    </p>
    EOD
    ◇
```

Macro referenced in 331.

12.1.55.3 Display global statistics mean trend change

Using the trend values from all accounts with complete logs which span the reporting period, we synthesise an overall mean trend for all users and public users only. This provides a sense of how the user community as a whole is progressing.

⟨ Display global statistics mean trend change 334 ⟩ ≡

```
        my $balunits = HDiet::monthlog::ENERGY_ABBREVIATIONS->[$ui->{energy_unit}] . "/day";
        my $wunits = HDiet::monthlog::DELTA_WEIGHT_ABBREVIATIONS->[$ui->{display_unit}] . "/week";

        my $fitter = HDiet::trendfit->new();
        my $pfitter = HDiet::trendfit->new();
        for (my $i = 1; $i <= $hndays; $i++) {
            $fitter->addPoint($ttrend[$i] / $ntrend[$i]);
            $pfitter->addPoint($pttrend[$i] / $nptrend[$i]);
        }
        my $ttslope = $fitter->fitSlope();
        my $pttslope = $pfitter->fitSlope();

        my $meanslopeweek = gs_snum(sprintf("%.2f", $ttslope * 7));
        my $meanslopebal = gs_snum(sprintf("%.0f ", $ttslope *
                    (HDiet::monthlog::CALORIES_PER_WEIGHT_UNIT->[$ui->{display_unit}] /
                     HDiet::monthlog::CALORIES_PER_ENERGY_UNIT->[$ui->{energy_unit}])));
        my $pmeanslopeweek = gs_snum(sprintf("%.2f", $pttslope * 7));
        my $pmeanslopebal = gs_snum(sprintf("%.0f ", $pttslope *
                    (HDiet::monthlog::CALORIES_PER_WEIGHT_UNIT->[$ui->{display_unit}] /
                     HDiet::monthlog::CALORIES_PER_ENERGY_UNIT->[$ui->{energy_unit}])));

        print $fh <<"EOD";
    <h2>Mean Gain/Loss</h2>

    <table class="global_stats">
        <tr>
            <th colspan="2" class="blr">All Accounts</th>
            <th colspan="2" class="blr">Public Accounts</th>
        </tr>

        <tr>
            <th>$balunits</th>
            <th>$wunits</th>
            <th class="bl">$balunits</th>
            <th>$wunits</th>
        </tr>

        <tr>
            <td>$meanslopebal</td>
            <td>$meanslopeweek</td>
            <td>$pmeanslopebal</td>
            <td>$pmeanslopeweek</td>
        </tr>
    </table>

    <p>
    Only accounts with weight entries in each month in the last
    $hndays days are included.
    </p>
    EOD
    ◇
```

Macro referenced in 331.

12.1.55.4 Compute global statistics gain and loss extrema

While receiving log items from the aggregator, the trend for each user with complete logs for the interval is computed and the minimum and maxinum for all accounts and public accounts is saved. We now edit the fit trend slope to the weekly gain/loss and energy balance to be displayed in the gain and loss extrema table. These values are displayed in the administrator's chosen display weight and energy units.

⟨ Compute global statistics gain and loss extrema 335 ⟩ ≡

```
my $minslopeweek = gs_snum(sprintf("%.2f", $minslope * 7));
my $minslopebal = gs_snum(sprintf("%.0f ", $minslope *
            (HDiet::monthlog::CALORIES_PER_WEIGHT_UNIT->[$ui->{display_unit}] /
             HDiet::monthlog::CALORIES_PER_ENERGY_UNIT->[$ui->{energy_unit}])));
my $qminslopeuser = quoteHTML($minslopeuser);

my $pminslopeweek = gs_snum(sprintf("%.2f", $pminslope * 7));
my $pminslopebal = gs_snum(sprintf("%.0f ", $pminslope *
            (HDiet::monthlog::CALORIES_PER_WEIGHT_UNIT->[$ui->{display_unit}] /
             HDiet::monthlog::CALORIES_PER_ENERGY_UNIT->[$ui->{energy_unit}])));
my $qpminslopeuser = quoteHTML($pminslopeuser);

my $maxslopeweek = gs_snum(sprintf("%.2f", $maxslope * 7));
my $maxslopebal = gs_snum(sprintf("%.0f ", $maxslope *
            (HDiet::monthlog::CALORIES_PER_WEIGHT_UNIT->[$ui->{display_unit}] /
             HDiet::monthlog::CALORIES_PER_ENERGY_UNIT->[$ui->{energy_unit}])));
my $qmaxslopeuser = quoteHTML($maxslopeuser);

my $pmaxslopeweek = gs_snum(sprintf("%.2f", $pmaxslope * 7));
my $pmaxslopebal = gs_snum(sprintf("%.0f ", $pmaxslope *
            (HDiet::monthlog::CALORIES_PER_WEIGHT_UNIT->[$ui->{display_unit}] /
             HDiet::monthlog::CALORIES_PER_ENERGY_UNIT->[$ui->{energy_unit}])));
my $qpmaxslopeuser = quoteHTML($pmaxslopeuser);

sub gs_snum {
    my ($v) = @_;
    $v =~ s/\-/&minus;/;
    $v =~ s/^(\d)/\+$1/;
    return $v;
}
```

◇

Macro referenced in 331.

12.1.55.5 Display global statistics gain and loss extrema

The gain and loss extrema are displayed in a table with separate sections for all accounts and public accounts only.

⟨ Display global statistics gain and loss extrema 336 ⟩ ≡

```
        print $fh <<"EOD";
    <h2>Gain and Loss Extrema</h2>

    <table class="global_stats">
        <tr>
            <th class="v"></th>
            <th colspan="3" class="blr">All Accounts</th>
            <th colspan="3" class="blr">Public Accounts</th>
        </tr>

        <tr>
            <th class="v"></th>
            <th class="bl">Name</th>
            <th>$balunits</th>
            <th>$wunits</th>
            <th class="bl">Name</th>
            <th>$balunits</th>
            <th>$wunits</th>
        </tr>

        <tr>
            <th class="l">Fastest loss</th>
            <td class="c">$qminslopeuser</td>
            <td>$minslopebal</td>
            <td>$minslopeweek</td>
            <td class="c">$qpminslopeuser</td>
            <td>$pminslopebal</td>
            <td>$pminslopeweek</td>
        </tr>


        <tr>
            <th class="l">Fastest gain</th>
            <td class="c">$qmaxslopeuser</td>
            <td>$maxslopebal</td>
            <td>$maxslopeweek</td>
            <td class="c">$qpmaxslopeuser</td>
            <td>$pmaxslopebal</td>
            <td>$pmaxslopeweek</td>
        </tr>
    </table>

    <p>
    Only accounts with $mincov% or more weight entries logged are included.
    </p>
    EOD
    ◇
```

Macro referenced in 331.

12.1.55.6 Display global statistics log update frequency

We display a table showing, for all accounts and public accounts, the number of accounts, percent of class, and cumulative percent of accounts which have been updated in intervals ranging from less than one day to one month or more. This gives a sense of how active the user community is in updating logs.

⟨ Display global statistics log update frequency 337 ⟩ ≡

```
    print $fh <<"EOD";
<h2>Log Update Frequency</h2>

<table class="global_stats">
    <tr>
        <th class="v"></th>
        <th colspan="3" class="blr">All Accounts</th>
        <th colspan="3" class="blr">Public Accounts</th>
    </tr>

    <tr>
        <th class="v">Days</th>
        <th class="bl">Accounts</th>
        <th>Percent</th>
        <th>Cumulative</th>

        <th class="bl">Accounts</th>
        <th>Percent</th>
        <th>Cumulative</th>
    </tr>
EOD

    my ($cum, $pcum) = (0, 0);
    for (my $i = 0; $i <= $hndays; $i++) {
        $acchist[$i] = 0 if !defined($acchist[$i]);
        $pacchist[$i] = 0 if !defined($pacchist[$i]);
        $cum += $acchist[$i];
        $pcum += $pacchist[$i];
        my $si = ($i < 1) ? "&lt;1" : (($i >= $hndays) ? "$hndays+" : $i);
        printf($fh "    <tr><td>%s</td> <td>%d</td> <td>%d%%</td> <td>%d%%</td> " .
                   "<td>%d</td> <td>%d%%</td> <td>%d%%</td></tr>\n",
            $si,
            $acchist[$i], int((($acchist[$i] / $acctotal) * 100) + 0.5),
                int((($cum / $acctotal) * 100) + 0.5),
            $pacchist[$i], int((($pacchist[$i] / $pacctotal) * 100) + 0.5),
                int((($pcum / $pacctotal) * 100) + 0.5));
    }
    printf($fh "    <tr><td>Total</td> <td>%d</td> <td>100%%</td> <td>100%%</td> " .
               "<td>%d</td> <td>100%%</td> <td>100%%</td></tr>\n",
        $acctotal, $pacctotal);

    print $fh <<"EOD";
</table>
</form>
EOD
◇
```

Macro referenced in 331.

12.1.55.7 Receive log records from the aggregator for global statistics

The following subroutine is called by the aggregator for each log item returned. The code assumes that all the records for a given user will be returned together, and that within a user's data, records will be in date order. Separate accumulations are done for all users and public users only, and a "coverage" statistic is computed indicating the percentage of days in the interval for which weight was logged.

⟨ Receive log records from the aggregator for global statistics 338 ⟩ ≡

```
    my $acctrend = 0;
    my $uljd;
    my @utrend;
    my $weightdays = 0;

    sub receive_aggregated_statistics_records {
        my ($user, $jd, $weight, $trend, $rung, $flag, $comment) = @_;

#if ($user->{login_name} eq 'astuemky') {
#    print(STDERR "User $user->{login_name} $jd ", jd_to_RFC_3339_date($jd),
#       " W = $weight  T = $trend  R = $rung  F = $flag  C = $comment\n");
#}
        if (($user->{login_name} ne $lastuser) &&
                defined($weight)) {
            if (($lastuser ne '') && ($lastacc >= 0)) {
                if ($lastacc > $hndays) {
                    $lastacc = $hndays;
                }
                $acchist[$lastacc]++;
                $acctotal++;
                if ($user->{public}) {
                    $pacchist[$lastacc]++;
                    $pacctotal++;
                }

                $badgetotal++ if ((defined($user->{badge_trend})) &&
                    ($user->{badge_trend} != 0));

                if ($acctrend && ($uljd == $jdnow)) {
                    ⟨ Update global statistics overall trend analysis 339 ⟩

#if (($#utrend + 1) == 0) {
#    my $sjd = jd_to_RFC_3339_date($jd);
#    print(STDERR "Utrend zero for user $user->{login_name} at JD $jd, $sjd  Lastuser = $lastuser\n");
#}
                    my $coverage = int((($weightdays / ($#utrend + 1)) * 100) + 0.5);

                    ⟨ Compute global statistics trend analysis for previous user 340 ⟩
                }
            }

            $lastuser = $user->{login_name};
            $lastpubname = $user->{public} ? $user->{public_name} : '';
            $weightdays = 0;
            $totuser++;

            $acctrend = 0;
            if ($jd == $jdthen) {
                $acctrend = 1;
                @utrend = ( );
            }
        }
        if ($acctrend && defined($trend)) {
            push(@utrend, $trend);
        }
        $uljd = $jd;
```

12.1.55.7.1 Update global statistics overall trend analysis If we have complete trend data for the user whose log items we have just completed receiving, add it to the global composite trend arrays (`@ttrend` for all accounts, `@pttrend` for public accounts only). For each day in the trend array, we keep track of the number of items added to the bin. This isn't strictly necessary since we currently only add the user's data if it's complete, but doing so permits loosening this constraint in the future should we judge that wise.

⟨ Update global statistics overall trend analysis 339 ⟩ ≡

```
#print(STDERR "$lastuser trend complete.\n");
    my $ufitter = HDiet::trendfit->new();
    for (my $i = 0; $i <= $#utrend; $i++) {
        $ufitter->addPoint($utrend[$i]);
#print(STDERR "$lastuser $user->{login_name} trend[$i] undefined.\n") if !defined($utrend[$i]);
        $ttrend[$i] += $utrend[$i];
        $ntrend[$i]++;
#print(STDERR "$lastuser trend $i: $ttrend[$i]  $ntrend[$i]\n");
        if ($user->{public}) {
            $pttrend[$i] += $utrend[$i];
            $nptrend[$i]++;
        }
    }
```

◇

Macro referenced in 338.

12.1.55.7.2 Compute global statistics trend analysis for previous user After we've received all of the records for a user, if the coverage (percentage of weights logged in the interval) is greater than our threshold, we fit a linear trend and, if it's greater than the previous maximum or less than the previous mininum, we save the user name as the current fastest gain or loss. This is done separately for all account and for public accounts only.

⟨ Compute global statistics trend analysis for previous user 340 ⟩ ≡

```
        if ($coverage >= $mincov) {
            my $uslope = $ufitter->fitSlope();
            if (($uslope < 0) && ($uslope < $minslope)) {
                $minslope = $uslope;
                $minslopeuser = $lastuser;
                $minslopecov = $coverage;
            }
            if (($uslope > 0) && ($uslope > $maxslope)) {
                $maxslope = $uslope;
                $maxslopeuser = $lastuser;
                $maxslopecov = $coverage;
            }

            if ($lastpubname ne '') {
                if (($uslope < 0) && ($uslope < $pminslope)) {
                    $pminslope = $uslope;
                    $pminslopeuser = $lastpubname;
                    $pminslopecov = $coverage;
                }
                if (($uslope > 0) && ($uslope > $pmaxslope)) {
                    $pmaxslope = $uslope;
                    $pmaxslopeuser = $lastpubname;
                    $pmaxslopecov = $coverage;
                }
            }
        }
```

◇

Macro referenced in 338.

12.1.56 Generate synthetic data for user account

The synthetic data generator allows an administrator to fill demonstration accounts with synthetic data generated by combining a linear trend with a variety of perturbation functions. The form is re-displayed after each generation request to facilitate piecing together multiple sequences of synthetic data.

⟨ Generate synthetic data for user account 341 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    if (!$assumed_identity) {
        ⟨ Verify that user has administrator privilege 315 ⟩
    }

    write_XHTML_prologue($fh, $homeBase, "Synthetic Data Generator", undef, $session->{handheld});
    generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $timeZ
    ⟨ Generate assumed identity notification 185 ⟩

    print $fh <<"EOD";
<h1 class="c">Synthetic Data Generator</h1>

<form id="Hdiet_synthdata" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩

EOD

    my $npert = 5;

    ⟨ Generate synthetic data as specified in form 342 ⟩

    ⟨ Generate synthetic data specification form 343 ⟩

    print $fh <<"EOD";
</form>
EOD

    write_XHTML_epilogue($fh, $homeBase);
```

◇

Macro referenced in 171.

12.1.56.1 Generate synthetic data as specified in form

If we arrived here from a previous instance of this form being submitted, parse and validate the form arguments and generate the synthetic data.

When generating synthetic data for the "flag" field, existing entries are replaced, and the "Percent to fill" field controls the percentage of days which are flagged.

⟨ Generate synthetic data as specified in form 342 ⟩ ≡

```
if (defined($CGIargs{from_y}) && ($CGIargs{from_y} ne '')) {
    my ($from_y, $from_m, $from_d) = ($CGIargs{from_y}, $CGIargs{from_m}, $CGIargs{from_d});
    my ($to_y, $to_m, $to_d) = ($CGIargs{to_y}, $CGIargs{to_m}, $CGIargs{to_d});
    my ($field, $fillfrac, $start_value, $end_value) =
        ($CGIargs{field}, $CGIargs{fill_frac}, $CGIargs{start_value}, $CGIargs{end_value});
    my $format = ($field eq 'weight') ? '%.1f' : '%.0f';

    $start_value =~ s/,/./;
    $end_value =~ s/,/./;

    my @pertarg;

    for (my $n = 1; $n <= $npert; $n++) {
        if (($CGIargs{"pf_$n"} ne '') && $CGIargs{"pm_$n"}) {
            $CGIargs{"pm_$n"} =~ s/,/./;
            $CGIargs{"po_$n"} =~ s/,/./;
            $CGIargs{"pp_$n"} =~ s/,/./;
            push(@pertarg, $CGIargs{"pf_$n"},  $CGIargs{"pm_$n"});
            if ($CGIargs{"pf_$n"} eq 'sine') {
                push(@pertarg, $CGIargs{"po_$n"},  $CGIargs{"pp_$n"});
            }
        }
    }

    my $hist = HDiet::history->new($ui, $user_file_name);

    if ($field eq 'flag') {
        $hist->syntheticData(
                sprintf("%04d-%02d-%02d", $from_y, $from_m, $from_d),
                sprintf("%04d-%02d-%02d", $to_y, $to_m, $to_d),
                $field, 1, 0, 0, '%d');
        $start_value = $end_value = 1;
        $format = '%d';
        @pertarg = ( );
    }

    $hist->syntheticData(
            sprintf("%04d-%02d-%02d", $from_y, $from_m, $from_d),
            sprintf("%04d-%02d-%02d", $to_y, $to_m, $to_d),
            $field, $fillfrac / 100, $start_value, $end_value, $format,
            @pertarg);

    propagate_trend($ui, sprintf("%04d-%02d", $from_y, $from_m), 0);
}
```

◇

Macro referenced in 341.

12.1.56.2 Generate synthetic data specification form

⟨ Generate synthetic data specification form 343 ⟩ ≡

```
    print $fh <<"EOD";
<table class="syndata">
EOD

    ⟨ Synthetic data start date 344 ⟩
    ⟨ Synthetic data end date 345 ⟩

    ⟨ Synthetic data field selection 346a ⟩

    print $fh <<"EOD";
<tr>
    <th class="l">Percent to fill:</th>
    <td colspan="4">
    <input type="text" name="fill_frac" value="100" size="4" maxlength="4" />%
    </td>
</tr>
EOD

    ⟨ Synthetic data start and end values 346b ⟩

    ⟨ Table of perturbation functions 347 ⟩

    print $fh <<"EOD";
<tr>
<td colspan="5" class="c">
    <input type="hidden" name="s" value="$session->{session_id}" />
    <input type="submit" name="q=synthdata" value=" Generate " />
     
    <input type="reset" value=" Reset " />
     
    <input type="submit" name="q=gonque" value=" Cancel " />
</td>
</tr>
</table>
EOD
◇
```

Macro referenced in 341.

12.1.56.2.1 Synthetic data start date These fields specify the date at which the synthetic data generation should start.

⟨ Synthetic data start date 344 ⟩ ≡

```
        my ($ysel, $msel, $dsel) = ("") x 3;
        my (@fm_selected, @fd_selected);

        for (my $i = 1; $i <= 31; $i++) {
            $fd_selected[$i] = '';
        }
        for (my $i = 1; $i <= 12; $i++) {
            $fm_selected[$i] = '';
        }

        print $fh <<"EOD";
    <tr>
        <th class="l">Start date:</th>
        <td colspan="4">
        <input type="text" name="from_y" value="" size="5" maxlength="5" />
        <select name="from_m" id="from_m"$msel>
    EOD

        my $mid = "fm_";
        ⟨ Generate option items for months 285c ⟩

        print $fh <<"EOD";
        </select>
    EOD

        print $fh <<"EOD";
        <select name="from_d" id="from_d"$dsel>
    EOD

        my $did = "fd_";
        ⟨ Generate option items for days 286a ⟩

        print $fh <<"EOD";
        </select>
    EOD

        print $fh <<"EOD";
        </td>
    </tr>
    EOD
    ◇
```

Macro referenced in 343.

12.1.56.2.2 Synthetic data end date These fields specify the date at which the synthetic data generation ends.

⟨ Synthetic data end date 345 ⟩ ≡

```
        print $fh <<"EOD";
    <tr>
        <th class="l">End date:</th>
        <td colspan="4">
        <input type="text" name="to_y" value="" size="5" maxlength="5" />
        <select name="to_m" id="to_m"$msel>
    EOD

        $mid = "tm_";
        ⟨ Generate option items for months 285c ⟩

        print $fh <<"EOD";
        </select>
    EOD

        print $fh <<"EOD";
        <select name="to_d" id="to_d"$dsel>
    EOD

        $did = "td_";
        ⟨ Generate option items for days 286a ⟩

        print $fh <<"EOD";
        </select>
    EOD

        print $fh <<"EOD";
        </td>
    </tr>
    EOD
    ◇
```

Macro referenced in 343.

12.1.56.2.3 Synthetic data field selection This selection box specifies which field is to be filled with the synthetic data. The data format is implicitly selected by the choice of field.

⟨ Synthetic data field selection 346a ⟩ ≡

```
     print $fh <<"EOD";
<tr>
    <th class="l">Field:</th>
    <td colspan="4">
    <select name="field">
        <option value="weight">Weight</option>
        <option value="rung">Exercise rung</option>
        <option value="flag">Flag</option>
    </select>
    </td>
</tr>
EOD
```

◇

Macro referenced in 343.

12.1.56.2.4 Synthetic data start and end values The underlying function for generating synthetic data is a linear trend from the start value to the end value, specified in the following two fields. This deterministic progression may be modified by the perturbation function specified below.

⟨ Synthetic data start and end values 346b ⟩ ≡

```
    print $fh <<"EOD";
<tr>
    <th class="l">Start value:</th>
    <td colspan="4">
    <input type="text" name="start_value" value="" size="6" maxlength="6" />
    </td>
</tr>
EOD

    print $fh <<"EOD";
<tr>
    <th class="l">End value:</th>
    <td colspan="4">
    <input type="text" name="end_value" value="" size="6" maxlength="6" />
    </td>
</tr>
EOD
```

◇

Macro referenced in 343.

12.1.56.2.5 Table of perturbation functions The user can supply a total of `$npert` perturbation functions which are applied to the linear data trend. Blank functions or those with an unspecified or zero range are ignored.

⟨ Table of perturbation functions 347 ⟩ ≡

```
    print $fh <<"EOD";
<tr>
    <td colspan="1"></td>
    <th>Function</th>
    <th>Range</th>
    <th>Period</th>
    <th>Phase</th>
</tr>
EOD
    for (my $p = 1; $p <= $npert; $p++) {
        print $fh <<"EOD";
<tr>
    <th class="l">Perturbation $p:</th>
    <td>
    <select name="pf_$p">
        <option value=""></option>
        <option value="uniform">Uniform</option>
        <option value="gaussian">Gaussian</option>
        <option value="sine">Sinusoidal</option>
    </select>
    </td>
    <td><input type="text" name="pm_$p" value="" size="5" maxlength="5" /></td>
    <td><input type="text" name="po_$p" value="" size="5" maxlength="5" /></td>
    <td><input type="text" name="pp_$p" value="" size="5" maxlength="5" /></td>
</tr>
EOD
    }
```

◇

Macro referenced in 343.

12.1.57 Send a feedback message

Display a form which allows users to send feedback to developers. The feedback is sent to an E-mail address which is never disclosed to users. The user may request a copy be sent to the E-mail address configured for the account.

⟨ Send a feedback message 348 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    write_XHTML_prologue($fh, $homeBase, "Send Feedback", undef, $session->{handheld});
    generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $timeZ
    ⟨ Generate assumed identity notification 185 ⟩

    print $fh <<"EOD";
<h1 class="c">Send Feedback</h1>
EOD

    my ($subject, $category, $message, $from) = ('') x 4;
    my @feedsel;
    if (defined($CGIargs{message})) {
        ⟨ Show preview of message being composed 350 ⟩
    }

    ⟨ Generate feedback message composition form 349 ⟩

    write_XHTML_epilogue($fh, $homeBase);
◇
```

Macro referenced in 169.

12.1.57.1 Generate feedback message composition form

The feedback composition form consists of a table containing the input fields, the checkbox which enables copying the message to the sender, and the action buttons to preview, submit, or cancel the message.

⟨ Generate feedback message composition form 349 ⟩ ≡

```
    $feedsel[$CGIargs{category}] = 1 if defined($CGIargs{category});
    my $ckcopy = defined($CGIargs{copy_sender}) ? ' checked="checked"' : '';
    my $qun = quoteHTML($user_name);
    my $em = $ui->{e_mail};
    if ("$ui->{first_name}$ui->{middle_name}$ui->{last_name}" ne "") {
        my $fname = "$ui->{first_name} $ui->{middle_name} $ui->{last_name}";
        $fname =~ s/\s+/ /g;
        $fname =~ s/^\s+//;
        $fname =~ s/\s+$//;
        $em = "$fname <$em>";
    }
    my $qem = quoteHTML($em);
    print $fh <<"EOD";
<form id="Hdiet_feedback" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩
<table border="border" class="feedback">
<tr>
<th>Name:<br />E-mail:</th> <td>$qun<br />$qem</td>
</tr>
EOD

    ⟨ Enumerate feedback message categories 352 ⟩

    my $qms = quoteHTML($message);
    print $fh <<"EOD";
<tr>
<th>Subject:</th>
<td>
    <input type="text" name="subject" value="$subject" size="64" maxlength="80" />
</td>
</tr>
<tr>
<th class="t">Message:</th>
<td>
    <textarea cols="64" rows="16" name="message">$qms</textarea>
</td>
</tr>
</table>

<p class="mlog_buttons">
<input type="checkbox" name="copy_sender" id="copy_sender"$ckcopy /> <label
    for="copy_sender">Send me a copy of the feedback message</label><br />
<input type="hidden" name="s" value="$session->{session_id}" />
<input type="submit" name="q=feedback" value=" Preview " onclick="return validateFeedback();" />
 
<input type="submit" name="q=send_feedback" value=" Send Feedback " onclick="return validateFeedback();" />
 
<input type="submit" name="q=account" value=" Cancel " />
</p>
</form>
EOD
◇
```

Macro referenced in 348.

12.1.57.2 Show preview of message being composed

If the "`message`" form argument is defined, we were invoked by the "Preview" button from a message being composed. Preset the fields to appear in the form to the values from the invoking form and show the message preview before the composition form.

⟨ Show preview of message being composed 350 ⟩ ≡

```
($subject, $category, $message, $from) =
        ($CGIargs{subject},
         $feedback_categories[$CGIargs{category}],
         $CGIargs{message},
         $ui->{e_mail});

$subject =~ s/[\r\n]/ /g;
$category =~ s/[\r\n]/ /g;
$message =~ s/\r\n/\n/g;
$message =~ s/\n\.\n/\n\. \n/g;

⟨ Show feedback message in reply page 356 ⟩

$feedsel[$CGIargs{category}] = 1;

⟨ Check spelling in subject and message 351 ⟩
```

◇

Macro referenced in 348.

12.1.57.2.1 Check spelling in subject and message Just like the friendly gas stations in the Era of Service—"Free fill-up if we forget to check the oil"—we provide a free spelling check every time the user requests a preview of the message. Someday we'll include a language preference in the `user` object to determine the language against which the message is checked.

⟨ Check spelling in subject and message 351 ⟩ ≡

```
        my $spell = 1;          # We may make this optional some day
        my $spellCmd = ⟨ Command to check spelling 10f ⟩;
        if ($spell && ($spellCmd ne '')) {
            my $sfn = "⟨ Users Directory 6h ⟩/$user_file_name/spell$$.tmp";
            if (open(SP, "|-:utf8", "$spellCmd >$sfn")) {
                print(SP $subject . "\n");
                print(SP $message . "\n");
                close(SP);
                open(SF, "<:utf8", $sfn) ||
                    die("Cannot reopen spelling file $sfn");
                my $pt = '';
                while (<SF>) {
                    $pt .= $_;
                }
                close(SF);
                unlink($sfn);
                $pt =~ s/^\s+//;
                $pt =~ s/\s+$//;
                $pt =~ s/\s+/ /g;
                if ($pt eq '') {
                    print $fh <<"EOD";
    <div class="spell_ok">
    <h4>No Misspelled Words</h4>
    </div>
    EOD
                } else {
                    $pt = quoteHTML(wrapText($pt, ⟨ Maximum line length in feedback E-mail messages 10d ⟩));
                    print $fh <<"EOD";
    <div class="spell_dubieties">
    <h4>Possibly Misspelled Words</h4>
    <pre>$pt
    </pre>
    </div>
    EOD
                }
            }
        }
    ◇
```

Macro referenced in 350.

12.1.57.3 Enumerate feedback message categories

Emit the selection list of feedback message categories. These are simply included as text in the feedback message, so the content of the option tags are used as their values.

⟨ Enumerate feedback message categories 352 ⟩ ≡

```
    print $fh <<"EOD";
<tr>
<th>Category:</th>
<td>
    <select name="category" id="category">
EOD

    for (my $i = 0; $i <= $#feedback_categories; $i++) {
        my $sel = $feedsel[$i] ? ' selected="selected"' : '';
        print($fh "        <option value=\"$i\"$sel>$feedback_categories[$i]</option>\n");
    }
    print $fh <<"EOD";
    </select>
</td>
</tr>
EOD
```

◇

Macro referenced in 349.

12.1.58 Send a message from the feedback form

When the user clicks the "Send Feedback" button in the feedback form, this transaction sends the E-mail message to the designated feedback address.

⟨ Send a message from the feedback form 353 ⟩ ≡

```
⟨ Retrieve active session information 193 ⟩
⟨ Retrieve user account information 194 ⟩

write_XHTML_prologue($fh, $homeBase, "Feedback Sent", undef, $session->{handheld});
generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $timeZ
⟨ Generate assumed identity notification 185 ⟩

my ($subject, $category, $message, $from) =
        ($CGIargs{subject},
         $feedback_categories[$CGIargs{category}],
         $CGIargs{message},
         $ui->{e_mail});

$subject =~ s/[\r\n]/ /g;
$category =~ s/[\r\n]/ /g;
$message =~ s/\r\n/\n/g;
$message =~ s/^\.\n/\. \n/;
$message =~ s/\n\.\n/\n\. \n/g;

if (!$readOnly) {
    ⟨ Send mail to feedback address 354 ⟩

    if ($CGIargs{copy_sender}) {
        ⟨ Send copy to the submitter 355 ⟩
    }
}

    print $fh <<"EOD";
<h1 class="c">Feedback Sent</h1>

<p class="justified">
<b>The following feedback message has been sent.  Thank you
for contributing to the improvement of The Hacker's Diet
<em>Online</em>.</b>
</p>
EOD

    ⟨ Show feedback message in reply page 356 ⟩

    print $fh <<"EOD";
<h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=account&amp;s=$session->{session_id}$tzOff">Back
    to account page</a></h4>
EOD

    write_XHTML_epilogue($fh, $homeBase);

    append_history($user_file_name, 16, $category) if !$readOnly;
◇
```

Macro referenced in 169.

12.1.58.1 Send mail to feedback address

Send the feedback message to the designated feedback E-mail address.

⟨ Send mail to feedback address 354 ⟩ ≡

```
    $from = "⟨ From address for mail sent to users 10b ⟩" if !defined($from);

    my $zto = '⟨ Address for feedback E-mail 10c ⟩';
    my $bn = <<"EOD";
⟨ Build Number 3c ⟩
EOD
    $bn =~ s/\s+$//;
    my $bt = <<"EOD";
⟨ Build Time 3d ⟩
EOD
    $bt =~ s/\s+$//;
    my $browser = defined($ENV{HTTP_USER_AGENT}) ? "\r\nBrowser:  $ENV{HTTP_USER_AGENT}" : '';
    my $fullName;
    if ("$ui->{first_name}$ui->{middle_name}$ui->{last_name}" ne "") {
        $fullName = "$ui->{first_name} $ui->{middle_name} $ui->{last_name}";
        $fullName =~ s/\s+/ /g;
        $fullName =~ s/^\s+//;
        $fullName =~ s/\s+$//;
        $fullName = "\r\nUser:     $fullName";
    }

    open(MAIL, "|-:utf8", "⟨ Path to Invoke Sendmail 10a ⟩",
            "-f$from",
            $zto) ||
        die("Cannot create pipe to ⟨ Path to Invoke Sendmail 10a ⟩");
    print MAIL <<"EOD";
From $from\r
To: $zto\r
Subject: [HackDiet Feedback] $category\r
Content-type: text/plain; charset=utf-8\r
\r
From:     $ui->{login_name} <$from>$fullName\r
Category: $category\r
Subject:  $subject$browser\r
Build:    $bn: $bt\r
\r
$message
.\r
EOD
    close(MAIL);
```

◇

Macro referenced in 353.

12.1.58.2 Send copy to the submitter

If requested, send a copy of the feedback message to the submitter's E-mail address.

⟨ Send copy to the submitter 355 ⟩ ≡

```
    open(MAIL, "|-:utf8", "⟨ Path to Invoke Sendmail 10a ⟩",
            "-f⟨ From address for mail sent to users 10b ⟩",
            $from) ||
        die("Cannot create pipe to ⟨ Path to Invoke Sendmail 10a ⟩");
    print MAIL <<"EOD";
From $from\r
To: $from\r
Subject: [Hacker's Diet Online Feedback] $category\r
Content-type: text/plain; charset=utf-8\r
\r
From:     $ui->{login_name} <$from>\r
Category: $category\r
Subject:  $subject\r
\r
$message
.\r
EOD
    close(MAIL);
```

◇

Macro referenced in 353.

12.1.58.3 Show feedback message in reply page

Format the feedback message as currently composed for inclusion in a Web page. This can be used either for a preview of the message during composition or as a confirmation after the message is sent.

⟨ Show feedback message in reply page 356 ⟩ ≡

```
        my $pt = $CGIargs{message};
        $pt =~ s/\r\n/\n/g;
        my $t = $pt;
        $pt = '';
        if (!($t =~ m/\n$/)) {
            $t .= "\n";
        }
        while ($t =~ s/^(.*\n)//) {
            my $l = $1;
            if (length($l) > ⟨ Maximum line length in feedback E-mail messages 10d ⟩) {
                $l = wrapText($l, ⟨ Maximum line length in feedback E-mail messages 10d ⟩);
            }
            $pt .= $l;
        }
        $pt =~ s/\n\n+$/\n/;
        $pt = quoteHTML($pt);

        my ($qli, $qem, $qcat, $qsub) = (quoteHTML($ui->{login_name}), quoteHTML($ui->{e_mail}),
            quoteHTML($category), quoteHTML($subject));
        print $fh <<"EOD";

    <pre class="preview"><b>From:</b>     $qli &lt;$qem&gt;
    <b>Category:</b> $qcat
    <b>Subject:</b>  $qsub

    $pt</pre>
    EOD
    ◇
```

Macro referenced in 350, 353.

12.1.59 Delete entire log database

Delete all logs for a user. This must be performed before we allow the user to close an account. The user is implored (but not required) to download a backup of the logs before deleting them from the database.

⟨ Delete entire log database 357 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩

    my @months = $ui->enumerateMonths();
    my $nmonths = $#months + 1;
    my $mont = 'month' . (($nmonths != 1) ? 's' : '');

    write_XHTML_prologue($fh, $homeBase, "Delete Entire Database", undef, $session->{handheld});
    generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $timeZ
    ⟨ Generate assumed identity notification 185 ⟩

     print $fh <<"EOD";
<h1 class="c">Delete Entire Log Database</h1>
EOD

    if ($nmonths == 0) {
        print $fh <<"EOD";
<h3>You have no logs in the database!  Either you have never entered
and saved any log items, or you have already deleted your logs.</h3>
EOD
    } else {
        print $fh <<"EOD";
⟨ Emit shrill warning about what is about to transpire 358a ⟩
⟨ Generate form permitting user to back up database 358b ⟩

<form id="Hdiet_wipedb" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩

<p class="justified">
In order to confirm your intention to
<span class="shrill">irreversibly delete</span> your entire
log database, please enter your user name and password in the fields
below, and type the one-time &ldquo;confirmation code&rdquo;
in the box.
</p>
EOD
        ⟨ Generate destructive operation confirmation form 359 ⟩
        print $fh <<"EOD";
<p class="mlog_buttons">
<input type="hidden" name="s" value="$session->{session_id}" />
<input type="hidden" name="c" value="$consig" />
<input type="submit" class="darwin" name="q=do_wipedb" value=" Delete Entire Log Database&#xa;(Cannot be undon
 
<input type="submit" name="q=account" value=" Cancel " />
</p>
</form>
EOD
    }
    write_XHTML_epilogue($fh, $homeBase);
```

◇

Macro referenced in 170a.

12.1.59.1 Emit shrill warning about what is about to transpire

The following message reminds the user what's going to happen if this request is allowed to proceed and beseeches him to download a backup of the database before destroying the online copy.

⟨ Emit shrill warning about what is about to transpire 358a ⟩ ≡

```
<p class="justified">
This page allows you to <span class="shrill">delete your entire log database</span>
of $nmonths $mont from The Hacker's Diet <em>Online</em>.  This operation is
<span class="shrill">irrevocable</span>&mdash;unless you have previously downloaded
a backup copy of your logs, all of the information you have entered
into them will be <span class="shrill">lost forever</span>.  Consequently, before
proceeding, we <span class="shrill">implore you</span> to make a database backup
now by pressing the button below.
</p>
```

◇

Macro referenced in 357.

12.1.59.2 Generate form permitting user to back up database

Having urged the user to back up the database, the following form provides a one-button means of doing so. A backup of the entire database in XML format is automatically selected.

⟨ Generate form permitting user to back up database 358b ⟩ ≡

```
<form id="Hdiet_exportdb" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩
<p class="mlog_buttons">
<input type="hidden" name="s" value="$session->{session_id}" />
<input type="hidden" name="format" value="xml" />
<input type="hidden" name="period" value="a" />

<input type="submit" name="q=do_exportdb" value=" Back Up Entire Log Database " />
</p>
</form>
```

◇

Macro referenced in 357.

12.1.59.3 Generate destructive operation confirmation form

Destructive operations, such as deleting all a user's logs from the database or closing an account, require confirmation by the user. We ask the user to enter their user name, password, and a randomly-generated "confirmation code" before proceeding with such an irrevocable operation.

⟨ Generate destructive operation confirmation form 359 ⟩ ≡

```
    my $concode = $ui->generatePassword(10);
    my $consig = sha1_hex($concode . ⟨ Confirmation signature encoding suffix 4c ⟩);
    $consig =~ tr/a-f/FGJKQW/;

    print $fh <<"EOD";
<table border="border" class="login">
<tr><th>User Name:</th>
    <td><input type="text" name="HDiet_username" size="60"
               maxlength="⟨ Maximum Text Input Field Length 9f ⟩" value="" /></td>
</tr>
<tr><th>Password:</th>
    <td><input type="password" name="HDiet_password" size="60"
               maxlength="⟨ Maximum Text Input Field Length 9f ⟩" value="" /></td>
</tr>
<tr><th>Confirmation:</th>
    <td><input type="text" name="HDiet_confirmation" size="15"
               maxlength="15" value="" />
        <span onmousedown="return false;" onmouseover="return false;"> 
        Enter code <tt><b>$concode</b></tt> in the box at the left.</span></td>
</tr>
</table>
EOD
◇
```

Macro referenced in 357, 363.

12.1.60 Process database delete

Perform the database delete operation requested by the above form. Before undertaking the deletion, we verify that the user name and password given in the request form match those for the account, and that the signature of the confirmation code matches that of the code the user was requested to enter.

⟨ Process database delete 360 ⟩ ≡

```
⟨ Retrieve active session information 193 ⟩
⟨ Retrieve user account information 194 ⟩

write_XHTML_prologue($fh, $homeBase, "Log Database Deletion", undef, $session->{handheld});
generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $timeZ
⟨ Generate assumed identity notification 185 ⟩

$CGIargs{c} = '' if !defined($CGIargs{c});
$CGIargs{HDiet_confirmation} = '' if !$CGIargs{HDiet_confirmation};
my $consig = sha1_hex($CGIargs{HDiet_confirmation} . ⟨ Confirmation signature encoding suffix 4c ⟩);
$consig =~ tr/a-f/FGJKQW/;

if (($CGIargs{HDiet_username} ne $ui->{login_name}) ||
    ($CGIargs{HDiet_password} ne $ui->{password})) {
    ⟨ Reject deletion request when user name and password fail to match 361a ⟩
} elsif ($consig ne $CGIargs{c}) {
    ⟨ Reject deletion request when confirmation code fails to match 361b ⟩
} else {
    if (!$readOnly) {
        ⟨ Backup user account before destructive operation 362a ⟩

        my @months = $ui->enumerateMonths();
        for my $m (@months) {
            unlink("⟨ Users Directory 6h ⟩/$user_file_name/$m.hdb") ||
               die("Cannot delete log file ⟨ Users Directory 6h ⟩/$user_file_name/$m.hdb");
            clusterDelete("⟨ Users Directory 6h ⟩/$user_file_name/$m.hdb");
        }

        append_history($user_file_name, 12);
    }

    print $fh <<"EOD";
⟨ Generate confirmation of database deletion 362b ⟩

<h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=account&amp;s=$session->{session_id}$tzOff">Back to
EOD
    }
    write_XHTML_epilogue($fh, $homeBase);
◇
```

Macro referenced in 170a.

12.1.60.1 Reject deletion request when user name and password fail to match

If the user name and password entered by the user to confirm the database deletion fail to match, issue a message indicating this is the case and reject the request. A link back to the deletion request page is provided in case the user wishes to try again.

⟨ Reject deletion request when user name and password fail to match 361a ⟩ ≡

```
    print $fh <<"EOD";
<h1 class="c">Log Database Deletion Rejected</h1>

<h3>The User Name and/or Password entered to confirm the log database
deletion did not match those of your user account.</h3>

<h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=wipedb&amp;s=$session->{session_id}$tzOff">Back
    to Delete Log Database Request</a></h4>
EOD
◇
```

Macro referenced in 360.

12.1.60.2 Reject deletion request when confirmation code fails to match

In addition to the user name and password, the user is required to enter a pseudorandomly generated "confirmation code" to proceed with the database deletion. If the code entered fails to match, the request is rejected. A link back to the deletion request allows the user to try again.

⟨ Reject deletion request when confirmation code fails to match 361b ⟩ ≡

```
    print $fh <<"EOD";
<h1 class="c">Log Database Deletion Rejected</h1>

<h3>The confirmation code entered for the deletion request did not match
that given in the request form.</h3>

<h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=wipedb&amp;s=$session->{session_id}$tzOff">Back
    to Delete Log Database Request</a></h4>
EOD
◇
```

Macro referenced in 360.

12.1.60.3 Backup user account before destructive operation

Before performing an operation which will result in irrevocable loss of user account data, if "Backups Directory" is defined, we make a backup of the user account prior to deletion of the data. This allows us to easily restore in cases of inadvertent deletion or errors in the application which cause deletion without the user's express intent. These backups should be purged by a CRON job after a decent interval in the interest of user privacy.

⟨ Backup user account before destructive operation 362a ⟩ ≡

```
my $tfn = timeXML(time());
$tfn =~ s/:/./g;            # Avoid idiot tar treating time as hostname
if ("⟨ Backups Directory 7b ⟩" ne '') {
    do_command("( cd ⟨ Backups Directory 7b ⟩; tar cfj ${user_file_name}_" .
        $tfn . ".bz2 -C ../Users $user_file_name )");
    clusterCopy("⟨ Backups Directory 7b ⟩/${user_file_name}_$tfn.bz2");
}
```

◇

Macro referenced in 317, 319, 360, 365.

12.1.60.4 Generate confirmation of database deletion

Confirm to the user that the databases have been deleted and provide links to proceed to account closure or, in the case of second thoughts, database restoration.

⟨ Generate confirmation of database deletion 362b ⟩ ≡

```
<h1 class="c">All Log Databases Deleted</h1>

<p class="justified">
Pursuant to your request, all logs have been deleted from your database
on The Hacker's Diet <em>Online</em>.  You can now
<a href="⟨ URL to invoke this program 12a ⟩?q=closeaccount&amp;s=$session->{session_id}$tzOff">close your account<
if you wish, or
<a href="⟨ URL to invoke this program 12a ⟩?q=importcsv&amp;s=$session->{session_id}$tzOff">restore your database<
from a backup copy you downloaded
before deleting the database.
</p>
```

◇

Macro referenced in 360.

12.1.61 Close this user account

The user's account is closed, and all account-related data are deleted. Before the account can be closed, the user must delete all logs from the database. We make this a separate operation to bring home to the user just what is being lost.

⟨ Close this user account 363 ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩
    ⟨ Retrieve user account information 194 ⟩


    my @months = $ui->enumerateMonths();
    my $nmonths = $#months + 1;
    my $mont = 'month' . (($nmonths != 1) ? 's' : '');

    write_XHTML_prologue($fh, $homeBase, "Close User Account", undef, $session->{handheld});
    generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $timeZ
    ⟨ Generate assumed identity notification 185 ⟩

     print $fh <<"EOD";
<h1 class="c">Close User Account</h1>
EOD

    if ($nmonths > 0) {
        ⟨ Reject request if logs remain in database 364a ⟩
    } else {

        my $qun = quoteHTML($ui->{login_name});
        print $fh <<"EOD";
⟨ Warn user about consequences of closing account 364b ⟩

<form id="Hdiet_wipedb" ⟨ Form processing action and method 12b ⟩>
⟨ Local time zone offset field 372b ⟩

<p class="justified">
If you wish to proceed with closing your account, please
confirm by entering your user name and password in the fields
below, and type the one-time &ldquo;confirmation code&rdquo;
in the box.
</p>
EOD
        ⟨ Generate destructive operation confirmation form 359 ⟩

        print $fh <<"EOD";
<p class="mlog_buttons">
<input type="hidden" name="s" value="$session->{session_id}" />
<input type="hidden" name="c" value="$consig" />
<input type="submit" class="darwin" name="q=do_closeaccount" value=" Close User Account&#xa;(Cannot be undone
 
<input type="submit" name="q=account" value=" Cancel " />
</p>
</form>
EOD
    }
    write_XHTML_epilogue($fh, $homeBase);
◇
```

Macro referenced in 170a.

12.1.61.1 Reject request if logs remain in database

One or more monthly logs remain in the database. Reject the account close request and issue a message explaining why.

⟨ Reject request if logs remain in database 364a ⟩ ≡

```
        print $fh <<"EOD";
<h3>You have $nmonths $mont of logs in the database.  Before you can
close your account, you must
<a href="⟨ URL to invoke this program 12a ⟩?q=wipedb&amp;s=$session->{session_id}$tzOff">delete
all of your logs</a> from the database.  Return here after the logs have been
deleted.</h3>
EOD
◇
```

Macro referenced in 363.

12.1.61.2 Warn user about consequences of closing account

The following message reminds the user that closing the account discards all preference settings and makes the account name available to other users.

⟨ Warn user about consequences of closing account 364b ⟩ ≡

```
<p class="justified">
This page allows you to <span class="shrill">close your account</span>
on The Hacker's Diet <em>Online</em>.  This will discard all the
preferences you have specified for your account and make your
present user name &ldquo;<b>$qun</b>&rdquo; available for
creation of a new account by another person.  Note that there
is no charge for maintaining an account, and that data are kept
in your account indefinitely even if your account is inactive.
</p>
◇
```

Macro referenced in 363.

12.1.62 Process user account close

The user has entered a close account request. Validate the user name, password, and confirmation code and, if all is well, delete everything associated with the account. Note that we must repeat the check for all logs having been deleted first, since a malicious user may cobble up a direct transaction request which bypasses the check in the usual request page above, or save a request from earlier, then transmit it after new logs have been created.

⟨ Process user account close 365 ⟩ ≡

```
⟨ Retrieve active session information 193 ⟩
⟨ Retrieve user account information 194 ⟩

write_XHTML_prologue($fh, $homeBase, "User Account Close", undef, $session->{handheld});
generate_XHTML_navigation_bar($fh, $homeBase, $session->{session_id}, undef, undef, $browse_public, $timeZ
⟨ Generate assumed identity notification 185 ⟩

$CGIargs{c} = '' if !defined($CGIargs{c});
$CGIargs{HDiet_confirmation} = '' if !$CGIargs{HDiet_confirmation};
my $consig = sha1_hex($CGIargs{HDiet_confirmation} . ⟨ Confirmation signature encoding suffix 4c ⟩);
$consig =~ tr/a-f/FGJKQW/;

if (($CGIargs{HDiet_username} ne $ui->{login_name}) ||
    ($CGIargs{HDiet_password} ne $ui->{password})) {
    ⟨ Reject account close for user name or password mismatch 366a ⟩
} elsif ($consig ne $CGIargs{c}) {
    ⟨ Reject account close for confirmation code mismatch 366b ⟩
} else {
    my @months = $ui->enumerateMonths();
    my $nmonths = $#months + 1;
    if ($nmonths > 0) {
        ⟨ Reject account close if logs remain in the database 367a ⟩
    } else {
        if (!$readOnly) {
            #   Delete active session file
            unlink("⟨ Session Directory 6g ⟩/$CGIargs{s}.hds");
            clusterDelete("⟨ Session Directory 6g ⟩/$CGIargs{s}.hds");
            unlink("⟨ Users Directory 6h ⟩/$user_file_name/ActiveSession.hda");
            clusterDelete("⟨ Users Directory 6h ⟩/$user_file_name/ActiveSession.hda");

            ⟨ Backup user account before destructive operation 362a ⟩

            #   At this point the user is logged out.  We can now delete
            #   the user directory and all its contents.
            do_command("rm -rf ⟨ Users Directory 6h ⟩/$user_file_name");
            clusterRecursiveDelete("⟨ Users Directory 6h ⟩/$user_file_name");
        }

        print $fh <<"EOD";
<h1 class="c">Account Closed</h1>

<p class="justified">
Pursuant to your request, your account
on The Hacker's Diet <em>Online</em> has been closed.  You can now
<a href="⟨ URL to invoke this program 12a ⟩/">log into another account</a>
if you wish, or
<a href="⟨ URL to invoke this program 12a ⟩?q=validate_user&amp;new=new_account$tzOff">create
a new account</a>.  Otherwise, thank you for participating and farewell!
</p>
EOD
    }
}
write_XHTML_epilogue($fh, $homeBase);
```

◇

Macro referenced in 170a.

12.1.62.1 Reject account close for user name or password mismatch

The user is required to confirm the account close by entering their user name and password. If the values entered do not match those of the account, the request is rejected.

⟨ Reject account close for user name or password mismatch 366a ⟩ ≡

```
        print $fh <<"EOD";
    <h1 class="c">User Account Close Rejected</h1>

    <h3>The User Name and/or Password entered to confirm the account
    close did not match those of your user account.</h3>

    <h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=closeaccount&amp;s=$session->{session_id}$tzOff">Bac
        to Close User Account Request</a></h4>
    EOD
    ◇
```

Macro referenced in 365.

12.1.62.2 Reject account close for confirmation code mismatch

To confirm the account close operation, the user is required to enter a randomly generated "confirmation code". If the code entered does not match that supplied in the close account form, the request is rejected.

⟨ Reject account close for confirmation code mismatch 366b ⟩ ≡

```
        print $fh <<"EOD";
    <h1 class="c">User Account Close Rejected</h1>

    <h3>The confirmation code entered for the account close request did not match
    that given in the request form.</h3>

    <h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=closeaccount&amp;s=$session->{session_id}$tzOff">Bac
        to Close User Account Request</a></h4>
    EOD
    ◇
```

Macro referenced in 365.

12.1.62.3 Reject account close if logs remain in the database

All monthly logs must be deleted from the database before an account may be closed. If logs remain, reject the request. This will usually be caught when the account close request form is displayed, but we must double-check here because the user may have directly crafted a transaction which bypasses the request form or recorded a previous request made before creating one or more new logs.

⟨ Reject account close if logs remain in the database 367a ⟩ ≡

```
    my $mont = 'month' . (($nmonths != 1) ? 's' : '');
    print $fh <<"EOD";
<h1 class="c">User Account Close Rejected</h1>

<h3>You have $nmonths $mont of logs in the database.  Before you can
close your account, you must
<a href="⟨ URL to invoke this program 12a ⟩?q=wipedb&amp;s=$session->{session_id}$tzOff">delete
all of your logs</a> from the database.  Return here after the logs have been
deleted.</h3>

<h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=closeaccount&amp;s=$session->{session_id}$tzOff">Bac
    to Close User Account Request</a></h4>
EOD
◇
```

Macro referenced in 365.

12.1.63 Generate test output page

The following code generates the HTML result when the CGI program is invoked with the "test" query. The contents of this page will vary as we use it to test various facilities under development.

⟨ Generate test output page 367b ⟩ ≡

```
◇
```

Macro referenced in 169.

12.1.64 Emit diagnostic for undefined query

⟨ Emit diagnostic for undefined query 367c ⟩ ≡

```
    ⟨ Retrieve active session information 193 ⟩

    write_XHTML_prologue($fh, $homeBase, "Undefined Query", undef, $session->{handheld});
    print $fh <<"EOD";
<h1 class="c">Undefined query: <tt>$CGIargs{q}</tt></h1>
<h4 class="nav"><a href="⟨ URL to invoke this program 12a ⟩?q=account&amp;s=$session->{session_id}$tzOff">Back to
<pre>
EOD
    ⟨ Dump CGI environment and parsed arguments 368a ⟩
    print $fh <<"EOD";
</pre>
EOD
    write_XHTML_epilogue($fh, $homeBase);
◇
```

Macro referenced in 169.

12.1.64.1 Dump CGI environment and parsed arguments

The `Data::Dumper` facility is used to dump the environment variables passed when we were invoked as a CGI program and the hash of arguments into which we parsed the arguments. If you use this in within HTML output, be sure it's within a `pre` sequence or the browser will mangle its rendering.

⟨ Dump CGI environment and parsed arguments 368a ⟩ ≡

```
    use Data::Dumper;
    print($fh Data::Dumper->Dump([\%CGIargs, \%ENV], ['*CGIargs', '*ENV']));
◇
```

Macro referenced in 367c.

12.1.65 JavaScript debugging console

⟨ JavaScript debugging console 368b ⟩ ≡

```
print $fh <<"EOD";
                      <h3>Debugging Console</h3>

<form id="debugging_console" action="#" onsubmit="return false;">

<p class="mlog_buttons">
<textarea id="log" rows="6" cols="80">
</textarea>
<br />
<input type="button" value=" Clear " onclick="document.getElementById('debugging_console').log.value = '';" />
 
<input type="button" value=" Test " onclick="TestSomething();" />
</p>
</form>

<h4 class="nav"><a href="javascript:">JavaScript Console</a></h4>
EOD
◇
```

Macro never referenced.

12.1.66 Template

⟨ Template 368c ⟩ ≡

```
◇
```

Macro never referenced.

12.2 Global declarations

⟨ Global declarations 369a ⟩ ≡

```
⟨ Perl language modes 369b ⟩

use Time::Local;
use Encode qw(decode_utf8);
use GD;
use Digest::SHA1  qw(sha1_hex);
use XML::LibXML;
use XML::LibXML::Common qw(:w3c);       # XML/DOM node type mnemonics
use HDiet::Julian qw(MONTH_ABBREVIATIONS :DEFAULT);
use Socket qw(inet_aton);
use Sys::Syslog;


use lib "⟨ CGI Support Directory 6a ⟩/Cgi";
use CGI;

use HDiet::Aggregator;
use HDiet::Cluster;
use HDiet::monthlog;
use HDiet::user;
use HDiet::history;
use HDiet::pubname;
use HDiet::session;
use HDiet::cookie;
use HDiet::hdCSV;
use HDiet::html;
use HDiet::xml;

use HDiet::Util::IDNA::Punycode;
use HDiet::Text::CSV;

⟨ Default parameter settings 370a ⟩

⟨ Global variables 370b ⟩
```

◇

Macro referenced in 163, 433, 434.

12.3 Perl language modes

⟨ Perl language modes 369b ⟩ ≡

```
require 5;
use strict;
use warnings;
use utf8;
```

◇

Macro referenced in 13, 16a, 19, 69, 106, 110, 140, 146, 154, 369a, 391, 397a, 407, 414, 419, 431.

12.3.1 Default parameter settings

The following variables contain parameters which control the operation of the program. All of these are defaults which can be overridden by command line options.

⟨ Default parameter settings 370a ⟩ ≡

◇

Macro referenced in 369a.

12.3.2 Global variables

The following variables are global to the entire `main` program. We could make many of these more local, but the entire program is sufficiently short and straightforward we'd probably only end up obfuscating things in the interest of "purity."

⟨ Global variables 370b ⟩ ≡

```
#   Processing arguments and options

my $verbose = 0;            # Verbose output for debugging

my $testmode = 0;           # Test mode: don't update real database

#   Handy constants

my %mnames = split(/,/, "Jan,1,Feb,2,Mar,3,Apr,4,May,5,Jun,6,Jul,7,Aug,8,Sep,9,Oct,10,Nov,11,Dec,12");

our @monthNames = (   "Zeroary",
                      "January", "February", "March",
                      "April", "May", "June",
                      "July", "August", "September",
                      "October", "November", "December"
                  );

my @chartSizes = ( '320x240', '480x360', '512x384', '640x480', '800x600', '1024x768', '1200x900', '1600x12

my @feedback_categories = ( ⟨ Categories of feedback messages 10e ⟩ );
```

◇

Macro referenced in 369a.

12.3.3 Process command line options

We use the `Getopt::Long` module to process command line options.

⟨ Process command line options 371a ⟩ ≡

```
use Getopt::Long;

GetOptions(
            'copyright' => sub { print("This program is in the public domain.\n"); exit(0); },
            'help' => sub { &print_command_line_help; exit(0); },
            'test' => \$testmode,
            'verbose' => \$verbose,
            'version' => sub { print("Version ⟨ Version 3a ⟩, ⟨ Release Date 3b ⟩\n"); exit(0); }
          );
```

◇

Macro referenced in 163.

12.3.4 Validate option specfications

Validate the option specifications before we begin processing. Pre-checking them avoids ugly pratfalls later on.

⟨ Validate option specifications 371b ⟩ ≡

```
{
    my $ok = 1;

    if (!$ok) {
        die("Invalid option specification(s)");
    }
}
```

◇

Macro referenced in 163.

12.4 XHTML generation

The following sections generate the XHTML result from a transaction. All of these write to a file handle named `$fh`, which should be set to the actual open UTF-8 file handle before using this code. When running as a CGI application, the file handle will be bound to `STDOUT`.

12.4.1 Mime Content-type specification

When we're invoked as a CGI program, the MIME `Content-type` must be specified before the actual text is returned. Although strictly speaking, we should specify a type of "`application/xhtml+xml`" for XHTML documents, many installed browsers do not handle this properly, so we fib and claim it to be HTML, having carefully fudged the syntax of the document to slip by benighted HTML-only browsers. Note that the header lines are terminated by both a carriage return and line feed, as required by the standard. In fact, most browsers don't require the carriage return, but why violate a standard and ask for trouble?

⟨ MIME Content-type specification 372a ⟩ ≡

```
print($fh "Content-type: text/html\r\n\r\n");
```

◇

Macro referenced in 163, 195, 239, 249.

12.4.2 Local time zone offset field

In assorted circumstances we'd like to know the offset between the user's local time zone and that of UTC. This allows us, for example, to define the "current day" in the user's time zone, not that of the prime meridian or some other location, and cope with collectivist "summer time" schemes.

HTTP does not provide this information directly, so we employ the following kludge. In each form we send to the user, a hidden field named "`HDiet_tzoffset`" is embedded. The JavaScript support code run at document initialisation time calls a `determineTimeZoneOffset()` function which calculates the local offset from UTC in minutes and plugs it into this field which is then submitted when the user transmits the form for processing. If JavaScript is unavailable or disabled, the field simply retains its original value of "`unknown`" and the server proceeds as if local time were UTC.

If you have more than one form in your document and need to distinguish the time zone offset fields (to avoid duplicating an `id` field, which is impermissible, you can pass a disambiguation string as a macro argument which will be appended to the ID value.

⟨ Local time zone offset field 372b ⟩ ≡

```
<div><input type="hidden" name="HDiet_tzoffset" id="tzoffset@1" value="unknown" /></div>◇
```

Macro referenced in 117, 182, 187, 191, 196, 210, 212, 213a, 214, 228, 237, 245, 257, 261, 280, 288a, 294, 295, 299, 301, 306, 307, 309, 320, 327, 331, 341, 349, 357, 358b, 363.

12.5 Utility Functions

The following utility functions are defined in the main program context to handle matters such as command line processing.

⟨ Utility functions 373 ⟩ ≡

⟨ Propagate trend through user's monthly logs 374 ⟩
⟨ Append entry to transaction history log 378 ⟩
⟨ Update time of user's last transaction 379a ⟩
⟨ Return time of user's last transaction 379b ⟩
⟨ Test whether user has a session active 380 ⟩

⟨ Parse weight value 381a ⟩
⟨ Parse signed weight value 381b ⟩

⟨ Wrap long lines onto multiple lines 382 ⟩

⟨ Print command line help information 383 ⟩
⟨ Minimum, Maximum, and Sign functions 384 ⟩
⟨ Execute system command 385a ⟩
⟨ Edit Unix time value to ISO 8601 local date and time 385b ⟩
⟨ Convert characters in a string to hexadecimal 386a ⟩
⟨ Test if month is the current month 386b ⟩
⟨ Encode international domain name 388a ⟩
⟨ Test domain valid for E-mail 388b ⟩
⟨ Draw text in a chart 387 ⟩

⟨ Parse CGI arguments 389 ⟩
⟨ Supply default values for undefined variables 390a ⟩

◇

Macro referenced in 163, 433.

12.5.1 Propagate trend through user's monthly logs

When a user changes a weight entry in a monthly log which is not the most recent in the database for that user, we must propagate the change in the trend at the end of the modified log to all subsequent logs, adjusting the `trend_carry_forward` field in each log which permits calculation of trend values for days within it in isolation.

The `propagate_trend` function performs this. It is called with the user's account information object and the year and month of the log in which the change occurred in *yyyy*-*mm* format. To recompute the trend carry-forward for all the logs for a user, specify "`0000-00`" or simply omit the second argument. The optional third argument, if nonzero, forces the conversion of all weight entries in the log into canonical form; it can be used to clean up oddly formatted numbers and nugatory decimal places which have crept into the database.

⟨ Propagate trend through user's monthly logs 374 ⟩ ≡

```
sub propagate_trend {
    my ($user, $first, $canon) = @_;

    $first = '0000-00' if !defined($first) || ($first eq '');
    $canon = 0 if !defined($canon);

    my @logs = $user->enumerateMonths();
    my $user_file_name = quoteUserName($user->{login_name});
    ⟨ Identify log where recomputation begins 375a ⟩
    ⟨ Load first log into memory 375b ⟩

    for ($i = $ifirst + 1; $i <= $#logs; $i++) {
        ⟨ Propagate trend to next log 376a ⟩
    }

    if ($user->{badge_trend} != 0) {
        open(FB, ">⟨ Users Directory 6h ⟩/$user_file_name/BadgeImageNew.png") ||
            die("Cannot update monthly log file ⟨ Users Directory 6h ⟩/$user_file_name/BadgeImageNew.png");
        my $hist = HDiet::history->new($user, $user_file_name);
        $hist->drawBadgeImage(\*FB, $user->{badge_trend});
        close(FB);
        do_command("mv ⟨ Users Directory 6h ⟩/$user_file_name/BadgeImageNew.png ⟨ Users Directory 6h ⟩/$user_file
        clusterCopy("⟨ Users Directory 6h ⟩/$user_file_name/BadgeImage.png");
    }
}
```

◇

Macro referenced in 373.

12.5.1.1 Identify log where recomputation begins

Now we scan the array containing the list of logs to find the index for the month whose log item was modified. We test for this using an inequality in order to select the first log when recomputation for all months has been selected. We assume that the trend carry-forward for this log is correct (or zero, indicating no previous month, if this is the first month in the database).

⟨ Identify log where recomputation begins 375a ⟩ ≡

```
    my $ifirst;

    for ($ifirst = 0; $ifirst <= $#logs; $ifirst++) {
        if ($logs[$ifirst] ge $first) {
            last;
        }
    }
```

◇

Macro referenced in 374.

12.5.1.2 Load first log into memory

We start the trend propagation by bringing the first log into memory. Note that loading the log causes its trend values to be filled in. We save the trend from the last day of the month as the value to carry over into the next.

⟨ Load first log into memory 375b ⟩ ≡

```
    my $mlog = HDiet::monthlog->new();

    my $i = $ifirst;
    open(FL, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/$logs[$i].hdb") ||
        die("Cannot open monthly log file ⟨ Users Directory 6h ⟩/$user_file_name/$logs[$i].hdb");
    $mlog->load(\*FL);
    close(FL);

    if ($canon) {
        ⟨ If requested, canonicalise weight entries in log 377b ⟩
        ⟨ Write modified log back to database 377a ⟩
    }

    my $ltrend = $mlog->{trend}[$mlog->monthdays()];
    my $lunit = $mlog->{log_unit};
    undef($mlog);

#print("First log: $logs[$ifirst]  Trend = $ltrend<br>\n");
```

◇

Macro referenced in 374.

12.5.1.3 Propagate trend to next log

For each subsequent log, we load it into memory, plug in the trend from the last day of the previous log, and recompute its trend with the new carry-forward. Then we save the log with the new carry-forward and get the trend from the last day to apply to the next log. The trend carry-forward is canonicalised to four decimal places with insignificant trailing zeroes and decimal places removed.

⟨ Propagate trend to next log 376a ⟩ ≡

```
        $mlog = HDiet::monthlog->new();

        open(FL, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/$logs[$i].hdb") ||
            die("Cannot open monthly log file ⟨ Users Directory 6h ⟩/$user_file_name/$logs[$i].hdb");
        $mlog->load(\*FL);
        close(FL);

        ⟨ Convert trend to weight unit in this log, if different 376b ⟩

        $ltrend = 0 if !defined($ltrend);
        $ltrend = sprintf("%.4f", $ltrend);
        $ltrend =~ s/(\.[^0]*)0+$/$1/;
        $ltrend =~ s/\.$//;

        ⟨ If requested, canonicalise weight entries in log 377b ⟩

    #print("Log: $logs[$i]  Trend = $ltrend, was $mlog->{trend_carry_forward}<br>\n");
        $mlog->{trend_carry_forward} = $ltrend;
        $mlog->computeTrend();

        ⟨ Write modified log back to database 377a ⟩

        $ltrend = $mlog->{trend}[$mlog->monthdays()];
        undef($mlog);
    ◇
```

Macro referenced in 374.

12.5.1.3.1 Convert trend to weight unit in this log, if different It may come to pass that logs in the database use different weight units (for example, if the user has moved from an area where one unit is prevalent to another, and changed his log unit preference accordingly). If the trend value we propagated from the previous log is in a different unit than that used in the present log, convert it to the current log's unit and note the change in trend units for the next time around.

⟨ Convert trend to weight unit in this log, if different 376b ⟩ ≡

```
        if ($lunit != $mlog->{log_unit}) {
            $ltrend *= WEIGHT_CONVERSION->[$lunit][$mlog->{log_unit}];
    #print("Log: $logs[$i]  Converted trend unit from $lunit to $mlog->{log_unit}<br>\n");
            $lunit = $mlog->{log_unit};
        }
    ◇
```

Macro referenced in 376a.

12.5.1.3.2 Write modified log back to database The current log is written back to the database directory. The `last_modification_time` in the log is updated to the current time.

⟨ Write modified log back to database 377a ⟩ ≡

```
    $mlog->{last_modification_time} = time();
    open(FL, ">:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/$logs[$i].hdb") ||
        die("Cannot create monthly log file ⟨ Users Directory 6h ⟩/$user_file_name/$logs[$i].hdb");
    $mlog->save(\*FL);
    close(FL);
    clusterCopy("⟨ Users Directory 6h ⟩/$user_file_name/$logs[$i].hdb");
```

◇

Macro referenced in 375b, 376a.

12.5.1.3.3 If requested, canonicalise weight entries in log If requested by the `$canon` argument, swoop through the weight entries in the log and transform them to canonical form (as defined by `monthlog::canonicalWeight`. This can be used to clean up ratty numbers which crept past our other canonicalisation safeguards.

⟨ If requested, canonicalise weight entries in log 377b ⟩ ≡

```
    if ($canon) {
        my $ncanon = 0;

        for (my $j = 1; $j <= $mlog->monthdays(); $j++) {
            if (defined($mlog->{weight}[$j]) && ($mlog->{weight}[$j] ne '')) {
                my $cw = canonicalWeight($mlog->{weight}[$j]);
                if ($cw ne $mlog->{weight}[$j]) {
#print("Log: $logs[$i]  Day $j:  $mlog->{weight}[$j] ==> $cw<br>\n");
                    $mlog->{weight}[$j] = $cw;
                    $ncanon++;
                }
            }
        }

        if ($ncanon > 0) {
#print("Log: $logs[$i]  $ncanon weight entries canonicalised.<br>\n");
        }
    }
```

◇

Macro referenced in 375b, 376a.

12.5.2 Append entry to transaction history log

An entry with the type given by the second argument is appended to the `History.hdh` log in the directory for the user specified by the first argument, which must be the user name encoded as a file name (we could do it here, but in fact everywhere this function is used we already have the file-encoded user name available in a variable, so why waste the time?). The optional third argument, if specified and nonblank, is appended to the history record with a leading comma. This allows additional fields to be added to the end of the log item, which always begins with the type, UNIX `time` of the entry, and the IP address of the client who submitted the transaction.

History log items are as follows:

Type	Transaction	Extra fields
1	Log in	Handheld,Cookie
2	Log out	
3	Session closed by log in	
4	Trend re-propagation	First month,Canonical weight
5	Monthly log update	Month,Total changes,weights,rungs,flags,comments
6	Password reset	
7	CSV import	Format, Overwrite, Imported, Not entry, No overwrite, No parse, Month, Changes,...
8	User account settings change	Settings changed
9	Invalid password reset	
10	Failed login attempt	
11	User attempted administrator command	Command
12	Delete all logs from database	
13	Administrator force session close	User file name
14	Administrator purge logs	Months purged
15	Diet calculator updated	
16	Feedback message sent	Category
17	Administrator deleted persistent login	User file name
18	Persistent logins cleared	Number cleared
19	Badge configuration changed	Trend interval

⟨ Append entry to transaction history log 378 ⟩ ≡

```
sub append_history {
    my ($user_file, $type, $extra) = @_;

    $extra = '' if !defined($extra);
    if ($extra ne '') {
        $extra = ',' . $extra;
    }
    open(FH, ">>:utf8", "⟨ Users Directory 6h ⟩/$user_file/History.hdh") ||
       die("Cannot append to history file ⟨ Users Directory 6h ⟩/$user_file/History.hdh");
    print(FH "$type," . time() . ",$ENV{REMOTE_ADDR}$extra\n");
    close(FH);
    clusterCopy("⟨ Users Directory 6h ⟩/$user_file/History.hdh");
}
```

◇

Macro referenced in 373.

12.5.3 Update time of user's last transaction

We keep track of the time and date of the last transaction processed for a user in the `LastTransaction.hdl` file in the user directory. This is a simple text file, with the first line a version identifier ("`1`" currently), and the second the UNIX `time()` of the transaction.

⟨ Update time of user's last transaction 379a ⟩ ≡

```
sub update_last_transaction {
    my ($user_file) = @_;

    #   Update the date and time of the last transaction by this user
    my $now = time();
    open(FL, ">:utf8", "⟨ Users Directory 6h ⟩/$user_file/LastTransaction.hdl") ||
        die("Cannot update last transaction file ⟨ Users Directory 6h ⟩/$user_file/LastTransaction.hdl");
    print FL <<"EOD";
1
$now
EOD
    close(FL);
    clusterCopy("⟨ Users Directory 6h ⟩/$user_file/LastTransaction.hdl");
}
```

◇

Macro referenced in 373.

12.5.4 Return time of user's last transaction

Return the UNIX `time()` value of the last transaction made by the user whose user file name is passed as the argument. If the user has no last transaction, a time of zero is returned.

⟨ Return time of user's last transaction 379b ⟩ ≡

```
sub last_transaction_time {
    my ($user_file) = @_;

    if (open(FL, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file/LastTransaction.hdl")) {
        my $lt = 0;
        my $s = in(\*FL);
        if ($s == 1) {          # Only proceed if version correct
            $s = in(\*FL);
            if ($s =~ m/^\d+$/) {
                $lt = $s;
            }
        }
        close(FL);
        return $lt;
    } else {
        return 0;
    }
}

⟨ Read line from persistent object file (379c main ) 390b ⟩
```

◇

Macro referenced in 373.

12.5.5 Test whether user has a session active

Check whether the user whose name is passed as the argument has a session open. This check assumes as little as possible about the correctness of the database, and can be used by the administrative routines to clean up messes. It looks for an `ActiveSession.hda` file in the user's directory. If one is present, it then reads the session ID from it and attempts to retrieve the active session file from the `Sessions` directory. If one is found, then the user does, indeed, have a session active. If no session file exists, then this is an "orphaned session" due to a "Lazarus file", mis-conceived restore, or some other vicissitude of computing. We delete the orphaned session and report no session open. The converse case, where there is a session file in the `Sessions` directory but no `ActiveSession.hda` in the user directory poses no problem, since the administrator terminate session facility will clean this up.

⟨ Test whether user has a session active 380 ⟩ ≡

```
sub is_user_session_open {
    my ($user_name) = @_;

    my $user_file_name = quoteUserName($user_name);
    if ((-f "⟨ Users Directory 6h ⟩/$user_file_name/ActiveSession.hda")
        && open(FS, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/ActiveSession.hda")) {
        my $asn = load_active_session(\*FS);
        close(FS);
        if (-f "⟨ Session Directory 6g ⟩/$asn.hds") {
            return 1;
        } else {
            unlink("⟨ Users Directory 6h ⟩/$user_file_name/ActiveSession.hda");
            clusterDelete("⟨ Users Directory 6h ⟩/$user_file_name/ActiveSession.hda");
#print(STDERR "is_user_session_open abstergifying orphaned session ⟨ Users Directory 6h ⟩/$user_file_name/ActiveS
        }
    }
    return 0;
}
◇
```

Macro referenced in 373.

12.5.6 Parse weight value

Parse a weight value given by the first string argument in the unit given by the second. If the unit is `WEIGHT_STONE`, the "stones and pound" syntax is accepted. If an invalid weight is specified `undef` is returned.

⟨ Parse weight value 381a ⟩ ≡

```
sub parseWeight {
    my ($w, $unit) = @_;

    $w =~ s/,/./g;
    my $n;
    if ($unit == WEIGHT_STONE) {
        if ($w =~ m/^\s*(\d+)\s+(\d*\.?\d*)\s*$/) {
            $n = ($1 * 14) + $2;
        } elsif ($w =~ m/^\s*(\d*\.?\d*)\s*$/) {
            $n = $1 * 14;
        }
    } else {
        if ($w =~ m/^\s*(\d*\.?\d*)\s*$/) {
            $n = $1;
        }
    }
    return $n;
}
```

◇

Macro referenced in 373.

12.5.7 Parse signed weight value

Parse a signed weight value from the first argument string in the unit given by the second. This is simply a wrapper for `parseWeight` which handles an optional prefix sign.

⟨ Parse signed weight value 381b ⟩ ≡

```
sub parseSignedWeight {
    my ($w, $unit) = @_;
    my $sgn = 1;

    if ($w =~ s/\s*([\+\-])//) {
        if ($1 eq '-') {
            $sgn = -1;
        }
    }
    my $v = parseWeight($w, $unit);
    if (defined($v)) {
        return $sgn * $v;
    }
    return undef;
}
```

◇

Macro referenced in 373.

12.5.8 Wrap long lines onto multiple lines

Wrap text in the first argument (which may contain explicit line breaks) so as not to exceed the maximum characters per line given by the second argument. If a line cannot be split due to absence of white space break points, it is allowed to overflow the column limit. Based upon the Perl module Text::Wrap by David Muir Sharnoff, much simplified for use here.

⟨ Wrap long lines onto multiple lines 382 ⟩ ≡

```
sub wrapText {
    my ($t, $columns) = @_;

    my ($ip, $xp) = ('', '');
    my $break = '\s';
    my $separator = "\n";

    my $r = "";
    my $tail = $t;
    my $lead = $ip;
    my $ll = $columns - length($ip) - 1;
    my $nll = $columns - length($xp) - 1;
    my $nl = "";
    my $remainder = "";

    pos($t) = 0;
    while ($t !~ /\G\s*\Z/gc) {
        if ($t =~ /\G([^\n]{0,$ll})($break|\z)/xmgc) {
            $r .= $nl . $lead . $1;
            $remainder = $2;
        } elsif ($t =~ /\G([^\n]*?)($break|\z)/xmgc) {
            $r .= $nl . $lead . $1;
            $remainder = $2;
        }

        $lead = $xp;
        $ll = $nll;
        $nl = $separator;
    }
    $r .= $remainder;

    $r .= $lead . substr($t, pos($t), length($t)-pos($t))
            if pos($t) ne length($t);

    return $r;
}
```

◇

Macro referenced in 373.

12.5.9 Print command line help information

⟨ Print command line help information 383 ⟩ ≡

```
    sub print_command_line_help {
        print << "EOD";
Usage: HackDiet.pl [ options ]
       Options:
             --copyright     Print copyright information
             --help          Print this message
             --test          Test mode: do not actually block hosts
             --verbose       Print verbose debugging information
             --version       Print version number
Version ⟨ Version 3a ⟩, ⟨ Release Date 3b ⟩
EOD
   }
```

◇

Macro referenced in 373.

12.5.10 Minimum, Maximum, Sign, and Round functions

The `min` and `max` functions return the largest and smallest of their arbitrarily long list of arguments. The `sgn` function returns 1 if the argument is positive, −1 if negative, and 0 if zero. The `round` function rounds a number to the nearest integer (positive or negative).

⟨ Minimum, Maximum, and Sign functions 384 ⟩ ≡

```
#   Return least of arguments
sub min {
    my $v = 1e308;

    my $a;
    while (defined($a = shift())) {
        $v = $a if $a < $v;
    }
    return $v;
}

#   Return greatest of arguments
sub max {
    my $v = -1e308;

    my $a;
    while (defined($a = shift())) {
        $v = $a if $a > $v;
    }
    return $v;
}

#   Return sign of argument
sub sgn {
    my $a = shift();

    return ($a == 0) ? 0 : (($a > 0) ? 1 : -1);
}

#   Round number to nearest integer
sub round {
    return sprintf("%.0f", shift());
}
```

◇

Macro referenced in 19, 69, 373.

12.5.11 Execute system command

Run a system command unless we're in `--test` mode, in which case we just print the command on standard error. An optional second argument may supply an annotation to be prefixed to the command when it is printed in `--verbose` mode.

⟨ Execute system command 385a ⟩ ≡

```
sub do_command {
    my ($cmd, $annotation) = @_;

    if ($verbose) {
        if (!defined($annotation)) {
            $annotation = '';
        } else {
            $annotation .= ": ";
        }
        print(STDERR "$annotation$cmd\n");
    }

    if (!$testmode) {
        system($cmd);
    }
}
```

◇

Macro referenced in 373.

12.5.12 Edit Unix time value to ISO 8601 local date and time

The Unix `time()` value argument is edited to a string in the ISO 8601 `YYYY-MM-DD HH:MM` format.

⟨ Edit Unix time value to ISO 8601 local date and time 385b ⟩ ≡

```
#    sub etime {
#        my ($sec, $min, $hour, $mday, $mon, $year) = localtime($_[0]);
#        return sprintf("%d-%02d-%02d %02d:%02d",
#            $year + 1900, $mon + 1, $mday, $hour, $min);
#    }
```

◇

Macro referenced in 373.

12.5.13 Convert characters in a string to hexadecimal

The characters in the string argument are converted to space separated two digit hexadecimal values if their code points are 255 or less. Unicode characters with higher code points are output in as many hexadecimal digits as required to represent them.

⟨ Convert characters in a string to hexadecimal 386a ⟩ ≡

```
sub toHex {
    my ($s) = @_;

    my $h = '';
    while ($s =~ s/^(.)//s) {
        $h .= sprintf("%02X ", ord($1));
    }
    $h =~ s/\s$//;
    return $h;
}
```

◇

Macro referenced in 373.

12.5.14 Test if month is the current month

Estimate, being conservative based upon differences in time zone between the user and the server, whether a month specified by the year and date arguments, is the current month or a month in the past. This test is used only for optimising things such as skipping the call on trend propagation when a change is made to the current month's log, and the like. It should never be used in circumstances where the user will perceive different behaviour, as opposed to internal optimisations.

⟨ Test if month is the current month 386b ⟩ ≡

```
sub isCurrentMonth {
    my ($lyear, $lmonth) = @_;

    #   Julian day at the server
    my $server_jd = unix_time_to_jd(time());

    #   JD at start of specified month
    my $this_month_jd = gregorian_to_jd($lyear, $lmonth, 1);

    #   JD at start of next month
    my ($nyear, $nmonth) = ($lyear, $lmonth + 1);
    if ($lmonth >= 12) {
        $lyear++;
        $lmonth = 1;
    }
    my $next_month_jd = gregorian_to_jd($nyear, $nmonth, 1);

    return ($server_jd >= ($this_month_jd + 1)) &&
           ($server_jd <= ($next_month_jd - 1));
}
```

◇

Macro referenced in 373.

12.5.15 Draw text in a chart

The specified `$text` is drawn into an open `GD` `$image` with the specified size, angle, position, and colour. The horizontal and vertical alignment may be specified as follows. The difference between the "origin" and the leftmost or bottom pixel is due to any space between the origin point of the first character and the presence of descenders in the typeset text.

Horizontal Alignment

Code	Alignment
`o`	Origin
`l`	Leftmost pixel
`c`	Centre
`r`	Rightmost pixel

Vertical Alignment

Code	Alignment
`o`	Origin
`b`	Bottom pixel
`c`	Centre
`t`	Top pixel

⟨ Draw text in a chart 387 ⟩ ≡

```
sub drawText {
    my ($img, $text, $font, $size, $angle, $x, $y, $alignh, $alignv, $colour) = @_;

    my $fontFile = "⟨ TrueType Font Directory 6b ⟩/$font.ttf";

    if (($alignh ne 'o') || ($alignv ne 'o')) {
        my @ext =  GD::Image->stringFT($colour, $fontFile, $size, $angle, $x, $y, $text);

        if ($alignh eq 'l') {
            $x -= ($ext[0] - $x);
        } elsif ($alignh eq 'c') {
            $x -= int(($ext[2] - $ext[0]) / 2);
        } elsif ($alignh eq 'r') {
            $x -= $ext[2] - $x;
        } else {
            die("drawText: invalid horizontal alignment '$alignh'") if $alignh ne 'o';
        }

        if ($alignv eq 'b') {
            $y -= ($ext[1] - $y);
        } elsif ($alignv eq 'c') {
            $y += int(($y - $ext[7]) / 2);
        } elsif ($alignv eq 't') {
            $y -= $ext[7] - $y;
        } else {
            die("drawText: invalid vertical alignment '$alignv'") if $alignv ne 'o';
        }
    }
    $img->stringFT($colour, $fontFile, $size, $angle, $x, $y, $text);
}
```

◇

Macro referenced in 373.

12.5.16 Encode international domain name

The argument is an Internet domain name which may (or may not) contain non-ASCII characters which must be encoded before querying a domain name server. If any characters which require such encoding are present, this function encodes them according to RFC 3490 into the "punycode" representation. Each dot-separated component of the domain name is encoded separately. An ASCII domain name or numeric IP address specification is not modified by this function.

⟨ Encode international domain name 388a ⟩ ≡

```
sub encodeDomainName {
    my ($idn) = @_;

    my $dn = '';
    my $w;

    foreach $w (split(/\./, $idn)) {
        $dn .= encode_punycode($w) . '.';
    }

    $dn =~ s/\.$//;
    return $dn;
}
```

◇

Macro referenced in 373.

12.5.17 Test domain valid for E-mail

Determine whether the argument is an Internet domain name which is able to receive E-mail. We use the "`dig`" program to retrieve the `MX` (mail exchanger) records for the domain, filter certain bogus results from parking services, and count the number of records obtained. If the domain has one or more valid mail exchanger records, we deem it valid as an E-mail destination. If we find no `MX` record, we test for address `A` records; this handles poorly-configured sites which run a mail exchanger on port 25 but which have no `MX` record in their DNS configuration.

⟨ Test domain valid for E-mail 388b ⟩ ≡

```
sub validMailDomain {
    my ($dn) = @_;

    my $nmx = `dig +short $dn MX | egrep -v ' 127\.0\.0.' | wc -l`;
    $nmx =~ s/\s//g;

    if ($nmx == 0) {
        $nmx = `dig +short $dn A | egrep -v ' 127\.0\.0.' | wc -l`;
        $nmx =~ s/\s//g;
    }

    return $nmx > 0;
}
```

◇

Macro referenced in 373.

12.5.18 Parse CGI arguments

When we are invoked as a CGI application by a Web server, the arguments will be passed either in the "QUERY_STRING" environment variable (for a "get" request) or via standard input (for a "post" request). The parse_cgi_arguments subroutine determines the request type, retrieves the arguments, and decodes them into a hash keyed by the field name which gives the decoded value of each. The hash is returned as the result of the subroutine. If the document is encoded in UTF-8, field names and values may contain Unicode characters.

If the form includes an uploaded file which should be processed as a raw sequence of bytes, the request URL should include a query string of "?enc=raw" (note that you can specify a query string on the URL even if the form is performing a POST submission of multipart form data). This causes the standard input stream to be put into raw mode, as opposed to UTF-8, and allows the uploaded file data to be processed without passing through the usual UTF-8 decoder.

⟨ Parse CGI arguments 389 ⟩ ≡

```
    sub parse_cgi_arguments {
        my $data;
#   NOTE: On Perl 5.8.5 we needed to read the CGI POST arguments
#   in UTF-8 mode.  On Perl 5.8.8 the decode_utf8() function
#   appears to double-decode POST (but not GET arguments) unless
#   we read the POST arguments in :raw mode.  I am not sure
#   I understand this (and there is much conflicting information
#   on this topic on the Web), but simply always reading STDIN
#   in :raw appears to work, at least at the moment.
#        if ($ENV{QUERY_STRING} =~ m/enc=raw/) {
#            binmode(STDIN, ":raw");
#        }
        binmode(STDIN, ":raw");

        my $query = new CGI;
#print("Content-type: text/plain\r\n\r\nQuery:\n");
#use Data::Dumper;
#print(Dumper($query));

        my %CGIfields = $query->Vars();
        my $uploaded = 0;
        if ($CGIfields{uploaded_file}) {
#print(STDERR "Uploading file...\n");
            my $uploaded_content = '';

            my $uf = $query->upload('uploaded_file');
            while (<$uf>) {
                $uploaded_content .= $_;
            }
            close($uf);
            $CGIfields{file} = $uploaded_content;
            $uploaded = 1;
#print(STDERR "Uploaded file of " . length($CGIfields{file}) . " bytes.\n");
        }
#   DECODE CGI ARGUMENTS FROM UTF-8.  THIS MAY BREAK POST AND
#   NEEDS MORE RESEARCH.  SEE COMMENTS FOR 2007-03-21.
for my $k (keys %CGIfields) {
    if ($k eq 'file') {
        #   IF ARGUMENT IS FILE, DO NOT DECODE FROM UTF-8.  THIS NEEDS
        #   TO BE THOUGHT OUT.  WE MAY TRY DECODING AND USE THE DECODE
        #   IF IT WORKS.
#print(STDERR "Uploaded " . length($CGIfields{file}) . "\n");
    } else {
        $CGIfields{$k} = decode_utf8($CGIfields{$k});
    }
#print(STDERR "Argument $k " . length($CGIfields{$k}) . " bytes.\n");
}
        return %CGIfields;
```

12.5.19 Supply default values for undefined variables

To conserve memory, we frequently leave inapplicable fields in the `%Sessions` hash undefined. The following little functions, whose names are pronounced "not-defined zero" and "not-defined blank" return their argument value if it is defined and zero or the null string respectively if it is not.

⟨ Supply default values for undefined variables 390a ⟩ ≡

```
sub ndz {
    return defined($_[0]) ? $_[0] : 0;
}

sub ndb {
    return defined($_[0]) ? $_[0] : '';
}
```

◇

Macro referenced in 373.

12.5.20 Read line from persistent object file

All of our persistent objects store their data in text files containing one line for each instance variable in the object. The following macro generates the `in` function used to read the next item from one of these files. The macro is called with an argument which specifies the name of the object, which is used in an error message if an unexpected end of file is encountered.

The function is called with the first argument the file handle from which the item is to be read. The optional second argument specifies the default value to be returned if an end of file is encountered. If no default is given and the end of file is encountered, the program will abort with an error message.

⟨ Read line from persistent object file 390b ⟩ ≡

```
sub in {
    my ($fh, $default) = @_;
    my $s;
    if ($s = <$fh>) {
        $s =~ s/\s+$//;
    } else {
        if (defined($default)) {
            $s = $default;
        } else {
            die("@1::in: Unexpected end of file");
        }
    }
    return $s;
}
```

◇

Macro referenced in 30b, 116a, 144a, 149a, 161b, 379b, 397a.

Chapter 13

Cluster file system support

Since we store all of our data in ordinary files in the host file system, server cluster support is simply a matter of replicating changes to these files made on one server to others in the cluster, much as is done by `rdist` for other server content. (We could, in fact, use `rdist`, but it's a heavyweight solution in this case, where we know precisely the changes we wish to make and are informed immediately when changes are made on the local server.)

Every committed change to a local file or directory is accompanied by a call on one of the methods of the `Cluster` object which queues a transaction to replicate the change on other servers in the cluster. The transactions are stored as files in a cluster synchronisation directory, which are deleted as the transactions are completed.

The stand-alone `ClusterSync.pl` program processes these transactions by submitting `scp` or `ssh` commands. It registers its process ID in a file so that it can be notified of the queueing of new transactions via `USR1` signals.

"HDiet/Cluster.pm" 391 ≡

```
#! ⟨ Perl directory 7d ⟩

    ⟨ Perl language modes 369b ⟩

    package HDiet::Cluster;

    use Time::HiRes qw( gettimeofday );
    use Digest::SHA1  qw(sha1_hex);

    require Exporter;

    our @ISA = qw(Exporter);
    our @EXPORT = qw( clusterConfiguration
                      clusterCopy clusterDelete clusterMkdir clusterRmdir clusterRecursiveDelete );
    our @EXPORT_OK = qw( command );

    my @clusterHosts = qw (⟨ Cluster Member Hosts 7f ⟩);
    my $hostname = $ENV{SERVER_NAME};
#$hostname = "server0.fourmilab.ch" if !defined($hostname);
    my $journal_sequence = 0;

    1;

    use constant FILE_VERSION => 1;     # If you change this, change in ClusterSync.pl below also!
◇
```

File defined by 391, 392, 394, 395abc, 396ab.

13.1 Display cluster configuration

The current cluster configuration is printed on the file handle passed as the argument, or `STDOUT` if omitted. If a cluster is configured, the presence or absence of the cluster transaction directory and each of its subsidiary cluster member subdirectories is checked and reported. For cluster member directories, the number of queued transaction files is listed.

"HDiet/Cluster.pm" 392 ≡

```
sub clusterConfiguration {
    my ($outfile) = @_;

    if (!(defined $outfile)) {
        $outfile = \*STDOUT;
    }
    print($outfile "Host name: $hostname\n");

    if ("⟨ Cluster Transaction Directory 7c ⟩" eq '') {
        print($outfile "Clustering disabled: Cluster Transaction Directory not specified\n");
    } elsif ($#clusterHosts < 0) {
        print($outfile "Clustering disabled: No Cluster Member Hosts configured\n");
    } else {
        print($outfile "Cluster members:");
        for (my $i = 0; $i <= $#clusterHosts; $i++) {
            print($outfile ' ');
            if ($clusterHosts[$i] eq $hostname) {
                print($outfile '[');
            }
            print($outfile $clusterHosts[$i]);
            if ($clusterHosts[$i] eq $hostname) {
                print($outfile ']');
            }
        }
        ⟨ Display summary of cluster transaction queue 393 ⟩
    }
}
```

◇

File defined by 391, 392, 394, 395abc, 396ab.

13.1.1 Display summary of cluster transaction queue

Walk through the cluster transaction directory and, for each cluster member other than ourselves, display the number of transactions queued for that member. Missing and unreadable directories are reported.

⟨ Display summary of cluster transaction queue 393 ⟩ ≡

```
print($outfile "\n");
print($outfile "Transaction directory: ⟨ Cluster Transaction Directory 7c ⟩  ");
if (-d "⟨ Cluster Transaction Directory 7c ⟩") {
    print($outfile "Exists\n");
    for (my $i = 0; $i <= $#clusterHosts; $i++) {
        if ($clusterHosts[$i] ne $hostname) {
            print($outfile "  Server directory: ⟨ Cluster Transaction Directory 7c ⟩/$clusterHosts[$i]: ");
            if (-d "⟨ Cluster Transaction Directory 7c ⟩/$clusterHosts[$i]") {
                my $n = 0;
                if (opendir(DI, "⟨ Cluster Transaction Directory 7c ⟩/$clusterHosts[$i]")) {
                    my $e;
                    while ($e = readdir(DI)) {
                        if ($e !~ m/^\./) {
                            $n++;
                        }
                    }
                    print($outfile "Queue length $n\n");
                    closedir(DI);
                } else {
                    print($outfile "*Unreadable*\n");
                }
            } else {
                print($outfile "*Missing*\n");
            }
        }
    }
} else {
    print($outfile "*Missing*\n");
}
```

◇

Macro referenced in 392.

13.2 Enqueue cluster synchronisation transaction

A synchronisation transaction is queued for all cluster hosts other than this one. The first argument is the operation name and the second is the file name within the Database Directory upon which the operation is to be performed. If the

"HDiet/Cluster.pm" 394 ≡

```
sub enqueueClusterTransaction {
    my ($operation, $filename) = @_;

    if (("⟨ Cluster Transaction Directory 7c ⟩" ne '') &&
        ($#clusterHosts >= 0) &&
        (-d "⟨ Cluster Transaction Directory 7c ⟩")) {
        my ($sec, $usec) = gettimeofday();
        my $efn = $filename;
        $efn =~ s:[\./]:_:g;
        my $transname = sprintf("T%d%06d_%03d_%s_%s.hdc", $sec, $usec,
                                ++$journal_sequence, $operation, $efn);
        for (my $i = 0; $i <= $#clusterHosts; $i++) {
            if ($clusterHosts[$i] ne $hostname) {
                if (-d "⟨ Cluster Transaction Directory 7c ⟩/$clusterHosts[$i]") {
                    open(TO, ">:utf8", "⟨ Cluster Transaction Directory 7c ⟩/$clusterHosts[$i]/$transname") ||
                        die("Unable to create cluster transaction " .
                            "⟨ Cluster Transaction Directory 7c ⟩/$clusterHosts[$i]/$transname");
                    print(TO FILE_VERSION . "\n");
                    print(TO "$operation\n");
                    print(TO "$filename\n");
                    print(TO sha1_hex(FILE_VERSION . $operation . $filename .
                        ⟨ Confirmation signature encoding suffix 4c ⟩) . "\n");
                    close(TO);
                    if (open(PI, "<⟨ Cluster Synchronisation Process ID File 8e ⟩")) {
                        my $syncpid = <PI>;
                        close(PI);
                        $syncpid =~ s/\s//g;
                        kill('⟨ Cluster Synchronisation Signal 8d ⟩', $syncpid);
#print(STDERR "Sending ⟨ Cluster Synchronisation Signal 8d ⟩ to process $syncpid\n");
                    } else {
#print(STDERR "Cannot open ⟨ Cluster Synchronisation Process ID File 8e ⟩\n");
                    }
                }
            }
        }
    }
◇
```

File defined by 391, 392, 394, 395abc, 396ab.

13.2.1 Copy a file to the cluster members

The file specified by the argument is copied to other members of the cluster. This, like the subsequent functions, simply calls enqueueClusterTransaction with the appropriate transaction code.

"HDiet/Cluster.pm" 395a ≡

```
        sub clusterCopy {
            my ($filename) = @_;

            enqueueClusterTransaction('copy', $filename);
        }
```

◇

File defined by 391, 392, 394, 395abc, 396ab.

13.2.2 Delete a file from cluster members

The file specified by the argument is deleted from other members of the cluster.

"HDiet/Cluster.pm" 395b ≡

```
        sub clusterDelete {
            my ($filename) = @_;

            enqueueClusterTransaction('delete', $filename);
        }
```

◇

File defined by 391, 392, 394, 395abc, 396ab.

13.2.3 Create a directory on cluster members

A directory whose name is given by the argument is created on the cluster members. The parent directory must already exist.

"HDiet/Cluster.pm" 395c ≡

```
        sub clusterMkdir {
            my ($filename) = @_;

            enqueueClusterTransaction('mkdir', $filename);
        }
```

◇

File defined by 391, 392, 394, 395abc, 396ab.

13.2.4 Delete a directory from cluster members

The directory specified by the argument is deleted from other members of the cluster. The directory must be empty.

"HDiet/Cluster.pm" 396a ≡

```
        sub clusterRmdir {
            my ($filename) = @_;

            enqueueClusterTransaction('rmdir', $filename);
        }
    ◇
```

File defined by 391, 392, 394, 395abc, 396ab.

13.2.5 Recursively delete a directory from cluster members

The directory specified by the argument and all its contents are recursively deleted from the cluster members.

"HDiet/Cluster.pm" 396b ≡

```
        sub clusterRecursiveDelete {
            my ($filename) = @_;

            enqueueClusterTransaction('rmrf', $filename);
        }
    ◇
```

File defined by 391, 392, 394, 395abc, 396ab.

13.3 Cluster synchronisation process

This stand-alone program performs synchronisation of changes made on a cluster machine with other members of the cluster. The program runs as an independent process. It periodically scans the cluster transaction directory and attempts to apply them to the other cluster members. It publishes its process ID and accepts `SIGUSR1` signals from the transaction processor which cause it to immediately process newly-queued transactions.

"HDiet/ClusterSync.pl" 397a ≡

```
#! ⟨ Perl directory 7d ⟩

    ⟨ Perl language modes 369b ⟩

    use File::Temp qw(tempfile);
    use Digest::SHA1  qw(sha1_hex);

    use constant FILE_VERSION => 1;

    binmode(STDOUT, ":utf8");

    ⟨ Assume group and user identity of cluster synchronisation process 398a ⟩

    my @clusterHosts = qw (⟨ Cluster Member Hosts 7f ⟩);
    my %failed_hosts;
    my %failed_transactions;

    my $tranqueue = 0;
    my ($logging, $cycleLog) = (0, 0);
    $SIG{⟨ Cluster Synchronisation Signal 8d ⟩} = sub { $tranqueue++; };
    $SIG{INT} = $SIG{TERM} =
        sub {
            unlink("⟨ Cluster Synchronisation Process ID File 8e ⟩");
            close(LOG) if $logging;
            exit(0);
        };

    my $verbose = 0;
    my $nosync = 0;

    $| = 0 if $verbose;

    ⟨ Save process ID of cluster synchronisation job 398b ⟩

    ⟨ Activate cluster synchronisation log file if configured 399a ⟩

    while (1) {
        my $transfound = 0;
        ⟨ Cycle active log file if HUP signal received 399b ⟩
        ⟨ Process queued cluster synchronisation transactions 400 ⟩
        if ($transfound == 0) {
            select(undef, undef, undef, ⟨ Cluster Synchronisation Time Interval 7g ⟩);
        }
    }

    ⟨ Read line from persistent object file (397b ClusterSync ) 390b ⟩
◇
```

File defined by 397a, 404, 405b, 406ab.

13.3.1 Assume group and user identity of cluster synchronisation process

If we were started as super-user and we've been configured to run under another identity, set our real and effective group and user ID to the configured values. If not started as super-user, no identity change is made, nor is a change made if the configured identity is the null string.

⟨ Assume group and user identity of cluster synchronisation process 398a ⟩ ≡

```
        if (($> == 0) && (($) + 0) == 0)) {
            if ("⟨ Cluster Synchronisation Group ID 9a ⟩" ne '') {
                my $gid = getgrnam("⟨ Cluster Synchronisation Group ID 9a ⟩");
                $( = $gid;
                $) = "$gid $gid";
            }
            if ("⟨ Cluster Synchronisation User ID 8g ⟩" ne '') {
                my $uid = getpwnam("⟨ Cluster Synchronisation User ID 8g ⟩");
                $< = $uid;
                $> = $uid;
            }
    #print("UID: $< $> GID: $( $) $$\n");
        }
    #else { print("Not started as root.\n"); }
    ◇
```

Macro referenced in 397a.

13.3.2 Save process ID of cluster synchronisation job

The process ID of the cluster synchronisation job is saved in the designated file so that signals may be sent to notify it when new transactions are queued.

⟨ Save process ID of cluster synchronisation job 398b ⟩ ≡

```
        open(PIDF, ">⟨ Cluster Synchronisation Process ID File 8e ⟩") ||
            die("Cannot create ⟨ Cluster Synchronisation Process ID File 8e ⟩");
        print(PIDF "$$\n");
        close(PIDF);
    ◇
```

Macro referenced in 397a.

13.3.3 Activate cluster synchronisation log file if configured

If a cluster synchronisation log file is configured, it is opened (or created if not previously existing) in append mode. The variable `$logging` is set to enable the generation of output to the log. When logging, we listen for the `HUP` signal and, upon receiving it, close and re-open the log file to permit cycling it by renaming the existing file and then signalling this program.

⟨ Activate cluster synchronisation log file if configured 399a ⟩ ≡

```
if ("⟨ Cluster Synchronisation Log File 8f ⟩" ne '') {
    open(LOG, ">>⟨ Cluster Synchronisation Log File 8f ⟩") ||
        die("ClusterSync: Unable to open log file ⟨ Cluster Synchronisation Log File 8f ⟩");
    my $oldfh = select LOG; $| = 1; select $oldfh;
    $logging = 1;
    $SIG{HUP} =
        sub {
            $cycleLog = 1;
        };
}
```

◇

Macro referenced in 397a.

13.3.4 Cycle active log file if `HUP` signal received

If we receive the `HUP` signal while logging is enabled, the variable `$cycleLog` is set from the signal handler and on the subsequent pass through the transaction processor we close and re-open the log file to allow it to be rotated by renaming it before sending the signal.

⟨ Cycle active log file if HUP signal received 399b ⟩ ≡

```
if ($cycleLog) {
    close(LOG);
    open(LOG, ">>⟨ Cluster Synchronisation Log File 8f ⟩") ||
        die("ClusterSync: Unable to reopen log file ⟨ Cluster Synchronisation Log File 8f ⟩");
    my $oldfh = select LOG; $| = 1; select $oldfh;
    $cycleLog = 0;
    print("ClusterSync: Log file cycled.\n") if $verbose;
}
```

◇

Macro referenced in 397a.

13.3.5 Process queued cluster synchronisation transactions

Walk through the cluster transaction directory and process the queued transactions. The transactions are sorted by file name which, given the naming scheme we use, guarantees that we'll execute them in the order they were queued.

⟨ Process queued cluster synchronisation transactions 400 ⟩ ≡

```
if (-d "⟨ Cluster Transaction Directory 7c ⟩") {
    for (my $i = 0; $i <= $#clusterHosts; $i++) {
        my $destHost = $clusterHosts[$i];
        if (defined $failed_hosts{$destHost}) {
            if (time() > $failed_hosts{$destHost}) {
                undef($failed_hosts{$destHost});
                logmsg("Removing $destHost from failed hosts list.");
            }
        }

        if ((-d "⟨ Cluster Transaction Directory 7c ⟩/$destHost") &&
            (!defined($failed_hosts{$destHost}))) {
            if (opendir(DI, "⟨ Cluster Transaction Directory 7c ⟩/$destHost")) {
                my @transactions = sort(grep(/\.hdc$/, readdir(DI)));

                for my $t (@transactions) {
                    $t = untaint($t);
                    if (defined($failed_hosts{$destHost})) {
                        last;
                    }
                    my $transfile = "⟨ Cluster Transaction Directory 7c ⟩/$destHost/$t";
                    if (defined($failed_transactions{$t})) {
                        ⟨ Determine if failed transaction should be retried 403 ⟩
                    }
                    eval {
                        ⟨ Execute cluster synchronisation transaction 401 ⟩
                    };
                    if ($@) {
                        ⟨ Recover from failure of a cluster synchronisation transaction 402 ⟩
                    } else {
                        $transfound++;
                        undef($failed_transactions{$t}) if defined $failed_transactions{$t};
                    }
                }
                closedir(DI);
            }
        }
    }
}
```

◇

Macro referenced in 397a.

13.3.5.1 Execute cluster synchronisation transaction

The cluster synchronisation transaction is executed by performing an `scp` or `ssh` command to effect the change on the local machine on the cluster host.

⟨ Execute cluster synchronisation transaction 401 ⟩ ≡

```
open(FI, "<:utf8", $transfile) ||
    die("ClusterSync: Unable to open $transfile");
my $file_version = in(\*FI);
if ($file_version != FILE_VERSION) {
    die("ClusterSync: Invalid file version in $transfile");
}
my $transaction = in(\*FI);
my $filename = in(\*FI);
my $signature = in(\*FI);
close(FI);
if ($verbose || $logging) {
    my ($sec, $min, $hour, $mday, $mon, $year) = localtime(time());
    my $dt = sprintf("%04d-%02d-%02d %02d:%02d",
        $year + 1900,, $mon + 1, $mday, $hour, $min);
    my $lm = "$dt $clusterHosts[$i]: $t\n" .
             "        Ver: $file_version\n" .
             "        Transaction: $transaction\n" .
             "        File: $filename";
    logmsg("$lm");
}
if (sha1_hex($file_version . $transaction . $filename .
    ⟨ Confirmation signature encoding suffix 4c ⟩) ne $signature) {
    die("ClusterSync: Invalid signature in transaction");
}
if ($filename !~ m:^⟨ Database Directory 6f ⟩:) {
    die("ClusterSync: Bogus file name ($filename) in transaction");
}
if (($filename =~ m/[;<>|#\$\*\?]/) || ($filename =~ m/\.\./)) {
    die("ClusterSync: Abusive character in file name ($filename) in transaction");
}

my $res;
if ($transaction eq 'copy') {
    $res = syncCommand("scp -q -p '$filename' '$destHost:$filename'",
        $destHost, $transfile);
} elsif ($transaction eq 'delete') {
    $res = syncCommand("ssh $destHost \"rm '$filename'\"",
        $destHost, $transfile);
} elsif ($transaction eq 'mkdir') {
    $res = syncCommand("ssh $destHost \"mkdir '$filename'\"",
        $destHost, $transfile);
} elsif ($transaction eq 'rmdir') {
    $res = syncCommand("ssh $destHost \"rmdir '$filename'\"",
        $destHost, $transfile);
} elsif ($transaction eq 'rmrf') {
    $res = syncCommand("ssh $destHost \"rm -rf '$filename'\"",
        $destHost, $transfile);
} else {
    die("ClusterSync: Invalid transaction \"$transaction\"");
}
logmsg("        Results: $res") if $res ne '';
```

◇

Macro referenced in 400.

13.3.5.2 Recover from failure of a cluster synchronisation transaction

Errors encountered in processing cluster synchronisations may be transient (for example, due to a race condition where we're trying to read a transaction file before the process queueing it has finished writing it, or permanent, as happens when a process generating a transaction crashes while generating the transaction, leaving it incomplete. To avoid crashing the synchronisation process, we attempt to execute transactions within an `eval` block and test for errors therein. When an error is detected, we handle it as follows.

Using the transaction file name as the key, we look up the transaction in the `%failed_transactions` hash to see whether it's an error we've previously dealt with. If not, then we make an entry in the hash with the transaction file name as the key and a value consisting of an array of two items, the first being the number of times the transaction has failed (initially 1), and the second the time of the most recent attempt to execute the transaction.

⟨ Recover from failure of a cluster synchronisation transaction 402 ⟩ ≡

```
my $whyFailed = $@;
$whyFailed =~ s/\s+$//;
if (!defined($failed_transactions{$t})) {
    $failed_transactions{$t} = [ 1, time() + ⟨ Cluster Failed Transaction Retry Interval 8b ⟩ ];
    if ($verbose || $logging) {
        my ($sec, $min, $hour, $mday, $mon, $year) = localtime(time());
        my $dt = sprintf("%04d-%02d-%02d %02d:%02d",
            $year + 1900,, $mon + 1, $mday, $hour, $min);
        my $lm = "$dt $clusterHosts[$i]: $t\n";
        ($sec, $min, $hour, $mday, $mon, $year) = localtime($failed_transactions{$t}[1]);
        $dt = sprintf("%04d-%02d-%02d %02d:%02d",
            $year + 1900,, $mon + 1, $mday, $hour, $min);
        logmsg("$lm        Failed ($whyFailed) on first attempt.  Retry at $dt.");
    }
} else {
    my ($nfails, $failtime) = ($failed_transactions{$t}[0], $failed_transactions{$t}[1]);
    $nfails++;
    if ($nfails >= ⟨ Cluster Failed Transaction Maximum Retries 8c ⟩) {
        undef($failed_transactions{$t});
        unlink($transfile) ||
            die("Cannot delete failed cluster transaction file $transfile");
            $tranqueue--;
        logmsg("        Failure limit exceeded.  Transaction deleted.");
    } else {
        $failed_transactions{$t} = [ $nfails, time() + ⟨ Cluster Failed Transaction Retry Interval 8b ⟩ ];
        my ($sec, $min, $hour, $mday, $mon, $year) = localtime(time());
        my $dt = sprintf("%04d-%02d-%02d %02d:%02d",
            $year + 1900,, $mon + 1, $mday, $hour, $min);
        my $lm = "$dt $clusterHosts[$i]: $t\n";
        ($sec, $min, $hour, $mday, $mon, $year) = localtime($failed_transactions{$t}[1]);
        $dt = sprintf("%04d-%02d-%02d %02d:%02d",
            $year + 1900,, $mon + 1, $mday, $hour, $min);
        logmsg("$lm        Failed ($whyFailed) on attempt $nfails.  Retry at $dt.");
    }
}
```

◇

Macro referenced in 400.

13.3.5.3 Determine if failed transaction should be retried

If the transaction has failed previously, we increment the number of attempts and, if we've reached the maximum number of retries, the transaction is deleted from the transaction directory and the failed transaction hash. Otherwise, we leave the transaction to be retried when the configured retry time arrives.

⟨ Determine if failed transaction should be retried 403 ⟩ ≡

```
        my ($nfails, $failtime) = ($failed_transactions{$t}[0], $failed_transactions{$t}[1]);
        if ($failtime > time()) {
    #logmsg("** Transaction $t: retry time has not arrived after try $nfails.");
            next;
        }
    ◇
```

Macro referenced in 400.

13.3.6 Execute cluster synchronisation command

This subroutine executes and optionally logs a system command executed to perform a cluster synchronisation transaction.

If the request fails with a message indicating a failure to contact the destination host, the host name is placed in the `%failed_hosts` hash, with a value indicating the time at which requests to that host will be tried again.

We handle failure to delete a file specially. Due to possible race conditions, particularly when replacing RememberMe files in logins, we may have a file deletion transaction queued for which the file has never been created on the cluster host. To avoid having this transaction be retried forever, we consider the deletion of a file which doesn't exist on the destination host as being successful.

"HDiet/ClusterSync.pl" 404 ≡

```
sub syncCommand {
    my ($cmd, $host, $tfile) = @_;

    logmsg("    Command: $cmd");

    if (!$nosync) {
        my $tfh = new File::Temp(TEMPLATE => '/tmp/HDClusterXXXXXXXXXXXX',
                         UNLINK => 1,
                         SUFFIX => '.hdc');
        $cmd = untaint($cmd);
        my $status = system($cmd . ">$tfh 2>&1");

        my @results;
        my $jres;
        if ($status != 0) {
            seek($tfh, 0, 0);
            @results = <$tfh>;
            close($tfh);
            my $jres = join("", @results);

            ⟨ Check for errors we deem harmless to cluster synchronisation 405a ⟩
        }

        if ($status == 0) {
            logmsg("        Executed OK.");
            unlink($tfile) ||
                die("Cannot delete cluster transaction file $tfile");
            $tranqueue--;
            return join("", @results);

        } else {
            logmsg("        ***Sync command failed, status $status: $cmd");

            if ($jres =~ m/(Connection timed out|Connection refused|lost connection)/) {
                $failed_hosts{$host} = time() + ⟨ Cluster Transaction Retry Time Interval 8a ⟩;
                logmsg("Marking host $host failed until " .
                       scalar(localtime($failed_hosts{$host})) . "\n");
            }
            return $jres;
        }
    }
    return undef;
}
```

◇

File defined by 397a, 404, 405b, 406ab.

13.3.6.1 Check for errors we deem harmless to cluster synchronisation

Due to race conditions, delays in processing transactions, recovery from transient failures, and other events which befall programs running in the real world, it may come to pass that we attempt to process a file delete or copy transaction for a file which no longer exists on the server. Rather than fail the transaction through the usual mechanism, which would cause it to be senselessly retried several times before being abandoned, we check for these cases specifically and deem the transaction successful if it's one of the harmless cases.

⟨ Check for errors we deem harmless to cluster synchronisation 405a ⟩ ≡

```
        if ($jres =~ m/rm: cannot remove\s.*No such file or directory/) {
            logmsg("        Deeming delete of nonexistent file successful.");
            $status = 0;
        }

        if (($cmd =~ m/^scp /) && ($jres =~ m/: No such file or directory/)) {
            logmsg("        Deeming copy of nonexistent file successful.");
            $status = 0;
        }

        if ($jres =~ m/mkdir:\s+cannot\s+create\s+directory.*:\s+File\s+exists/) {
            logmsg("        Deeming creation of already-extant directory successful.");
            $status = 0;
        }

        if ($jres =~ m/rmdir:\s+.*No\s+such\s+file\s+or\s+directory/) {
            logmsg("        Deeming removal of nonexistent directory successful.");
            $status = 0;
        }
    ◇
```

Macro referenced in 404.

13.3.7 Output a log message

The message argument is output to the log file if logging is configured and copied to standard output if verbose mode is on.

"HDiet/ClusterSync.pl" 405b ≡

```
        sub logmsg {
            my ($msg) = @_;

            print("$msg\n") if $verbose;
            print(LOG "$msg\n") if $logging;

        }
    ◇
```

File defined by 397a, 404, 405b, 406ab.

13.3.8 Conditionally un-taint a variable

The first argument is tested to match the pattern given by the second argument (an arbitrary string if no pattern is specified). If it matches, an untainted instance of the string is returned. This can be used to sanitise input from untrusted sources based upon matching defined patterns.

"HDiet/ClusterSync.pl" 406a ≡

```
        sub untaint {
            my ($val, $pat) = @_;
            $pat = qr/.*/ if !defined($pat);
            if (!($val =~ m/^($pat)$/)) {
                die("Failure to validate pattern in untaint");
            }
            return $1;
        }
    ◇
```

File defined by 397a, 404, 405b, 406ab.

13.3.9 Display if variable is tainted

For diagnostic purposes, it's handy to be able to test whether a value is tainted. This function is called with an arbitrary name and a static value and prints a message indicating whether the value is tainted.

"HDiet/ClusterSync.pl" 406b ≡

```
        sub taintso {
            my ($name, $var) = @_;
            my $zip = substr($var, 0, 0);
            local $@;
            eval { eval "# $zip" };
            if (length($@) != 0) {
                print("$name tainted.\n");
            } else {
                print("$name clean.\n");
            }
        }
    ◇
```

File defined by 397a, 404, 405b, 406ab.

Chapter 14

HTML utilities

The following utility functions are used in the generation of HTML files.

"HDiet/html.pm" 407 ≡

```
#! ⟨ Perl directory 7d ⟩

    ⟨ Perl language modes 369b ⟩

    package HDiet::html;

    require Exporter;

    our @ISA = qw(Exporter);
    our @EXPORT = qw( write_XHTML_prologue
                      generate_XHTML_navigation_bar
                      write_XHTML_epilogue
                      quoteHTML quoteHTMLFile );
    our @EXPORT_OK = qw( );
    1;

    ⟨ Write XHTML prologue 408a, ... ⟩
    ⟨ Generate XHTML navigation bar 410a, ... ⟩
    ⟨ Write XHTML epilogue 412 ⟩

    ⟨ Quote text for inclusion in HTML 413 ⟩

◇
```

14.1 Write XHTML prologue

The prologue for an XHTML result file is written to the already-open UTF-8 file handle `$fh`. Note that if we're generating the result from invocation as a CGI program, the MIME `Content-type` must previously have been output. The optional `$onload` argument allows specification of additional code to be executed in the `onload` event handler for the page. If the optional `$handheld` argument is true, the prologue will be adapted to the small screen of a handheld device.

⟨ Write XHTML prologue 408a ⟩ ≡

```
sub write_XHTML_prologue {
    my ($fh, $homeBase, $pageTitle, $onload, $handheld, $noheader) = @_;

    $onload = '' if !$onload;
    $noheader = 0 if !$noheader;
    my $stylesheet = $handheld ? 'hdiet_handheld' : 'hdiet';
    print $fh <<"EOD";
```

◇

Macro defined by 408ab, 409.
Macro referenced in 407.

Our documents all use the XHTML 1.0 Strict Document Type Definition. We use the UTF-8 character set, which is declared by a "`meta http-equiv`" tag. If any additional HTTP header items have been queued in the `HTTP_header` array, they are emitted in the header.

⟨ Write XHTML prologue 408b ⟩ ≡

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<title>The Hacker's Diet Online: $pageTitle</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
EOD

    my $umeta;
    while ($umeta = shift(@::HTTP_header)) {
        $umeta =~ s/^(\S+):\s+//;
        my $mtype = $1;
        print($fh "<meta http-equiv=\"$1\" content=\"$umeta\" />\n");
    }

    print $fh <<"EOD";
<link rel="stylesheet" href="$homeBase/$stylesheet.css" type="text/css" />
<link rel="shortcut icon" href="$homeBase/figures/hdicon.ico" />
<script type="text/javascript" src="$homeBase/hdiet.js">
</script>
</head>
```

◇

Macro defined by 408ab, 409.
Macro referenced in 407.

When the document is loaded, we call the `initialiseDocument` JavaScript function which transforms the target specifications of external links and determines the time zone offset from Universal Time. The caller may supply additional `onload` code via the `$onload` argument.

We generate a standard heading at the top of the page: a fancy table-based header for the desktop version and a simple list of destinations for handheld devices.

⟨ Write XHTML prologue 409 ⟩ ≡

```
<body onload="initialiseDocument();$onload">

EOD

        if (!$noheader) {
            if ($handheld) {
                print $fh <<"EOD";
<h1 class="c"><a href="⟨ Application documentation URL 11h ⟩"><span class="title1">The Hacker's Diet <em>Online</

EOD
            } else {
                print $fh <<"EOD";
<table class="title">
<tr>
<td class="licon">
<a href="⟨ Site home URL 11f ⟩/" class="i"><img src="$homeBase/figures/swlogo.png"
    id="flicon"
    class="b0" width="82" height="74"
    alt="Fourmilab home" />
</a>
</td>
<td align="center" valign="top">
<a href="⟨ Application documentation URL 11h ⟩"><span class="title1">The Hacker's Diet <em>Online</em></span></a>
<span class="title2">How to lose weight and hair<br />
through stress and poor nutrition</span>
</td>
<td class="ricon">
<a href="⟨ Book home URL 11g ⟩" class="i"><img src="$homeBase/figures/titleicon.png"
    id="hdicon"
    class="b0" width="82" height="80"
    alt="The Hacker's Diet Home" /></a>
</td>
</tr>
</table>

EOD
            }
        }
    }
```

◇

Macro defined by 408ab, 409.
Macro referenced in 407.

14.2 Generate XHTML navigation bar

Every page has a standard navigation bar at the top. This allows random access among the most frequently referenced pages. The caller must supply, in addition to the customary output stream and application URL, the session ID and the identifier (as defined in the `%dest` hash below) of its page; the latter is used to highlight that item in the navigation bar as the current page and disable its link. When displaying the navigation bar on a minor page (for example, CSV import) which does not appear within the navigation bar itself, omit the page identifier or specify "`Other`".

The optional `$linkspec` argument specifies text which is included verbatim in the links included in the navigation bar. This may be used, for example, to invoke JavaScript code which warns the user of unsaved changes before navigating away from a page.

The optional `$browse_public` argument disables the "`Settings`" item in the navigation bar while the user is browsing a public account.

The `$timeZoneOffset` argument specifies the time zone offset between the user's site and UTC (if known), or "`unknown`" if it could not be determined.

⟨ Generate XHTML navigation bar 410a ⟩ ≡

```
sub generate_XHTML_navigation_bar {
    my ($fh, $homeBase, $session, $thispage, $linkspec, $browse_public, $timeZoneOffset) = @_;

    $thispage = "Other" if !defined($thispage);
    $linkspec = $linkspec ? (' ' . $linkspec) : '';

    my $lurl = "<a class=\"navbar\"$linkspec href=\"⟨ URL to invoke this program 12a ⟩?s=$session&amp;q=";
    my $eurl = "\">";

    my $tz = "&amp;HDiet_tzoffset=$timeZoneOffset";
}
```

◇

Macro defined by 410ab, 411.
Macro referenced in 407.

The following hash defines the destinations for the navigation bar. These are the transaction names and optional other CGI arguments for the destinations.

⟨ Generate XHTML navigation bar 410b ⟩ ≡

```
    my %dest = (
                    Log => "log&amp;m=now",
                    History => "calendar",
                    Chart => "histreq",
                    Trend => "trendan",
                    Settings => "modacct",
                    Utilities => "account",
                    Signoff => "logout"
                );

    $dest{$thispage} = '';
```

◇

Macro defined by 410ab, 411.
Macro referenced in 407.

The `$thispage` argument allows specifying which page we are currently displaying. That item is displayed highlighted in the navigation bar and no link is attached to it, since there's no point in navigating back to the same page we're on. It is perfectly valid to specify `undef` for `$thispage`—this indicates you're on a page which isn't linked directly to the navigation bar; in this case all of the links in the bar will be active.

⟨ Generate XHTML navigation bar 411 ⟩ ≡

```
            my (%elink, %active);
            for my $k (keys %dest) {
               if ($dest{$k} ne '') {
                    $dest{$k} = $lurl . $dest{$k} . $tz . $eurl;
                    $elink{$k} = '</a>';
                    $active{$k} = '';
                } else {
                    $elink{$k} = '';
                    $active{$k} = ' class="active"';
                }
            }

            if ($browse_public) {
                $dest{Settings} = '';
                $elink{Settings} = '';
                $active{Settings} = ' class="disabled"';
            }

            print $fh <<"EOD";
        <table class="navbar">
            <tr>
                <td title="Display the current monthly log"$active{Log}>$dest{Log}Log$elink{Log}</td>
                <td title="Show a calendar of all monthly logs"$active{History}>$dest{History}History$elink{History}</
                <td title="Generate historical charts"$active{Chart}>$dest{Chart}Chart$elink{Chart}</td>
                <td title="Analyse weight trend and energy balance"$active{Trend}>$dest{Trend}Trend$elink{Trend}</td>
                <td title="Edit your account settings"$active{Settings}>$dest{Settings}Settings$elink{Settings}</td>
                <td title="Perform various utility functions"$active{Utilities}>$dest{Utilities}Utilities$elink{Utilit
                <td class="pad"></td>
                <td title="Sign off from The Hacker's Diet Online"$active{Signoff}>$dest{Signoff}Sign Out$elink{S
            </tr>
        </table>
        EOD
            }
        ◇
```

Macro defined by 410ab, 411.
Macro referenced in 407.

14.3 Write XHTML epilogue

Our XHTML files have a stereotyped epilogue which is generated by the following code.

⟨ Write XHTML epilogue 412 ⟩ ≡

```
        sub write_XHTML_epilogue {
            my ($fh, $homeBase) = @_;

            print $fh <<"EOD";
    </body>
    </html>
    EOD
        }
    ◇
```

Macro referenced in 407.

14.4 Quote text for inclusion in HTML

The `quoteHTML` function quotes all HTML metacharacters in its argument and expands characters which are not Latin-1 graphics to HTML numeric entities. The quoted string is returned.

The `quoteHTMLFile` copies text in an input file handle to the output file handle, applying `quoteHTML` to all lines.

⟨ Quote text for inclusion in HTML 413 ⟩ ≡

```
sub quoteHTML {
    my ($s) = @_;

    my $os = '';

    while ($s =~ s/^(.)//s) {
        my $o = ord($1);
        if ($1 eq "\n") {
            $os .= $1;
        } elsif (($1 eq '<') || ($1 eq '>') || ($1 eq '&') || ($1 eq '"') ||
            ($o < 32) ||
            (($o >= 127) && ($o < 161)) || ($o > 255)) {
            $os .= "&#$o;";
        } else {
            $os .= $1;
        }
    }
    return $os;
}

sub quoteHTMLFile {
    my ($ifh, $ofh) = @_;

    while (<$ifh>) {
        print($ofh quoteHTML($_));
    }
}
```

◇

Macro referenced in 407.

Chapter 15

XML utilities

The following utility functions are used in the generation of XML files.

"HDiet/xml.pm" 414 ≡

```
#! ⟨ Perl directory 7d ⟩

    ⟨ Perl language modes 369b ⟩

    package HDiet::xml;

    require Exporter;

    our @ISA = qw(Exporter);
    our @EXPORT = qw(
                      generateXMLprologue
                      generateXMLepilogue
                      textXML
                      quoteXML
                      timeXML
                    );
    our @EXPORT_OK = qw( );
    1;
```

◇

File defined by 414, 415ab, 416ab, 417.

15.1 Generate XML prologue

The standard XML prologue, including XML version, character set, DOCTYPE, and our universal root element are written to the file handle argument.

"HDiet/xml.pm" 415a ≡

```
        sub generateXMLprologue {
            my ($fh) = @_;

            print $fh <<"EOD";
    <?xml version="1.0" encoding="UTF-8"?>
    <?xml-stylesheet type="text/css" href="http://www.fourmilab.ch/hackdiet/online/hackdiet_db.css"?>
    <!DOCTYPE hackersdiet SYSTEM
              "http://www.fourmilab.ch/hackdiet/online/hackersdiet.dtd">
    <hackersdiet version="1.0">
    EOD
        }
    ◇
```

File defined by 414, 415ab, 416ab, 417.

15.2 Generate XML epilogue

The standard XML epilogue, which simply closes the root element from the prologue, is written to the file handle argument.

"HDiet/xml.pm" 415b ≡

```
        sub generateXMLepilogue {
            my ($fh) = @_;

            print $fh <<"EOD";
    </hackersdiet>
    EOD
        }
    ◇
```

File defined by 414, 415ab, 416ab, 417.

15.3 QuoteXML

The `quoteXML` function replaces XML metacharacters with named entities; only ampersand and the less and greater than signs need be replaced in this application. In addition, if the `$safe` argument is true, all character with Unicode code points of 128 decimal and above are replaced with XML hexadecimal numeric entities, resulting in a file which, even though nominally encoded in UTF-8, can be edited with an editor aware only of the 7-bit ASCII set.

"HDiet/xml.pm" 416a ≡

```
sub quoteXML {
    my ($s, $safe) = @_;

    $s =~ s/&/&amp;/g;
    $s =~ s/</&lt;/g;
    $s =~ s/>/&gt;/g;
    if ($safe) {
        $s =~ s/([\x{80}-\x{FFFF}])/sprintf("&#x%x;", ord($1))/eg;
    }
    return $s;
}
```

◇

File defined by 414, 415ab, 416ab, 417.

15.4 TextXML

The `textXML` function generates an XML element containing the text given by the argument, which is quoted if necessary to escape any metacharacters and/or avoid non-ASCII character if `$safe` is specified. The text is wrapped in an element with the specified `$tagname`.

"HDiet/xml.pm" 416b ≡

```
sub textXML {
    my ($tagname, $s, $safe) = @_;

    $s = quoteXML($s, $safe);
    my $etagname = $tagname;
    $etagname =~ s/\s+.*$//;
    return "<$tagname>$s</$etagname>";
}
```

◇

File defined by 414, 415ab, 416ab, 417.

15.5 UNIX time to ISO date and time

The `textXML` function generates an XML element containing the text given by the argument, which is quoted if necessary to escape any metacharacters and/or avoid non-ASCII character if `$safe` is specified. The text is wrapped in an element with the specified `$tagname`.

"HDiet/xml.pm" 417 ≡

```
sub timeXML {
    my ($utime) = @_;

    my @lmod = gmtime($utime);
    my $lm = sprintf("%04d-%02d-%02dT%02d:%02d:%02dZ",
        $lmod[5] + 1900, $lmod[4] + 1, $lmod[3], $lmod[2], $lmod[1], $lmod[0]);

    return $lm;
}
```

◇

File defined by 414, 415ab, 416ab, 417.

Chapter 16

Julian date utilities

We use Julian day numbers to represent times and dates. This avoids all cultural bias and eliminates worries about overflow apocalypses in 2038 and beyond. Time is represented as a fractional day. Note that Julian day numbers start at noon. This package makes no aassuptions regarding the time zone in which Julian day numbers are defined, but they should be used only for UTC/GMT times and dates. This program conforms to that convention.

"HDiet/Julian.pm" 419 ≡

```
#! ⟨ Perl directory 7d ⟩

    ⟨ Perl language modes 369b ⟩

    package HDiet::Julian;

    require Exporter;

    our @ISA = qw(Exporter);
    our @EXPORT = qw(gregorian_to_jd jd_to_gregorian jd_to_weekday
        civil_time_to_jd jd_to_civil_time
        unix_time_to_jd jd_to_unix_time unix_time_to_civil_date_time
        jd_to_RFC_822_date jd_to_RFC_3339_date jd_to_old_cookie_date);
    our @EXPORT_OK = qw(leap_gregorian GREGORIAN_EPOCH WEEKDAY_NAMES
        MONTH_ABBREVIATIONS);
    1;

    ⟨ Julian date constant definitions 420a ⟩
    ⟨ Julian date support functions 420b ⟩

    ⟨ Gregorian leap year computation 421a ⟩
    ⟨ Gregorian date to Julian day number 421b ⟩

    ⟨ Julian day to Gregorian date 422 ⟩
    ⟨ Julian day to day of week 423a ⟩

    ⟨ Civil time to Julian day fraction 423b ⟩
    ⟨ Julian day fraction to civil time 424a ⟩

    ⟨ Unix time to Julian day and fraction 424b ⟩
    ⟨ Julian day and fraction to Unix time 424c ⟩
    ⟨ Unix time to civil date and time 425a ⟩

    ⟨ Julian day and fraction to RFC 822 time and date 425b ⟩
    ⟨ Julian day and fraction to RFC 3339 time and date 426a ⟩
    ⟨ Julian day and fraction to old HTTP cookie time and date 426b ⟩

◇
```

16.1 Julian date constant definitions

The following constants are used in Julian date computation and/or furnished to users of this package for convenience. In the latter category are the names of weekdays and abbreviations for month names.

⟨ Julian date constant definitions 420a ⟩ ≡

```
use constant GREGORIAN_EPOCH => 1721425.5;
use constant WEEKDAY_NAMES => [ "Sunday", "Monday", "Tuesday", "Wednesday",
                                "Thursday", "Friday", "Saturday" ];
use constant MONTH_ABBREVIATIONS => [
    "Jan", "Feb", "Mar", "Apr", "May", "Jun",
    "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" ];

use constant J1970 => 2440587.5;    # Julian date at Unix epoch: 1970-01-01
```

◇

Macro referenced in 419.

16.2 Julian date support functions

Our Julian day functions require a proper modulus function which works for non-integers and a floor function which behaves properly for negative numbers. If these prove useful elsewhere, they may be promoted to their own package and imported where needed.

⟨ Julian date support functions 420b ⟩ ≡

```
#   MOD  --  Modulus function which works for non-integers.

sub mod {
    my ($a, $b) = @_;

    return $a - ($b * floor($a / $b));
}

#   FLOOR  -- Round number to the nearest integer less than
#             the argument.  Note that, unlike int(), floor(-1.5) = -2.

sub floor {
    my $x  = shift;
    my $ix = int($x);
    return (($x >= 0) || ($x == $ix)) ? $ix : ($ix - 1);
}
```

◇

Macro referenced in 419.

16.3 Gregorian leap year computation

Determine if the Gregorian year argument is a leap year. Returns 1 if the year is a leap year, 0 otherwise. This works for years prior to the adoption of the Gregorian calendar including negative year numbers.

⟨ Gregorian leap year computation 421a ⟩ ≡

```
sub leap_gregorian {
    my $year = shift;

    return (($year % 4) == 0) &&
            (!((($year % 100) == 0) && (($year % 400) != 0)));
}
```

◇

Macro referenced in 419.

16.4 Gregorian date to Julian day number

The `$year`, `$month`, and `$day` arguments specifying a date in the (proleptic) Gregorian calendar are converted to the corresponding Julian day number at noon on that date. Note the `$month` is a calendar month, with 1 denoting January.

⟨ Gregorian date to Julian day number 421b ⟩ ≡

```
sub gregorian_to_jd {
    my ($year, $month, $day) = @_;

    return (GREGORIAN_EPOCH - 1) +
           (365 * ($year - 1)) +
           floor(($year - 1) / 4.0) +
           (-floor(($year - 1) / 100.0)) +
           floor(($year - 1) / 400.0) +
           floor((((367 * $month) - 362) / 12) +
           (($month <= 2) ? 0 :
                            (leap_gregorian($year) ? -1 : -2)
           ) +
           $day);
}
```

◇

Macro referenced in 419.

16.5 Julian day to Gregorian date

Convert the Julian day argument (which may contain a day fraction) the a proleptic Gregorian date, adjusting for the noon origin of Julian days and midnight for Gregorian dates. The year, calendar month (1=January), and day of the month are returned as an array.

⟨ Julian day to Gregorian date 422 ⟩ ≡

```
sub jd_to_gregorian {
    my $jd = shift;

    my ($wjd, $depoch, $quadricent, $dqc, $cent, $dcent, $quad, $dquad,
        $yindex, $yearday, $leapadj, $year, $month, $day);

    $wjd = floor($jd - 0.5) + 0.5;
    $depoch = $wjd - GREGORIAN_EPOCH;
    $quadricent = floor($depoch / 146097);
    $dqc = mod($depoch, 146097);
    $cent = floor($dqc / 36524);
    $dcent = mod($dqc, 36524);
    $quad = floor($dcent / 1461);
    $dquad = mod($dcent, 1461);
    $yindex = floor($dquad / 365);
    $year = ($quadricent * 400) + ($cent * 100) + ($quad * 4) + $yindex;
    if (!(($cent == 4) || ($yindex == 4))) {
        $year++;
    }
    $yearday = $wjd - gregorian_to_jd($year, 1, 1);
    $leapadj = (($wjd < gregorian_to_jd($year, 3, 1)) ? 0
                                                      :
                    (leap_gregorian($year) ? 1 : 2)
               );
    $month = int((((($yearday + $leapadj) * 12) + 373) / 367);
    $day = ($wjd - gregorian_to_jd($year, $month, 1)) + 1;

    return ($year, $month, $day);
}
```

◇

Macro referenced in 419.

16.6 Julian day to day of week

An index denoting the day of the week (0=Sunday... 6=Saturday) is returned for the Julian day argument.

⟨ Julian day to day of week 423a ⟩ ≡

```
sub jd_to_weekday {
    my $j = shift;
    my $ij = int($j + 1.5);

    $ij %= 7;

    return ($ij < 0) ? (7 + $ij) : $ij;
}
```

◇

Macro referenced in 419.

16.7 Civil time to Julian day fraction

Three arguments specifying the midnight-based hour, minute, and second in civil time are converted to a day fraction which may be added to the result of `gregorian_to_jd`. The seconds argument may contain a fraction of seconds.

⟨ Civil time to Julian day fraction 423b ⟩ ≡

```
sub civil_time_to_jd {
    my ($hour, $min, $sec) = @_;

    my $s = $sec + (60 * ($min + (60 * $hour)));
    return ($s / (24 * 60 * 60));
}
```

◇

Macro referenced in 419.

16.8 Julian day fraction to civil time

Convert the day fraction in the noon-based Julian day argument of civil hour, minute, and second, which are returned as an array. Note that seconds are rounded to the nearest integer.

⟨ Julian day fraction to civil time 424a ⟩ ≡

```
sub jd_to_civil_time {
    my $j = shift;
    my ($ij, $hh, $mm, $ss);

    $j += 0.5;                    # Astronomical to civil
    $ij = int(($j - floor($j)) * 86400.5);
    $hh = int($ij / 3600);
    $mm = int(($ij / 60) % 60);
    $ss = int(($ij % 60) + 0.5);
    return ($hh, $mm, $ss);
}
```

◇

Macro referenced in 419.

16.9 UNIX time to Julian day and fraction

UNIX `time()` values are defined as the number of seconds elapsed since 1970-01-01 at 00:00 UTC, not counting leap seconds. This is a pure offset from Julian Day number, so we need only add the offset and scale appropriately. The argument will usually be an integer, but need not be; it may contain fractional seconds.

⟨ Unix time to Julian day and fraction 424b ⟩ ≡

```
sub unix_time_to_jd {
    my $ut = shift;

    return J1970 + ($ut / (24 * 60 * 60));
}
```

◇

Macro referenced in 419.

16.10 Julian day and fraction to UNIX time

Even though we support arbitrary day fractions (up to the precision of Perl numbers), UNIX `time()` values are traditionally integers, so we force the result to be an integer here.

⟨ Julian day and fraction to Unix time 424c ⟩ ≡

```
sub jd_to_unix_time {
    my $j = shift;

    return int((($j - J1970) * (24 * 60 * 60)) + 0.5);
}
```

◇

Macro referenced in 419.

16.11 UNIX time to civil date and time

This function isn't properly a Julian date utility, but a convenience for code which needs to work with UNIX `time()` values which may exceed the "doomsday date" of 2038-01-19. Perl, as of version 5.8, relies upon the C library's `gmtime` function to implement its own `gmtime`, which results in disastrous truncation for days after doomsday on 32-bit machines.

This function can be used to replace most (but not all) applications of Perl's `gmtime`, and avoids some of its C-legacy eccentricity. It is called with a generalised UNIX `time` value which may be negative or a positive value requiring more than 32 bits to represent—any value corresponding to a non-negative Julian day number is accepted. A list is returned, containing values as follows:

```
($year, $month, $day, $hour, $minute, $second) = unix_time_to_civil_date_time(time());
```

The year is the full Gregorian calendar year (no need to add 1900), and the month value for January is 1.

⟨ Unix time to civil date and time 425a ⟩ ≡

```
sub unix_time_to_civil_date_time {
    my $j = unix_time_to_jd(shift);

    my @dt = jd_to_gregorian($j);
    push(@dt, jd_to_civil_time($j));
    return @dt;
}
```

◇

Macro referenced in 419.

16.12 Julian day and fraction to RFC 822 time and date

Comvert a Julian day and fraction to a date and time conforming to the RFC 822 specification for Internet mail. We assume that the Julian day represents UTC and unconditionally specify the time zone as "+0000" in the result.

⟨ Julian day and fraction to RFC 822 time and date 425b ⟩ ≡

```
sub jd_to_RFC_822_date {
    my $j = shift;

    my ($uy, $um, $ud) = jd_to_gregorian($j);
    my ($uhh, $umm, $uss) = jd_to_civil_time($j);

   return sprintf("%02d/%s/%04d:%02d:%02d:%02d +0000",
        $ud, MONTH_ABBREVIATIONS->[$um], $uy, $uhh, $umm, $uss);
}
```

◇

Macro referenced in 419.

16.13 Julian day and fraction to RFC 3339 time and date

Comvert a Julian day and fraction to a date and time conforming to the RFC 3339 date and time specification format, confirming to ISO 8601. We assume that the Julian day represents UTC and unconditionally specify the time zone as "Z" in the result.

⟨ Julian day and fraction to RFC 3339 time and date 426a ⟩ ≡

```
sub jd_to_RFC_3339_date {
    my $j = shift;

    my ($uy, $um, $ud) = jd_to_gregorian($j);
    my ($uhh, $umm, $uss) = jd_to_civil_time($j);

    return sprintf("%04d-%02d-%02dT%02d:%02d:%02dZ",
        $uy, $um, $ud, $uhh, $umm, $uss);
}
```

◇

Macro referenced in 419.

16.14 Julian day and fraction to old HTTP cookie time and date

Comvert a Julian day and fraction to a date and time conforming to the format used in "old" HTTP cookies, as defined in the "HISTORICAL" section of RFC 2109 . (This is, in fact, a decade after the RFC, the format which most sites continue to use for cookies. We assume that the Julian day represents UTC.

⟨ Julian day and fraction to old HTTP cookie time and date 426b ⟩ ≡

```
sub jd_to_old_cookie_date {
    my $j = shift;

    my ($uy, $um, $ud) = jd_to_gregorian($j);
    my ($uhh, $umm, $uss) = jd_to_civil_time($j);
    my $wdn = substr(WEEKDAY_NAMES->[jd_to_weekday($j)], 0, 3);
    my $mabb = MONTH_ABBREVIATIONS->[$um - 1];

    return sprintf("$wdn, %02d-$mabb-%04d %02d:%02d:%02d GMT",
        $ud, $uy, $uhh, $umm, $uss);
}
```

◇

Macro referenced in 419.

Chapter 17

Documentation in POD format

⟨ Documentation in POD format 427 ⟩ ≡

```
=head1 NAME

HackDiet - Hacker's Diet Online Database Interface

=head1 SYNOPSIS

B<HackDiet.pl>
[I<options>]

=head1 DESCRIPTION

```

⟨ Options documentation 428a, ... ⟩

```
=head1 VERSION

This is B<HackDiet> version ⟨ Version 3a ⟩, released ⟨ Release Date 3b ⟩.
The current version of this program is always posted at
⟨ Book home URL 11g ⟩/online/.

=head1 AUTHOR

John Walker
(⟨ Site home URL 11f ⟩/)

=head1 BUGS

Please report any bugs to bugs@fourmilab.ch.

=head1 SEE ALSO

B<nuweb> (http://nuweb.sourceforge.net/),
S<Literate Programming> (http://www.literateprogramming.com/).

=head1 COPYRIGHT

This program is in the public domain.

=cut
```

◇

Macro referenced in 163.

17.0.1 Options

Here we document the command-line options.

⟨ Options documentation 428a ⟩ ≡

```
=head1 OPTIONS

All options may be abbreviated to their shortest
unambiguous prefix.

=over 5
◇
```

Macro defined by 428abcd, 429.
Macro referenced in 427.

17.0.1.1 `--copyright`

⟨ Options documentation 428b ⟩ ≡

```
=item B<--copyright>

Display copyright information.
◇
```

Macro defined by 428abcd, 429.
Macro referenced in 427.

17.0.1.2 `--help`

⟨ Options documentation 428c ⟩ ≡

```
=item B<--help>

Display how to call information.

◇
```

Macro defined by 428abcd, 429.
Macro referenced in 427.

17.0.1.3 `--verbose`

⟨ Options documentation 428d ⟩ ≡

```
=item B<--verbose>

Generate verbose output to indicate what's going on.

◇
```

Macro defined by 428abcd, 429.
Macro referenced in 427.

17.0.1.4 `--version`

⟨ Options documentation 429 ⟩ ≡

```
=item B<--version>

Display version number.

=back
=cut
◇
```

Macro defined by 428abcd, 429.
Macro referenced in 427.

Chapter 18

`HackDietBadge.pl`: Return badge for user

`HackDietBadge.pl` is invoked by a URL provided to a user who wishes to display a "status badge" on their Web page or log. If a badge image is configured, every time a change is made which might affect the trend, the `BadgeImage.png` image in the user's directory is updated with the current data. This is a lightweight program which doesn't pull in any of the heavy machinery of the main application. All it does is parse the query, extract the encrypted user name specification, verify it, and if everything is valid copy the badge image as the CGI result. If an error is detected, a canned "Invalid request" image is returned and a diagnostic message is output to `STDERR` which will appear in the Web server's error log.

18.1 HackDietBadge

"HackDietBadge.pl" 431 ≡

```
#! ⟨ Perl directory 7d ⟩

    ⟨ Perl language modes 369b ⟩

    use Crypt::OpenSSL::AES;
    use Crypt::CBC;
    use HDiet::Digest::Crc32;

    print <<"EOD";
Content-type: image/png\r
Pragma: no-cache\r
EOD

    if (defined($ENV{QUERY_STRING}) && ($ENV{QUERY_STRING} =~ m/b=([0-9FGJKQW]+)/)) {
        my $cuserid = $1;

        my $btype;
        if ($ENV{QUERY_STRING} =~ m/t=(\d+)/) {
            $btype = $1;
        } else {
            $btype = 0;
        }

        my $user_file_name;

        eval {
            $user_file_name = decodeEncryptedUserID($cuserid);
        };

        if ((!$@) && defined($user_file_name) &&
            open(BI, "<⟨ Users Directory 6h ⟩/$user_file_name/BadgeImage.png")) {
        } else {
            open(BI, "<⟨ Image and Icon Directory 6c ⟩/steenkin_badge.png") ||
                die("Cannot open ⟨ Image and Icon Directory 6c ⟩/steenkin_badge.png");
            print(STDERR "HackDietBadge: Bogus or corrupted user specification\n");
        }
    } else {
        open(BI, "<⟨ Image and Icon Directory 6c ⟩/steenkin_badge.png") ||
            die("Cannot open ⟨ Image and Icon Directory 6c ⟩/steenkin_badge.png");
        print(STDERR "HackDietBadge: Invalid or missing query string\n");
    }

    print("Content-Length: ", -s BI, "\r\n\r\n");

    my $iobuf;
    while (read(BI, $iobuf, 65536)) {
        print($iobuf);
    }
    close(BI);

    ⟨ Decode encrypted user ID 136 ⟩
◇
```

Chapter 19

jig.pl: Test Jig

The test jig is an executable program that includes all of the modules of the CGI application which is used to run specific module tests from the command line. The nature of the tests evolves as the program is developed and depends upon the current state of the project.

"jig.pl" 433 ≡

```
#! ⟨ Perl directory 7d ⟩

    ⟨ Global declarations 369a ⟩

    binmode(STDOUT, ":utf8");
    binmode(STDIN, ":utf8");

use Data::Dumper;


    my $user_file_name = quoteUserName('John Walker');
    if (!(-f "⟨ Users Directory 6h ⟩/$user_file_name/UserAccount.hdu")
        || (!open(FU, "<:utf8", "⟨ Users Directory 6h ⟩/$user_file_name/UserAccount.hdu"))) {
        die("Cannot open ⟨ Users Directory 6h ⟩/$user_file_name/UserAccount.hdu");
    }

    my $ui = HDiet::user->new();
    $ui->load(\*FU);
    close(FU);

#    $ui->describe();

    my $uec = $ui->generateEncryptedUserID();
    my $duec = decodeEncryptedUserID($uec);
#    print(Dumper($uec, $duec));
    print("Encoded ID: ($uec)\n");
    print("User ID: ($duec)\n");

    my $hist = HDiet::history->new($ui, $user_file_name);
    open(BF, ">/tmp/steenk.png") || die("Cannot create /tmp/steenk.png");
    $hist->drawBadgeImage(\*BF, 14);
    close(BF);

    exit(0);

    ⟨ Decode encrypted user ID 136 ⟩

    ⟨ Utility functions 373 ⟩

◇
```

Chapter 20

Bowdler.pl: Bowdlerise Source for Publication

This program is developed as a ready to run application for Fourmilab. Necessarily, it contains various private information (for example, the incantation which feedback messages use to glide past the junk mail filter with impunity). This little filter, which reads from standard input and writes to standard output, has hard-coded pattern matches which strip this information and replace it with innocuous strings one wishing to install the application at another site can replace with their own private codes. Note that running this program over itself will convert all of the substitutions to the identity transform and hence bowdlerise itself.

"Bowdler.pl" 434 ≡

```
#! ⟨ Perl directory 7d ⟩

    ⟨ Global declarations 369a ⟩

    my $l;
    while ($l = <>) {
        #   Beta test backdoor
        $l =~ s/'Beta luck next time'/'Beta luck next time';# Beta test backdoor
        #   Confirmation signature encoding suffix
        $l =~ s/"Sodium Chloride"/"Sodium Chloride"/;
        #   Address for feedback E-mail
        $l =~ s/bitbucket/bitbucket/;
        #   Master encryption key
        $l =~ s/"Super duper top secret!"/"Super duper top secret!"/;

        print($l);
    }
◇
```

Chapter 21

bump: Increment build number

This little shell script increments a number in the file given by the argument. It is used in the `Makefile` to increment the build number each time the Web file is extracted.

"bump" 435 ≡

```
#! /bin/sh

#   Increment a number kept in a file and echo it
#   to standard output.

LAST=‘cat $1‘
NEXT=‘expr $LAST \+ 1‘
echo $NEXT >$1
echo $NEXT
◇
```

Chapter 22

XHTML Style Sheet

The `hdiet.css` style sheet is shared by all documents in the Web tree, both static and dynamically generated.

22.1 Global document properties

"hdiet.css" 436 ≡

```
body {
    margin-left: 10%;
    margin-right: 10%;
    background-color: #FFFFFF;
    color: #000000;
}
```

◇

File defined by 436, 437, 438, 439ab, 440, 441ab, 442ab, 443, 444, 445a.

22.2 Links

The following define the default style for links within the various documents. The class "`i`" links have no static or dynamic decoration whatsoever; they are used in links which wrap images and buttons where it should be self-evident that they constitute a link (and the browser's default link rendering is usually unspeakably ugly).

"hdiet.css" 437 ≡

```
a:link, a:visited {
    background-color: inherit;
    color: rgb(0%, 0%, 80%);
    text-decoration: none;
}

a:hover  {
    background-color: rgb(30%, 30%, 100%);
    color: rgb(100%, 100%, 100%);
}

a:active {
    color: rgb(100%, 0%, 0%);
    background-color: rgb(30%, 30%, 100%);
}

a.i:link, a.i:visited, a.i:hover {
    background-color: inherit;
    color: inherit;
    text-decoration: none;
}
```

◇

File defined by 436, 437, 438, 439ab, 440, 441ab, 442ab, 443, 444, 445a.

22.3 Headings and titles

"hdiet.css" 438 ≡

```
h1.c, h2.c {
    text-align: center;
}

h1.monthyear, h1.pr_monthyear, h1.pr_mo_monthyear, h1.mo_monthyear {
    font-family: Helvetica, Arial, sans-serif;
    text-align: center;
}

h3.acct_category {
    margin-top: 0px;
    text-align: center;
}

h3.browsing {
    font-family: Helvetica, Arial, sans-serif;
    text-align: center;
    color: #FFFFFF;
    background-color:  #00A000;
    width: 66%;
    margin-left: auto;
    margin-right: auto;
    padding-top: 2px;
    padding-bottom: 3px;
}

h3.warning {
    font-family: Helvetica, Arial, sans-serif;
    text-align: justify;
    color: #000000;
    background-color:  #F0F000;
    width: 80%;
    margin-left: auto;
    margin-right: auto;
    padding-left: 8px;
    padding-right: 8px;
    padding-top: 4px;
    padding-bottom: 4px;
}

.monthyear span {
    background-color: #0000FF;
    color: #FFFF00;
    border-width: 4px;
    border-color: #8080FF;
    border-style: outset;
    padding: 6px;
}

span.title1 {
    font-size: x-large;
}

span.title2 {
    font-size: large;
    font-style: italic;
}
```

◇

File defined by 436, 437, 438, 439ab, 440, 441ab, 442ab, 443, 444, 445a.

22.4 Blocks of text

"hdiet.css" 439a ≡

```
        .justified {
            text-align: justify;
        }

        .centred {
            text-align: center;
        }
    ◇
```

File defined by 436, 437, 438, 439ab, 440, 441ab, 442ab, 443, 444, 445a.

22.5 Block text styles

"hdiet.css" 439b ≡

```
        p.acct_summary {
            text-align: center;
            margin-top: 0px;
        }

        p.build {
            text-align: right;
            font-family: sans-serif;
            font-size: smaller;
            color: #909090;
            background-color: inherit;
        }
    ◇
```

File defined by 436, 437, 438, 439ab, 440, 441ab, 442ab, 443, 444, 445a.

22.6 Inline text decoration

"hdiet.css" 440 ≡

```
span.imported {
    background-color: #FFFFFF;
    color: inherit;
}

span.notentry {
    background-color: #FFFF00;
    color: inherit;
}

span.noparse {
    background-color: #FFA0A0;
    color: inherit;
}

span.conflict {
    background-color: #A0FFFF;
    color: inherit;
}

span.administrator {
    color: #00A000;
    background-color: inherit;
}

span.shrill {
    color: #FF0000;
    background-color: inherit;
    font-weight: bold;
    font-style: italic;
}

.darwin {
    /* Why not use background-image?  Because it doesn't work
       with Exploder (neither 6 nor 7).  See Microsoft Krap Bulletin 322240:
           http://support.microsoft.com/kb/322240  */
    background: url(⟨ Web Document Home 5a ⟩/figures/darwin.png);
}
```

◇

File defined by 436, 437, 438, 439ab, 440, 441ab, 442ab, 443, 444, 445a.

22.7 Fieldsets

We use `<fieldset>` tags in the CSV import form to make it clear that importing by uploading a file is a completely distinct operation from importing CSV entries pasted into a text area. The style of this page owes a great deal to the W3C XHTML validator.

"hdiet.css" 441a ≡

```
fieldset {
    padding: .5em;
    background: white;
    border: 1px dotted #9090FF;
    margin-left: 20px;
    margin-right: 20px;
    margin-top: .5em;
}

fieldset legend {
    color: #FFFFFF;
    background-color: #9090FF;
    font-size: smaller;
    padding: .1ex .5ex;
    border-right: 1px solid gray;
    border-bottom: 1px solid gray;
    font-weight: bold;
}

input {
    vertical-align: middle;
}

input.reset {
    text-align: center;
}

input.default {
    text-align: center;
    font-weight: bold;
}
```

◇

File defined by 436, 437, 438, 439ab, 440, 441ab, 442ab, 443, 444, 445a.

22.8 Images

"hdiet.css" 441b ≡

```
img.b0 {
    border: 0px;
}
```

◇

File defined by 436, 437, 438, 439ab, 440, 441ab, 442ab, 443, 444, 445a.

22.9 Canvas

A “canvas” division is positioned by the JavaScript code to overlap the chart image in a monthly log page. This permits the live update code to draw new entries into the chart as they are made in the log.

"hdiet.css" 442a ≡

```
        div.canvas {
            border: 0px;
            position: absolute;
            left: 0px;
            top: 0px;
            height: 100px;
            width: 100px;
            visibility: hidden;
        }
```

◇

File defined by 436, 437, 438, 439ab, 440, 441ab, 442ab, 443, 444, 445a.

22.10 Lists

"hdiet.css" 442b ≡

```
        li.skip {
            margin-top: 1ex;
        }

        ul.goofs {
            font-family: Helvetica, Arial, sans-serif;
            text-align: justify;
            color: #FF0000;
            background-color:  inherit;
            width: 80%;
            margin-left: auto;
            margin-right: auto;
        }
```

◇

File defined by 436, 437, 438, 439ab, 440, 441ab, 442ab, 443, 444, 445a.

22.11 Navigation

"hdiet.css" 443 ≡

```
        p.mlog_buttons {
            text-align: center;
        }

        p.trendan {
            font-weight: bold;
            text-align: center;
        }

        span.accesskey {
            text-decoration: underline;
        }

        span.required {
            background-color: inherit;
            color: #FF0000;
            font-weight: bold;
        }

        h4.nav {
            margin-top: 0px;
            margin-bottom: 0px;
        }
```

◇

File defined by 436, 437, 438, 439ab, 440, 441ab, 442ab, 443, 444, 445a.

22.12 Paper log forms

The following style definitions are used when printing paper log forms. We explicitly insert page breaks before all but the first division container in which the log pages are enclosed.

"hdiet.css" 444 ≡

```
div.plog_subsequent {
    page-break-before: always;
}

h1.plog {
    margin-bottom: 0px;
    text-align: center;
}

h2.plog {
    margin-top: 0px;
    text-align: center;
}

table.plog {
    margin-left: auto;
    margin-right: auto;
    width: 100%;
}

table.plog tr.heading {
    font-weight: bold;
}

table.plog th.h1 {
    text-align: center;
}

table.plog th.h7 {
    text-align: center;
    width: 33em;
}

table.plog th.c1 {
    text-align: right;
    width: 2em;
}

table.plog td.s1 {
    width: 0.5em;
}

table.plog td.c2 {
    text-align: left;
    width: 2em;
}

table.plog td.s2 {
    width: 1em;
}

table.plog td.c3 {
    border-bottom: 1px solid black;
    width: 3em;
}
```

22.13 Tables

"hdiet.css" 445a ≡

⟨ Page title table 445b ⟩

⟨ Sign in and account management tables 446 ⟩

⟨ Monthly log table 447 ⟩

⟨ Navigation bar table 449 ⟩

⟨ Trend analysis table 448 ⟩

⟨ Calendar Navigation Tables 450a ⟩

⟨ Persistent login manager table 450b ⟩

⟨ Feedback message table 451 ⟩

⟨ Global statistics tables 452 ⟩

◇

File defined by 436, 437, 438, 439ab, 440, 441ab, 442ab, 443, 444, 445a.

22.13.1 Page title table

We use a uniform design for the title of all pages. This consists of a table containing two navigation icons at the left and right and the main title centred in the middle.

⟨ Page title table 445b ⟩ ≡

```
table.title {
    width: 100%;
}

table.title td.licon {
    width: 90px;
    text-align: left;
}

table.title td.ricon {
    width: 90px;
    text-align: right;
}
```

◇

Macro referenced in 445a.

22.13.2 Sign in and account management tables

The following styles are used by all of the forms related to account management: new account creation, account settings changes, and sign in.

⟨ Sign in and account management tables 446 ⟩ ≡

```
table.login {
    background-color: #E0E0FF;
    color: #000000;
    margin-left: auto;
    margin-right: auto;
}

table.login th {
    text-align: left;
    padding-right: 1em;
}
```

◇

Macro referenced in 445a.

22.13.3 Monthly log table

The monthly log table includes both static text and editable fields. Span definitions are used to colour code variance items according to their sign.

⟨ Monthly log table 447 ⟩ ≡

```
table.mlog, table.mo_mlog {
    background-color: #D0D0D0;
    color: #000000;
    margin-left: auto;
    margin-right: auto;
}

table.pr_mlog, table.pr_mo_mlog {
    margin-left: auto;
    margin-right: auto;
    border: thin solid;
    border-collapse: collapse;
}

table.pr_mlog td, table.pr_mlog th, table.pr_mo_mlog td, table.pr_mo_mlog th {
    padding-left: 0.2em;
    padding-right: 0.2em;
}

td.r {
    text-align: right;
}

span.r {
    color: #FF0000;
    background-color: #D0D0D0;
}

span.bk {
    color: #000000;
    background-color: #D0D0D0;
}

span.g {
    color: #00A000;
    background-color: #D0D0D0;
}

span.pr_r {
    color: #FF0000;
    background-color: inherit;
}
span.pr_bk {
    color: #000000;
    background-color: inherit;
}

span.pr_g {
    color: #00A000;
    background-color: inherit;
}
```

◇

Macro referenced in 445a.

22.13.4 Trend analysis table

The trend analysis table presents the weight and energy balance of the standard trend intervals up to the most recent log entry and, optionally, a user-specified custom trend interval. Items in this table "borrow" the `span` styles from the monthly log table to colour code positive and negative numbers.

⟨ Trend analysis table 448 ⟩ ≡

```
table.trendan {
    background-color: #D0D0D0;
    color: #000000;
    margin-left: auto;
    margin-right: auto;
}

table.trendan td.w, td.e {
    text-align: center;
}

table.trendan th.custitle {
    background-color: #E0E0FF;
    color: #000000;
    border-left-style: none;
    border-right-style: none;
 }

table.trendan th {
    padding-left: 4px;
    padding-right: 4px;
}
```

◇

Macro referenced in 445a.

22.13.5 Navigation bar table

The following declarations define the style of the navigation bar which appears at the top of all pages displayed during a session. The navigation bar is a full page width table with a single row which contains columns for each of the destinations available from it. The `td` item representing the present page (if any) is given a class of `active`.

⟨ Navigation bar table 449 ⟩ ≡

```
table.navbar {
    background-color: #6060FF;
    color: #FFFFFF;
    width: 100%;
    font-family: Helvetica, Arial, sans-serif;
    font-weight: bold;
    font-size: larger;
}

table.navbar td:first-child {
    border-left: none;
    text-align: center;
}

table.navbar td {
    border-left: 2px solid #FFFFFF;
    text-align: center;
    padding-left: 8px;
    padding-right: 8px;
}

table.navbar td.active {
    color: #FFFF60;
    background-color: #6060FF;
}

table.navbar td.disabled {
    color: #D0D0D0;
    background-color: #6060FF;
}

table.navbar td.pad {
    width: 99%;
}

a.navbar:link, a.navbar:visited {
    background-color: inherit;
    color: #FFFFFF;
    text-decoration: none;
}

a.navbar:hover  {
    background-color: #30D030;
    color: inherit;
}

a.navbar:active {
    color: #FF3030;
    background-color: #30D030;
}
```

◇

Macro referenced in 445a.

22.13.6 Calendar Navigation Tables

Access to all historical logs in the database is via the calendar page, which shows a list of calendars, each representing a year. Within each year's calendar, months present in the database are linked to a monthly log page which will display them.

⟨ Calendar Navigation Tables 450a ⟩ ≡

```
table.list_of_calendars {
    margin-left: auto;
    margin-right: auto;
}

table.calendar {
    margin: 16px;
    color: inherit;
    background-color: #E0E0E0;
}

table.calendar th {
    font-size: larger;
}

table.calendar td {
    padding: 4px;
}
```

◇

Macro referenced in 445a.

22.13.7 Persistent login manager table

The only peculiarity with the persistent login manager table is the token (cookie) field, which we want to display in a monospace font and smaller so the table isn't so wide. This class definition is not bound to the persistent login table and may be used elsewhere should the need arise.

⟨ Persistent login manager table 450b ⟩ ≡

```
td.monospace {
    font-family: monospace;
    font-size: smaller;
}
```

◇

Macro referenced in 445a.

22.13.8 Feedback message table

Access to all historical logs in the database is via the calendar page, which shows a list of calendars, each representing a year. Within each year's calendar, months present in the database are linked to a monthly log page which will display them.

⟨ Feedback message table 451 ⟩ ≡

```
table.feedback {
    width: 80%;
    color: inherit;
    background-color: #E0E0E0;
    margin-left: auto;
    margin-right: auto;
}

table.feedback th.t {
    vertical-align: top;
}

.preview {
    padding: 8px;
    background-color: #FFFFA0;
    width: 80%;
    margin-left: auto;
    margin-right: auto;
}

div.spell_ok {
    background-color: #00C000;
    color: #FFFFFF;
    width: 80%;
    margin-left: auto;
    margin-right: auto;
    padding-left: 8px;
    padding-right: 8px;
}

div.spell_ok h4 {
    padding-top: 2px;
    padding-bottom: 4px;
    font-family: sans-serif;
}

div.spell_dubieties {
    background-color: #FFA0A0;
    color: inherit;
    width: 80%;
    margin-left: auto;
    margin-right: auto;
    padding-left: 8px;
    padding-right: 8px;
}

div.spell_dubieties h4 {
    padding-top: 2px;
    font-family: sans-serif;
    margin-bottom: 0px;
}
```

◇

Macro referenced in 445a.

22.13.9 Global statistics tables

The global statistics are presented in a series of tables which share the following styles.

⟨ Global statistics tables 452 ⟩ ≡

```
table.global_stats {
    background-color: #D0D0D0;
    color: #000000;
    border: 3px ridge;
    margin-left: auto;
    margin-right: auto;
    border-collapse: collapse;
}

table.global_stats td {
    text-align: right;
    border: 3px ridge;
    padding-left: 1em;
    padding-right: 1em;
}

table.global_stats td.c {
    text-align: center;
}

table.global_stats th.l {
    text-align: left;
    border: 3px ridge;
    padding-right: 1em;
}

table.global_stats th.l {
    border-right: none;
}

table.global_stats th.blr {
    border-left: 3px ridge;
    border-right: 3px ridge;
}

table.global_stats th.bl {
    border-left: 3px ridge;
}
```

◇

Macro referenced in 445a.

Chapter 23

XHTML Handheld Style Sheet

The `hdiet_handheld.css` style sheet supplies the rules used when pages are displayed on handheld devices such as personal digital assistants or mobile telephones. Because these devices typically have slow Internet connections, the size of this file should be kept as small as possible.

23.1 Global document properties

"hdiet_handheld.css" 453 ≡

```
table.login {
    background-color: #FFFFFF;
    color: #000000;
    margin-left: 0px;
    margin-right: auto;
    border: none;
}

table.login td, table.login th {
    border: none;
}
```

◇

Chapter 24

JavaScript Utilities

The `hdiet.js` JavaScript program includes common utilities shared by all pages and the dynamic update facilities used by the log pages.

24.1 Global definitions

The following definitions are used in various functions below. Most of these have the same names and values as constants in the Perl application.

"hdiet.js" 454 ≡

```
var WEIGHT_KILOGRAM = 0;
var WEIGHT_STONE = 2;
var WEIGHT_ABBREVIATIONS = [ "kg", "lb", "st" ];
var CALORIES_PER_WEIGHT_UNIT = [ 7716, 3500, 3500 ];
var WEIGHT_CONVERSION = [
/*  Entries for pounds and stones are identical because
    even if stones are selected, entries in log items are
    always kept in pounds.

   To:         kg              lb             st
                                                        From   */
          [ 1.0,           2.2046226,     2.2046226    ],  //  kg
          [ 0.45359237,    1.0,           1.0          ],  //  lb
          [ 0.45359237,    1.0,           1.0          ]   //  st
];
var CALORIES_PER_ENERGY_UNIT = [ 1, 0.239045 ];
var ENERGY_CONVERSION = [
//
//  To:         cal         kJ                From
            [   1.0,        4.18331  ],     //  cal
            [   0.239045,   1.0      ]      //  kJ
];
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.2 Unicode text entity definitions

We define symbolic names for the following Unicode characters which are used in the program. The names are their Unicode names with spaces replaced with underscores.

"hdiet.js" 455a ≡

```
        var U_MINUS_SIGN = "\u2212";
    ◇
```

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.3 Document load-time processing

The `initialiseDocument` function is called from the "`onload`" event handler in our HTML documents. It performs document-level initialisation and configuration functions.

"hdiet.js" 455b ≡

```
        function initialiseDocument() {
            externalLinks();
            determineTimeZoneOffset();
        }
    ◇
```

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.4 Warn if document accessed insecurely

The `checkSecure` function, usually called from the `onload` event handler of the `body` tag of a document, verifies that the document was loaded securely and complains if it wasn't. This warns the user who is using an HTTP connection vulnerable to sniffing when contacting the server. An inexcusable kludge disables this check when testing development builds on the Fourmilab backup server.

"hdiet.js" 455c ≡

```
        function checkSecure() {
            if ((!location.protocol.match(/^https:/i)) &&
                (location.hostname != "server1.fourmilab.ch")) {
                alert("Warning!  This document appears to have been " +
                      "received over an insecure Internet link (http: " +
                      "as opposed to https:).  It is possible the data " +
                      "you submit may be intercepted by an " +
                      "eavesdropper between your computer and The " +
                      "Hacker's Diet Online server.\n\n" +
                      "To be safe, please re-submit your query to the secure server:\n " +
                      "    https://www.fourmilab.ch⟨URL to invoke this program 12a⟩");
            }
        }
    ◇
```

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.5 Record unsaved changes

Every time the user modifies a field in the monthly log, `countChange` should be called to increment `unsavedChanges` which, if nonzero when the user clicks on a link which departs the monthly log form, will give the user an opportunity to cancel following the link and save the changes before leaving.

"hdiet.js" 456a ≡

```
var unsavedChanges = 0;

function countChange() {
    unsavedChanges++;
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.6 Document departure processing

The `leaveDocument` function should be called by any link or button in a monthly log form which leaves the document for another. This function checks whether the user has unsaved changes and allows returning to the form so that they may be saved with the "Update" button.

"hdiet.js" 456b ≡

```
function leaveDocument() {
    if (unsavedChanges > 0) {
        return window.confirm("You have " + unsavedChanges +
            " unsaved change" + (unsavedChanges > 1 ? "s" : "") +
            " to this form.  To discard " +
            "these changes and navigate away from this page " +
            "press OK.  Otherwise, press Cancel and save your " +
            "changes before leaving this page.");
    }
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.7 Format weight to display unit

The `editWeight` function returns a string with its first weight argument formatted appropriate for the display unit specified by the second argument.

"hdiet.js" 457 ≡

```
        var decimalCharacter = ".";             // User decimal separator character

        function editWeight(weight, unit) {
             if (unit == WEIGHT_STONE) {
                var sgn = (weight < 0) ? "-" : "";
                weight = Math.abs(weight);
                var stones = Math.floor(weight / 14);
                var lbs = weight - (stones * 14);
    //alert("Stoner " + weight + "  " + stones + "  " + lbs);
                return (sgn + stones.toFixed(0)) + " " +
                    ((lbs < 10) ? " " : "") + lbs.toFixed(1).replace(/\./, decimalCharacter);
             } else {
                return weight.toFixed(1).replace(/\./, decimalCharacter);
            }
        }
    ◇
```

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.8 Parse weight

The `parseWeight` function returns the contents of a weight field as a number, If the display unit is stones and pounds, the value is returned in pounds. We accept either a comma or period as the decimal character. If the value is invalid, -1 is returned.

"hdiet.js" 458a ≡

```
        function parseWeight(weight, unit) {
            weight = weight.replace(/,/g, ".");
            if (unit == WEIGHT_STONE) {
                var comp = weight.match(/^\s*(\d+)\s+(\d+\.?\d*)\s*$/);
                if (comp != null) {
                    return (Number(comp[1]) * 14) + Number(comp[2]);
                }
    //            alert("Sproink (" + weight + ")");
                if (!weight.match(/^\s*(\d+\.?\d*)\s*$/)) {
                    return -1;
                }
                return Number(weight) * 14;
             } else {
                if (!weight.match(/^\s*(\d+\.?\d*)\s*$/)) {
                    return -1;
                }
                return Number(weight);
            }
        }
    ◇
```

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.8.1 Parse signed weight

The `parseSignedWeight` works precisely like `parseWeight`, but the weight may be preceded by an optional sign.

"hdiet.js" 458b ≡

```
        function parseSignedWeight(weight, unit) {
            var sgn = 1;
            var ms = weight.match(/\s*([\+\-])/);
            if (ms != null) {
                if (ms[1] == '-') {
                    sgn = -1;
                }
                weight = weight.replace(/\s*[\+\-]/, "");
            }
            return parseWeight(weight, unit) * sgn;
        }
    ◇
```

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.9 Trend fitting utilities

The following functions perform a linear regression fit on a set of points. The intermediate values are kept in globals and hence only one fit may be underway at a time. Since that's all we require within the document, there's no need for an object-oriented interface as in the Perl implementation on the server.

24.9.1 Start fit

The `fitStart` function initialises the fit accumulation variables for a new linear regression fit.

"hdiet.js" 459a ≡

```
        var fit_n, fit_s1, fit_s2, fit_s3, fit_s4;

        function fitStart() {
            fit_n = fit_s1 = fit_s2 = fit_s3 = fit_s4 = 0;
        }
    ◇
```

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.9.2 Add Point

The `fitAddPoint` function adds a point value to the trend we're fitting.

"hdiet.js" 459b ≡

```
        function fitAddPoint(value) {
            fit_s1 += (fit_n + 1) * value;
            fit_s2 += (fit_n + 1);
            fit_s3 += value;
            fit_s4 += (fit_n + 1) * (fit_n + 1);
            fit_n++;
        }
    ◇
```

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.9.3 Fit Slope

The `fitSlope` function fits a linear trend to the points supplied so far and returns its slope. You are free to continue adding points after returning the trend.

"hdiet.js" 459c ≡

```
        function fitSlope() {
    //alert(fit_n + " " + fit_s1 + " " + fit_s2 + " " + fit_s3 + " " + fit_s4);
            return ((fit_s1 * fit_n) - (fit_s2 * fit_s3)) /
                    ((fit_s4 * fit_n) - (fit_s2 * fit_s2));
        }
    ◇
```

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.10 Expand abbreviated weight entry

We allow the user to abbreviate weight entries in the log in various ways. If this is a valid abbreviation, find the previous weight log item and expand it accordingly.

"hdiet.js" 460 ≡

```
function expandAbbreviatedWeight(day, unit) {
    var w = document.getElementById("w" + day).value;
    w = w.replace(/^\s+/, "");
    w = w.replace(/\s+$/, "");
    w = w.replace(/,/g, ".");

    //   In stones, all abbreviations have a decimal
    if ((unit == WEIGHT_STONE) && (!w.match(/\d*[\.,]\d*/))) {
        //  Canonicalise weight
        if (w != '') {
            document.getElementById("w" + day).value =
                            editWeight(parseWeight(w, unit), unit);
        }
        return true;
    }

    if ((w == '.') || (w == ',') || (w.match(/^[\.,]\d+$/)) ||
        (w.match(/^\d([\.,]\d*)?$/)) ||
        ((unit == WEIGHT_STONE) && w.match(/^\d\d[\.,]\d*$/))) {
        var p = 0, pd =  0;
        for (var i = day - 1; i >= 1; i--) {
            p = document.getElementById("w" + i).value;
            if (p.match(/^\d/)) {
                pd = p.replace(/,/g, ".");
                break;
            }
        }
        if (pd <= 0) {
            alert("Cannot abbreviate weight.  No previous weight in this month's log.");
            return false;
        }
        if ((w == '.') || (w == ',')) {
            document.getElementById("w" + day).value = p;
        } else {
            var pn = Number(pd);
            if (unit == WEIGHT_STONE) {
                ⟨ Expand abbreviated stones and pounds entry 461 ⟩
            } else {
                if (w.match(/^[\.,]\d+$/)) {
                    document.getElementById("w" + day).value =
                        editWeight(Math.floor(pn) + Number(w), unit);
                } else if (w.match(/^\d([\.,]\d*)?$/)) {
                    document.getElementById("w" + day).value =
                        editWeight(((Math.floor(pn  / 10)) * 10) + Number(w), unit);
                }
//else { alert("Failed to parse (" + w + ")"); }
            }
        }
        return true;
    }
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.10.1 Expand abbreviated stones and pounds entry

When the weight unit is set to stones an abbreviation may be used to change the pounds and decimal place of the previous stone and pound display just as when the units are pounds. In addition, when the display unit is set to stones, if the previous entry has a pounds field between 10 and 13 and the user enters a single digit, decimal character, and optional decimal digit, the action taken depends on the units digit entered. If it's between 0 and 3, it replaces the last digit of the pounds in the last entry, but if the digit is 4 or greater (which is invalid in a stones and pounds display), that digit replaces the two digit pounds field in the previous entry. This reduces the scribbling required when the weight happens to fluctuate around X stones 10. In addition, when the display unit is stones, the user can enter two digits followed by the decimal character and an optional decimal digit to replace the pounds field of the last stones and pounds entry; the decimal character must be entered to distinguish the entry from one denoting an even number of stones.

⟨ Expand abbreviated stones and pounds entry 461 ⟩ ≡

```
        var sf = p.match(/^(\d+)\s+(\d*[\.,]?\d*)$/);
        var stones, pounds;
        if (sf != null) {
            stones = Number(sf[1]);
            pounds = Number(sf[2].replace(/,/g, "."));
    //alert("Previous st=" + stones + " lbs=" + pounds);
        }
    //else { alert("Unable to parse previous stones value (" + p + ")"); }
        var nw = Number(w);
        if (pounds >= 10) {
            if (nw < 4) {
                if (w.match(/^[\.,]\d+$/)) {
                    pounds = Math.floor(pounds) + nw;// alert("gonk 5");
                } else {
                    pounds = ((Math.floor(pounds  / 10)) * 10) + nw;// alert("gonk 6");
                }
            } else {
                pounds = nw;// alert("gonk 2");
            }
        } else {
            if (w.match(/^[\.,]\d+$/)) {
                pounds = Math.floor(pounds) + nw;// alert("gonk 3");
            } else {
                pounds = nw;// alert("gonk 4");
            }
        }
    //alert("New st=" + stones + " lbs=" + pounds);
        document.getElementById("w" + day).value = editWeight((stones * 14) + pounds, unit);
    ◇
```

Macro referenced in 460.

24.11 Create canvas to draw in chart image

We plot new entries in the monthly chart by overlaying a `<div>` on it, which is dynamically sized and positioned to precisely overlay the chart. The following code, which is called whenever we need to plot in the chart, positions the division over the chart and creates a graphics object to draw in it. Subsequent calls simply return the existing object.

"hdiet.js" 462 ≡

```
        var plot;
        var plotChart;

        function getCanvas(imageID) {
            if (!plot) {
                var canvas = document.getElementById("canvas");
                plotChart = document.getElementById(imageID);
                var elementChain = plotChart;
                var offsetLeft = 0, offsetTop = 0;
                while (elementChain) {
                    offsetLeft += elementChain.offsetLeft;
                    offsetTop += elementChain.offsetTop;
                    elementChain = elementChain.offsetParent;
                }
                canvas.style.width = plotChart.width + "px";
                canvas.style.height = plotChart.height + "px";
                canvas.style.left = offsetLeft + "px";
                canvas.style.top = offsetTop + "px";
                canvas.style.visibility = "visible";
                plot = new jsGraphics(canvas);
    //alert("Create canvas");
            }
            return plot;
        }
    ◇
```

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.12 Handle resize of window

This event handler is invoked when the user resizes the window. It is responsible for repositioning the canvas over the current location of the monthly chart after the resize.

"hdiet.js" 463 ≡

```
function resizeEvent(e) {
    var canvas = document.getElementById("canvas");
    var chart = plotChart;
    var elementChain = plotChart;
    var offsetLeft = 0, offsetTop = 0;
    while (elementChain) {
        offsetLeft += elementChain.offsetLeft;
        offsetTop += elementChain.offsetTop;
        elementChain = elementChain.offsetParent;
    }
    canvas.style.left = offsetLeft + "px";
    canvas.style.top = offsetTop + "px";
}

function setResizeEventHandle() {
    //  For competently-implemented and standards-compliant browsers
    if (document.implementation.hasFeature("Events", "2.0")) {
        this.addEventListener("resize", resizeEvent, false);
    //  For Exploder
    } else if (document.attachEvent) {
        this.attachEvent("onresize", resizeEvent);
    }
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.13 Recompute after weight change

When a weight field is modified, we need to recompute the trend and variances for this and subsequent days in the log. The `changeWeight` function is invoked from the `onchange` handler in the weight fields of the monthly log form.

⟨ Maximum Expected Weight Variance 464a ⟩ ≡

```
0.06◇
```

Macro referenced in 465.

"hdiet.js" 464b ≡

```
function changeWeight(day) {
    var n = Number(document.getElementById("md").getAttribute("value"));    // Number of days
    var t = Number(document.getElementById("t0").getAttribute("value"));    // Trend carry-forward
    var unit = Number(document.getElementById("du").getAttribute("value")); // Display unit
    var height = Number(document.getElementById("hgt").getAttribute("value")); // Height in centimetres
    decimalCharacter = document.getElementById("dc").getAttribute("value");

    if (!expandAbbreviatedWeight(day, unit)) {
        document.getElementById("w" + day).value = "";
        return;
    }

    var ckw = 0;
    if (document.getElementById("w" + day).value.match(/^\s*$/)) {
        document.getElementById("w" + day).value = "";
    } else {
        var ckw = parseWeight(document.getElementById("w" + day).value, unit);
        if (ckw < 0) {
            alert("Weight entry invalid.");
            resetFocus("w", day);
            return;
        }
    }

    countChange();

    ⟨ Find most recent trend value before this day 469 ⟩

    ⟨ Check for implausibly large change in weight 465 ⟩

    ⟨ Undraw trend values starting at this day 466 ⟩

    plotWeightOnChart(day, unit);

    ⟨ Update the trend and variance for this and subsequent days 470 ⟩

    ⟨ Fit a linear trend and update weight and energy balance 472 ⟩

    ⟨ Update the mean and most recent body mass index 473 ⟩
}
◇
```

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.13.1 Check for implausibly large change in weight

The following code applies heuristics which attempt to detect weight entries which are inconsistent with earlier log entries and possibly erroneous. The eccentricities of users' log-keeping habits complicate this substantially. A user who enters their weight almost every day is easy to cope with; we can simply compare the trend with the weight entry and flag it if the change exceeds the maximum seen in a large sample of data. The user who enters weight only infrequently poses more of a problem. We take the following approach. First, we test the weight against the trend as if the user were assiduous in logging weight. If that test indicates an anomaly, we then see if there is a previous weight entry in the log and, if so, apply the fraction change test against that weight as opposed to the trend (as the user has presumably already confirmed that change if it triggers the dubiety test). If no previous weight has been entered in the month, we compute the number of days so far with no weight entries and simulate a trend driven by a linear progression from the starting trend to the entered weight and use that in the plausibility test. All of this is, of course, *ad hack*, but then all we're trying to do is catch fat-finger errors, and any alert we flash can be dismissed by the user simply by pressing the OK button.

⟨ Check for implausibly large change in weight 465 ⟩ ≡

```
        if ((t > 0) && (ckw > 0) && ((Math.abs(t - ckw) / t) > ⟨ Maximum Expected Weight Variance 464a ⟩)) {
            var deltad = -1, lastw = 0;
            for (var ld = day - 1; ld >= 1; ld--) {
                if (document.getElementById("w" + ld).value != "") {
                    deltad = day - ld;
                    lastw = parseWeight(document.getElementById("w" + ld).value, unit);
                    break;
                }
            }
            if (deltad == -1) {
                deltad = day;
            }
    //alert("deltad " + deltad + " lastw " + lastw);
            if (lastw > 0) {
                if (Math.abs(lastw - ckw) > Math.abs(t - ckw)) {
                    lastw = t;
                }
            } else {
                var simt = t;
                for (var i = 1; i < deltad; i++) {
                    simt = simt + (((t + (((ckw - t) * i) / deltad)) - simt) / 10);
                }
    //alert("simt " + simt);
                lastw = simt;
            }
    //alert("Adjusted lastw " + lastw);
            if ((Math.abs(ckw - lastw) / lastw) > ⟨ Maximum Expected Weight Variance 464a ⟩) {
                if (!confirm("This weight is a " +
                    (((ckw - lastw) * 100) / lastw).toFixed(1).replace(/\./, decimalCharacter) +
                    "% change\u2014possibly incorrect.\n" +
                    "Press OK to accept weight as entered, Cancel to correct.")) {
                    resetFocus("w", day);
                    return;
                }
            }
        }
```

◇

Macro referenced in 464b.

24.13.2 Undraw trend values starting at this day

⟨ Undraw trend values starting at this day 466 ⟩ ≡

```
/* ******

    var scaling = document.getElementById("sc").getAttribute("value").
            match(/^([^,]+),([^,]+),([^,]+),([^,]+),([^,]+),([^,]+),([^,]+)$/);
    for (var i = 1; i <= 7; i++) {
        scaling[i] = Number(scaling[i]);
    }

    for (var d = day; d < (nd - 1); d++) {
        var tfrom = document.getElementById("T" + (d + 1)).getAttribute("value"),
            tto = document.getElementById("T" + (d + 2)).getAttribute("value");
        if (tfrom.match(/^\d/) && tto.match(/^\d/)) {
            tfrom = Number(tfrom);
            tto = Number(tto);
            var plot = getCanvas("chart");
            var px1 = scaling[1] + (scaling[2] * (d));
            var py1 = scaling[3] - Math.floor(((tfrom - scaling[4]) * scaling[5]) / scaling[6]);
            var px2 = scaling[1] + (scaling[2] * (d + 1));
            var py2 = scaling[3] - Math.floor(((tto - scaling[4]) * scaling[5]) / scaling[6]);
            plot.setColor("#FFFF00");
            plot.drawLine(px1, py1, px2, py2);
            plot.paint();
        }
    }

****** */
◇
```

Macro referenced in 464b.

24.13.3 Plot weight on chart image

Using the chart scale information supplied in the hidden "sc" field, compute the pixel co-ordinates of the weight just entered in the canvas which overlays the monthly chart image and plot the weight. Since we're just providing real-time feedback, and the most common case is a user entering new weights in blank fields, we don't worry about trying to erase a previous weight entry. We could handle that by embedding hidden fields with the old value, but I'm not going to go to all that trouble unless this proves to be a genuine annoyance.

Weights which are off-scale high or low are not plotted on the chart, and blank weights or those which fail to parse are likewise ignored.

"hdiet.js" 467 ≡

```
function plotWeightOnChart(day, unit) {
    ⟨ Extract scaling information for chart 468 ⟩
    var dweight = parseWeight(document.getElementById("w" + day).value, unit);
    if ((dweight >= scaling[4]) && (dweight <= (scaling[4] + scaling[6]))) {
        var plot = getCanvas("chart");
        var px = scaling[1] + (scaling[2] * (day - 1));
        var py = scaling[3] - Math.floor(((dweight - scaling[4]) * scaling[5]) / scaling[6]);

        var sinkerSize = 4;

        //  Fill float/sinker with white or yellow, if it's flagged.

        plot.setColor(document.getElementById("f" + day).checked ? "#FFFF00" : "#FFFFFF");
        for (var j = -sinkerSize; j <= sinkerSize; j++) {
            var dx = Math.abs(j) - sinkerSize;

            plot.drawLine(px - dx, py + j, px + dx, py + j);
        }

        //  Trace the outline of the float/sinker in blue

        plot.setColor("#0000FF");
        plot.drawLine(px - sinkerSize, py, px, py - sinkerSize);
        plot.drawLine(px, py - sinkerSize, px + sinkerSize, py);
        plot.drawLine(px + sinkerSize, py, px, py + sinkerSize);
        plot.drawLine(px, py + sinkerSize, px - sinkerSize, py);

        plot.paint();
    }
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.13.4 Extract scaling information for chart

The scaling of day numbers, weight, and exercise rungs to the chart is given by the values passed in the hidden "sc" field. Extract these values and convert them to numbers in an array for use in subsequent plotting operations.

⟨ Extract scaling information for chart 468 ⟩ ≡

```
var scaling = document.getElementById("sc").getAttribute("value").
        match(/^([^,]+),([^,]+),([^,]+),([^,]+),([^,]+),([^,]+),([^,]+)$/);
for (var i = 1; i <= 7; i++) {
    scaling[i] = Number(scaling[i]);
}
```

◇

Macro referenced in 467, 475.

24.13.5 Find most recent trend value before this day

In order to update the trend of the day in which the new weight has been entered and that of subsequent days up to the last weight specified in the log, we need to find the most recent known trend value. This will be the value given with the most recent previous weight, or the trend carry-forward if this is the first day of the month. If this is the first day and there is no trend carry-forward, the trend is started at the entered weight.

⟨ Find most recent trend value before this day 469 ⟩ ≡

```
/* Find the last non-blank weight entry in the log. */
var nd = n;

while ((nd > 0) && (!document.getElementById("w" + nd).value.match(/^\d/))) {
    nd--;
}
nd = Math.max(nd, day);

/* If this is not the first day of the month, get the trend from
   the previous day's entry. */

if (day > 1) {
    var lt = document.getElementById("t" + (day - 1)).firstChild.data;
    if (lt.match(/^\d/)) {
        t = parseWeight(lt, unit);
    } else {
        var jt = "", j, k;
        for (j = day - 2; j >= 1; j--) {
            jt = document.getElementById("t" + j).firstChild.data;
            if (jt.match(/^\d/)) {
                break;
            }
        }
        if (j == 0) {
            jt = document.getElementById("t0").getAttribute("value");
        }
        if (jt != "" && jt != 0) {
            t = parseWeight(jt, unit);
            for (k = j + 1; k < day; k++) {
                replaceText("t" + k, editWeight(t, unit));
                document.getElementById("T" + k).setAttribute("value", t.toFixed(4));
            }
        }
    }
}

/* If this is the first day of the month, use the trend
   carry-forward as the previous trend value.  If no trend
   carry-forward is specified, simply use the current weight
   to start the trend. */

if (t == 0) {
    t = parseWeight(document.getElementById("w" + day).value, unit);
}
```

◇

Macro referenced in 464b.

24.13.6 Update the trend and variance for this and subsequent days

Starting with the weight which was changed and proceeding to the end of the monthly log, update the trend and variance columns to reflect the newly entered weight. If no previous trend was defined, we blank out trend and variance fields until we encounter the first specified weight, then use it as the starting point for the trend.

⟨ Update the trend and variance for this and subsequent days 470 ⟩ ≡

```
//alert("Change weight " + day + " " + t + "  (" + document.getElementById("w" + day).value + ") t = " + t);
    for (var i = day; i <= n; i++) {
        var w = document.getElementById("w" + i).value;
        if (w.match(/^\d/)) {
            if (t < 0) {
                t = parseWeight(w, unit);
            } else {
                t = t + ((parseWeight(w, unit) - t) / 10);
            }
            replaceText("t" + i, editWeight(t, unit));
            updateVariance("v" + i, parseWeight(w, unit) - t);
            document.getElementById("T" + i).setAttribute("value", t.toFixed(4));
        } else {
            replaceText("v" + i, "");
            if ((i <= nd) && (t > 0)) {
                replaceText("t" + i, editWeight(t, unit));
                document.getElementById("T" + i).setAttribute("value", t.toFixed(4));
            } else {
                replaceText("t" + i, "");
                document.getElementById("T" + i).setAttribute("value", "");
            }
        }
    }
    ⟨ Plot the updated trend 471 ⟩
```

◇

Macro referenced in 464b.

24.13.7 Plot the updated trend

⟨ Plot the updated trend 471 ⟩ ≡

```
/* ******

    for (var d = day; d < (n - 1); d++) {
        var tfrom = document.getElementById("T" + (d + 1)).getAttribute("value"),
            tto = document.getElementById("T" + (d + 2)).getAttribute("value");
        if (tfrom.match(/^\d/) && tto.match(/^\d/)) {
            tfrom = Number(tfrom);
            tto = Number(tto);
            var plot = getCanvas("chart");
            var px1 = scaling[1] + (scaling[2] * (d));
            var py1 = scaling[3] - Math.floor(((tfrom - scaling[4]) * scaling[5]) / scaling[6]);
            var px2 = scaling[1] + (scaling[2] * (d + 1));
            var py2 = scaling[3] - Math.floor(((tto - scaling[4]) * scaling[5]) / scaling[6]);
            plot.setColor("#FF0000");
            plot.drawLine(px1, py1, px2, py2);
            plot.paint();
        }
    }

****** */
```

◇

Macro referenced in 470.

24.13.8 Fit a linear trend and update weight and energy balance

If the log contains two or more entries, fit a linear trend to it and update the weekly gain/loss and calorie balance.

⟨ Fit a linear trend and update weight and energy balance 472 ⟩ ≡

```
        if (nd > 1) {
            var np = 0;
            fitStart();
            for (var i = 1; i <= nd; i++) {
                var w = document.getElementById("t" + i).firstChild.data;
                if (w.match(/^\d/)) {
                    var nw = parseWeight(w, unit);
                    if (nw > 0) {
                        fitAddPoint(nw);
                        np++;
                    }
                }
            }

            var tslope = fitSlope();
            if (np < 2) {
                tslope = 0;
            }
            replaceText("delta_sign", tslope > 0 ? "gain" : "loss");
            replaceText("weekly_delta", Math.abs(tslope * 7).toFixed(2).replace(/\./, decimalCharacter));

            replaceText("calorie_sign", tslope > 0 ? "excess" : "deficit");
            replaceText("daily_calories", Math.round(Math.abs(tslope) * CALORIES_PER_WEIGHT_UNIT[unit]));
        }
    ◇
```

Macro referenced in 464b.

24.13.9 Update the mean and most recent body mass index

The pseudoscientific "body mass index" is computed from the mean and final trend value in the log and the user's height in centimetres. Using the trend rather than the weight minimises psychologically jarring jitter.

⟨ Update the mean and most recent body mass index 473 ⟩ ≡

```
var tweight = 0, lweight = 0, nw = 0;

for (var i = 1; i <= n; i++) {
    var w = document.getElementById("w" + i).value;
    if (w.match(/^\d/)) {
        lweight = parseWeight(document.getElementById("t" + i).firstChild.data, unit);
        tweight += lweight;
        nw++;
    }
}

if ((nw > 0) && (height > 0)) {
    tweight /= nw;
    tweight *= WEIGHT_CONVERSION[unit][WEIGHT_KILOGRAM];
    lweight *= WEIGHT_CONVERSION[unit][WEIGHT_KILOGRAM];
    height /= 100;
    height *= height;
    replaceText("mean_bmi", (tweight / height).toFixed(1).replace(/\./, decimalCharacter));
    replaceText("last_bmi", (lweight / height).toFixed(1).replace(/\./, decimalCharacter));
    document.getElementById("bmi").style.display = "inline";
} else {
    document.getElementById("bmi").style.display = "none";
}
```

◇

Macro referenced in 464b.

24.14 Change exercise rung field

Check whether the user entered just a period in the exercise rung field. If so, copy the most recently specified exercise rung, if any. The period may be preceded and/or followed by white space. The rung value is syntax and range checked, and any fractional part is discarded. The cleaned-up value is placed in the rung field.

"hdiet.js" 474 ≡

```
function changeRung(day) {
    if (document.getElementById("r" + day).value.match(/^\s*[\.,\+\-]\s*$/)) {
        var r = 0;
        for (var i = day - 1; i >= 1; i--) {
            r = document.getElementById("r" + i).value;
            if (r.match(/^\d/)) {
                break;
            }
            r = Number(r);
        }
        if (r <= 0) {
            alert("Cannot copy rung.  No previous rung in this month's log.");
            document.getElementById("r" + day).value = "";
            return;
        }
        if (document.getElementById("r" + day).value.match(/^\s*[\+]\s*$/)) {
            r++;
        } else if (document.getElementById("r" + day).value.match(/^\s*[\-]\s*$/)) {
            r--;
        }
        document.getElementById("r" + day).value = r;
    }
    if (document.getElementById("r" + day).value.match(/^\s*$/)) {
        document.getElementById("r" + day).value = "";
    } else {
        var r = Math.floor(Number(document.getElementById("r" + day).value));
        if (isNaN(r) || (r < 1) || (r > 48)) {
            alert("Rung value invalid.  Must be integer between 1 and 48.");
            resetFocus("r", day);
        } else {
            document.getElementById("r" + day).value = r;
            ⟨ Plot exercise rung on chart image 475 ⟩
        }
    }
    countChange();
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.14.1 Plot exercise rung on chart image

Using the chart scale information supplied in the hidden "sc" field, compute the pixel co-ordinates of the weight just entered in the canvas which overlays the monthly chart image and plot the rung. We simply plot the rung as a horizontal line from the current to the next day. For the most common case of no daily change in rung this is what is expected. Entry of several different rungs in one session will produce disconnected lines (which are still strictly correct) that will be connected when the changes are saved and the chart is regenerated.

This code is based on a bob-tailed version of the algorithm for plotting sparse data detailed in section 5.12.1. Here, we know that today's point is defined—we wouldn't be here plotting it otherwise—so the only question is whether yesterday's and tomorrow's points are defined.

⟨ Plot exercise rung on chart image 475 ⟩ ≡

```
⟨ Extract scaling information for chart 468 ⟩

if ((r >= 1) && (r <= scaling[7])) {
    var n = Number(document.getElementById("md").getAttribute("value")); // Days in month

    var plot = getCanvas("chart");
    plot.setColor("#0000FF");

    var cx = scaling[1] + (scaling[2] * (day - 1)),
        cy = scaling[3] - Math.floor(((r - 1) * scaling[5]) / scaling[7]);

    if (day == n) {
        var lx = scaling[1] + (scaling[2] * (day - 2));
        if (document.getElementById("r" + (day - 1)).value != "") {
            //  Yesterday defined--plot from yesterday to today
            var ly = scaling[3] - Math.floor(((Number(document.getElementById("r" + (day - 1)).value) - 1)
            plot.drawLine(lx, ly, cx, cy);
        } else {
            //  Yesterday not defined--plot a flat line from yesterday to today
            plot.drawLine(lx, cy, cx, cy);
        }
    } else {
        if ((day > 1) && (document.getElementById("r" + (day - 1)).value != "")) {
            //  Yesterday defined--plot from yesterday to today
            var lx = scaling[1] + (scaling[2] * (day - 2)),
                ly = scaling[3] - Math.floor(((Number(document.getElementById("r" + (day - 1)).value) - 1)
            plot.drawLine(lx, ly, cx, cy);
        } else {
            if (document.getElementById("r" + (day + 1)).value != "") {
                //  Tomorrow defined--plot from today to tomorrow
                var nx = scaling[1] + (scaling[2] * day),
                    ny = scaling[3] - Math.floor(((Number(document.getElementById("r" + (day + 1)).value)
                plot.drawLine(cx, cy, nx, ny);
            } else {
                //  Tomorrow not defined--plot a flat line from today to tomorrow
                var nx = scaling[1] + (scaling[2] * day);
                plot.drawLine(cx, cy, nx, cy);
            }
        }
    }

    plot.paint();
}
```

◇

Macro referenced in 474.

24.15 Change comment field

If the user enters just a period in a comment field, the most most recently entered comment will be copied into the field. The period must be the only character in the comment field. To enter a comment which is just a single period, enter a space after the period.

"hdiet.js" 476a ≡

```
function changeComment(day) {
    if ((document.getElementById("c" + day).value == ".") ||
        (document.getElementById("c" + day).value == ",")) {
        var r = "";
        for (var i = day - 1; i >= 1; i--) {
            r = document.getElementById("c" + i).value;
            if (!r.match(/^\s*$/)) {
                break;
            }
        }
        if (r == "") {
            alert("Cannot copy comment.  No previous comment in this month's log.");
            document.getElementById("c" + day).value = "";
            return;
        }
        document.getElementById("c" + day).value = r;
    }
    countChange();
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.16 Diet calculator support

Functions in this section support the interactive diet calculator. We begin by defining the variables which represent the active fields in the diet calculator. They are loaded from the form fields by `loadDietCalcFields()` below whenever the user changes a field and we need to reflect the change in the other fields.

"hdiet.js" 476b ≡

```
var calc_calorie_balance, calc_energy_unit,
    calc_start_weight, calc_weight_unit,
    calc_goal_weight, calc_weight_change,
    calc_weight_week, calc_weeks, calc_months,
    calc_start_date, calc_end_date;
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.16.1 Load diet calculator values

Load the current values from the diet calculator fields into our working variables. The fact that we convert these values to and from the symbolic representation shown to the user in the form fields every time guarantees they are always in canonical form.

"hdiet.js" 477 ≡

```
function loadDietCalcFields() {
    decimalCharacter = document.getElementById("dc").getAttribute("value");
    calc_energy_unit = document.getElementById("calc_energy_unit").selectedIndex;
    calc_calorie_balance = Number(document.getElementById("calc_calorie_balance").value.replace(/,/g, ".")
        CALORIES_PER_ENERGY_UNIT[calc_energy_unit];
    calc_weight_unit = document.getElementById("calc_weight_unit").selectedIndex;
    calc_start_weight =
        parseWeight(document.getElementById("calc_start_weight").value, calc_weight_unit);
    calc_goal_weight =
        parseWeight(document.getElementById("calc_goal_weight").value, calc_weight_unit);
    calc_weight_change = parseSignedWeight(document.getElementById("calc_weight_change").value, calc_weigh
    calc_weight_week =
        parseSignedWeight(document.getElementById("calc_weight_week").value, calc_weight_unit);
    calc_weeks = Number(document.getElementById("calc_weeks").value);
    calc_months = Number(document.getElementById("calc_months").value);
    calc_start_date = get_selected_date("from");
    calc_end_date = get_selected_date("to");
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.16.2 Recalculate diet calculator values

Recalculate derived quantities from primary quantities and update the fields in the diet calculator.

"hdiet.js" 478 ≡

```
        function dietCalcRecalculate() {
            calc_weight_change = calc_goal_weight - calc_start_weight;
            calc_weight_week = (calc_calorie_balance * 7) / CALORIES_PER_WEIGHT_UNIT[calc_weight_unit];
            calc_weeks = Math.round(calc_weight_change / calc_weight_week);
            calc_months = Math.round(((calc_weight_change / calc_weight_week) * 7.0) / 30.44);
            calc_end_date = calc_start_date + (calc_weeks * 7 * 24 * 60 * 60 * 1000);

            //  Update the form fields with the new values

            document.getElementById("calc_calorie_balance").value = Math.round(calc_calorie_balance /
                CALORIES_PER_ENERGY_UNIT[document.getElementById("calc_energy_unit").selectedIndex]);
            document.getElementById("calc_start_weight").value =
                editWeight(calc_start_weight, calc_weight_unit);
            document.getElementById("calc_goal_weight").value =
                editWeight(calc_goal_weight, calc_weight_unit);
            document.getElementById("calc_weight_change").value =
                editWeight(calc_weight_change, calc_weight_unit);
            document.getElementById("calc_weight_week").value =
                editWeight(calc_weight_week, calc_weight_unit);
            document.getElementById("calc_weeks").value = calc_weeks;
            document.getElementById("calc_months").value = calc_months;
            set_date_selection("from", calc_start_date);
            set_date_selection("to", calc_end_date);

            if (calc_end_date <= calc_start_date) {
                document.getElementById("end_date").style.display = "none";
                document.getElementById("endless_date").style.display = "inline";
            } else {
                document.getElementById("end_date").style.display = "inline";
                document.getElementById("endless_date").style.display = "none";
            }

            countChange();
        }
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.16.2.1 Set date selection

Set the components of a calendar date in the group of selection fields with ID `which` to represent the JavaScript millisecond UTC date `ms`.

"hdiet.js" 479a ≡

```
        function set_date_selection(which, ms) {
            var date = new Date(ms);
            var year = date.getUTCFullYear(),
                month = date.getUTCMonth(),
                day = date.getUTCDate();

            var i;
            for (i = 0; i < document.getElementById(which + "_y").length; i++) {
                if (year == Number(document.getElementById(which + "_y").options[i].value)) {
                    document.getElementById(which + "_y").selectedIndex = i;
                    i = -1;
                    break;
                }
            }
            if (i != -1) {
    //alert("Added year " + year + " to " + which + " selection");
                document.getElementById(which + "_y").options[document.getElementById(which + "_y").length] =
                    new Option(year, year);
                document.getElementById(which + "_y").selectedIndex = document.getElementById(which + "_y").length
            }
            document.getElementById(which + "_m").selectedIndex = month;
            document.getElementById(which + "_d").selectedIndex = day - 1;
        }
    ◇
```

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.16.3 Change energy balance

Handle a change in the energy balance.

"hdiet.js" 479b ≡

```
        function change_calc_calorie_balance() {
            loadDietCalcFields();
    //alert("cccb " + calc_calorie_balance);
            dietCalcRecalculate();
        }
    ◇
```

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.16.4 Change energy unit

Handle a change in the energy unit.

"hdiet.js" 480a ≡

```
function change_calc_energy_unit() {
    var old_calc_energy_unit = calc_energy_unit;
    loadDietCalcFields();
    calc_calorie_balance *= ENERGY_CONVERSION[old_calc_energy_unit][calc_energy_unit];
    dietCalcRecalculate();
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.16.5 Change starting weight

Handle a change in the starting weight.

"hdiet.js" 480b ≡

```
function change_calc_start_weight() {
    loadDietCalcFields();
    if (calc_start_weight > 0) {
        dietCalcRecalculate();
    } else {
        alert("Invalid initial weight.");
        resetFocus("calc_start_weight");
    }
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.16.6 Change weight unit

Handle a change in the weight unit. There is some fancy footwork below due to our parsing stones and pounds differently from sane units like kilograms or pounds. If the user has changed the units from another unit to stones, the primary weight fields will have been parsed incorrectly as a number of stones. We temporarily reinstate the former unit, re-parse the primary weight fields, convert them to stones (actually pounds), and then we're finally ready to edit them in stones and pounds format into the form fields. The lengths we'll do to get stoned!

"hdiet.js" 481a ≡

```
function change_calc_weight_unit() {
    var old_calc_weight_unit = calc_weight_unit;
    loadDietCalcFields();
    var new_calc_weight_unit = calc_weight_unit;
    calc_weight_unit = old_calc_weight_unit;
    calc_start_weight =
        parseWeight(document.getElementById("calc_start_weight").value, calc_weight_unit);
    calc_goal_weight =
        parseWeight(document.getElementById("calc_goal_weight").value, calc_weight_unit);
    calc_weight_unit = new_calc_weight_unit;
    calc_start_weight *= WEIGHT_CONVERSION[old_calc_weight_unit][calc_weight_unit];
    calc_goal_weight *= WEIGHT_CONVERSION[old_calc_weight_unit][calc_weight_unit];
    dietCalcRecalculate();
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.16.7 Change goal weight

Handle a change in the goal weight.

"hdiet.js" 481b ≡

```
function change_calc_goal_weight() {
    loadDietCalcFields();
    if (calc_goal_weight > 0) {
        dietCalcRecalculate();
    } else {
        alert("Invalid goal weight.");
        resetFocus("calc_goal_weight");
    }
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.16.8 Change desired weight gain/loss

Handle a change in the desired weight gain or loss. Changing this derivative field modifies the goal weight to result in the specified delta weight.

"hdiet.js" 482a ≡

```
        function change_calc_weight_change() {
            loadDietCalcFields();
            calc_goal_weight = calc_start_weight + calc_weight_change;
            dietCalcRecalculate();

        }
    ◇
```

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.16.9 Change weekly weight gain/loss

Handle a change in the weekly weight gain/loss. Changing this derivative field adjusts the energy balance to achieve the desired weekly change.

"hdiet.js" 482b ≡

```
        function change_calc_weight_week() {
            loadDietCalcFields();
            calc_calorie_balance = calc_weight_week * (CALORIES_PER_WEIGHT_UNIT[calc_weight_unit] / 7);
    //alert(calc_calorie_balance);
            dietCalcRecalculate();
        }
    ◇
```

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.16.10 Change weeks duration

Changing the weeks to go field adjusts the energy balance to achieve the specified weight delta in the given number of weeks.

"hdiet.js" 483a ≡

```
        function change_calc_weeks() {
            loadDietCalcFields();
            if (calc_weeks > 0) {
                calc_calorie_balance = Math.round(((calc_weight_change / calc_weeks) *
                    (CALORIES_PER_WEIGHT_UNIT[calc_weight_unit] / 7)));
                dietCalcRecalculate();
            } else {
                alert("Weeks duration must be greater than zero.");
                resetFocus("calc_weeks");
            }
        }
    ◇
```

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.16.11 Change months duration

Changing the months to go field triggers recomputation of the calorie balance needed to accomplish the current weight change in the specified number of months. We take the number of days in a month as the mean for the Gregorian calendar.

"hdiet.js" 483b ≡

```
        function change_calc_months() {
            loadDietCalcFields();
            if (calc_months > 0) {
                calc_calorie_balance = Math.round(((calc_weight_change / calc_months) *
                    (CALORIES_PER_WEIGHT_UNIT[calc_weight_unit] / 30.44)));
                dietCalcRecalculate();
            } else {
                alert("Months duration must be greater than zero.");
                resetFocus("calc_months");
            }
        }
    ◇
```

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.16.12 Change start date

The starting date is a primary field. When changed, the ending date is adjusted to reflect the current duration.

"hdiet.js" 484a ≡

```
        function change_from_date() {
            calc_start_date = get_selected_date("from");
            dietCalcRecalculate();
        }

        function change_from_y() {
            change_from_date();
        }

        function change_from_m() {
            change_from_date();
        }

        function change_from_d() {
            change_from_date();
        }
    ◇
```

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.16.12.1 Get selected date

Extract the components of a calendar date from the group of selection fields with ID `which` and return a JavaScript millisecond UTC time quantity for that date. No range checking is done; the selection field is presumed to have worried about that, and there's no serious damage a user can do by dummying up ridiculous values, since this is only used in the diet calculator.

"hdiet.js" 484b ≡

```
        function get_selected_date(which) {
            var year = document.getElementById(which + "_y").options[document.getElementById(which + "_y").selecte
                month = document.getElementById(which + "_m").selectedIndex,
                day = document.getElementById(which + "_d").selectedIndex + 1;
            return Date.UTC(year, month, day);

        }
    ◇
```

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.16.13 Change end date

When the ending date is changed, we adjust the energy balance to achieve the desired weight change in the interval between the start and end dates.

"hdiet.js" 485a ≡

```
        function change_to_date() {
            calc_end_date = get_selected_date("to");
            if (calc_end_date > calc_start_date) {
                calc_calorie_balance = Math.round((calc_weight_change /
                    ((calc_end_date - calc_start_date) / (24 * 60 * 60 * 1000))) *
                    CALORIES_PER_WEIGHT_UNIT[calc_weight_unit]);
            } else {
                alert("End date must be after start date.");
                resetFocus("to_y");
            }
            dietCalcRecalculate();
        }
        function change_to_y() {
            change_to_date();
        }

        function change_to_m() {
            change_to_date();
        }

        function change_to_d() {
            change_to_date();
        }
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.16.14 Change plot diet plan in chart

Note when the user changes the state of the "Plot plan in chart" checkbox. The only reason we care about this is to note that something has changed which will trigger a warning if the user is about to quit the page without saving the changes.

"hdiet.js" 485b ≡

```
        function change_calc_plot_plan() {
            countChange();
        }
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.17 Validate feedback form

"hdiet.js" 486a ≡

```
function validateFeedback() {
    if (document.getElementById("category").selectedIndex <= 0) {
        alert("Please choose a category for your feedback message.");
        return false;
    }
    return true;
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.18 Reset keyboard focus

This function resets the keyboard focus to the field with ID consisting of the `fieldname` concatenated with the optional `day` number. We make this a function because a crappy work-around is required for Firefox, and in case it breaks some other browser or eventually may be retired, it only needs to be fixed here.

"hdiet.js" 486b ≡

```
function resetFocus(fieldname, day) {
    if (arguments.length < 2) {
        day = "";
    }
    setTimeout("document.getElementById(\"" + fieldname + day + "\").focus()", 1);
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.19 External link window management

In Strict XHTML 1.0, the "`target=`" attribute of the `<a>` tag is forbidden, having been deemed a matter of "presentation"... *hrrrmph*! So, we run this little JavaScript hack after loading every page, which riffles through the links, checking for "`rel=`" attributes, which *are* permitted, looking for those with a prefix of "`Target:`". For each of these, we extract the balance of the relation specification and set it as the DOM target property of the link. The things we do to comply with that standard!

"hdiet.js" 487 ≡

```
/*
    externalLinks  --  Emulate "target=" in XHTML 1.0 Strict <a> tags

    http://www.sitepoint.com/article/standards-compliant-world

    Modified by John Walker to only extract and modify links with
    rel="Target:<frame>" and extract the frame name from that
    specification. */

function externalLinks() {
    if (!document.getElementsByTagName) {
        return;
    }
    var anchors = document.getElementsByTagName("a");
    for (var i = 0; i < anchors.length; i++) {
        var anchor = anchors[i], target;
        if (anchor.getAttribute("href") &&
            anchor.getAttribute("rel") &&
            anchor.getAttribute("rel").match(/^Target:/)) {
            target = anchor.getAttribute("rel").match(/(^Target:)(\w+$)/);
            anchor.target = target[2];
        }
    }
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.20 Determine local time zone offset

On the server we express all times in UTC, but the user may be accessing from any time zone, and the local time zone may vary as the user travels. If JavaScript is enabled, we determine the current offset between UTC and the local time zone (taking into account summer time), and fill this into a hidden form element named "`tzoffset`" if such exists. This allows request handlers to behave intelligently based on the user's time zone (for example, forbidding the entry of precognitive log entries from the future). If JavaScript is disabled, the hidden `tzoffset` field will be sent with its default value of "`unknown`".

"hdiet.js" 488a ≡

```
function determineTimeZoneOffset() {
    if (document.getElementById && document.getElementById("tzoffset")) {
        document.getElementById("tzoffset").value = (new Date()).getTimezoneOffset();
    }
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.21 Express number in canonical form

Convert a the number given by the first argument to canonical form by expressing it to the number of decimal places given by the second argument and then removing trailing zeroes after the decimal point and, if the number is an integer, the decimal point as well. The optional third argument specifies the user's preferred decimal separator character; if unspecified, a period is used.

"hdiet.js" 488b ≡

```
function canonicalNumber(value, places, decimal) {
    var v = value.toFixed(places);

    if (arguments.length < 3) {
        decimal = '.';
    }
    v = v.replace(/0+$/, "");
    v = v.replace(/\.$/, "");
    v = v.replace(/\./, decimal);
    return v;
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.22 Height specification unit conversion

We invite the user to specify their height in order that we might compute the Body Mass Index. We allow the height to be entered either in centimetres or as feet an inches (or inches alone). The following JavaScript code is triggered when the user changes the value in one of these fields, and propagates the change to the other fields accordingly. If JavaScript is not enabled on the client side, no harm will be done: the server will independently validate the specifications and reject nonsense.

24.22.1 Centimetres

The user has modified the centimetres field. Sanity check the entry and update the feet and inches fields to correspond.

"hdiet.js" 489 ≡

```
        function height_changed_cm() {
            var thisform = document.getElementById("Hdiet_newacct");
            var cm = thisform.HDiet_height_cm.value;
            cm = cm.replace(/,/, ".");
            if (cm > 244) {
                if (!confirm("That's awfully tall (" + cm + " centimetres).  Are you sure?")) {
                    thisform.HDiet_height_cm.focus();
                    thisform.HDiet_height_cm.select();
                    return false;
                }
            }
             if (cm < 122) {
                if (!confirm("That's awfully short (" + cm + " centimetres).  Are you sure?")) {
                    thisform.HDiet_height_cm.focus();
                    thisform.HDiet_height_cm.select();
                    return false;
                }
           }
            var inches = cm / 2.54;
            thisform.HDiet_height_ft.value = Math.floor(inches / 12);
            thisform.HDiet_height_in.value =
                canonicalNumber(inches % 12, 1, thisform.decimal_character.value);
        }
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.22.2 Feet

The user has modified the feet field. Check the value for reasonableness and update the centimetres field.

"hdiet.js" 490 ≡

```
function height_changed_ft() {
    var thisform = document.getElementById("Hdiet_newacct");
    var ft = thisform.HDiet_height_ft.value;
    if (ft > 7) {
        if (!confirm("That's awfully tall (" + ft + " feet).  Are you sure?")) {
            thisform.HDiet_height_ft.focus();
            thisform.HDiet_height_ft.select();
            return false;
        }
    }
     if (ft < 4) {
        if (!confirm("That's awfully short (" + ft + " feet).  Are you sure?")) {
            thisform.HDiet_height_ft.focus();
            thisform.HDiet_height_ft.select();
            return false;
        }
    }
    var cm = ft * 2.54 * 12;
    if (thisform.HDiet_height_in.value != '') {
        cm += thisform.HDiet_height_in.value * 2.54;
    }
    thisform.HDiet_height_cm.value =
        canonicalNumber(cm, 1, thisform.decimal_character.value);;
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.22.3 Inches

The user has modified the inches field. If the value entered is greater than 12, we assume it is intended as a complete specification of height, so we update the feet and inches fields accordingly. The centimetres field is updated to agree with the specification given.

"hdiet.js" 491 ≡

```
function height_changed_in() {
    var thisform = document.getElementById("Hdiet_newacct");
    var inches = thisform.HDiet_height_in.value;
    inches = inches.replace(/,/, ".");
    if (inches > 12) {
        if (inches > 7 * 12) {
            if (!confirm("That's awfully tall (" + inches + " inches).  Are you sure?")) {
                thisform.HDiet_height_in.focus();
                thisform.HDiet_height_in.select();
                return false;
            }
        }
         if (inches < 4 * 12) {
            if (!confirm("That's awfully short (" + inches + " inches).  Are you sure?")) {
                thisform.HDiet_height_in.focus();
                thisform.HDiet_height_in.select();
                return false;
            }
       }

        var feet = Math.floor(inches / 12);
        thisform.HDiet_height_ft.value = feet;
        inches -= feet * 12;
        thisform.HDiet_height_in.value = inches;
    }
    var cm = inches * 2.54;
    if (thisform.HDiet_height_ft.value != '') {
        cm += thisform.HDiet_height_ft.value * 2.54 * 12;
    }
    thisform.HDiet_height_cm.value =
        canonicalNumber(cm, 1, thisform.decimal_character.value);
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.23 Handle change to weight unit in new account creation

When creating a new account (but not when editing the settings for an existing account), these functions are called from the "`onclick`" event handlers of the radio buttons which select the log and display weight units. New users who do not understand the distinction between these units may accidentally set them differently simply by checking one and forgetting the other. Since almost all new accounts will want these units to be the same, if a radio button in one group is clicked and no button in the other group has been clicked, we preset it to be the same as the first one clicked. Afterward, the user is free to change either unit without it affecting the other.

"hdiet.js" 492a ≡

```
var logunit_spec = -1, dispunit_spec = -1;

function set_logunit(t) {
    if (dispunit_spec < 0) {
        var newu = t.value;
        document.getElementById("HDiet_dunit_" +
            WEIGHT_ABBREVIATIONS[newu]).checked = true;
    }
    logunit_spec = newu;
}

function set_dispunit(t) {
    if (logunit_spec < 0) {
        var newu = t.value;
        document.getElementById("HDiet_wunit_" +
            WEIGHT_ABBREVIATIONS[newu]).checked = true;
    }
    dispunit_spec = newu;
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.24 Replace a text node in an HTML document

The `replaceText` function replaces the first child of the node with the specified `id` with `newtext`. This is used to modify table items and other text fields in the monthly log, providing a spreadsheet-style dynamic update when the user modifies editable fields.

"hdiet.js" 492b ≡

```
function replaceText(id, newtext) {
    var n = document.getElementById(id);
    n.replaceChild(document.createTextNode(newtext), n.firstChild);
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.25 Update a variance field

The value `newvar` is placed in the variance field with `id`. The variance is formatted with a leading plus or Unicode minus sign, and the class of the enclosing container is set so that the text is displayed in red or green according to the sign.

"hdiet.js" 493a ≡

```
function updateVariance(id, newvar) {
    var n = document.getElementById(id);
    var fn = Math.abs(newvar).toFixed(1).replace(/\./, decimalCharacter);
    var svar = ((fn == 0) ? "" :
        ((newvar > 0) ? "+" : U_MINUS_SIGN)) + fn;
    n.replaceChild(document.createTextNode(svar), n.firstChild);
    n.setAttribute('class', (fn == 0) ? "bk" :
        (newvar < 0) ? 'g' : 'r');
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.26 Recompute flagged fraction

When the user checks or unchecks a flag box, this function is invoked to recompute the fraction of days flagged. If there is a weight entered for the day, we repaint it to reflect the status of the flag.

"hdiet.js" 493b ≡

```
function updateFlag(day) {
    var unit = Number(document.getElementById("du").getAttribute("value"));
    plotWeightOnChart(day, unit);

    var ndays = document.getElementById("md").getAttribute("value");
    var i, nflagged = 0;
    for (i = 1; i <= ndays; i++) {
        if (document.getElementById("f" + i).checked) {
            nflagged++;
        }
    }
    var fracflagged = Math.round((nflagged * 100) / ndays);
    if (fracflagged > 0) {
        replaceText("percent_flagged", fracflagged + "%");
        document.getElementById("fracf").style.display = "inline";
    } else {
        document.getElementById("fracf").style.display = "none";
    }
    countChange();
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.27 Password strength estimation

As the user enters a password, we provide a subjective indication of its strength in a read-only text field to its right. The strength is given as a number from 0 to 10, 0 indicating no password at all, and 10 a password of excellent security for an application of this kind. We compute the strength by estimating the probability of guessing the password by random assembly of characters with the frequencies present in a large corpus of text. Note that we *do not* test for common words, idiot passwords, or those easily guessed from information available about the user. That would be nice, but we're just a humble client-side utility and can't afford large databases or extensive computation.

For Unicode characters with code points above the Latin-1 range, for which we lack statistics, we make a kludge estimate of the probability of the character's being guessed as $(1/2^{16})(((\log_2 n) - 8)/16)$ where n is the Unicode code point for the character.

We return the inverse probability of (or, in other words, the expected number of guesses needed to determine) the password. This is expressed to the user by taking the $\log_{10}$ of the value, subtracting 9, and scoring the result as between 1 and 10.

"hdiet.js" 494 ≡

```
function passwordStrength(s) {
    ⟨ Character frequency table 496 ⟩

    var pprob = 1.0;

    ⟨ Ad hoc tests for bad passwords 495 ⟩

    for (i = 0; i < s.length; i++) {
        var c = s.charCodeAt(i), p;
        if (c > 0xFF) {
            p = (1.0 / 65536.0) * ((psLog2(c) - 8) / 16);
        } else {
            p = characterFrequency[(c < 128) ? (c - 32) : (c - 65)];
            if (p == 0) {
                p = 1.0e-7;
            }
        }
        pprob *= p;
    }
    return 1 / pprob;
}

function showPasswordStrength() {
    var thisform = document.getElementById("Hdiet_newacct");
    var ps = passwordStrength(thisform.HDiet_password.value);
    thisform.HDiet_password_strength.value =
        (thisform.HDiet_password.value.length < 6) ? 0 :
    Math.round(Math.min(Math.max(psLog10(ps) - 9, 1), 10));
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.27.1 *Ad hoc* tests for bad passwords

Here we check for some common cases of "idiot passwords" which naïvely might appear far more secure than they are, in fact. We handle these by deleting the characters which contribute little to the password's security and scoring it as if it were a shorter string without them.

⟨ Ad hoc tests for bad passwords 495 ⟩ ≡

```
        //  The string "password" and other bozo classics
        s = s.replace(/password|secret|qwerty|cookie|loveyou|/ig, "");

        //  Consecutive identical characters
        s = s.replace(/(.)\1+/g, "$1");

        //  Three or more characters in code point order or decending order
        for (i = 0; i < s.length - 2; i++) {
            if (((s.charCodeAt(i) == (s.charCodeAt(i + 1) - 1)) &&
                 (s.charCodeAt(i + 1) == (s.charCodeAt(i + 2) - 1))) ||
                 ((s.charCodeAt(i) == (s.charCodeAt(i + 1) + 1)) &&
                 (s.charCodeAt(i + 1) == (s.charCodeAt(i + 2) + 1)))
                ) {
                s = s.substring(0, i) + s.substring(i + 1);
                i = 0;
            }
        }
    ◇
```

Macro referenced in 494.

24.27.2 Character frequency table

This table, which is indexed rather curiously with the first 95 entries representing the ASCII character between space (32) and tilde (127) inclusive, and the following 96 the Latin-1 graphics between nonbreaking space (160) and "ÿ" inclusive, gives the empirical fraction that each character made up of a large corpus of text in a variety of human and programming languages. This is used to estimate the entropy of characters in order to determine the cryptographic strength of a password.

⟨ Character frequency table 496 ⟩ ≡

```
var characterFrequency = new Array (
    0.10696, 0.00081822, 0.0023291, 4.3716e-05, 0.00015954, 1.8698e-05,
    8.4113e-05, 0.0030053, 0.00047366, 0.00047334, 5.0773e-05, 4.7613e-05,
    0.0074841, 0.003832, 0.0073566, 0.0022768, 0.0006086, 0.0010785,
    0.00065979, 0.00050631, 0.00045059, 0.00044005, 0.00038765,
    0.00035741, 0.00034319, 0.00035236, 0.00050331, 0.0013616, 0.005661,
    0.0012761, 0.0055853, 0.00077171, 0.00013994, 0.0035894, 0.0013236,
    0.0019242, 0.0014263, 0.003019, 0.00099098, 0.0010051, 0.001466,
    0.0034202, 0.00031154, 0.00053323, 0.0018927, 0.0016076, 0.0019353,
    0.0020567, 0.0012778, 0.00019367, 0.0018611, 0.0029033, 0.0026777,
    0.0010777, 0.00044205, 0.0012584, 7.0946e-05, 0.00058568, 0.00012246,
    0.00020457, 0.00019619, 0.00020383, 7.005e-06, 0.00018455, 2.4702e-05,
    0.065786, 0.014786, 0.027696, 0.029905, 0.097183, 0.011098, 0.016708,
    0.027406, 0.062681, 0.0013139, 0.0057647, 0.04161, 0.023039, 0.058548,
    0.056328, 0.020771, 0.0023996, 0.054953, 0.057549, 0.055857, 0.031333,
    0.008424, 0.0082229, 0.0021771, 0.014088, 0.0025955, 0.00022901,
    9.1645e-05, 0.00022885, 5.7936e-06, 0, 0.00014447, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0.00022864, 0, 0, 0, 0, 6.1623e-06, 0, 0, 0, 0, 0, 0, 0, 0,
    5.2669e-08, 0, 0.00022595, 0, 0, 0, 5.2406e-05, 1.4747e-06,
    2.1068e-07, 1.5801e-07, 0, 1.0007e-06, 0, 0, 5.2669e-07, 8.9538e-07,
    5.7568e-05, 6.847e-07, 1.5801e-07, 5.2669e-07, 1.5801e-07, 4.2136e-07,
    1.5801e-07, 0, 0, 1.5801e-07, 1.5801e-07, 1.9172e-05, 0, 4.2136e-07,
    0, 0, 8.9538e-07, 1.5801e-07, 1.5801e-07, 5.7936e-06, 0, 0,
    0.00014258, 0.0004153, 0.00039244, 6.3467e-05, 0, 0.00017786, 0, 0,
    3.5815e-05, 0.00024918, 0.0017644, 0.00013225, 3.3708e-06, 1.5801e-07,
    0.0001953, 4.7455e-05, 1.4905e-05, 0, 7.5107e-05, 1.3641e-05,
    0.00011571, 2.2279e-05, 0, 0.00010207, 0, 0, 3.0127e-05, 3.039e-05,
    4.282e-05, 0.00026688, 0, 0, 0
);
```

◇

Macro referenced in 494.

24.28 Password match indication

Since the user enters and confirms their password in a field whose contents are not displayed by the browser, we provide a little read-only check box to the right of the "Retype password" field which indicates whether it and the "Password" field are actually the same. The will prevent, in most cases, the need for bouncing the entire form in the case of a mismatch, which forces the user (due to brower password caching security rules) to re-enter both the password and the confirmation.

"hdiet.js" 497a ≡

```
function checkPasswordMatch() {
    var thisform = document.getElementById("Hdiet_newacct");
    thisform.HDiet_password_match.checked =
        thisform.HDiet_password.value == thisform.HDiet_rpassword.value;
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.29 Mathematical functions

The following functions provide useful mathematical functions absent in JavaScript such as $\log_2$ and $\log_{10}$.

"hdiet.js" 497b ≡

```
function psLog2(x) {
    return Math.LOG2E * Math.log(x);
}

function psLog10(x) {
    return Math.LOG10E * Math.log(x);
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

24.30 Debugging console support

The `dump` function allows easy access to the JavaScript debugging console embedded in pages under development. It is called with pairs of arguments which give the name and values to be displayed on the console.

"hdiet.js" 498 ≡

```
function dump()
{
    var t = "", i;

    for (i = 0; i < arguments.length; i += 2) {
        if (t.length > 0) {
            t += ", ";
        }
        t += arguments[i] + " = " + arguments[i + 1];
    }
    document.getElementById("debugging_console").log.value += t + "\n";
}
```

◇

File defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.

Chapter 25

XML Database Export Document Type Definition

The `hackersdiet.dtd` file contains the Document Type Definition (DTD) for the XML export files generated from the database. The DTD exists purely to document the format of a compliant document and permit validation; nothing in it is required to parse an XML export file (which uses non non-standard entities), which is logically, if not officially "`standalone`".

25.1 Overall document structure

The `hackersdiet` element is the unique container which serves as the root of the document tree.

"hackersdiet.dtd" 500 ≡

```
<!--              The Hacker's Diet Online
            http://www.fourmilab.ch/hackdiet/online/

              XML Database Document Type Definition

    This definition is cited with a:
        <!DOCTYPE hackersdiet SYSTEM
            "http://www.fourmilab.ch/hackdiet/online/hackersdiet.dtd">
    declaration in compliant XML files.

-->

    <!-- Root element -->

    <!ELEMENT hackersdiet
        (epoch?, account?, monthlogs?)
    >

    <!ATTLIST hackersdiet
        version     CDATA   #REQUIRED
    >

    <!ELEMENT epoch                 (#PCDATA)>

    <!ELEMENT account
        (user?, preferences?, diet-plan?)
    >

    <!ATTLIST account
        version     CDATA   #REQUIRED
    >

    <!ELEMENT monthlogs
        (monthlog*)
    >

    <!ATTLIST monthlogs
        version     CDATA   #REQUIRED
    >
◇
```

File defined by 500, 501, 502, 503, 504.

25.2 User information

User information is encoded within the `user` element. The distinction between user information and preferences (see below) is a bit fuzzy, and both are kept in the `user` object in the application, but in general the `user` elements describes invariant properties of the user, while `preferences` are whims changeable at the will.

"hackersdiet.dtd" 501 ≡

```
<!-- User information.  This element is optional but,
     if present, must be first in the file. -->

<!ELEMENT user
    (login-name?,
     first-name?,
     middle-name?,
     last-name?,
     e-mail?,
     height?,
     account-created?)
>

<!ATTLIST user
    version     CDATA   #REQUIRED
>

<!ELEMENT login-name            (#PCDATA)>
<!ELEMENT first-name            (#PCDATA)>
<!ELEMENT middle-name           (#PCDATA)>
<!ELEMENT last-name             (#PCDATA)>
<!ELEMENT e-mail                (#PCDATA)>
<!ELEMENT height                (#PCDATA)>
<!ELEMENT account-created       (#PCDATA)>
```

◇

File defined by 500, 501, 502, 503, 504.

25.3 Preferences

The `preferences` element contains the user's choice of units and other items which can be changed at will.

"hackersdiet.dtd" 502 ≡

```
<!-- Preferences.  This element is optional but,
     if present, must be after the user information,
     if present.  -->

<!ELEMENT preferences
    (log-unit?,
     display-unit?,
     energy-unit?,
     current-rung?,
     decimal-character?)
>

<!ATTLIST preferences
    version     CDATA   #REQUIRED
>

<!ELEMENT log-unit              (#PCDATA)>
<!ELEMENT display-unit          (#PCDATA)>
<!ELEMENT energy-unit           (#PCDATA)>
<!ELEMENT current-rung          (#PCDATA)>
<!ELEMENT decimal-character     (#PCDATA)>
```

◇

File defined by 500, 501, 502, 503, 504.

25.4 Diet Plan

The `diet-plan` element contains the quantities established by the user in the diet calculator, and controls whether the plan is plotted in charts.

"hackersdiet.dtd" 503 ≡

```
<!-- Diet plan.  This element is optional but,
     if present, must be after the preferences,
     if present.  -->

<!ELEMENT diet-plan
    (calorie-balance?,
     start-weight?,
     goal-weight?,
     start-date?,
     show-plan?)
>

<!ATTLIST diet-plan
    version     CDATA   #REQUIRED
>

<!ELEMENT calorie-balance       (#PCDATA)>
<!ELEMENT start-weight          (#PCDATA)>
<!ELEMENT goal-weight           (#PCDATA)>
<!ELEMENT start-date            (#PCDATA)>
<!ELEMENT show-plan             (#PCDATA)>
```

◇

File defined by 500, 501, 502, 503, 504.

25.5 Monthly Log

A `monthlog` container will be present for each month of the database present in the export file. This is composed of a header followed by a `day` element for each day of the month which gives the log entries for that day. All days are present, even if all of the fields for that day are empty.

"hackersdiet.dtd" 504 ≡

```
<!-- Monthly log.  Any number of monthly logs may
     appear, in any order. -->

<!ELEMENT monthlog
    (properties, days)
>

<!ATTLIST monthlog
    version     CDATA   #REQUIRED
>

<!ELEMENT properties
    (year,
     month,
     weight-unit,
     trend-carry-forward?,
     last-modified?)
>

<!ELEMENT year                  (#PCDATA)>
<!ELEMENT month                 (#PCDATA)>
<!ELEMENT weight-unit           (#PCDATA)>
<!ELEMENT trend-carry-forward   (#PCDATA)>
<!ELEMENT last-modified         (#PCDATA)>

<!ELEMENT days
    (day+)
>

<!ATTLIST days
    ndays       CDATA   #REQUIRED
>

<!ELEMENT day
    (date,
     weight,
     rung,
     flag,
     comment)
>

<!ELEMENT date                  (#PCDATA)>
<!ELEMENT weight                (#PCDATA)>
<!ELEMENT rung                  (#PCDATA)>
<!ELEMENT flag                  (#PCDATA)>
<!ELEMENT comment               (#PCDATA)>
```

◇

File defined by 500, 501, 502, 503, 504.

Chapter 26

XML Database Export Style Sheet

The `hackdiet_db.css` file provides a style sheet which renders XML database export files in a more or less primate-readable format. This is not intended for presentation, but simply to make examination of databases easier than digging into raw XML.

26.1 Overall document structure

The `hackersdiet` element is the unique container which serves as the root of the document tree.

"hackdiet_db.css" 505 ≡

```
hackersdiet {
    display: block;
    text-align: center;
}

hackersdiet:before {
    content: "Hacker's Diet Online Database Export";
    font-size: xx-large;
    font-family: sans-serif;
}

hackersdiet * {
    padding-left: 8%;
}
```

◇

File defined by 505, 506ab, 507, 508, 509, 510, 511, 512.

26.2 Epoch of export

"hackdiet_db.css" 506a ≡

```
        epoch {
            display: block;
            margin-top: 1em;
            text-align: center;
            font-family: sans-serif;
            font-weight: bolder;
            font-size: larger;
        }

        epoch:before {
            display: inline;
            content: "Epoch: ";
        }
```

◇

File defined by 505, 506ab, 507, 508, 509, 510, 511, 512.

26.3 Account information

"hackdiet_db.css" 506b ≡

```
        account {
            display: block;
            margin-top: 1em;
            text-align: center;
        }

        account:before {
            content: "Account Information";
            font-family: sans-serif;
            font-weight: bolder;
            font-size: large;
        }
```

◇

File defined by 505, 506ab, 507, 508, 509, 510, 511, 512.

26.4 User identification

"hackdiet_db.css" 507 ≡

```
user {
    display: block;
    margin-top: 1em;
    text-align: left;
}

user:before {
    display: block;
    content: "User:";
    font-family: sans-serif;
    font-weight: bolder;
    font-size: larger;
}

user * {
    display: block;
}

user *:before {
    font-weight: bolder;
    margin-right: 1em;
}

user login-name:before {
    content: "Account name:";
}

user first-name:before {
    content: "First name:";
}

user middle-name:before {
    content: "Middle name:";
}

user last-name:before {
    content: "Last name:";
}

user e-mail:before {
    content: "E-mail address:";
}

user height:before {
    content: "Height:";
}

user height:after {
    content: " cm";
}

user account-created:before {
    content: "Account created:";
}
```

◇

File defined by 505, 506ab, 507, 508, 509, 510, 511, 512.

26.5 Preferences

"hackdiet_db.css" 508 ≡

```
        preferences {
            display: block;
            margin-top: 1em;
            text-align: left;
        }

        preferences:before {
            display: block;
            content: "Preferences:";
            font-family: sans-serif;
            font-weight: bolder;
            font-size: larger;
        }

        preferences * {
            display: block;
        }

        preferences *:before {
            font-weight: bolder;
            margin-right: 1em;
        }

        preferences log-unit:before {
            content: "Log weight unit:";
        }

        preferences display-unit:before {
            content: "Display weight unit:";
        }

        preferences energy-unit:before {
            content: "Energy unit:";
        }

        preferences current-rung:before {
            content: "Current rung:";
        }

        preferences decimal-character:before {
            content: "Decimal character:";
        }
```

◇

File defined by 505, 506ab, 507, 508, 509, 510, 511, 512.

26.6 Diet plan

"hackdiet_db.css" 509 ≡

```
diet-plan {
    display: block;
    margin-top: 1em;
    text-align: left;
}

diet-plan:before {
    display: block;
    content: "Diet Plan:";
    font-family: sans-serif;
    font-weight: bolder;
    font-size: larger;
}

diet-plan * {
    display: block;
}

diet-plan *:before {
    font-weight: bolder;
    margin-right: 1em;
}

diet-plan calorie-balance:before {
    content: "Calorie balance:";
}

diet-plan start-weight:before {
    content: "Starting weight:";
}

diet-plan goal-weight:before {
    content: "Goal weight:";
}

diet-plan start-date:before {
    content: "Start date:";
}

diet-plan show-plan:before {
    content: "Show plan in charts:";
}
```

◇

File defined by 505, 506ab, 507, 508, 509, 510, 511, 512.

26.7 Monthly logs

The monthly logs receive the fanciest formatting—the equivalent of a table in XHTML, defined entirely in CSS. We start by generating a header for the logs from the `monthlogs` container in which they are enclosed.

"hackdiet_db.css" 510 ≡

```
monthlogs {
    display: block;
    margin-top: 1em;
    text-align: center;
}

monthlogs:before {
    content: "Monthly Logs";
    font-family: sans-serif;
    font-weight: bolder;
    font-size: large;
}
```

◇

File defined by 505, 506ab, 507, 508, 509, 510, 511, 512.

Then, for each individual log we generate a header with a light grey background with the properties of the log labeled on separate lines.

"hackdiet_db.css" 511 ≡

```
monthlog {
    display: block;
    margin-top: 1em;
    text-align: left;
}

monthlog * {
    display: block;
}

monthlog *:before {
    font-weight: bolder;
    margin-right: 1em;
}

monthlog properties {
    display: block;
    background-color: #E0E0E0;
    width: 75%;
}

monthlog year:before {
    content: "Year:";
}

monthlog month:before {
    content: "Month:";
}

monthlog weight-unit:before {
    content: "Weight unit:";
}

monthlog trend-carry-forward:before {
    content: "Trend carry-forward:";
}

monthlog last-modified:before {
    content: "Last modified:";
}


monthlog last-modified {
    margin-bottom: 1ex;
}
```

◇

File defined by 505, 506ab, 507, 508, 509, 510, 511, 512.

The entries for days of the month are enclosed in a `days` container, with each day defined in a `day` container within it. We format the items for each day in columns.

"hackdiet_db.css" 512 ≡

```
monthlog days {
    display: table;
}

monthlog day {
    display: table-row;
}

monthlog day * {
    display: table-cell;
}

monthlog day date {
    text-align: right;
    width: 2em;
}

monthlog day weight {
    text-align: right;
    width: 4em;
}

monthlog day rung {
    text-align: right;
    width: 4em;
}

monthlog day flag {
    text-align: right;
    width: 1em;
}

monthlog day comment {
    text-align: left;
    padding-left: 48px;
}
```

◇

File defined by 505, 506ab, 507, 508, 509, 510, 511, 512.

Chapter 27

webapp.html: Main Web Page

This is the main Web page for The Hacker's Diet Online. It contains the user documentation and the initial request form.

"webapp.html" 513 ≡

```
⟨ HTML header section 514 ⟩

<body bgcolor="#FFFFFF" onload="initialiseDocument();">

 <p>


<div class="bodycopy">

<p>
<hr>

<h3><a href="/">Fourmilab Home Page</a></h3>

<address>
by <a href="/">John Walker</a><br />
⟨ Release Date 3b ⟩
</address>
<center>
<em>This document is in the public domain.</em>
<br /> 
</center>

</div>

</body>
</html>
◇
```

27.1 HTML Header Section

⟨ HTML header section 514 ⟩ ≡

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html lang="en">
<head>
<title>The Hacker's Diet Online</title>
<style type="text/css">
    div.bodycopy {
        margin-left: 10%;
        margin-right: 10%
    }
</style>

<meta name="keywords" content="hacker, hacker's, diet, online, john, walker" />
<meta name="description" content="The Hacker's Diet Online" />
<meta name="author" content="John Walker" />
<meta name="robots" content="index" />

</head>
◇
```

Macro referenced in 513.

Chapter 28

Makefile

This is the `Makefile` for Hdiet. Of course, generating the `Makefile` from the Nuweb invites infinite regress, since it's the `Makefile` which invokes `nuweb` to create.... But as long as we include the generated `Makefile` in the source distribution, all will be well, and we do that below, in the definition of the `Makefile` in the Nuweb. **Slap!** Thanks—I needed that.

Since, in the interest of preserving formatting in the LaTeX code documentation, we edit this file with hardware tabs disabled, we must cope with the regrettable detail that `make` uses tabs as a significant character.

"Makefile.mkf" 515 ≡

```
WEBDIR = ⟨ Web Directory 5d ⟩

CGIDIR = ⟨ CGI Installation Directory 5f ⟩

EXEDIR = ⟨ Executable Installation Directory 6d ⟩

WEBDIR_PRODUCTION = ⟨ Production Web Directory 5e ⟩

CGIDIR_PRODUCTION = ⟨ Production CGI Installation Directory 5g ⟩

EXEDIR_PRODUCTION = ⟨ Production Executable Installation Directory 6e ⟩

PROGRAMS = jig.pl bump

duh:
        @echo "Please choose: check dist publish test weblint"

⟨ Extract source code from Nuweb 516 ⟩

⟨ Installation 520 ⟩

⟨ Source installation 521 ⟩

⟨ Source distribution 517 ⟩

⟨ Documentation 518 ⟩

⟨ Testing 519a ⟩

⟨ Clean-up 519b ⟩
```

◇

28.1 Extract source code from Nuweb

All of the source code for Hdiet, its support files, documentation, and the tools used to build it are defined in the Nuweb file `hdiet.w`. Processing this file with `nuweb` suffices to extract all the contents, so we can use the Perl source code `jig.pl` as a proxy for all the files generated from the Nuweb program. Any `Makefile` target which requires a file from the Nuweb can simply specify `jig.pl` as a dependency and be sure everything is up to date.

One little detail... since the `Makefile` itself is defined here, when you make a change you must first do something that processes the Nuweb ("`make check`" is a good choice) before the `Makefile` will contain the changes you made.

⟨ Extract source code from Nuweb 516 ⟩ ≡

```
jig.pl:    hdiet.w
        @echo -n 'Build '
        @./bump buildno.txt
        @date -u '+%F %R %Z' >buildtime.txt
        nuweb hdiet
        chmod 755 $(PROGRAMS)
        unexpand -a <Makefile.mkf >Makefile
```

◇

Macro referenced in 515.

28.2 Source distribution

Build a source distribution archive for the current version. This process is complicated by the need to Bowdlerise the source code, removing Fourmilab-specific passwords and other security-related information. Since these are defined in the master program web, we must create a sanitised Web in a temporary directory, then re-generate everything from it. This is done in a `Bowdler` directory, and the results are collected into a `hdiet-`*version*`.tar.gz` archive in that directory. The `Bowdler` directory is left around, but will be deleted when the next distribution is built.

⟨ Source distribution 517 ⟩ ≡

```
dist:
        rm -f hdiet.tar hdiet-*.tar.gz
        tar cfv hdiet.tar hdiet.w Bowdler.pl Makefile bump HDiet buildno.txt buildtime.txt wz_jsgraphics.js
        rm -rf Bowdler
        mkdir Bowdler
        ( cd Bowdler ; tar xfv ../hdiet.tar )
        ( cd Bowdler ; perl Bowdler.pl hdiet.w >hdiet.w1 ; mv hdiet.w1 hdiet.w )
        ( cd Bowdler ; make clean )
        ( cd Bowdler ; make check )
        ( cd Bowdler ; latex hdiet; nuweb hdiet ; latex hdiet )
        ( cd Bowdler ; make pdf )
        ( cd Bowdler ; tar cfvz hdiet-⟨ Version 3a ⟩.tar.gz hdiet.w jig.pl Bowdler.pl \
                HackDiet.pl HackDietBadge.pl Makefile \
                bump buildno.txt buildtime.txt \
                webapp.html hdiet.tex hdiet.pdf \
                HDiet \
                hdiet.css hdiet_handheld.css hdiet.js wz_jsgraphics.js \
                hackdiet_db.css hackersdiet.dtd )
        rm -f hdiet.tar
```

◇

Macro referenced in 515.

28.3 Documentation

⟨ Documentation 518 ⟩ ≡

```
view:   jig.pl
        latex hdiet
        nuweb hdiet
        latex hdiet
        xdvi hdiet

viewman: jig.pl
        pod2man hdiet.pl >ZZhdiet.1
        groff -X -man ZZhdiet.1
        rm -f ZZhdiet.1

pdf:    jig.pl
        sed 's///' <hdiet.tex >ZZhdiet.tex
        latex ZZhdiet
        latex ZZhdiet
        pdflatex ZZhdiet
        pdflatex ZZhdiet
        mv ZZhdiet.pdf hdiet.pdf
        rm -f ZZhdiet*

viewpdf: pdf
        acroread hdiet.pdf
```

◇

Macro referenced in 515.

28.4 Testing

⟨ Testing 519a ⟩ ≡

```
check:  $(PROGRAMS)
        perl -c HackDiet.pl
        perl -c HDiet/Aggregator.pm
        perl -c HDiet/Cluster.pm
        perl -c HDiet/ClusterSync.pl
        perl -c HDiet/cookie.pm
        perl -c HDiet/hdCSV.pm
        perl -c HDiet/history.pm
        perl -c HDiet/html.pm
        perl -c HDiet/Julian.pm
        perl -c HDiet/monthlog.pm
        perl -c HDiet/pubname.pm
        perl -c HDiet/session.pm
        perl -c HDiet/trendfit.pm
        perl -c HDiet/user.pm
        perl -c HDiet/xml.pm
        perl -c Bowdler.pl
        perl -c HackDietBadge.pl
        perl -c jig.pl
        weblint webapp.html

test:   $(PROGRAMS)
        perl jig.pl --verbose --test

weblint:    jig.pl
        weblint webapp.html
```

◇

Macro referenced in 515.

28.5 Clean-up

The `clean` target deletes intermediate files but preserves files which are present in the distribution and Web page documentation. The `cvsclean` target deletes everything except for the Nuweb program definition and the Makefile which permits building everything which it defines.

⟨ Clean-up 519b ⟩ ≡

```
clean:
        rm -f hdiet.dvi hdiet.toc hdiet.aux hdiet.log Makefile.mkf hdiet.pdf HDiet/*.pm jig.pl

cvsclean:  clean
        rm -f hdiet.pdf jig.pl *.pm hdiet.tex webapp.html
```

◇

Macro referenced in 515.

28.6 Installation

Our installation targets are a tag tacky at present. Since we've become habituated to testing on the backup server, the "`publish`" target installs in that server. The "`production`" target installs on the live server, whose directory names are prefixed by "Production" in the definitions at the top of the program.

⟨ Installation 520 ⟩ ≡

```
publish:
        cp -p webapp.html hdiet.css hdiet_handheld.css hdiet.js hackersdiet.dtd \
              hackdiet_db.css wz_jsgraphics.js $(WEBDIR)
        cp -p HackDiet.pl $(CGIDIR)/HackDiet.NEW
        chmod 755 $(CGIDIR)/HackDiet.NEW
        mv $(CGIDIR)/HackDiet.NEW $(CGIDIR)/HackDiet
        cp -p HackDietBadge.pl $(CGIDIR)/HackDietBadge.NEW
        chmod 755 $(CGIDIR)/HackDietBadge.NEW
        mv $(CGIDIR)/HackDietBadge.NEW $(CGIDIR)/HackDietBadge
        cp -pr HDiet/* $(CGIDIR)/HDiet
        mv $(CGIDIR)/HDiet/ClusterSync.pl $(EXEDIR)/ClusterSync.NEW
        chmod 755 $(EXEDIR)/ClusterSync.NEW
        mv $(EXEDIR)/ClusterSync.NEW $(EXEDIR)/ClusterSync

production:
        cp -p webapp.html hdiet.css hdiet_handheld.css hdiet.js hackersdiet.dtd \
              hackdiet_db.css wz_jsgraphics.js $(WEBDIR_PRODUCTION)
        cp -p HackDiet.pl $(CGIDIR_PRODUCTION)/HackDiet.NEW
        chmod 755 $(CGIDIR_PRODUCTION)/HackDiet.NEW
        mv $(CGIDIR_PRODUCTION)/HackDiet.NEW $(CGIDIR_PRODUCTION)/HackDiet
        cp -p HackDietBadge.pl $(CGIDIR_PRODUCTION)/HackDietBadge.NEW
        chmod 755 $(CGIDIR_PRODUCTION)/HackDietBadge.NEW
        mv $(CGIDIR_PRODUCTION)/HackDietBadge.NEW $(CGIDIR_PRODUCTION)/HackDietBadge
        cp -pr HDiet/* $(CGIDIR_PRODUCTION)/HDiet
        mv $(CGIDIR_PRODUCTION)/HDiet/ClusterSync.pl $(EXEDIR_PRODUCTION)/ClusterSync.NEW
        chmod 755 $(EXEDIR_PRODUCTION)/ClusterSync.NEW
        mv $(EXEDIR_PRODUCTION)/ClusterSync.NEW $(EXEDIR_PRODUCTION)/ClusterSync
```

◇

Macro referenced in 515.

28.7 Source installation

The source distribution is installed in the test server or production server download directory by the following two targets. The download directory has a subdirectory for each version, so previous versions remain accessible. New directories for versions are created automatically as required. It's up to you to perform a "`make dist`" before installing the distribution.

⟨ Source installation 521 ⟩ ≡

```
pubsrc:
        if [ ! -d $(WEBDIR)/download/⟨ Version 3a ⟩ ] ; then mkdir $(WEBDIR)/download/⟨ Version 3a ⟩ ; fi
        cp -p Bowdler/hdiet-⟨ Version 3a ⟩.tar.gz $(WEBDIR)/download/⟨ Version 3a ⟩
        cp -p Bowdler/hdiet.pdf $(WEBDIR)/download/⟨ Version 3a ⟩

prodsrc:
        if [ ! -d $(WEBDIR_PRODUCTION)/download/⟨ Version 3a ⟩ ] ; then mkdir $(WEBDIR_PRODUCTION)/download/⟨ Vers
        cp -p Bowdler/hdiet-⟨ Version 3a ⟩.tar.gz $(WEBDIR_PRODUCTION)/download/⟨ Version 3a ⟩
        cp -p Bowdler/hdiet.pdf $(WEBDIR_PRODUCTION)/download/⟨ Version 3a ⟩
◇
```

Macro referenced in 515.

Chapter 29

Indices

Three indices are created automatically: an index of file names, an index of macro names, and an index of user-specified identifiers. An index entry includes the name of the entry, where it was defined, and where it was referenced.

29.1 Files

"Bowdler.pl" Defined by 434.
"bump" Defined by 435.
"HackDiet.pl" Defined by 163.
"HackDietBadge.pl" Defined by 431.
"hackdiet_db.css" Defined by 505, 506ab, 507, 508, 509, 510, 511, 512.
"hackersdiet.dtd" Defined by 500, 501, 502, 503, 504.
"hdiet.css" Defined by 436, 437, 438, 439ab, 440, 441ab, 442ab, 443, 444, 445a.
"hdiet.js" Defined by 454, 455abc, 456ab, 457, 458ab, 459abc, 460, 462, 463, 464b, 467, 474, 476ab, 477, 478, 479ab, 480ab, 481ab, 482ab, 483ab, 484ab, 485ab, 486ab, 487, 488ab, 489, 490, 491, 492ab, 493ab, 494, 497ab, 498.
"HDiet/Aggregator.pm" Defined by 106, 107, 108.
"HDiet/Cluster.pm" Defined by 391, 392, 394, 395abc, 396ab.
"HDiet/ClusterSync.pl" Defined by 397a, 404, 405b, 406ab.
"HDiet/cookie.pm" Defined by 146, 147, 148ab, 149ac, 150ab, 151, 152, 153ab.
"HDiet/hdCSV.pm" Defined by 13, 14, 15.
"HDiet/history.pm" Defined by 69, 70, 71, 72, 73, 75, 76, 93ab, 94abc, 95, 96, 97, 100, 101a, 103.
"HDiet/html.pm" Defined by 407.
"HDiet/Julian.pm" Defined by 419.
"HDiet/monthlog.pm" Defined by 19, 21, 22, 23, 24, 25, 26, 27, 28abc, 29ab, 30ab, 31, 36ab, 37abc, 38abc, 39, 40, 41, 50b, 57, 58, 59, 60, 62, 63, 64, 65, 66, 67.
"HDiet/pubname.pm" Defined by 154, 155, 156, 157, 158, 159ab, 160b, 161ab.
"HDiet/session.pm" Defined by 140, 141, 142, 143, 144ac, 145ab.
"HDiet/trendfit.pm" Defined by 16ab, 17ab, 18ab.
"HDiet/user.pm" Defined by 110, 112, 113, 114, 115, 116a, 117, 118, 119, 127ab, 128, 129, 130, 131, 132, 133, 134, 135, 137, 138ab, 139.
"HDiet/xml.pm" Defined by 414, 415ab, 416ab, 417.
"hdiet_handheld.css" Defined by 453.
"jig.pl" Defined by 433.
"Makefile.mkf" Defined by 515.
"webapp.html" Defined by 513.

29.2 Macros

⟨ Account management transactions 170b ⟩ Referenced in 169.
⟨ Activate cluster synchronisation log file if configured 399a ⟩ Referenced in 397a.
⟨ Ad hoc tests for bad passwords 495 ⟩ Referenced in 494.

⟨ Add login to history database 177b ⟩ Referenced in 173.
⟨ Add standard intervals to analysis list 252b ⟩ Referenced in 252a.
⟨ Address for feedback E-mail 10c ⟩ Referenced in 354.
⟨ Administrator object dump selection 204 ⟩ Referenced in 203.
⟨ Administrator-only functions 183 ⟩ Referenced in 179.
⟨ Allocate colours for chart 42 ⟩ Referenced in 41, 75, 93b.
⟨ Append entry to transaction history log 378 ⟩ Referenced in 373.
⟨ Append summary of records imported 227 ⟩ Referenced in 215b.
⟨ Application documentation URL 11h ⟩ Referenced in 409.
⟨ Apply changes to comment 56 ⟩ Referenced in 50b.
⟨ Apply changes to exercise rung 54 ⟩ Referenced in 50b.
⟨ Apply changes to flag 55 ⟩ Referenced in 50b.
⟨ Apply changes to weight 51 ⟩ Referenced in 50b.
⟨ Apply perturbation functions to value 99 ⟩ Referenced in 97.
⟨ Assume group and user identity of cluster synchronisation process 398a ⟩ Referenced in 397a.
⟨ Backup user account before destructive operation 362a ⟩ Referenced in 317, 319, 360, 365.
⟨ Backups Directory 7b ⟩ Referenced in 362a.
⟨ Beta Test Invitations Directory 7a ⟩ Referenced in 290b, 291a, 308.
⟨ Beta test backdoor 4a ⟩ Referenced in 290b.
⟨ Beta test invitation field 121b ⟩ Referenced in 119.
⟨ Beta test 3e ⟩ Referenced in 118, 119, 171, 179, 183, 184, 290b, 291a, 306, 307.
⟨ Book Directory 5b ⟩ Referenced in 5d.
⟨ Book home URL 11g ⟩ Referenced in 11h, 409, 427.
⟨ Browsing public account functions 182 ⟩ Referenced in 179.
⟨ Build Number 3c ⟩ Referenced in 184, 354.
⟨ Build Time 3d ⟩ Referenced in 184, 354.
⟨ Build table of intervals and compute date span of union 74a ⟩ Referenced in 73.
⟨ CGI Execution Directory 5h ⟩ Referenced in 6a.
⟨ CGI Installation Directory 5f ⟩ Referenced in 515.
⟨ CGI Support Directory 6a ⟩ Referenced in 6bc, 369a.
⟨ CSV Format version 4b ⟩ Referenced in 58.
⟨ CSV direct upload import form 213a ⟩ Referenced in 211.
⟨ CSV file upload import form 212 ⟩ Referenced in 211.
⟨ Calculate dependent variables from primary variables 270b ⟩ Referenced in 261, 272.
⟨ Calendar Navigation Tables 450a ⟩ Referenced in 445a.
⟨ Cancel used beta test invitation code 291a ⟩ Referenced in 289.
⟨ Categories of feedback messages 10e ⟩ Referenced in 370b.
⟨ Character frequency table 496 ⟩ Referenced in 494.
⟨ Characters Permissible in File Names 9b ⟩ Referenced in 137.
⟨ Check for Excel CSV record 221 ⟩ Referenced in 220.
⟨ Check for Palm/HDREAD CSV record 225 ⟩ Referenced in 220.
⟨ Check for errors we deem harmless to cluster synchronisation 405a ⟩ Referenced in 404.
⟨ Check for implausibly large change in weight 465 ⟩ Referenced in 464b.
⟨ Check spelling in subject and message 351 ⟩ Referenced in 350.
⟨ Civil time to Julian day fraction 423b ⟩ Referenced in 419.
⟨ Clean-up 519b ⟩ Referenced in 515.
⟨ Close previous session if still open 176a ⟩ Referenced in 173, 188, 323.
⟨ Close this user account 363 ⟩ Referenced in 170a.
⟨ Cluster Failed Transaction Maximum Retries 8c ⟩ Referenced in 402.
⟨ Cluster Failed Transaction Retry Interval 8b ⟩ Referenced in 402.
⟨ Cluster Member Hosts 7f ⟩ Referenced in 391, 397a.
⟨ Cluster Synchronisation Group ID 9a ⟩ Referenced in 398a.
⟨ Cluster Synchronisation Log File 8f ⟩ Referenced in 399ab.
⟨ Cluster Synchronisation Process ID File 8e ⟩ Referenced in 394, 397a, 398b.
⟨ Cluster Synchronisation Signal 8d ⟩ Referenced in 394, 397a.
⟨ Cluster Synchronisation Time Interval 7g ⟩ Referenced in 397a.
⟨ Cluster Synchronisation User ID 8g ⟩ Referenced in 398a.
⟨ Cluster Transaction Directory 7c ⟩ Referenced in 392, 393, 394, 400.

⟨ Cluster Transaction Retry Time Interval 8a ⟩ Referenced in 404.
⟨ Command to check spelling 10f ⟩ Referenced in 351.
⟨ Compute diet plan extrema on chart 47 ⟩ Referenced in 45.
⟨ Compute global statistics gain and loss extrema 335 ⟩ Referenced in 331.
⟨ Compute global statistics trend analysis for previous user 340 ⟩ Referenced in 338.
⟨ Configure Web page status badge 228, 229, 230 ⟩ Referenced in 169.
⟨ Confirm a persistent login is selected 330 ⟩ Referenced in 329.
⟨ Confirm a session is selected 325 ⟩ Referenced in 323.
⟨ Confirmation signature encoding suffix 4c ⟩ Referenced in 149c, 153a, 359, 360, 365, 394, 401.
⟨ Constants and conversion tables 20 ⟩ Referenced in 19.
⟨ Convert characters in a string to hexadecimal 386a ⟩ Referenced in 373.
⟨ Convert trend to weight unit in this log, if different 376b ⟩ Referenced in 376a.
⟨ Cookie name 11a ⟩ Referenced in 151, 172, 178.
⟨ Create new month for synthetic data 98 ⟩ Referenced in 97.
⟨ Create new user account request 288a ⟩ Referenced in 173.
⟨ Create the new user account 291b ⟩ Referenced in 289.
⟨ Custom start and end date selection boxes 258a ⟩ Referenced in 257, 281a.
⟨ Custom trend end date 260 ⟩ Referenced in 238a, 246a, 258a, 269a.
⟨ Custom trend start date 259 ⟩ Referenced in 238a, 246a, 258a, 269a.
⟨ Cycle active log file if HUP signal received 399b ⟩ Referenced in 397a.
⟨ Database Directory 6f ⟩ Referenced in 6ghi, 7abc, 11e, 163, 401.
⟨ Decimal character selection 126a ⟩ Referenced in 119.
⟨ Decode encrypted user ID 136 ⟩ Referenced in 431, 433.
⟨ Default cookie retention time 11b ⟩ Referenced in 147.
⟨ Default parameter settings 370a ⟩ Referenced in 369a.
⟨ Define “cachebuster” argument 199b ⟩ Referenced in 199a, 281b.
⟨ Define chart geometry 43a ⟩ Referenced in 40, 41.
⟨ Define historical chart geometry 80b ⟩ Referenced in 75.
⟨ Define requests permissible whilst browsing public account 164b ⟩ Referenced in 163.
⟨ Delete a persistent login token 329 ⟩ Referenced in 171.
⟨ Delete entire log database 357 ⟩ Referenced in 170a.
⟨ Delete existing public name 160a ⟩ Referenced in 158, 159b.
⟨ Determine first and last days in database 240 ⟩ Referenced in 239, 252a, 280.
⟨ Determine if failed transaction should be retried 403 ⟩ Referenced in 400.
⟨ Determine range of dates to plot in historical chart 282 ⟩ Referenced in 281b.
⟨ Determine scale for weight and trend plot 45, 46 ⟩ Referenced in 40, 41.
⟨ Determine the number of days in the historical interval 77 ⟩ Referenced in 73, 75, 97.
⟨ Determine vertical weight scaling based on extrema 81ab, 82a ⟩ Referenced in 75.
⟨ Determine which monthly log to display 197a ⟩ Referenced in 196.
⟨ Diet calculator form action buttons 270a ⟩ Referenced in 261.
⟨ Diet calculator 261 ⟩ Referenced in 169.
⟨ Dispatch administrator requests 171 ⟩ Referenced in 169.
⟨ Dispatch requests which return HTML result documents 169 ⟩ Referenced in 163.
⟨ Dispatch requests which return non-HTML results 168 ⟩ Referenced in 163.
⟨ Display CSV import request form 211 ⟩ Referenced in 169.
⟨ Display administrator account manager 309, 310, 311, 312 ⟩ Referenced in 171.
⟨ Display administrator global statistics 331 ⟩ Referenced in 171.
⟨ Display administrator persistent login manager 327 ⟩ Referenced in 171.
⟨ Display administrator session manager 320 ⟩ Referenced in 171.
⟨ Display calendar navigation page 208 ⟩ Referenced in 169.
⟨ Display global statistics gain and loss extrema 336 ⟩ Referenced in 331.
⟨ Display global statistics log update frequency 337 ⟩ Referenced in 331.
⟨ Display global statistics mean trend change 334 ⟩ Referenced in 331.
⟨ Display monthly log 196 ⟩ Referenced in 169.
⟨ Display password reset request form 187 ⟩ Referenced in 170b.
⟨ Display summary of cluster transaction queue 393 ⟩ Referenced in 392.
⟨ Display trend summary below monthly chart 202 ⟩ Referenced in 196.
⟨ Documentation in POD format 427 ⟩ Referenced in 163.

⟨ Documentation 518 ⟩ Referenced in 515.
⟨ Domain for cookies 11c ⟩ Referenced in 150ab.
⟨ Download backup copy of all logs for user 249 ⟩ Referenced in 168.
⟨ Download monthly log as CSV file 235b ⟩ Referenced in 168.
⟨ Download monthly log as XML file 236 ⟩ Referenced in 168.
⟨ Draw axes for chart and label date axis 43b ⟩ Referenced in 41.
⟨ Draw axes for historical chart 82b ⟩ Referenced in 75.
⟨ Draw caption with trend summary 92 ⟩ Referenced in 87.
⟨ Draw text in a chart 387 ⟩ Referenced in 373.
⟨ Draw title with date range 91 ⟩ Referenced in 87.
⟨ Dump CGI environment and parsed arguments 368a ⟩ Referenced in 367c.
⟨ Dump XML database file 219 ⟩ Not referenced.
⟨ Dump objects if requested by administrator 205 ⟩ Referenced in 196.
⟨ E-mail address text field 123a ⟩ Referenced in 119.
⟨ Edit Unix time value to ISO 8601 local date and time 385b ⟩ Referenced in 373.
⟨ Embed historical chart image in request/result page 281b ⟩ Referenced in 280.
⟨ Emit diagnostic for undefined query 367c ⟩ Referenced in 169.
⟨ Emit historical chart request form 280 ⟩ Referenced in 279.
⟨ Emit shrill warning about what is about to transpire 358a ⟩ Referenced in 357.
⟨ Emit trend anlysis page 252a ⟩ Referenced in 251.
⟨ Empty monthly log cache 105b ⟩ Referenced in 75.
⟨ Encode international domain name 388a ⟩ Referenced in 373.
⟨ Encode preset values for use in HTML 120 ⟩ Referenced in 119.
⟨ Encoding for Space in File Name Characters 9c ⟩ Referenced in 137.
⟨ Ensure month is in cache 102 ⟩ Referenced in 100, 101a.
⟨ Enumerate feedback message categories 352 ⟩ Referenced in 349.
⟨ Estimate local time at user site 167 ⟩ Referenced in 163.
⟨ Examine dates, fitting those within intervals 74b ⟩ Referenced in 73.
⟨ Executable Installation Directory 6d ⟩ Referenced in 515.
⟨ Execute cluster synchronisation transaction 401 ⟩ Referenced in 400.
⟨ Execute system command 385a ⟩ Referenced in 373.
⟨ Expand abbreviated stones and pounds entry 461 ⟩ Referenced in 460.
⟨ Expand weight entry if abbreviated 52 ⟩ Referenced in 51.
⟨ Expand weight entry in stones and pounds 53 ⟩ Referenced in 52.
⟨ Export database as Hacker's Diet Online CSV 242 ⟩ Referenced in 239.
⟨ Export database as Legacy Excel Eat Watch CSV 244 ⟩ Referenced in 239.
⟨ Export database as Palm Eat Watch CSV 243 ⟩ Referenced in 239.
⟨ Export database as XML 241 ⟩ Referenced in 239.
⟨ Export log database 237 ⟩ Referenced in 169.
⟨ Extract brain-dead Internet Exploder request field 166 ⟩ Referenced in 163.
⟨ Extract scaling information for chart 468 ⟩ Referenced in 467, 475.
⟨ Extract source code from Nuweb 516 ⟩ Referenced in 515.
⟨ Feedback message table 451 ⟩ Referenced in 445a.
⟨ Fill cache with monthly logs in the date range 104 ⟩ Referenced in 73, 97.
⟨ Fill in trend carry-forward from most recent previous log, if required 201 ⟩ Referenced in 197b.
⟨ Find diet plan extrema on historical chart 79 ⟩ Referenced in 75.
⟨ Find most recent trend value before this day 469 ⟩ Referenced in 464b.
⟨ Find weight and trend extrema in log entries to be plotted 78 ⟩ Referenced in 75.
⟨ Fit a linear trend and update weight and energy balance 472 ⟩ Referenced in 464b.
⟨ Force re-login if session terminated or invalid 186 ⟩ Referenced in 170a.
⟨ Force termination of user session 323 ⟩ Referenced in 171.
⟨ Forget all persistent logins 300 ⟩ Referenced in 170b.
⟨ Form processing action and method 12b ⟩ Referenced in 117, 181, 182, 183, 185, 187, 191, 196, 210, 214, 228, 237, 245, 257, 261, 280, 288a, 294, 295, 299, 301, 306, 307, 309, 320, 327, 331, 341, 349, 357, 358b, 363.
⟨ From address for mail sent to users 10b ⟩ Referenced in 128, 354, 355.
⟨ Generate XHTML navigation bar 410ab, 411 ⟩ Referenced in 407.
⟨ Generate array of years for diet calculator selection 264a ⟩ Referenced in 261, 272.
⟨ Generate assumed identity notification 185 ⟩ Referenced in 179, 196, 208, 211, 214, 228, 231, 234, 237, 239, 245, 247, 249,

251, 261, 279, 295, 296, 297, 299, 300, 301, 304, 305, 306, 307, 309, 314, 315, 316, 318, 320, 324, 325, 326, 327, 330, 331, 341, 348, 353, 357, 360, 363, 365.
⟨ Generate comment column 35a ⟩ Referenced in 31.
⟨ Generate confirmation of database deletion 362b ⟩ Referenced in 360.
⟨ Generate date column 32b ⟩ Referenced in 31.
⟨ Generate destructive operation confirmation form 359 ⟩ Referenced in 357, 363.
⟨ Generate diet calculator form 265ab, 266ab, 267ab, 268, 269a ⟩ Referenced in 261.
⟨ Generate exercise rung column 34a ⟩ Referenced in 31.
⟨ Generate feedback message composition form 349 ⟩ Referenced in 348.
⟨ Generate flag column 34b ⟩ Referenced in 31.
⟨ Generate form fields for custom chart interval 281a ⟩ Referenced in 280.
⟨ Generate form fields for custom trend interval 257 ⟩ Referenced in 252a.
⟨ Generate form permitting user to back up database 358b ⟩ Referenced in 357.
⟨ Generate global statistics for open accounts 333 ⟩ Referenced in 331.
⟨ Generate hidden monthly log property fields 200 ⟩ Referenced in 196.
⟨ Generate historical chart 287 ⟩ Referenced in 168.
⟨ Generate invitation codes 307, 308 ⟩ Referenced in 171.
⟨ Generate monthly chart 250a ⟩ Referenced in 168.
⟨ Generate option items for chart sizes 286b ⟩ Referenced in 280.
⟨ Generate option items for days 286a ⟩ Referenced in 259, 260, 344, 345.
⟨ Generate option items for months 285c ⟩ Referenced in 259, 260, 344, 345.
⟨ Generate option items for years in database 285b ⟩ Referenced in 259, 260.
⟨ Generate paper log form for month 248 ⟩ Referenced in 247.
⟨ Generate paper log forms 247 ⟩ Referenced in 169.
⟨ Generate synthetic data as specified in form 342 ⟩ Referenced in 341.
⟨ Generate synthetic data for user account 341 ⟩ Referenced in 171.
⟨ Generate synthetic data specification form 343 ⟩ Referenced in 341.
⟨ Generate table of open sessions 322 ⟩ Referenced in 320.
⟨ Generate table of persistent logins 328b ⟩ Referenced in 327.
⟨ Generate table of public accounts 303 ⟩ Referenced in 301.
⟨ Generate table of yearly calendars 209 ⟩ Referenced in 208.
⟨ Generate test output page 367b ⟩ Referenced in 169.
⟨ Generate warning if JavaScript disabled in diet calculator form 263a ⟩ Referenced in 261.
⟨ Generate weight, trend, and variance columns 33 ⟩ Referenced in 31.
⟨ Global declarations 369a ⟩ Referenced in 163, 433, 434.
⟨ Global statistics tables 452 ⟩ Referenced in 445a.
⟨ Global variables 370b ⟩ Referenced in 369a.
⟨ Gregorian date to Julian day number 421b ⟩ Referenced in 419.
⟨ Gregorian leap year computation 421a ⟩ Referenced in 419.
⟨ HTML header section 514 ⟩ Referenced in 513.
⟨ Height (for body mass index) 124 ⟩ Referenced in 119.
⟨ Identify log where recomputation begins 375a ⟩ Referenced in 374.
⟨ If requested, canonicalise weight entries in log 377b ⟩ Referenced in 375b, 376a.
⟨ Image and Icon Directory 6c ⟩ Referenced in 93b, 431.
⟨ Import log items from CSV database file 220 ⟩ Referenced in 215b.
⟨ Import log items from XML database file 217 ⟩ Referenced in 215b.
⟨ Import uploaded CSV log entries 214, 215ab, 216 ⟩ Referenced in 169.
⟨ Installation 520 ⟩ Referenced in 515.
⟨ JavaScript debugging console 368b ⟩ Not referenced.
⟨ Julian date constant definitions 420a ⟩ Referenced in 419.
⟨ Julian date support functions 420b ⟩ Referenced in 419.
⟨ Julian day and fraction to RFC 3339 time and date 426a ⟩ Referenced in 419.
⟨ Julian day and fraction to RFC 822 time and date 425b ⟩ Referenced in 419.
⟨ Julian day and fraction to Unix time 424c ⟩ Referenced in 419.
⟨ Julian day and fraction to old HTTP cookie time and date 426b ⟩ Referenced in 419.
⟨ Julian day fraction to civil time 424a ⟩ Referenced in 419.
⟨ Julian day to Gregorian date 422 ⟩ Referenced in 419.
⟨ Julian day to day of week 423a ⟩ Referenced in 419.

⟨ Label date axis at the bottom 83 ⟩ Referenced in 82b.
⟨ Label date axis with year numbers only 85 ⟩ Referenced in 83.
⟨ Label date axis with years, months, and possibly weeks 84 ⟩ Referenced in 83.
⟨ Label exercise rung axis if any plotted 86b ⟩ Referenced in 87.
⟨ Label trend report for custom interval 256 ⟩ Referenced in 255.
⟨ Label weight axis at the left 86a ⟩ Referenced in 87.
⟨ Label weight axis 50a ⟩ Referenced in 41.
⟨ Left Delimiter for Quoted File Name Characters 9d ⟩ Referenced in 137.
⟨ Length of automatically generated passwords 10g ⟩ Referenced in 188.
⟨ List publicly-visible accounts 301 ⟩ Referenced in 169.
⟨ Load first log into memory 375b ⟩ Referenced in 374.
⟨ Load or create monthly log containing imported record 223 ⟩ Referenced in 218, 221, 225.
⟨ Local time zone offset field 372b ⟩ Referenced in 117, 182, 187, 191, 196, 210, 212, 213a, 214, 228, 237, 245, 257, 261, 280, 288a, 294, 295, 299, 301, 306, 307, 309, 320, 327, 331, 341, 349, 357, 358b, 363.
⟨ Log changes to account settings 298 ⟩ Referenced in 297.
⟨ Log failed login attempt in system log 175a ⟩ Referenced in 174b, 175b.
⟨ Log out user: end session 192 ⟩ Referenced in 170a.
⟨ Login-related transactions 170a ⟩ Referenced in 169.
⟨ Look up public account and verify it exists 305 ⟩ Referenced in 304.
⟨ MIME Content-type specification 372a ⟩ Referenced in 163, 195, 239, 249.
⟨ Main account dispatch page 179 ⟩ Referenced in 170b.
⟨ Master encryption key 4d ⟩ Referenced in 135, 136.
⟨ Maximum Expected Weight Variance 464a ⟩ Referenced in 465.
⟨ Maximum File Length 7e ⟩ Referenced in 137.
⟨ Maximum Text Input Field Length 9f ⟩ Referenced in 35a, 118, 121ab, 122, 123ab, 187, 312, 320, 327, 359.
⟨ Maximum line length in feedback E-mail messages 10d ⟩ Referenced in 351, 356.
⟨ Minimum, Maximum, and Sign functions 384 ⟩ Referenced in 19, 69, 373.
⟨ Modify user account request 295 ⟩ Referenced in 170b.
⟨ Monthly Log Weight Range in Kilograms 4e ⟩ Referenced in 4f, 45.
⟨ Monthly Log Weight Range in Pounds 4f ⟩ Referenced in 45.
⟨ Monthly log control panel 203 ⟩ Referenced in 196.
⟨ Monthly log table 447 ⟩ Referenced in 445a.
⟨ Monthly log title and navigation buttons 198 ⟩ Referenced in 196.
⟨ Navigation bar table 449 ⟩ Referenced in 445a.
⟨ New monthly log creation form 210 ⟩ Referenced in 208.
⟨ Obtain list of open accounts 313 ⟩ Referenced in 311.
⟨ Obtain list of open sessions 321 ⟩ Referenced in 320, 323.
⟨ Obtain list of persistent login tokens 328a ⟩ Referenced in 300, 327, 329.
⟨ Obtain list of public accounts 302 ⟩ Referenced in 301.
⟨ Obtain list of years 101b ⟩ Referenced in 100, 101a.
⟨ Obtain trend carry-forward for Excel CSV 61 ⟩ Referenced in 60.
⟨ Open new session and link to user directory 176b ⟩ Referenced in 173.
⟨ Options documentation 428abcd, 429 ⟩ Referenced in 427.
⟨ Output table rows for each interval analysed 255 ⟩ Referenced in 254.
⟨ Output trend analysis report for intervals evaluated 254 ⟩ Referenced in 252a.
⟨ Override diet calculator primary fields from form fields 271 ⟩ Referenced in 262.
⟨ Page title table 445b ⟩ Referenced in 445a.
⟨ Parse CGI arguments 389 ⟩ Referenced in 373.
⟨ Parse Excel CSV date field 222 ⟩ Referenced in 221.
⟨ Parse signed weight value 381b ⟩ Referenced in 373.
⟨ Parse weight value 381a ⟩ Referenced in 373.
⟨ Password and password confirmation fields 122 ⟩ Referenced in 119.
⟨ Path for cookies 11d ⟩ Referenced in 150ab.
⟨ Path to Invoke Sendmail 10a ⟩ Referenced in 128, 354, 355.
⟨ Perform static update of diet calculator 272 ⟩ Referenced in 261.
⟨ Perl directory 7d ⟩ Referenced in 13, 16a, 19, 69, 106, 110, 140, 146, 154, 163, 391, 397a, 407, 414, 419, 431, 433, 434.
⟨ Perl language modes 369b ⟩ Referenced in 13, 16a, 19, 69, 106, 110, 140, 146, 154, 369a, 391, 397a, 407, 414, 419, 431.
⟨ Persistent login manager table 450b ⟩ Referenced in 445a.

⟨ Plot exercise rung information 44 ⟩ Referenced in 41.
⟨ Plot exercise rung on chart image 475 ⟩ Referenced in 474.
⟨ Plot multiple days per pixel 90 ⟩ Referenced in 87.
⟨ Plot multiple pixels per day 88 ⟩ Referenced in 87.
⟨ Plot the diet plan if defined and requested 48a ⟩ Referenced in 41.
⟨ Plot the diet plan on historical chart 80a ⟩ Referenced in 75.
⟨ Plot the updated trend 471 ⟩ Referenced in 470.
⟨ Plot weight and rung data on historical chart 87 ⟩ Referenced in 75.
⟨ Plot weight entries as floats and sinkers 49 ⟩ Referenced in 41.
⟨ Plot weight entry as float or sinker 89 ⟩ Referenced in 88.
⟨ Plot weight trend line on chart 48b ⟩ Referenced in 41.
⟨ Preset diet calculator start and end dates 264b ⟩ Referenced in 261.
⟨ Print command line help information 383 ⟩ Referenced in 373.
⟨ Process XML daily log entry 218 ⟩ Referenced in 217.
⟨ Process administrator account delete 318, 319 ⟩ Referenced in 171.
⟨ Process administrator database purge 316, 317 ⟩ Referenced in 171.
⟨ Process command line options 371a ⟩ Referenced in 163.
⟨ Process custom interval specification, if any 253 ⟩ Referenced in 239, 252a, 280.
⟨ Process database delete 360 ⟩ Referenced in 170a.
⟨ Process database export 239 ⟩ Referenced in 168.
⟨ Process new user account request 289 ⟩ Referenced in 170b.
⟨ Process queued cluster synchronisation transactions 400 ⟩ Referenced in 397a.
⟨ Process user account close 365 ⟩ Referenced in 170a.
⟨ Process user account modification 297 ⟩ Referenced in 170b.
⟨ Production Book Directory 5c ⟩ Referenced in 5e.
⟨ Production CGI Installation Directory 5g ⟩ Referenced in 515.
⟨ Production Executable Installation Directory 6e ⟩ Referenced in 515.
⟨ Production Web Directory 5e ⟩ Referenced in 515.
⟨ Prohibit password reset on read-only account 190a ⟩ Referenced in 188.
⟨ Propagate handheld setting to subsequent forms 288b ⟩ Referenced in 187, 191, 288a, 294.
⟨ Propagate trend through user's monthly logs 374 ⟩ Referenced in 373.
⟨ Propagate trend to next log 376a ⟩ Referenced in 374.
⟨ Provide administrator access to user account 314 ⟩ Referenced in 171.
⟨ Provide browse access to public account 304 ⟩ Referenced in 169.
⟨ Public Name Directory 6i ⟩ Referenced in 156, 157, 158, 159a, 160a, 302.
⟨ Public name settings 126b ⟩ Referenced in 119.
⟨ Quit browsing another account 235a ⟩ Referenced in 169.
⟨ Quote text for inclusion in HTML 413 ⟩ Referenced in 407.
⟨ Read line from persistent object file 390b ⟩ Referenced in 30b, 116a, 144a, 149a, 161b, 379b, 397a.
⟨ Read log if in database or create blank log if it's not 197b ⟩ Referenced in 196, 206, 235b, 236, 250a.
⟨ Recalculate trend carry-forward for all logs for a user 234 ⟩ Referenced in 169.
⟨ Receive log records from the aggregator for global statistics 338 ⟩ Referenced in 331.
⟨ Recover from failure of a cluster synchronisation transaction 402 ⟩ Referenced in 400.
⟨ Reject account close for confirmation code mismatch 366b ⟩ Referenced in 365.
⟨ Reject account close for user name or password mismatch 366a ⟩ Referenced in 365.
⟨ Reject account close if logs remain in the database 367a ⟩ Referenced in 365.
⟨ Reject deletion request when confirmation code fails to match 361b ⟩ Referenced in 360.
⟨ Reject deletion request when user name and password fail to match 361a ⟩ Referenced in 360.
⟨ Reject login: Incorrect password 175b ⟩ Referenced in 174a.
⟨ Reject login: Unknown user name 174b ⟩ Referenced in 173, 174a, 188.
⟨ Reject request if logs remain in database 364a ⟩ Referenced in 363.
⟨ Reject setting query or change from cookie-based login 296 ⟩ Referenced in 295.
⟨ Release Date 3b ⟩ Referenced in 371a, 383, 427, 513.
⟨ Remember Me Directory 11e ⟩ Referenced in 151, 152, 300, 328a, 329.
⟨ Report errors in account modification request and re-issue form 299 ⟩ Referenced in 297.
⟨ Report errors in new account request and re-issue form 294 ⟩ Referenced in 289, 291b.
⟨ Report warnings from static diet calculator update 263b ⟩ Referenced in 261.
⟨ Request historical chart 279 ⟩ Referenced in 169.

⟨ Request invitation codes 306 ⟩ Referenced in 171.
⟨ Request log records from the aggregator and compute global statistics 332 ⟩ Referenced in 331.
⟨ Request paper log forms 245 ⟩ Referenced in 169.
⟨ Reset a user's password 188 ⟩ Referenced in 170b.
⟨ Retrieve active session information 193 ⟩ Referenced in 179, 192, 196, 206, 208, 211, 214, 228, 231, 234, 235ab, 236, 237, 239, 241, 242, 243, 244, 245, 247, 249, 250a, 251, 261, 278, 279, 287, 295, 297, 300, 301, 304, 306, 307, 309, 314, 316, 318, 320, 323, 327, 329, 331, 341, 348, 353, 357, 360, 363, 365, 367c.
⟨ Retrieve user account information 194 ⟩ Referenced in 179, 196, 206, 208, 211, 214, 228, 231, 234, 235ab, 236, 237, 239, 241, 242, 243, 244, 245, 247, 249, 250a, 251, 261, 278, 279, 287, 295, 297, 300, 301, 304, 306, 307, 309, 314, 316, 318, 320, 323, 327, 329, 331, 341, 348, 353, 357, 360, 363, 365.
⟨ Return log items for aggregation from this user 109 ⟩ Referenced in 108.
⟨ Return login request form 172 ⟩ Referenced in 170a.
⟨ Return password reset confirmation page 191 ⟩ Referenced in 188.
⟨ Return time of user's last transaction 379b ⟩ Referenced in 373.
⟨ Right Delimiter for Quoted File Name Characters 9e ⟩ Referenced in 137.
⟨ Sanity check year and month specification 195 ⟩ Referenced in 197b, 234, 250a.
⟨ Save diet calculator settings 278 ⟩ Referenced in 169.
⟨ Save process ID of cluster synchronisation job 398b ⟩ Referenced in 397a.
⟨ Selection of months for paper logs 246a ⟩ Referenced in 245.
⟨ Selection of months to export from database 238a ⟩ Referenced in 237.
⟨ Send E-mail confirming password reset 190b ⟩ Referenced in 188.
⟨ Send a feedback message 348 ⟩ Referenced in 169.
⟨ Send a message from the feedback form 353 ⟩ Referenced in 169.
⟨ Send copy to the submitter 355 ⟩ Referenced in 353.
⟨ Send mail to feedback address 354 ⟩ Referenced in 353.
⟨ Session Directory 6g ⟩ Referenced in 176ab, 192, 193, 235a, 304, 314, 321, 322, 323, 326, 365, 380.
⟨ Set handler for termination signals 164a ⟩ Referenced in 163.
⟨ Set monthly log entry from Excel CSV record 224 ⟩ Referenced in 221.
⟨ Set monthly log entry from Palm CSV record 226a ⟩ Referenced in 225.
⟨ Set monthly log property variables 199a ⟩ Referenced in 196.
⟨ Set primary diet calculator fields from user object 262 ⟩ Referenced in 261.
⟨ Set variables to default to previous request settings 283, 284, 285a ⟩ Referenced in 237, 245, 247, 252a, 261, 280.
⟨ Show build number and date 184 ⟩ Referenced in 179.
⟨ Show feedback message in reply page 356 ⟩ Referenced in 350, 353.
⟨ Show preview of message being composed 350 ⟩ Referenced in 348.
⟨ Show user name and account being browsed 180a ⟩ Referenced in 179.
⟨ Sign in and account management tables 446 ⟩ Referenced in 445a.
⟨ Site home URL 11f ⟩ Referenced in 11g, 190b, 233, 409, 427.
⟨ Source distribution 517 ⟩ Referenced in 515.
⟨ Source installation 521 ⟩ Referenced in 515.
⟨ Specify Content-type for PNG image 250b ⟩ Referenced in 250a, 287.
⟨ Standard navigation bar functions 180b ⟩ Referenced in 179.
⟨ Static change to daily balance 273b ⟩ Referenced in 272.
⟨ Static change to desired weight change 274c ⟩ Referenced in 272.
⟨ Static change to diet duration in months 276a ⟩ Referenced in 272.
⟨ Static change to diet duration in weeks 275b ⟩ Referenced in 272.
⟨ Static change to end date 277 ⟩ Referenced in 272.
⟨ Static change to energy unit 273a ⟩ Referenced in 272.
⟨ Static change to goal weight 274b ⟩ Referenced in 272.
⟨ Static change to initial weight 274a ⟩ Referenced in 272.
⟨ Static change to start date 276b ⟩ Referenced in 272.
⟨ Static change to weight change per week 275a ⟩ Referenced in 272.
⟨ Static change to weight unit 273c ⟩ Referenced in 272.
⟨ Store settings for user account 292 ⟩ Referenced in 291b, 297.
⟨ Supply default values for undefined variables 390a ⟩ Referenced in 373.
⟨ Synthetic data end date 345 ⟩ Referenced in 343.
⟨ Synthetic data field selection 346a ⟩ Referenced in 343.
⟨ Synthetic data start and end values 346b ⟩ Referenced in 343.

⟨ Synthetic data start date 344 ⟩ Referenced in 343.
⟨ Table of perturbation functions 347 ⟩ Referenced in 343.
⟨ Template 368c ⟩ Not referenced.
⟨ Test domain valid for E-mail 388b ⟩ Referenced in 373.
⟨ Test if month is the current month 386b ⟩ Referenced in 373.
⟨ Test whether user has a session active 380 ⟩ Referenced in 373.
⟨ Testing 519a ⟩ Referenced in 515.
⟨ Trend analysis table 448 ⟩ Referenced in 445a.
⟨ Trend analysis 251 ⟩ Referenced in 169.
⟨ TrueType Font Directory 6b ⟩ Referenced in 83, 387.
⟨ URL to invoke this program 12a ⟩ Referenced in 12b, 118, 179, 180b, 181, 182, 183, 189, 190ab, 195, 196, 198, 209, 211, 212, 213a, 233, 234, 239, 249, 251, 281b, 297, 299, 304, 305, 314, 315, 316, 317, 318, 319, 324, 325, 326, 330, 353, 360, 361ab, 362b, 364a, 365, 366ab, 367ac, 410a, 455c.
⟨ Undraw trend values starting at this day 466 ⟩ Referenced in 464b.
⟨ Unix time to Julian day and fraction 424b ⟩ Referenced in 419.
⟨ Unix time to civil date and time 425a ⟩ Referenced in 419.
⟨ Update Web page badge 207b ⟩ Referenced in 207a, 231.
⟨ Update Web page status badge 231, 232, 233 ⟩ Referenced in 169.
⟨ Update global statistics overall trend analysis 339 ⟩ Referenced in 338.
⟨ Update last login and transaction time 177a ⟩ Referenced in 173.
⟨ Update monthly log 206 ⟩ Referenced in 169.
⟨ Update persistent login state 178 ⟩ Referenced in 173.
⟨ Update the mean and most recent body mass index 473 ⟩ Referenced in 464b.
⟨ Update the trend and variance for this and subsequent days 470 ⟩ Referenced in 464b.
⟨ Update time of user's last transaction 379a ⟩ Referenced in 373.
⟨ Update user account information 293 ⟩ Referenced in 188, 231, 278, 291b, 297.
⟨ User login name text field 121a ⟩ Referenced in 119.
⟨ User's full name optional text fields 123b ⟩ Referenced in 119.
⟨ Users Directory 6h ⟩ Referenced in 78, 102, 104, 105a, 108, 109, 132, 133, 173, 174a, 176ab, 177a, 188, 192, 194, 197b, 201, 207ab, 223, 226b, 231, 241, 242, 243, 244, 249, 290a, 291b, 293, 303, 311, 313, 314, 317, 319, 351, 360, 365, 374, 375b, 376a, 377a, 378, 379ab, 380, 431, 433.
⟨ Utility functions for regular session 181 ⟩ Referenced in 179.
⟨ Utility functions 373 ⟩ Referenced in 163, 433.
⟨ Validate E-mail address agrees with specification in reset request 189 ⟩ Referenced in 188.
⟨ Validate administrator password 324 ⟩ Referenced in 316, 318, 323, 329.
⟨ Validate beta test invitation code 290b ⟩ Referenced in 289.
⟨ Validate option specifications 371b ⟩ Referenced in 163.
⟨ Validate specified session 326 ⟩ Referenced in 323.
⟨ Validate user login request 173 ⟩ Referenced in 170a.
⟨ Validate user name and password 174a ⟩ Referenced in 173.
⟨ Validate user name for new account 290a ⟩ Referenced in 289.
⟨ Verify that user has administrator privilege 315 ⟩ Referenced in 306, 307, 309, 314, 316, 318, 320, 323, 327, 329, 331, 341.
⟨ Version 3a ⟩ Referenced in 371a, 383, 427, 517, 521.
⟨ Warn user about consequences of closing account 364b ⟩ Referenced in 363.
⟨ Web Directory 5d ⟩ Referenced in 515.
⟨ Web Document Home 5a ⟩ Referenced in 163, 196, 233, 440.
⟨ Weight and energy unit radio buttons 125 ⟩ Referenced in 119.
⟨ Wrap long lines onto multiple lines 382 ⟩ Referenced in 373.
⟨ Write HTML table footer 35b ⟩ Referenced in 31.
⟨ Write HTML table header 32a ⟩ Referenced in 31.
⟨ Write XHTML epilogue 412 ⟩ Referenced in 407.
⟨ Write XHTML prologue 408ab, 409 ⟩ Referenced in 407.
⟨ Write back all items in the cache 105a ⟩ Referenced in 97.
⟨ Write back logs modified by database import 226b ⟩ Referenced in 215b.
⟨ Write modified log back to database 377a ⟩ Referenced in 375b, 376a.
⟨ Write updated log item back to database 207a ⟩ Referenced in 206.

29.3 Identifiers

Sections which define identifiers are underlined.

`$chartSizes`: 285a, 286b, 370b.
`$feedback_categories`: 350, 353, 370b.
`$monthNames`: 196, 198, 209, 370b.
`$testmode`: 370b, 371a.
`$verbose`: 370b, 371a, 385a, 397a, 399b, 401, 402, 405b.
`addPoint`: 16a, 17b, 24, 71, 74b, 334, 339.
`analyseTrend`: 73, 74a, 92, 94b, 202, 254.
`append_history`: 175b, 176a, 177b, 188, 189, 192, 207a, 216, 233, 234, 278, 298, 300, 315, 317, 323, 324, 329, 353, 360, 378.
`assignPublicName`: 158, 292.
`bnd`: 34a, 37c, 58, 59, 60, 62.
`bodyMassIndex`: 25, 199a.
`calc_calorie_balance`: 112, 114, 115, 116a, 131, 134, 242, 262, 265b, 270b, 271, 273ab, 275ab, 276a, 277, 278, 476b, 477, 478, 479b, 480a, 482b, 483ab, 485a.
`calc_end_date`: 262, 264ab, 270b, 476b, 477, 478, 485a.
`calc_energy_unit,`: 476b.
`calc_goal_weight`: 112, 114, 115, 116a, 131, 134, 242, 261, 262, 266b, 270b, 271, 273c, 274bc, 278, 476b, 477, 478, 481ab, 482a.
`calc_months`: 262, 268, 270b, 276a, 476b, 477, 478, 483b.
`calc_months,`: 262, 476b.
`calc_start_date`: 112, 114, 115, 116a, 131, 134, 242, 262, 264ab, 270b, 271, 276b, 277, 278, 476b, 477, 478, 484a, 485a.
`calc_start_weight`: 112, 114, 115, 116a, 131, 134, 242, 261, 262, 266a, 270b, 271, 273c, 274ac, 278, 476b, 477, 478, 480b, 481a, 482a.
`calc_weeks`: 262, 268, 270b, 275b, 476b, 477, 478, 483a.
`calc_weight_change,`: 261, 262, 476b, 478.
`calc_weight_unit,`: 261, 262, 476b.
`calc_weight_week`: 261, 262, 267b, 270b, 275a, 476b, 477, 478, 482b.
`CALORIES_PER_ENERGY_UNIT`: 20, 92, 94b, 202, 255, 334, 335, 454, 477, 478.
`CALORIES_PER_WEIGHT_UNIT`: 19, 20, 92, 94b, 134, 202, 255, 270b, 275ab, 276a, 277, 334, 335, 454, 472, 478, 482b, 483ab, 485a.
`canonicalNumber`: 120, 138a, 139, 488b, 489, 490, 491.
`canonicalWeight`: 36b, 37a, 377b.
`changeComment`: 35a, 476a.
`changeRung`: 34a, 474.
`changeWeight`: 33, 464b.
`change_calc_calorie_balance`: 265b, 479b.
`change_calc_energy_unit`: 265b, 480a.
`change_calc_goal_weight`: 266b, 481b.
`change_calc_plot_plan`: 270a, 485b.
`change_calc_start_weight`: 266a, 480b.
`change_calc_weight_change`: 267a, 482a.
`change_calc_weight_unit`: 266a, 481a.
`change_calc_weight_week`: 267b, 482b.
`change_from_d`: 259, 484a.
`change_from_date`: 484a.
`change_from_m`: 259, 484a.
`change_from_y`: 259, 484a.
`change_to_d`: 260, 485a.
`change_to_date`: 485a.
`change_to_m`: 260, 485a.
`change_to_y`: 260, 485a.
`characterFrequency`: 494, 496.
`checkCookieSignature`: 146, 152, 153a.
`checkPasswordMatch`: 122, 497a.

checkSecure: 172, 174b, 175b, 186, 455c.
civil_time_to_jd: 419, 423b.
computeChartScale: 40, 199a.
computeTrend: 24, 29a, 202, 376a.
convertWeight: 36b, 51.
countChange: 456a, 464b, 474, 476a, 478, 485b, 493b.
decodeComments: 30b, 67.
decodeEncryptedUserID: 136, 431, 433.
deletePublicName: 159b, 292.
describe: 23, 97, 114, 142, 148a, 160b, 205, 300, 433.
determineTimeZoneOffset: 455b, 488a.
dietCalcRecalculate: 478, 479b, 480ab, 481ab, 482ab, 483ab, 484a, 485a.
dietPlanLimits: 134, 199a, 250a, 287.
dnz: 27, 37b, 41, 44, 45, 49.
do_command: 105a, 207b, 319, 362a, 365, 374, 385a.
drawBadgeImage: 93a, 105a, 207b, 374, 433.
drawChart: 74a, 75, 77, 287.
drawText: 43b, 44, 50a, 84, 85, 86ab, 91, 92, 94c, 95, 96, 387.
editWeight: 36a, 38c, 50a, 86a, 95, 255, 261, 457, 460, 461, 469, 470, 478.
encodeComments: 27, 66.
encodeCSV: 13, 15, 58, 242.
encodeDomainName: 289, 297, 388a.
ENERGY_CALORIE: 19, 20, 112, 262, 270b, 275a, 277, 278.
ENERGY_CONVERSION: 19, 20, 262, 270b, 273a, 275a, 277, 278, 454, 480a.
ENERGY_KILOJOULE: 19, 20.
ENERGY_UNITS: 20, 92, 114, 130, 202, 242, 254.
enumerateMonths: 100, 101a, 132, 201, 209, 238a, 241, 242, 243, 244, 258a, 303, 311, 317, 319, 357, 360, 363, 365, 374.
enumerateYears: 101b, 133, 208, 237, 251, 264a, 279.
etime: 385b.
expandAbbreviatedWeight: 460, 464b.
expireCookie: 150b, 178.
exportCSV: 58, 235b, 242.
exportDietPlanXML: 131, 241.
exportExcelCSV: 60, 244.
exportHDReadCSV: 59, 243.
exportPreferencesXML: 130, 241.
exportUserInformationXML: 129.
exportXML: 62, 236, 241.
externalLinks: 455b, 487.
findPublicName: 159a, 194, 305.
firstDay: 94a, 101a, 202, 240.
firstDayOfInterval: 94b, 103, 202, 252b, 256, 282.
fitAddPoint: 459b, 472.
fitSlope: 16a, 18a, 24, 73, 334, 340, 459c, 472.
fitStart: 459a, 472.
fit_n: 459a, 459bc.
fit_s1: 459a, 459bc.
fit_s2: 459a, 459bc.
fit_s3: 459a, 459bc.
fit_s4: 459a, 459bc.
fixo: 33, 38b.
floor: 420b, 421b, 422, 424a, 457, 460, 461, 466, 467, 471, 474, 475, 489, 491.
fractionFlagged: 26, 199a.
generateCookie: 150a, 151.
generateCookieID: 147, 153b.
generateEncryptedUserID: 135, 232, 433.
generateRandomName: 156, 157, 158.

generateSessionID: 141, 145b.
generateUniqueName: 156, 157.
generateXMLepilogue: 236, 241, 414, 415b.
generateXMLprologue: 236, 241, 414, 415a.
generate_XHTML_navigation_bar: 179, 196, 208, 211, 214, 228, 231, 234, 237, 239, 245, 249, 251, 261, 279, 295, 296, 297, 299, 300, 301, 304, 305, 306, 307, 309, 314, 315, 316, 318, 320, 324, 325, 326, 327, 330, 331, 341, 348, 353, 357, 360, 363, 365, 407, 410a.
getDays: 71, 74b, 88, 90.
get_selected_date: 477, 484a, 484b, 485a.
GREGORIAN_EPOCH: 419, 420a, 421b, 422.
gregorian_to_jd: 31, 47, 60, 74a, 77, 83, 84, 85, 94ab, 103, 109, 150b, 202, 240, 247, 248, 252b, 253, 256, 276b, 277, 278, 282, 332, 386b, 419, 421b, 422.
height_changed_cm: 124, 489.
height_changed_ft: 124, 490.
height_changed_in: 124, 491.
html: 11h, 19, 110, 369a, 372a, 407, 408b, 412, 513, 514, 517, 519ab, 520.
importCSV: 57, 226a.
in: 4f, 14, 15, 20, 23, 28abc, 29ab, 30ab, 45, 49, 52, 69, 73, 75, 85, 86b, 87, 89, 97, 100, 101a, 114, 116a, 119, 128, 137, 144a, 145a, 149a, 159a, 161b, 163, 174b, 175b, 187, 188, 190a, 196, 197ab, 206, 211, 213a, 232, 235b, 236, 239, 248, 250a, 252a, 259, 260, 261, 263b, 264a, 270a, 272, 275b, 276a, 280, 281b, 289, 291b, 294, 295, 296, 297, 299, 300, 301, 309, 319, 323, 330, 331, 333, 334, 341, 350, 351, 353, 356, 357, 359, 361b, 363, 364ab, 365, 366b, 367a, 371a, 373, 375b, 376a, 379b, 389, 390b, 391, 401, 406a, 407, 427, 435, 445a, 454, 460, 464b, 467, 469, 474, 475, 476a, 487, 495, 500, 501, 504, 509, 513.
initialiseDocument: 409, 455b, 513.
isCurrentMonth: 386b.
is_user_session_open: 317, 319, 380.
J1970: 419, 420a, 424bc.
jd_to_civil_time: 419, 424a, 425ab, 426ab.
jd_to_gregorian: 47, 71, 77, 97, 103, 108, 246a, 264ab, 332, 419, 422, 425ab, 426ab.
jd_to_old_cookie_date: 150ab, 419, 426b.
jd_to_RFC_3339_date: 338, 419, 426a.
jd_to_RFC_822_date: 419, 425b.
jd_to_unix_time: 271, 276b, 277, 278, 419, 424c.
jd_to_weekday: 31, 60, 248, 419, 423a, 426b.
Julian: 19, 47, 69, 106, 110, 146, 248, 369a, 386b, 419, 420a, 519a.
lastDay: 94a, 100, 202, 240, 262.
last_transaction_time: 303, 313, 379b.
leap_gregorian: 419, 421a, 421b, 422.
leaveDocument: 196, 198, 261, 456b.
load: 28a, 28bc, 29ab, 30a, 78, 93b, 102, 104, 109, 116a, 144a, 149a, 152, 159a, 161b, 173, 174a, 188, 193, 194, 197b, 201, 223, 241, 242, 243, 244, 302, 303, 311, 313, 317, 319, 321, 322, 328a, 375b, 376a, 433.
loadDietCalcFields: 261, 477, 479b, 480ab, 481ab, 482ab, 483ab.
load_active_session: 140, 145a, 176a, 380.
localiseDecimal: 92, 138b, 139, 202.
localiseNumber: 120, 139.
login: 113, 114, 118, 119, 129, 142, 143, 145b, 151, 153b, 160b, 170a, 171, 173, 174ab, 175b, 177a, 187, 188, 189, 190a, 191, 192, 265a, 288a, 291b, 294, 295, 300, 327, 328a, 329, 330, 359, 445a, 446, 453, 501, 507.
login_form: 117, 172, 174b, 175b, 186.
max: 17ab, 18b, 45, 47, 78, 79, 264a, 281b, 307, 384, 469, 494.
min: 17ab, 18b, 45, 47, 78, 79, 84, 197a, 210, 249, 281b, 307, 384, 385b, 401, 402, 423b, 494.
minMaxMean: 18b, 73.
mod: 420b, 422.
monthdays: 23, 24, 25, 26, 27, 29a, 31, 41, 43ab, 44, 45, 47, 48a, 49, 50b, 57, 58, 59, 60, 61, 62, 63, 66, 78, 100, 101a, 103, 109, 199a, 201, 248, 375b, 376a, 377b.
ndb: 390a.
ndz: 390a.
new_account_form: 119, 288a, 294, 295, 299.
nextMonth: 64, 78, 104, 198.

parseCSV: 13, 14, 57.
parseSignedWeight: 274c, 275a, 381b, 458b, 477.
parseWeight: 274ab, 381a, 381b, 458a, 458b, 460, 464b, 465, 467, 469, 470, 472, 473, 477, 481a.
parse_cgi_arguments: 163, 389.
passwordStrength: 494.
plotChart: 41, 250a, 462, 463.
previousMonth: 64, 103, 198.
print_command_line_help: 163, 371a, 383.
propagate_trend: 207a, 234, 342, 374.
psLog10: 494, 497b.
psLog2: 494, 497b.
quoteHTML: 35a, 117, 120, 126b, 179, 180a, 185, 187, 189, 190a, 191, 195, 199a, 205, 208, 218, 220, 289, 297, 303, 305, 308, 311, 314, 317, 319, 322, 323, 328b, 335, 349, 351, 356, 363, 407, 413.
quoteHTMLFile: 205, 407, 413.
quoteUserName: 108, 110, 132, 133, 135, 137, 157, 158, 159a, 160a, 173, 174a, 188, 193, 194, 290a, 301, 303, 304, 314, 317, 319, 323, 329, 374, 380, 433.
quoteXML: 129, 130, 414, 416a, 416b.
replaceText: 469, 470, 472, 473, 492b, 493b.
resetFocus: 464b, 465, 474, 480b, 481b, 483ab, 485a, 486b.
resetPassword: 127a, 127b, 188.
retrieve: 108, 332.
Round: 46, 384, 420b.
RUNG_MAX: 20, 40, 44, 54, 76.
save: 27, 105a, 115, 143, 148b, 151, 158, 161a, 176b, 207a, 226b, 235a, 293, 304, 314, 377a, 456b.
save_active_session: 144c, 176b.
sendMail: 128, 190b.
set_date_selection: 478, 479a.
set_dispunit: 119, 492a.
set_logunit: 119, 492a.
sgn: 36a, 49, 89, 134, 381b, 384, 457, 458b.
showPasswordStrength: 122, 494.
signCookie: 149c, 150a.
start: 16ab, 17a, 47, 74a, 77, 79, 131, 238a, 246a, 247, 253, 257, 258a, 261, 262, 264b, 269a, 272, 277, 281ab, 287, 343, 386b, 469, 485a, 503, 509.
storeCookie: 146, 151, 178.
syntheticData: 97, 99, 342.
testCookiePresent: 146, 152, 172, 178.
textXML: 62, 414, 416b.
timeXML: 62, 129, 131, 241, 242, 362a, 414, 417.
toHex: 386a.
toHTML: 31, 196.
unix_time_to_civil_date_time: 167, 195, 197a, 210, 218, 221, 225, 249, 419, 425a.
unix_time_to_jd: 134, 150a, 246a, 264ab, 332, 386b, 419, 424b, 425a.
unsavedChanges: 203, 270a, 456a, 456b.
updateFlag: 34b, 493b.
updateFromCGI: 50b, 206.
updateVariance: 470, 493a.
update_last_transaction: 177a, 207a, 208, 211, 216, 233, 234, 249, 250a, 251, 261, 278, 279, 287, 295, 296, 297, 300, 379a.
U_MINUS_SIGN: 455a, 493a.
validateFeedback: 349, 486a.
validMailDomain: 289, 297, 388b.
var: 33, 39, 406b, 454, 455a, 456a, 457, 458ab, 459a, 460, 461, 462, 463, 464b, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476ab, 479a, 480a, 481a, 484b, 487, 488b, 489, 490, 491, 492ab, 493ab, 494, 496, 497a, 498.
verbose: 21, 22, 65, 370b, 371a, 383, 385a, 397a, 399b, 401, 402, 405b, 428d, 519a.
WEEKDAY_NAMES: 19, 32b, 60, 69, 248, 419, 420a, 426b.
WEIGHT_KILOGRAM: 19, 20, 21, 25, 45, 47, 78, 79, 112, 134, 217, 221, 225, 262, 278, 454, 473.
WEIGHT_POUND: 19, 20, 95, 217, 221.

WEIGHT_STONE: 19, 20, 36a, 43a, 46, 51, 52, 80b, 95, 217, 221, 225, 271, 278, 381a, 454, 457, 458a, 460.
WEIGHT_UNITS: 20, 23, 59, 60, 62, 114, 130, 242.
wgt: 33, 38c.
wrapText: 351, 356, 382.
write_XHTML_epilogue: 172, 174b, 175b, 179, 186, 187, 189, 190a, 191, 195, 196, 208, 211, 215b, 230, 233, 234, 237, 239, 245, 247, 249, 251, 261, 279, 288a, 294, 295, 296, 297, 299, 300, 301, 304, 305, 306, 308, 312, 314, 315, 316, 317, 318, 319, 320, 324, 325, 326, 327, 330, 331, 341, 348, 353, 357, 360, 363, 365, 367c, 407, 412.
write_XHTML_prologue: 172, 174b, 175b, 179, 186, 187, 189, 190a, 191, 195, 196, 208, 211, 214, 228, 231, 234, 237, 239, 245, 247, 249, 251, 261, 279, 288a, 294, 295, 296, 297, 299, 300, 301, 304, 305, 306, 307, 309, 314, 315, 316, 318, 320, 324, 325, 326, 327, 330, 331, 341, 348, 353, 357, 360, 363, 365, 367c, 407, 408a.
xml: 19, 110, 215b, 236, 237, 239, 241, 358b, 369a, 408b, 414, 415a, 519a.
znd: 31, 33, 38a, 51, 54, 55, 58, 59, 60, 61.

Chapter 30

Feature Requests

2007 April 30

Add items to the Diet Calculator to display the "band" and the "brick wall" limits around the goal weight. (Suggested by Rob Campbell.)

2007 May 2

Custom colour selection for chart components. (Suggested by Bruce Lokeinsky.)

2007 May 4

In the current day's log, only display days up to and including the current day. (Suggested by Bruce Lokeinsky.)

2007 May 19

Check boxes in the Chart Workshop to select which components are charted.

Perhaps it would make sense (in particular in conjunction with the cookies for "remember me") to define an alias on the HTTP server so that the application could be accessed with a URL like `http://www.fourmilab.ch/hdo` instead of all of the `cgi-bin` rigmarole.

Option to suppress monthly header records in CSV data exports. (Suggested by Robert Ewing.)

Ability to configure user-defined numeric and string fields to be added to the monthly log form. These fields will be exported, and numeric fields may be plotted in charts.

Chapter 31

Development Log

2006 January 21

Created Nuweb `hdiet.w`.

2006 March 19

Added a `hdiet.js` target for the common JavaScript support and included the `externalLinks()` function to handle window targets in XHTML 1.0 Strict compliant documents.

Implemented a `trendfit` module to perform linear regression trend fitting and used it within the `monthlog::computeTrend` method, which now returns the slope of the fit trend.

Added a `monthlog::fractionFlagged` method which returns the fraction (between 0 and 1) of days in a month which are flagged.

Added a line above the log summarising the weekly gain/loss, daily calorie excess/deficit, and percent of days flagged. The latter is shown only if at least one day is flagged.

Added a title showing the month and year of the log, nicely styled as in the `hdread` output.

2006 April 4

Completed the first cut implementation of chart generation from monthly logs with the `monthlog::plotChart` method.

2006 April 5

Cleaned up the `monthlog::toHTML` method, breaking the monolithic per-day loop up into scraps which generate each column (or column group) of the table.

2006 April 6

Implemented the `monthlog::save` method, which saves a monthly log item in a compact format that preserves all archival information.

Implemented the `monthlog::load` method, which loads a log in the format written by `save`.

Implemented the initial version of the `user` module, which manages user accounts. The `quoteUserName` function handles the problem of turning arbitrary UTF-8 user login names into file names acceptable for a Unix file system. We assume the underlying file system distinguishes upper and lower case letters, but do not require it to handle Latin-1 or Unicode characters. The user name transformation is reversible, but we do not presently rely on this.

2006 April 7

Implemented the initial version of the `session` object. This object contains information for an active session (logged-in user). The `generateSessionID` function creates a unique session ID from a UTF-8 login name by appending 16 pseudorandom bytes and then forming the SHA1 signature of the value. The session ID is embedded in documents sent back and forth to the user over the course of the session.

2006 May 21

Changed the code which writes the `test.html` document to create a UTF-8 document. This will permit arbitrary Unicode in text fields. The `charset` declaration in the document header was changed accordingly.

If the monthly log contains editable fields, `monthlog::toHTML` now wraps the log in a form and generates "Update" and "Reset" buttons at the end of the table which allow submitting the form to the server or resetting all fields to their original values.

2006 May 22

Added code to the test jig to bulk convert all the CSV month databases in a directory created with `hdread` into `monthlog::save` files in another directory (both hard-coded for the moment—this *is* the test jig!). The log unit, last modification time, and trend carry-forward are parsed from the "`StartTrend`" line in the CSV files. Something like this may eventually find its way into the online application, but for the nonce it's a handy way to populate test databases prior to any data import facility's existing in the online edition.

Added a new "`administrator`" field to the `user` object which indicates whether the user has administrative privileges. At the moment this is a simple 1 or 0 Boolean, but it could be extended to a numeric privilege level should the need arise.

2006 May 23

Added `save` and `load` methods to the `session` object to save and restore sessions to the active session database directory.

Added fields to support the diet calculator, current exercise rung persistence, and energy unit selection to the `user` object. Removed the last login field, as we'll keep this in a separate file in the user directory to avoid the risk of rewriting the `user` database item for every session.

2006 May 26

Integrated the new uniform HTML title into a subroutine that all HTML generation may now call: `write_XHTML_prologue`. This routine takes arguments which specify the file handle to which the prologue should be written, the base URL for links back to the site, and the specific page subtitle to be included in the page title.

Modified the test CGI log generator to use the new prologue generator in order to test it.

To make things symmetrical, added a `write_XHTML_epilogue`, taking file handle and base URL arguments, which generates the standard HTML file epilogue.

2006 May 27

Created a new package, `html` to contain common subroutines used to generate HTML in other packages. The first member function is `quoteHTML`. formerly in `monthlog`.

Added a new `new_account_form` method to the `user` object. This generates the form the user fills in to create a new account. Fields are filled in with values stored in the parent `user` object, permitting defaults to be pre-specified simply by setting the appropriate fields.

Moved the `write_XHTML_prologue` and `write_XHTML_epilogue` to the `html` object so other packages can use them, should that eventually make sense. They were already entirely free of references to context in the main program.

2006 May 28

Modified `quoteUserName` in the `user` object to handle user names which, when quoted, exceed the host system's maximum file name length. If the quoted name is too long, it is truncated to the maximum length less 40 characters and the 40 hexadecimal character SHA1 digest of the entire name is appended to the end.

Added `save` and `load` methods to the `user` object which save and restore user information in an already open text file.

2006 May 29

Added `save_active_session` and `load_active_session` methods to the `session` object to save and restore the active session file which, kept in the user directory, provides a back-link to the current session when a user is logged in. Note that while `save_active_session` is a method of `session`, `load_active_session` is simply a subroutine which reads an active session file and returns its ID.

2006 June 2

Moved the current test page generation in the CGI request processing to a separate section to clean up the main loop for addition of the actual query dispatching logic.

2006 June 3

Added logic to create the session, back-link from the user directory to the session, last login indication, and last transaction time when a new login is validated. If a session is already open for this user, it is automatically closed.

Added a corresponding logout mechanism which cleans up all the active session information.

2006 June 4

Added a "chart" query to generate a monthly chart. This involved shuffling some code in the main loop, since the chart must be generated before we send the usual "`Content-type: text/html` and set `STDOUT` to UTF-8.

Added an image tag to the monthly log output to embed the chart for that month into the log document. Fixed a few XHTML validation nits introduced in yesterday's additions to the log document.

Swatted another swarm of bugs in `parse_cgi_arguments`, which I shall surely still be fixing when the protons in my body are rent asunder by the Big Rip. This time it had to do with UTF-8 arguments which are bytewise encoded with "`%XX`" escapes, which must be decoded back to the binary codes, and the resulting string then converted to the native wide character encoding with `Encode::decode_utf8`. Failure to do this broke the new code to apply user changes in UTF-8 comment fields. I modified the code so that it should support UTF-8 in both argument names and values, but anybody who indulges in UTF-8 argument names entirely deserves whatever surprises may lurk there. I also fixed an eccentric and ill-considered attempt to escape quotes in CGI arguments. This shouldn't be necessary, as they should already be encoded as hex.

Completed the initial implementation of the `monthlog::updateFromCGI` method, which applies changes made by a user in a form containing a monthly log table in the format written by `monthlog::toHTML`. The handling of omitted fields, canonical value comparison, and deletion of existing fields is rather intricate, and I'm sure some problems remain to be discovered, but detailed testing will remain difficult until we actually apply the changes to the log, which will be implemented in the near future. The code *does not* presently handle unit conversion when the log and display units are different; this isn't a big thing, but I'll defer it until enough of the related infrastructure is in place that it isn't hideously painful to test.

Broke up the CSS definition into logical pieces. Documentation is still sketchy, and all the table styles are still in one big messy box.

2006 June 5

Implemented proper handling for the case where the user submits an invalid session ID (usually due to a session having timed out or the user's having logged out from another window). The login page is re-issued along with a diagnostic indicating that the session has been terminated. This is complicated by the fact that the code which validates the session ID may be called from request handlers such as those which generate charts, which are invoked before we've output the "`Content-type`" for an HTML result, as the login page expects. A little state variable directs traffic to avoid collision in this circumstance.

Added `convertWeight` and `canonicalWeight` functions in the `monthlog` object to transform weight values from one unit to another and express weight values in canonical form.

Implemented a `propagate_trend` function which propagates the trend carry-forward from a specified month's log (or throughout the database if no month is specified), and a new "update_trend" request which uses this function to recompute all the trend carry-forwards in the database for a user.

2006 June 6

Modified `propagate_trend` to correctly handle the case where logs in the database use different weight units. The trend is now properly converted to the weight unit used in each log before being saved in the log.

Added the ability to re-canonicalise weight entries in logs when processing them with `propagate_trend`. Note that the first log processed, which is not usually modified by trend propagation, will be updated if weight canonicalisation is requested.

Added the ability to specify the first month of the logs for trend propagation with an "m" argument in the "update_trend" request and force re-canonicalisation of weight entries with a "canon" argument.

Fixed the "log" request to force the "m" argument to "now" if no such argument is specified. Since the case of returning to the current month's log is so common, this allows abbreviating the request in URLs for this destination.

Replaced the code scattered all over the place which appended records to the user history log with calls to a new subroutine, `append_history`, which permits the caller to append optional fields to the standard contents of the log item. This provides a handy place to document the format of the log items.

Made `update_last_transaction`, which updates the time of the last transaction processed for a user, into a subroutine which can be called from any code which knows the file name encoded user name.

Completed the initial implementation of the "update_log" transaction. This now writes the modified log back to the database if any changes were made, appends a history item recording the month changes and the number of changes made by column, updates the last transaction time, and, if any weight entries were changed and this is not the current month, propagates trend changes to subsequent months. The user is returned to the log page following the updates so that they can be reviewed.

2006 June 7

Laid in infrastructure for new account creation: handling of the "new_account" request, validation of the request arguments, and re-issuing of the request form with valid arguments pre-specified in case of error. We currently validate that the account name does not already exist, but do not actually create the new account; this will be deferred until we decide on how account creation will be validated.

Integrated the CPAN `IDNA::Punycode` module (version 0.03) in our `HDiet` library, redesignating the module `HDiet::Util::IDNA::Punycode`. This module is sufficiently obscure that few potential users of this code will have it installed, and it's hard to justify the bother of installing it server-wide for a single, rather obscure, application.

2006 June 8

Added a new `encodeDomainName` function which transforms a fully qualified domain name piecewise to its RFC 3490 "punycode" representation (if the argument is a numeric IP address or pure ASCII name, no change is made and no harm is done). This is now used in verification of the domain name specified in the user's E-mail address, permitting the use of international domain names in this field.

Cleaned up `user:quoteUserName` to make all the testing for permissible characters, special case quoting of spaces, and delimiters for characters quoted as hexadecimal global macros defined at the start of the program. This allows the administrator to make their own trade-off between easy-to-type user directory names versus those which resemble what the user actually types to log in.

Broke up the `user::new_account_form` method, which had grown to unwieldy length by literate programming standards, into functional chunks. It is really cool to be able to specify pieces of a Perl "here" document with Nuweb!

2006 June 9

Added fields to the new account creation form (`user::new_account_form`) which invite the user to specify their height (in either centimetres, feet and inches, or just inches). This will be used for Body Mass Index computations. JavaScript validation of the fields propagates changes in one set of units to another, but ultimately it's up to the server to deal with whatever nonsense is entered.

Added a `monthlog::bodyMassIndex` method to compute the notorious BMI value. It called with an argument from the `user` object which gives the user's height in centimetres and an optional second argument which selects either the mean for the month (zero, or omitted), the most recent value (negative), or a specific day given by number.

2006 June 10

Changed the name of the user account information (`.hdu`) file from that of the file name encoded user name to the constant `UserAccount.hdu`. Since these files exist with the user's directory, there is no need for their names to also encode the user name, and it's just one more fussy thing to enter when the administrator wants to poke around in the file structure.

Implemented the "backup" request, which allows the user to download a Zipped archive containing all their monthly logs. This is done by invoking the "`zip`" program via the `system` function, directing the output to standard output. The default file name is "`hackdiet_log_backup_`*YYYY-MM-DD*`.zip`", based on the current date in UTC.

Fixed a bug where the generation of a monthly chart by the "chart" request erroneously had the HTML for an undefined request appended to the end of the PNG file. All binary result handlers should `exit(0)` upon successful completion to avoid falling into the regular HTML request dispatcher.

Added a `monthlog::exportCSV` method to export a monthly log in a format essentially compatible with `hdread` (the only difference is that Unicode characters, which `hdread` doesn't handle, are encoded as Perl-like escapes as defined by our `hdCSV` package). This is used by a new "csvout" request which replies with the monthly log given by the "m" CGI argument encoded as CSV. Note that the CSV is log is returned with a `charset` of "`iso-8859-1`", as Latin-1 graphics characters are not quoted in our CSV output, but all characters not present in Latin-1 are quoted.

2006 June 11

Added code to create a new account when everything validates in the new account request form. This drops the user back to the login form for the newly-created account, which I twiddled so the user name for the new account is pre-entered in the field in the login form. The password must, of course, be re-specified.

Implemented the `user::sendMail` method which will be used to send mail to users for validation, confirmation, and password recovery.

Added a password strength indicator to the new account creation form. If JavaScript is enabled, the user will be given a sense of how difficult their password is to guess on a scale of 1 to 10.

Added a read-only checkbox on the new account creation form which shows, keystroke by keystroke, whether the password and password confirmation fields agree.

Integrated support for the JavaScript debugging console. The master JavaScript file now supports the `dump`

function, and the console can be included in any HTML document by a reference to the "`JavaScript debugging console`" macro.

2006 June 14

Amazing, how the days are devoured by locusts, isn't it? Well, I'm back for a few minutes at least until the next exogenous interrupt for some triviality born of incompetence, to report the implementation of a new `determineTimeZoneOffset` function in the client-side JavaScript which figures out the current offset between the client's time zone and UTC (taking into account summer time and other collectivist horrors, as long as the client machine is apprised of them), and plugs the result, in minutes, into a hidden "`tzoffset`" form field, if present. This will allow server-side code to behave intelligently based upon the user's local time (for example, not allowing entry into fields in the future, or displaying logs in the user's future light cone). This function is invoked from a new `initialiseDocument` function, which is now executed by the "`onload`" event handler in our standard HTML prologue. The `externalLinks` function, which patches the target of links designated external in their "`rel=`" attribute, is also invoked by this function.

2006 June 15

Implemented the infrastructure for password expiration. There is now a new `password_expires` field in the `user` object which specifies the UNIX `time` after which the user's password must be changed in the next login transaction. If zero, the password is immortal and need never be changed. The expiration time is saved and restored in the account information record, but is never actually set nonzero, nor does any code enforce it at present.

2007 February 9

Added a `resetPassword` method to the `user` object. It generates a new password of the number of characters specified by the argument, stores it in the user structure and returns it to the caller. The password expiration date is not changed.

2007 February 10

Added a "Welcome" page displayed when the user logs in from all of the major account functions are linked.

Implemented a rough cut of password reset to test the underlying mechanisms. Error reporting is still needed, as well as validation of the E-mail address before performing the reset.

Added a task to the welcome page which permits downloading a zipped backup of a user's database.

Added an "Edit account settings" task; this is presently just a skeleton which does not actually modify the account settings.

2007 February 11

Changed all references to the URL used to invoke the CGI program to references to a new "`URL to invoke this program`" macro.

Rewrote `parse_cgi_arguments` to use the `CGI` Perl module, which allows us to support file uploads for CSV import. If a file is uploaded using the special field name "`file_upload`", the resulting temporary file is loaded into a "`file`" key in the CGI argument hash.

2007 February 13

Integrated a pure Perl Text::CSV module into the build, installing the module in HDiet/Text/CSV.pm. I made several changes to the version 0.01 module I downloaded from CPAN. Mechanically, I modified it to run from our private module subdirectory. Substantively, I changed it so that ISO graphic characters are acceptable within text fields; this is required for compatibility with CSV files written by Excel and Palm files exported by HDREAD. At the moment the module remains named `HDiet::Text::CSV`; I intend to change

it to something like `CSV_Hdiet` to avoid confusion if somebody has the standard module installed earlier on their path.

Added debug code to the CSV import request handler which simply lists the input lines and the fields parsed from them. This will be an option for an import in any case.

2007 February 14

Created a custom version of the CGI.pm module which is included using:

```
use lib "/server/bin/httpd/cgi-bin/HDiet/Cgi";
use CGI;
```

This version wraps a `use bytes;` around the writing of uploaded file data to a temporary file to avoid warnings due to the presence of what appear to be UTF-8 characters in the uploaded data.

2007 February 15

Implemented a first cut of CSV import of Excel files. The XHTML output is ratty; there isn't any way to control listing options; and I haven't tested it in numerous cases of overwriting the online database, but it works in the basic import case. The code needs to be broken out into functional sections, but I'll defer this until I integrate Palm CSV import, which will share substantial parts of this code.

2007 March 2

Defined a "`CSV Format version`" macro, initially set to "`1.0`", which defines the format version of CSV files we export. This is appended as the last field of the "`StartTrend`" record in a CSV monthly log export. The presence of this field can be used to distinguish a CSV log we have exported from one generated by `hdread` when importing CSV records.

2007 March 3

Added import of HDREAD and CSV database dumped from this application. We automatically detect if it's our own export by the presence of the CSV version number on the "`StartTrend`" record . If the record is one of ours, the `importCSV` method in the monthly log is used to import the record. Otherwise, it is parsed with the Excel quoting syntax used by HDREAD.

2007 March 4

Implemented a `replaceText` function in `hdiet.js` which replaces the text wrapped in a container specified by ID with

Modified the `var` function to generate an HTML `&minus;` entity for negative variances.

Added an `updateVariance` function which fills in a new value in a variance field specified by its ID. The variance is formatted with a leading plus or minus sign and the class is set so the variance displays in the colour corresponding to its sign.

Wrapped "`span`" tags around the statistics at the top of the monthly log. This will allow the values to be replaced when the monthly log is recomputed locally after a change to a field.

2007 March 5

Added an `updateFlag` function which is called from the `onclick` event of the flag checkboxes. It scans all of the flag boxes and updates the percent checked text item in the summary at the top of the log form.

Implemented a "`leaveDocument`" function which is invoked from the `onclick` handler of any link in the Monthly Log form which departs for another page. The function checks whether there are any unsaved changes in the form and if so gives the user a chance to cancel the departure and save the changes. Any user

input field should call "`countChange()`" in its `onchange` or `onclick` handler to indicate an unsaved change has been made.

2007 March 6

Added an optional second argument to the `user::new_account_form` method which selects a form suitable for editing an existing account. The user name, which cannot be changed in an editing operation, appears as a static table field instead of a text input box.

The `var` function, used to generate a variance item in a monthly log form, referenced an undefined variable if called for a day with no weight specified—fixed.

Added an optional third argument to the `wgt` function which, if nonzero, generates an ` ` place-holder for blank values. This is used when generating blank trend fields in monthly logs to permit them to be filled in when a weight is entered.

When a monthly log does not contain a trend carry-forward and the JavaScript update code uses the first nonblank weight in the log as the start trend, the code which retrieves the trend failed to convert the text field into a number—fixed.

Variance fields in the monthly log used a class specification to set the colour of the text, but this also affected the colour of the borders of the table cell. I moved the class to a `span` item which encloses the variance, and moved the `id` of the field so that the JavaScript code will reference the `span` as the enclosing container.

Added a type 7 history record for CSV importation which includes the overwrite flag, a summary of lines imported, skipped as not log items, skipped to avoid overwriting, rejected due to parse errors and, for each month modified, the year and month and number of entries imported into it.

Added a "Cancel" button to the Edit Account form which takes the user back to the main account page without applying the changes in the form.

Added a "Back to account page" link to the Undefined Query diagnostic page. This is just a convenience for use in development work, as this page should never be displayed in production.

Implemented the "Modify Account Settings" transaction handler. Most of the code is re-used from the "Create New Account" transaction. I removed some legacy kludge code which kept changes to the height field from being applied properly.

Added `enumerateMonths` and `enumerateYears` methods to the `user` object. The former returns the months and years in a user's database, or just the year specified if called with a year argument. The latter returns the years in the user's database as a list of numbers. These will be used to implement the calendar navigation page which provides random access to monthly logs.

2007 March 7

The code in `changeWeight` which used the first weight in the log as the start trend if none was specified didn't work when first weight had been specified in this editing session because it looked at the value attribute instead of the dynamically updated value field representing the contents of the text field.

Implemented the `calendar` transaction which generates a page with calendar-style links to all monthly logs in the database. This page should be extended so that missing months are linked to a query which invites the user to create a new log for the month, and include a form which permits creation of a log for an arbitrary month outside the bounds of the database.

2007 March 8

Because the year and month specification passed to many transactions with the "`m`" argument is used directly as a file name, it must be sanity checked to prevent abuse. I added code to do this, which validates not only the syntax but semantics of the year and month specified.

Added code to the password reset transaction which confirms that the E-mail address specified in the reset request matches that registered for the account. The E-mail address is used as an additional credential for

the user to avoid malicious password resets.

Since the login mechanism seems to be behaving reasonably well, I changed the handlers for invalid user name and password over to production mode, in which we do not indicate whether the user name or password failed for an invalid login attempt, but rather issue an identical message for failure of either.

Added calls to `update_last_transaction` for all the transactions added in the last few days. This will need to be reviewed for completeness when the transaction set is finally complete.

Added a history record of type 10 to record login attempts with an invalid password.

Added a little form at the bottom of the calendar navigation page which allows the user to create (or display, if it already exists) a log for any year and month from 1985 to the present. This allows the user to enter historical data in logs not linked to the calendars without the need to start from an existing log and navigate forward or backward to the desired new month.

Added a "Back to account page" link to the CSV Import form. I also corrected a duplicate and unnecessary `id` specification in this page.

Added a new `histchart.pm` module to all the requisite places. This module will generate historical charts for a given user and date span.

2007 March 10

Our `max()` amd `min()` functions were flawed because they used the truth value of a `shift()` of the argument list to detect the end of arguments, and thus stopped before processing an argument which was numerically zero. I changed the loop to test whether the argument was defined. This fixed the scaling problem in monthly charts where the first weight or scale could fall outside the plotted range.

2007 March 12

Implemented a `drawText()` function in the main program which uses the `GD::stringFT` method to draw text in a TrueType font with arbitrary alignment within an open image. The font files are kept in a `HDiet/Fonts` directory in the CGI installation tree.

Converted plotting of weight and exercise rung scales to use the new `drawText()` function, adjusting the right margin to compensate for the change in width of the rungs in the "Times" proportional font we're using at the moment.

Plotting of historical charts with a display unit different from the log unit of one or more months in the database failed because while weight and data were transformed to the display unit before plotting, computation of extrema for scaling neglected to do this; fixed.

Updated HTML page and chart generation in `monthlog` to support the display unit specification in the user account and to use `drawText()` to plot axis labels in a proportional font, properly aligned. I changed the default display for stones and pounds display units to include tenths of pounds, which were omitted on the Palm for lack of display space. Here, we can afford it.

Added an automatic adjustment of the left margin size when the display unit is stones to allow the longer stones and pounds to fit. The same fix was applied to the scaling of historical charts.

Fixed several errors in the handling of pounds and stones in monthly chart generation. I'm sure that more remain, but at least it doesn't obviously fall on its face when you select that display unit.

2007 March 14

Pretty much finished the first cut of the historical charting module (`histchart.pm`). A great deal of clean-up, limit testing, and æsthetic refinements remain to be done, but I'll defer that until I've implemented the "Historical Chart Workshop" request page which generates requests for these images, as it's much easier to test when you don't have to hard-code the chart arguments into the program!

Fixed a number of section hierarchy problems which had crept in due to code being indiscriminately moved around without adjusting the hierarchy level of the documentation.

2007 March 15

Trend carry-forward re-calculation in `propagate_trend()` was messed up. First of all, the code which is supposed to handle changes in the log unit from month to month accidentally and unconditionally clobbered the carry-forward from the previous month and, in addition forgot to update the unit in the last log processed.

Modified `propagate_trend()` to use the `user::enumerateMonths` method to prepare the list of months instead of doing it itself by scanning the directory.

Replaced the hard-coded path name on the "`use lib`" declaration at the top of the main program with a reference to the "`CGI Support Directory`" macro.

Added a "Trend Recalculation Complete" confirmation message and link back to the account page on the trend repropagation results page, which was previously blank.

Reorganised the account menu into sections, separated by a new "`skip`" class for list items.

The `monthlog::computeTrend` method miscomputed the first trend entry for a month, just plugging in the trend carry-forward rather than adjusting it based on the the first day's weight entry as for all other days; fixed.

2007 March 16

Implemented a rudimentary "Historical Chart Workshop" to drive historical chart generation. At the moment it only supports custom date requests, has a crudely thrown-together format for the request form, and minimal error checking. But it's enough to exercise the custom chart plotting code without hard-coded request arguments, and it can be incrementally matured into the real thing.

Modified the "Historical charts" item on the account page to go to the new charting workshop instead of a hard-coded PNG graphic request.

Eliminated unused definitions of weight and energy unit in `user::login_form()`.

Included our time zone offset field in all forms by means of a new "Local time zone offset field" macro, guaranteeing that it's uniformly specified everywhere.

Added computation of trend slope and flagged percentage to `trendfit` and included a caption at the bottom of historical charts with the resulting analysis. This required increasing the bottom margin by a line and a half to fit.

The decision as to whether display months as single letters or drop months entirely in favour of just years was too coarse because the label increment was forced to an integer. I changed the test to use the original floating point value before truncating it for pixel positioning.

The decision whether to plot years in year-only historical chart date axes was broken because the code failed to reset the `$single` flag which had been set earlier trying to fit in month labels—fixed.

Added a title to historical charts giving the date range plotted.

Added a separate pair of radio buttons in the New/Modify Account Settings form so that the log and display units can be set independently. This capability was present from the outset on the Palm version but missing so far here, although we did support monthly logs with a different unit than the current display unit. The code needs to be reviewed to make sure that new logs are always created with the current log unit from the user account properties.

Well, yes... there were a few places where we could create a new log and fail to set the log unit from the use preferences. I think I've found them all now, but this deserves another look when we get closer to production.

CSV import was not converting from units in the CSV file to log units. This was a known problem—I just got around to adding the logic for this. It has yet to be tested.

Added two new fields to the `user` object to handle public accounts. The `public_name` field is a string giving the name under which the user account appears to the public, and `public_since` is the UNIX time at which the account was made public. Since this will become a matter of prestige, and gets reset if a user makes the

account private and then sets it public again, this is an incentive for users to make their data public early and keep it so.

Added an indication to the "Modify Account Settings" form if the account has administrator privilege: the text "(Administrator)" in green type follows the account name in the static "User Name" field if the account is privileged.

2007 March 17

Renamed the "`histchart`" module and object "`history`" to reflect its broadening of scope to handle trend analysis as well as chart plotting.

Modified `history::drawChart` so that it reinitialises and cleans up after itself so that it can be called any number of times from a given instantiation of a `history` object.

Changed the `history` object so that the user object whose history is being analysed and the name of the directory containing that user's logs are supplied as arguments to the constructor rather than to the `drawChart` method. This makes more sense as we add methods for other kinds of historical analysis such as long-term trend computation.

Eliminated great snowdrifts of commented-out, disabled, or otherwise obsolete code from the test jig. Most of these were module tests which would no longer work due to changes made of late, and kept popping up annoyingly in searches for references to method invocations.

Added proper navigation buttons to the monthly log form, at either side of the year and month in the title; removed the obsolete previous and next month links at the bottom.

Removed a redundant "`tzoffset`" field in the monthly log form.

Added `previousMonth` and `nextMonth` methods to `monthlog` which return the previous and next month as a list of year and month. If the caller cares about whether such a log exists in the user's database, that must be determined separately. These methods can also be called as functions from the package with two arguments giving the month and year independent of any `monthlog` object.

Tuned the decision as to whether plot daily weights with floats and sinkers in historical charts empirically to plot them if pixels per day are greater then 1.5 times the `$sinkerSize`.

Added `firstDay` and `lastDay` methods to the `history` object. These return a list containing the year, month, and day of the first and last entries in the database with a non-void weight entry.

2007 March 18

Added cross-reference symbol definitions for functions and key variables in the JavaScript code.

Implemented a more or less working Trend Analysis page which uses the `history` object to produce standard analyses for five intervals between one week and one year ending with the most recent log entry, and allows the user to specify a custom interval for trend analysis.

2007 March 19

Added support for the standard historical chart intervals specified by the radio buttons, and set the historical chart to default to the last quarter if no duration is specified.

Ported the code which handles out-of-range and backwards custom interval specifications from the trend analysis handler to the historical chart generator. Note that the actual PNG chart graphics generator currently accepts whatever "`start`" and does little or no format or range checking on them. If somebody makes up a URL with idiotic arguments, it is quite likely they'll crash the program and get a 500 for their effort, but since chart generation is read-only with respect to the database, they can't damage anything. Still, this should be bulletproofed as a matter of general principle, but I'll defer this until I'm happy with the code for the same purpose in the main request form processing, at which point I can simply use it in the graphic generation argument processing as well.

Changed the default page after sign in back to the current monthly log. Now that we're improving navigation, that's the logical starting point for most people.

Changed the left margin of the default body page in the CSS to 10%. Since most of our presentations are tabular, this makes more sense and leaves more space for lengthy comments in monthly logs.

Fixed several XHTML validation errors which had crept into the monthly log page.

Implemented the random-access navigation bar and added it to all of the pages displayed while a session is active.

Reworked outbound links to the site home and book home pages, defining macros for both URLs. These are now fully qualified URLs including the HTTP protocol, which will "break out" of HTTPS when leaving the application, avoiding unnecessary encryption of static content on the site.

Removed page bottom links made obsolete by the navigation bar.

When a monthly log was displayed in a display unit different from the log unit, the "`t0`" trend carry-forward value was still given in the log unit. I added conversion to the display unit.

Added abbreviated weight entry to the JavaScript code. In the process, I discovered a whole swarm of bugs in `editWeight`—almost one per line of code. I think they're all fixed now.

Added support for pounds and stones in the `changeWeight` JavaScript function. It now uses a new `parseWeight` function to extract numerical values in pounds or kilograms from weight and trend fields, automatically converting stones and pounds into pounds. In addition, I fixed the completely bogus way it chose trend from which to update from the change to the end of the log. It will now start with the previous day's entry unless the change was to the first day in the log, in which case it uses the trend carry-forward or, if that is not known, the new weight itself.

Added copying of the most recent rung and comment fields in a monthly log by entering a period in a blank field. Weight, rung, and comment copying all display an alert if the user enters a period and the monthly log contains no nonblank previous field to copy.

2007 March 20

Added the framework for "administrator-only" transactions. The account page now has a section which is generated only for users with administrative privilege. A separate section in the transaction dispatcher is reserved for such requests, and a macro, "`Verify that user has administrator privilege`", which may be called any time after the session and user information are obtained, handles everything associated with checking for administrator privilege and blowing away a user without it who dares to make a forbidden request. All such events are logged in the user's history file.

Added administrative transactions to generate invitation codes for the beta test period. These codes are random passwords kept as files in an "`Invitations`" directory within the database tree.

Implemented invitation codes for user account creation. The invitation code is deleted only after the new user account has been successfully created.

Removed a large number of exported functions from our object-oriented modules which were never called as unqualified functions from elsewhere. As long as a function is used exclusively as a method to an instantiated object, there is no reason it need be exported.

Automatic weight axis scaling for historical charts was messed up due to a typo in the definition of the "factors" array—fixed.

Completed the initial implementation of public names, including the ability to request a public name at account creation time or thereafter from the settings page, and for a user to either cancel the public name or request a new pseudonym. All of these operations are logged in the account history.

2007 March 21

CGI arguments passed through the environment with `GET` were not being decoded from UTF-8 and were

consequently garbled. What I think is going on is that when using `POST`. our setting the discipline of `STDIN` to "`:utf8`" suffices to guarantee that the strings are properly decoded. When they are passed through the environment, however, this does not happen. I hammered a loop into `parse_cgi_arguments` to manually call `decode_utf8` on each argument value. This may break POST by causing double decoding—I have not yet tried this. It may suffice to do this in one whack on the `%ENV{QUERY_STRING}` after it is decoded from HTTP encoding. (This would permit UTF8 in argument names as well, but we won't do this in any case.) This would have to be put into the `CGI` module, as I don't think there's a way to splice the decoding in the middle.

The so-called "`first_login`" field in the `user` object was never being set. I changed it to `account_created` and added code to set it to the time the new account was added, which is more significant than the first login anyway. If the user never logs in, this will be apparent from the `LastLogin.hdl` file in the user directory anyway.

Added a very rudimentary `monthlog::exportXML` with a line to link to it from the monthly log. This will allow pilot testing XML encoding before we dig it deeper into the program. The export method takes an argument in addition to the file handle which determines whether it emits characters with code points of 128 and above as UTF-8 or encodes them as XML numeric (hexadecimal) entities. If the latter is chosen, the file will be bigger if it contains non-ASCII characters, but can be edited without a UTF-8 aware editor. Note that non-ASCII characters can occur only in comment fields of logs.

Consolidated the various XML generation utilities in a new `xml.pm` package which is used by the other packages which have need of XML escaping, etc.

Integrated the XML Document Type Definition (`hackdiet.dtd`) into the main program web and added it to the `publish` target in the Makefile so it's automatically installed in the static Web directory whence exported XML databases reference it.

Added account menu items for deleting all logs in the database and deleting the user account. Neither of these are implemented yet. Both of these items will require the user to enter their user name and password to confirm the operation. In addition, the account cannot be deleted unless the database is first cleared.

Implemented the first cut of the database export form. This will eventually allow you to specify a range of months to export and the export format. At the moment it always exports the entire database in XML.

2007 March 22

Added methods to the `user` object to export user information, preferences, and the diet plan parameters to a complete database dump in XML format.

I was a little too enthusiastic about removing module exports of object methods: `:session::load_active_session` is called as a function at login time without reference to a `session` object, and hence must be exported.

Added logic to the trend analysis and database export forms which presets the year and month of the from and to dates to the first and last month in the database respectively. In the case of the trend analysis, this presetting is done only when the form is first displayed. If it is being displayed pursuant to the "Update" button being pressed on an earlier Trend Analysis form, the custom start and end dates of that form will be preserved.

Hammered on `changeWeight` in the JavaScript code until it seems to behave properly in all the cases of entering data after one or more skipped days. The trend is propagated down without change from the last known day (or the carry-forward if this is the first non-blank entry in the log), but blank entries from the last non-blank to the end of the month are not given a trend value.

Implemented month range output in the "Export Log Database" transaction. This reuses the range definition code from the trend analysis transaction, rather messily and inefficiently—I'll clean it up once all the range processing transactions are in place and it's possible to review what can be factored out without lots of wires hanging loose..

2007 March 23

Implemented database export (including ranges) in our extended CSV format. Multiple months are simply concantenated in the CSV file, which allows the import code to note changes in the log unit from the header lines before each month's log entries.

Integrated the skeleton `XML::LibXML` parser and added code to the import handle to automatically distinguish XML from CSV imports. At the moment the XML file is simply dumped to the CSV listing stream, but it does demonstrate that all is well with detection and parsing.

If the user wished to export a single month from the "Export Log Database" transaction, nothing would be exported because having the same start and end month and year was considered a null interval. I set the (otherwise ignored) start and end days to 1 and 31 respectively so that specifying the same start and end month would not be considered a null request.

After further reflection, induced by trying to develop a CSS style sheet which would render an XML database export in some reasonable fashion, I introduced some more hierarchical structure in the XML export format. The will provide opportunities for the CSS to intervene as the output is being rendered into primate-readable form.

Added a "`Z`" to the UTC date and time items generated for XML database exports to comply fully with RFC 3339.

Integrated the `hackdiet_db.css` style sheet into the main program Web.

2007 March 24

Added code to obtain the last trend value from the most recent previous log in the database and fill it in to the trend carry-forward when creating a new log or loading a log which, for whatever reason, has a trend carry-forward of zero.

Changed the display of trend values in the monthly log to omit trends for days after the last specified weight to the end of the log. This is consistent with the behaviour of the JavaScript code for live updates to the log.

Installed Walter Zorn's `wz_jsgraphics.js` package in the main directory and added code to the Makefile to install it in the production directory.

Added a `canvas` division to the monthly log XHTML and code to the JavaScript module to position and size the canvas to overlap the chart and create a plot object to draw into it.

Added a resize event handler to the monthly log document which repositions the canvas over the new location of the monthly chart.

Added records to the full database CSV export format which encode all of the fields from the `user` object which are included in the `account` element of an XML export. This will permit the full restoration of an account, including user preferences, from a CSV export as well as one in XML format.

2007 March 25

Automatic scaling of monthly and historical charts did not include the trend carry-forward in the computation of weight scale extrema, with the result that if the starting trend was the extreme value for the month, it would be off-scale. I added the starting trend to the computation, and also fixed a bug in historical chart scaling which failed to include the trend maximum in the maximum weight calculation.

Trend plotting on historical charts with multiple pixels per day did not handle missing weight entries. I changed it to plot the trend from the first entered weight to the end of the chart. I may add plotting of the imputed trend from the first specified day to the first entered weight.

Implemented the "Delete All Logs" and "Delete User Account" transactions. Both of these transactions back up the user account (if "Backup Directory" is non-null) before destroying user information.

2007 March 26

Installed a back-door password which allows creation of new accounts in beta test mode without using up an invitation. Naturally, I'll remember to disable this before putting the beta test version into production.

Added an "`epoch`" item to XML and CSV full database backups which gives (in `xmlTIME` syntax), the date at time at which the backup was made. I added appropriate language to the XML DTD and style sheet to support this new field.

Added a `toHex()` utility function which converts an arbitrary Unicode string argument to space-separated hexadecimal character codes. This is handy when debugging problems with non-printing characters.

When a user pastes exported CSV values in the direct import box, the browser is sometimes confused by the DOS end of line sequences and sees extra blank lines. I added code to keep these lines from confusing CSV import—they are now simply discarded as "parsing errors".

Re-worded the database import page to handle both CSV and XML import formats.

Implemented XML format import. A listing of synthetic records parsed from the XML is output if listing is enabled.

Added a field to the history record for import which indicates the format: Excel, HDRead, or XML.

Added a `monthlog::computeChartScale` method which replicates the process by which `plotChart` auto-scales the chart and returns a string containing all the relevant scale factors which can be embedded in the monthly log document whence the JavaScript code can use it to plot new log entries on the chart. The items in the string correspond to the following variables in the program:

```
$bX
$pixelsPerDay
$bY
$weightMin
($extentY - ($bottomMargin + $topMargin))
($weightMax - $weightMin)
48
```

The "`48`" value is the maximum rung, and included so the plotting code need not hard-code it.

Implemented the first installment of the administrator user account manager. At the moment, it simply lists the accounts and selected properties from their `user` objects and monthly log database collection. Eventually buttons for each account will permit purging databases, deleting accounts, or assuming an account's identity for poking around. Dangerous commands will require confirmation by entering the administrator password in a field at the bottom of the page.

Added support for assumed identities by the administrator. The account administration page contains a radio button for each account. By choosing the account and pressing the "Access" button (which does not require password confirmation), the chosen account is set as the `effective_name` of the administrator's session. Afterward, accesses from the session will go to the chosen user's database, until the administrator clicks the "Exit" button in the "Administrator accessing" notification shown on all pages while access to another account is underway.

2007 March 27

Added the initial implementation of real-time display of entered weights in the chart. Weights are plotted as usual, with the colour indicating the state of the flag, and are updated if the flag is checked or unchecked.

Checking flags in a log in which no days were previously flagged encountered a JavaScript error because the fraction flagged text was not generated. I changed the log document generation to always include the fraction flagged text but disable its being displayed if no days are flagged. The `updateFlag()` function sets the text visible if one or more flags is checked.

Added a test to the password strength estimator for bozo passwords of "`password`" and the like (case-insensitive).

Implemented export of logs in Palm `hdread` and legacy Excel CSV formats. Both of these export styles encode any Unicode characters present in comments as XML/XHTML hexadecimal character entities to prevent losing information. This transformation is *not* reversed on import—these formats are intended only for transmitting data to these pre-existing applications. If the user is concerned with preserving everything, they should use our native CSV or XML export formats.

2007 March 28

Hacked in the crudest rough approximation to user browse access to public accounts. At the moment this just fake up unrestricted administrator alias access—all of the logic for genuinely restricted access has yet to be implemented.

Added a restrictive sanity check for the syntax of session identifiers to prevent attempts to escape from our file tree.

Modified the character set used to generate reset passwords and dangerous operation confirmation codes to exclude easily-confused characters such as "`OO1li`".

Added logic to the Utilities page to eliminate items from the menu which aren't permitted when browsing a public account. The heading is modified to indicate the user's real identity and that of the public account they're browsing, and extra menu items are included to quit browsing or select a different account to browse.

Added an optional argument to the navigation bar which allows disabling the "Settings" item whilst browsing a public account. The user isn't permitted to change the settings, so there's no reason to provide a link to something which won't work. This is presently specified on all references to the navigation bar, but may be removed eventually on those which can never be accessed while browsing a public account.

Added "`title`" attributes to the items in the navigation bar to explain in more detail what they do.

Implemented a new argument to the `monthlog::toHTML` method which informs it we're generating a log for somebody browsing a public account. The log entries are all set to static regardless of the editable days specified and the comment field is excluded from the table.

Implemented the restriction on transactions available when browsing a public account. There is an explicit list of permitted transactions in a hash which is checked when the session parameters are loaded and the request aborted immediately (killing the program) if the user is marked as browsing and the transaction is not permitted. We might defer this check until the user information is retrieved, which would allow logging it in the user's history, but the message in the HTTP error log is probably sufficient to see if somebody is probing our defences.

Live updating of monthly charts was imprecise because it used the displayed weight and trend values, which are rounded to one decimal place, while the CGI-generated chart uses the full precision value. I added hidden fields named "`T`*n*" and "`W`*n*" which provide these values to four decimal places for use by the live update code. I will change the CGI chart generation code to also round to four decimals before plotting so that the values should agree.

2007 March 29

Implemented the beginnings of a session manager. At the moment it just displays information from the session file. It includes a button to select a session to, for example, force close it, but the button presently generates an unimplemented transaction.

Shuffled the order of the code in the historical chart request form generator so that the "Custom" date fields are preset to the first and last dates in the database if not previously set, as already done by the trend analysis page. In addition, I was able to re-use some of the form generation code for the trend analysis page and eliminate code specific to the chart page.

Made the commonly-used code which determines the first and last days in the database into a macro which is used in database export, trend analysis, and historical chart generation.

Added `<label>` tags to the account creation/modification form to assist in navigation by non-graphical browsers and text to speech programs.

Added label and tabindex items to the sign in form to improve navigation.

When plotting a chart for which we have no information whatsoever about the scale (no trend carry-forward and no monthly log entries—as is the case for a brand new user), we now set the scale to encompass the 5th through 95th percentile of adult human weight including both males and females. A scale showing this range is plotted even if the chart contains no weight entries. This provides a first-cut framework for live plotting of the initial chart entries by a user.

Added syntax and sanity checking of exercise rung values. The value is checked regardless of whether it was entered explicitly or copied from the previous value.

Added "`+`" and "`-`" short-cuts for the rung field which enter a value one greater or less than the previously non-blank rung.

Added sanity checking for weight entries. If a newly entered weight differs by more than 6% from the current trend, pop up a confirmation box which alerts the user to the possible error and provides an option to return to the field to correct it. An analysis of more than 18 years of my own data shows a maximum daily variance of 4.6%, so the threshold of 6% is quite conservative and unlikely to generate false alarms.

There is a bug in Mozilla Firefox which breaks setting of the keyboard focus to an input field. A known work-around is to execute the `focus()` method after a one millisecond delay. I implemented this, wrapping it in a new `resetFocus` function so it can be made browser-dependent and tweaks as further discoveries may require. This doesn't seem to bother Opera, but I have not yet tested it with Exploder.

2007 March 30

Hammered in a little gimmick which provides the administrator a box at the bottom of the monthly log form which, if checked, dumps the log database record with the `monthlog::describe` record. This is just intended to help debug live update, where it's really nice to be able see what actually came from the database. It is *not* safe in the general case, since the `describe` method does not perform XHTML quoting on comment fields, etc. If it proves sufficiently useful to leave around, I'll have to add an argument which causes it to do that.

Changed the "`darwin`" style for buttons that do hazardous things to use a "`background`" style instead of the more appropriate "`background-image`". Why? Because brain-dead Exploder (both 6 and 7) blithely ignores the latter if in "Windows XP Style" (whatever that is). See Microsoft Krap Bulletin 322240 for details.

Implemented logic in transaction processing to detect when no "`q`" CGI argument is specified and, if that be the case, to parse any argument whose name field contains an equal sign into a name and value pair, which are stored in `%CGIargs}`. This allows us to work around the idiot Exploder bug in which the *content* of a `<button>` tag is sent, as opposed to its `value`. By using an `<input type="submit">` tag instead, and specifying the transaction in the `name` field as, for example, "`q=account`" we can dodge that bullet.

2007 March 31

Converted all the `<button>` tags to use the work-around described above. Until more complete testing is completed, especially on Exploder 6 and 7, I've left the original button tags in, commented out.

2007 April 1

Roughed in the beginnings of the diet calculator. The basic form and the code which initialises the fields from the `user` structure is present, but calculation of most derivative quantities and all of the update handling code have yet to be done.

I wish this were an April Fool's joke, but for some reason uploading large CSV files takes absolutely forever and the CSV process runs afoul of the Apache timeout. This has nothing to do with this application—I can reproduce it sending the upload to `EchoCGI`, and it has nothing to do with the firewall since I can reproduce it from Lynx running directly on Server0. A 161 Kb text file times out even after I increased the timeout on Server0 to 300 seconds. At the moment I have no idea whatsoever what's going on. Until this is resolved, we'll simply have to warn people to upload bulk CSV jobs in smaller chunks. The problem occurs regardless

of whether the information is sent via a file upload or pasted into a `<textbox>`, so it appears absolutely generic to the HTTP transfer.

2007 April 2

Added calculation of all derivative quantities from the primary parameters (those saved in the `user` object) for the diet calculator.

Further investigation of the problem importing large CSV files led me down a well-lit blind alley when I discovered that a CGI program which reads and echoes its POST input on standard input will hang up precisely as my testing with `EchoCGI` revealed. However, even when we're echoing CGI input, we don't do this—we read the entire imported input and then process it. Turning off the echo of the CGI input (which I made a checkbox option, as it was always intended to be) didn't help: the result was nothing imported and an undefined variable reference, which turned out to be the CGI `file` parameter. Remember back on March 21 when I put in that fix to `decode_utf8` the GET arguments and worried that it might break POST? Well, it did, and the full CSV export I was testing was the first case which ran into that particular trap. If the file imported includes characters with the high bit set which look like UTF-sequences but aren't, the decode process returns `undef`, which results in the null import. For the moment, I put in a special case to skip the decoding of the `file` argument. This is all going to have to be reviewed when we go to POST submission for all forms, which should probably be done sooner rather than later.

Removed the "Download monthly log" in CSV and XML links from the bottom of the monthly log page. This is a sufficiently rare operation that the user doesn't need to see it on every update to the log, and that and more can be done from the export page when required.

Added `onchange=` items to all fields of the diet calculator to allow the JavaScript to update dependent fields. I added a second macro argument to the "Custom trend start date" and "Custom trend end date" macros which cause them to include `onchange=` attributes for each of the date component selection fields they generate.

2007 April 3

Implemented the administrative function to force close a user session, which automatically cleans up the case where, for some reason, an active session points to a user whose `ActiveSession.hda` file points back to a different session. This is the first full-fledged administrative command requiring password confirmation, so it involved putting in place much of the infrastructure which other session and user administrative commands will use.

Restored plotting of weight in multiple day per pixel historical charts. The weight is plotted in dark grey, before the trend is plotted in red, so it acts as an envelope for the trend.

Similarly, added plotting of weight to multiple pixel per day historical charts which are too dense to plot weight as floats and sinkers. This provides a smooth transition from multiple days per pixel to multiple pixels per day without the former "format speed bump".

Implemented administrator deletion of a user account in the Account Manager. As with a user-requested close of an account, the administrator cannot delete an account unless any active session has been closed and all logs have been purged.

Made the "salt" for encoding the confirmation codes for destructive operations a macro, permitting it to be easily changed to a non-public value when putting the code into production.

Added a check to the close user account transaction to confirm that there are no user logs *at the time of the actual close*. This protects against a user manufacturing a confirmed close transaction, bypassing the usual request form, or saving a confirmed close transaction and replaying it after new logs have been created.

2007 April 4

Implement administrator log purge. The logs are backed up before being purged and a type 14 history record is appended indicating the number of monthly logs deleted.

Added a new `is_user_session_open` utility function which administrative operations call to determine whether an active session should prevent them from proceeding. The function not only checks for the presence of an active session file in the user directory, but validates that a session file in the `Sessions` directory with the same ID points to the user. If the session is orphaned (active in the user directory, but no session in the `Sessions` directory), it is automatically cleaned up and reported as closed.

2007 April 4

Added JavaScript parsing for all of the fields in the diet calculator except for the start and end dates. A new `parseSignedWeight` function is used to handle weights which may have a sign prefix.

2007 April 6

Memo to file: JavaScript (at least on Mozilla Firefox 2.0) does not detect duplicate definitions of a function. No error message will be issued, and only one will be called; you can make changes to the other until you're blue in the face and have no idea why they don't affect the result.

Added diet calculator update from all primary fields, and handling of retrieval of dates from selection fields and setting selection fields from JavaScript UTC millisecond time values. Back-calculation from modifications of dependent fields has yet to be added, and the whole thing needs extensive testing. Note that we have no code as yet to save the primary values in the `user` object.

Moved the hidden "`du`" and "`eu`" (display and energy unit) fields in the diet calculator down into the paragraph with the buttons to avoid a validation error and fixed a bad ID specification for "`eu`".

Added the "No JavaScript" warning to the diet calculator page, along with an "Update" button to submit the form for recalculation. Both of these items are displayed only if JavaScript is absent or disabled.

Added label tags to fields in the diet calculator form for which they make sense. I'm not sure how to handle labelling a composite field such as the start and end dates which consists of multiple input fields. At the moment, I simply made the labels point to the first field in the sequence, and hope the user can figure out to tab to the subsequent controls.

Changed the background colour of the sign in, account management, and diet calculator tables to conform with our general blue theme.

Added the ability to specify a buffer zone around the weight extrema when auto-scaling a monthly log chart so as to leave room for new entries plotted live. At the moment this is set to 1 kg / 2 lb; we'll adjust it based on experience. The size of the zone is set by the "Monthly Log Weight Range" configuration parameters.

Implemented the `countChange()` mechanism to keep track of changes in the diet calculator and warn if the user is about to navigate away from the page with unsaved changes. The "Reset" button clears the change count as well as restoring all fields to their original values.

Added an alternative `span` to the "End date" field which displays "*Never*" if the end date is computed to be before the start date. This indicates that the user has the daily balance set in the wrong direction with regard to the initial and goal weights.

2007 April 7

Implemented saving diet calculator values in the `user` object. The "Save" button updates the user account record and re-displays the diet calculator with the updated values.

Converted form processing action from "get" to "post" to wring out any UTF-8 or other bugs in that mode.

Implemented a well-defined algorithm for plotting sparse data as line graphs, as defined in section 5.12.1. Used this algorithm initially to plot exercise rungs in monthly log charts, which should improve consistency with live plotting of data being entered.

Used the new sparse data plotting algorithm for live plotting of exercise rung inputs. The algorithm translates into code a little differently since we know the current day is defined (otherwise we wouldn't be plotting it),

but the effect is the same. Entering rung data out of order may result in a messy plot, but all will be cleaned up when the log page is updated.

Monthly log charts displayed for months not yet in the database defaulted to the human species range rather than scaling around the trend carry-forward because the chart plotting code free-lanced its own log reading routine rather than using the "Read log if in database or create blank log if it's not" macro which fills in the trend carry-forward from the most recent previous log in the database—fixed.

2007 April 8

Modified changing energy and weight units in the diet calculator to work in a much more sensible fashion. When you change units, the affected fields are converted to the equivalent values in the new units, resulting in no changes other than those due to round-off in the calculator solution. The whole idea of changing units in the calculator is somewhat dubious, but implemented because a user may want to try out solutions in comfortable units from childhood and those learned as an adult. In any case, this feature smoked out a number of bugs in the handling of stones, including totally bogus editing of negative values in stones.

Added column headings to the "Browse Public Accounts" form. The fields in this form should mostly be centred, but I deferred that for the moment until we're sure the current set is complete.

The `monthlog::editWeight` function mis-handled negative weights in stones and pounds mode. I fixed this, and changed the output in all units to use the XHTML `&minus;` entity for the minus sign on negative quantities.

Fixed a zero-based/one-based bug in saving the diet calculator settings which caused the start and end dates to walk forward one month and one day every time they were saved.

Calculation of the rate of weight change failed on initial display of the diet calculator when the energy unit was set to kJ—fixed.

After further reflection, I decided to modify the diet calculator to always keep the primary quantities saved in the `user` structure in terms of calories and kilograms, and convert to whatever energy and display weight unit the user has set when they are loaded, converting back when they are updated. This requires care to prevent rounding problems, but avoids all difficulties with the user changing units which make the saved diet calculator settings meaningless.

2007 April 9

Implemented plotting the diet plan in monthly charts. The plan is plotted as a dashed yellow line.

Implemented plotting the diet plan in historical charts. Verified that it plots correctly with units set to kilograms, pounds, and stones.

2007 April 10

Backed out the gimmick in `monthlog::editWeight` which replaced "`-`" with "`&minus;`". The latter looks nice, but JavaScript doesn't understand how to parse numbers with a leading Unicode minus sign.

2007 April 12

Legacy Excel CSV exports were missing the carriage return line terminator on the column heading lines—fixed.

2007 April 13

Added logic to the "Update" button handler to detect if more than one field has been changed and issue a warning that this may result in infelicitous results.

2007 April 15

More whacking away at handling static updates to the diet calculator. Fixed a number of lost-through-distraction problems with weight and energy unit storage in the `user` object vs. their display values in the diet calculator form. Many bugs have been fixed, but much remains to be done.

2007 April 16

Fixed a rounding error when parsing the diet duration in weeks in a static update to the diet calculator—it no longer "creeps" upward.

Parsing of the existing start date in a static update of the diet calculator was completely bogus—fixed.

At this point I am now ready to assert that diet calculator static (no JavaScript) updates are correct. Much more testing is required to justify this triumph of confidence over confirmation but, hey, I haven't found any bugs in my five or ten minutes of testing so far!

Turned off, for the nonce, debugging output from static diet calculator updates. It'd still all in the code, commented out, and can be re-enabled as required when arthropods exsquiggle from the cracks in our crystalline code.

2007 April 17

Added code to log failed login attempts with `Sys::Syslog` in a format which will cause Gardol to treat them like failed FTP logins and eventually block the sending IP address after too many consecutive failures.

2007 April 18

Implemented static dittoing of rung and comment fields in monthly logs and dittoing and abbreviated entry for weight fields for user without JavaScript. All of the logic is implemented in the `monthlog::updateFromCGI` method and simply ignores the entry if no previous entry can be found.

Bashed in a rough but usable feedback page so beta testers can send feedback without running the gauntlet of the regular feedback form. I may not be able to resist the temptation to spiff this up with JavaScript, a preview button, etc. but on the other hand I may.

2007 April 20

The `quoteHTML` function did not work on arguments which contained more than one line of text—fixed.

Added an echo of the message sent to the feedback confirmation page.

2007 April 21

Added a checkbox to the Send Feedback form which causes a copy to be sent to the user who submitted it.

Corrected a validation error in the feedback form due to an extraneous table row tag.

Dern tootin' I couldn't resist putting in a preview feature for the feedback form. I can hardly gripe about badly composed messages if I don't let folks see how they're going to look. I also put in a little JavaScript validator which natters if the user has not chosen a message category before pressing the "Preview" or "Send Feedback" buttons. The categories have been moved up from being embedded in the feedback form to the feedback configuration section so it's easy to find them when the time comes to revise them.

Added a type 16 history log item for the sending of a feedback message, including the category as text.

2007 April 22

Added a "`production` target to the `Makefile` to install the application in the production server directory. This uses variants of the various directory definitions with a "Production" prefix.

Fixed unequal separation between the previous and next month buttons and the month and year heading in the monthly log form.

Deleted all of the commented out "button, button" tags remaining from the fix for the Exploder button problem. See the third paragraph under March 30th above for details.

Added a `Bowdler.pl` program to bowdlerise the source code before publication and distribution. All sensitive information (for example passwords, "salt" for encryption, etc.) are changed to innocuous strings. The program is, of course, able to bowdlerise itself, becoming the identity transform.

Implemented the `history::syntheticData` method, which fills a field in a specified date range with synthetic data specified by exquisiteky subtle arguments. This is presently crudely invoked from a no-parameters item in the administrator account page; this will be replaced by a form with all parameters variable in due course.

2007 April 23

Implemented a "`checkSecure`" JavaScript function which validates that the current page has been invoked with an "`https:`" URL and, if not, pops up an alert warning the user of the possible security risk. This is currently used only on the login pages; it may be extended to other pages if a justification for such paranoia can be ginned up. The check is brutally suppressed when testing development versions on Server0.

2007 April 24

Removed the explicit "`http:`" on the URLs used to reference the logo images in the heading of CGI result pages. When the application is invoked from an "`https:`" URL, Exploder complains about "mixed secure and insecure data" if any content on the page is included with a non-HTTPS URL.

A nonzero variance less than 0.1 units would display as "0.0" or "−0.0" in red or green. I modified it so that any variance which shows as 0.0 will always appear in blank. Further, variances were not right justified in the monthly log column, so variances without a sign did not line up properly; they're right justified now.

The synthetic data generator fell flat on its face when attempting to generate data for a month not already present in the database. I added logic to create new months not present in the cache as data generation proceeds.

2007 April 25

Implemented read-only logins to demonstration accounts. An account must be marked as read-only in the `user` object to be accessed this way. If so marked, users can log in with a blank password, creating a non-exclusive read-only session. All operations which modify the database are silently ignored when logged in read-only, allowing the visitor to go through the motions without actually changing anything.

Added column headings to the administrator account manager table.

Brain damaged Exploder does not understand a link wrapped around an input button—it lets you press the button but does precisely nothing. I converted the "Exit" buttons in the administrator and public user browse notifications to little forms wrapped around input items which Exploder appears able to comprehend.

The XML DTD and stylesheet specifications require an absolute URL, which was broken by our going to relative URLs in the interest of transparent support for HTTPS. I added explicit, fully qualified, URLs for these items.

2007 April 26

The trend analysis form labeled both the starting and ending custom dates as "From"—fixed.

2007 April 29

The exercise rung field was missing from Excel CSV export files—fixed.

Added display of the user's real name to the "Send Feedback" form. It is displayed in the form used by most E-mail clients, with the full name first and the E-mail address in angle brackets.

Added body mass calculation and display to historical charts. As with monthly charts, the body mass index is calculated only if the height is specified.

The diet calculator could not be displayed when browsing a public account because I forgot to explicitly allow the transaction in `%browsing_user_requests`. I added it, along with logic to remove the action buttons from the diet calculator when displayed by a browsing user. (They wouldn't do any harm due to the transaction filtering, but there's no reason to confuse the user by showing them.)

Announced the opening of the beta test program.

2007 April 30

An undefined variable warning was issued the first time the feedback form was displayed due to a typo in the test for whether a category was already specified—fixed.

Added a "Browser" header line to feedback E-mail sent to the feedback address (but not the copy sent back to the user, to avoid confusion). Since we're likely to get reports of browser-specific problems, this identifies the browser (from the "`HTTP_USER_AGENT`" environment variable, if present) in case the sender omits this detail.

Plotting of a "flat-line" diet plan in a monthly log failed to convert the kilogram quantity from the `user` object into the display unit for the chart—fixed. (Reported by Rob Campbell.)

Further, when the log and display unit differed, the trend carry-forward (which is kept in the log unit) was not converted to the display unit, resulting in a vertical scale which squashed the actual data into a small part of the vertical extent—fixed.

Changed the link on "The Hacker's Diet *Online*" in the centre of the title to point to the Online documentation, not the main page for the book, which is already linked to the title page logo in the right part of the title. (Reported by Rob Campbell.)

When displaying a log for other than the current month, the "Log" item in the navigation bar was highlighted and disabled, which prevented it from being clicked to display the current month. I added logic to test whether the current month is displayed (this, as the handling of the "`now`" month specification, is presently based on UTC and needs to be reviewed with regard to `tzoffset`). If the month displayed is not the current month, the "Log" item may now be clicked to jump to the current month. (Reported by Rob Campbell.)

2007 May 1

Added display of both the log and display weight unit to the Account Manager form. The "Weight" column now shows "*log/display*".

Added a little JavaScript glue to make it less likely new users will accidentally set the log and display units to different values. When setting the log or display weight unit in the new account form (but not the edit settings form for an existing account), checking a radio button in either group will set the corresponding button in the other group unless it has already been explicitly set.

Added a dynamically updated build number and build time and date to the `Makefile`, which is included in the "Utilities" page in Beta Test mode and always in feedback messages. This identified the version to the tester and the version which sent a feedback report.

2007 May 2

The `get_selected_date` JavaScript function ran afoul of another eccentricity of "Internet Exploder" (both 6 and 7). All reasonable browsers interpret the `value` field of an `Option` object within a selection field as the value which will be sent to the server when the value is selected, regardless of whether it was specified as the text within the `option` tag or as a `value=` attribute within the tag itself. But not Exploder—if you specify the value by the text within the tags, the `value` property is blank; you have to use the `text` property to obtain the value. This resulted in changes to the diet calculator causing the year within the start and end dates to warp back to 1900. I modified the function to use the `text` property (which, thankfully, also works in Firefox and Opera. (Reported by Robert Ewing.)

Changed the background of the monthly and historical charts to "`rgb(160, 160, 160)`" to make the green float/sinker lines stand out better against the background. (Reported by Bruce Lokeinsky.)

The check for whether the domain name in an E-mail address was valid tested whether the domain could be resolved to an A record, which caused mail-relay-only domains, which have one or more MX record but no A records, to fail. I created a new `validMailDomain` function which uses `dig` to query the domain for MX records, filters the result, and returns Boolean `true` if the domain has one or more valid MX records. The validation of E-mail addresses in both new account creation and modification of an existing account now call this function to verify domain in the E-mail address. (Reported by Joshua Carpoff.)

If a transaction request which assumes an active session was sent with no session identifier at all, a harmless "uninitialised variable" warning was issued. I fixed it to set the session ID to the null string in this case. The invalid session warning will continue to appear in the HTTP error log, but the Perl warnings which preceded it have now been banished.

The state of the "Send me a copy of the feedback message" check box was lost when a preview of a feedback message in progress was displayed. I fixed it to preserve the state and check the button if it was checked in the sending transaction. (Reported by Robert Ewing.)

Added the user's full name (if entered) to the feedback message sent to the designated feedback address. This allows giving credit for feedback message without looking up the user's full name in the Account Manager page.

2007 May 3

Added a message to the sign in page, conditional on "Beta Test", that provides a link to the development Web log.

Fixed an error in displaying the user name in a feedback message which caused it to be concatenated with the "From" line.

The Opera browser and some intermediate Web cache servers appear to ignore our specification of an HTTP header item of "`Cache-Control: private`" for all dynamically-generated documents. In particular the "stateless" chart images embedded in the monthly log and chart workshop pages, purely on the basis that the URL is identical, are served an obsolete image from the cache rather than requesting an image from the server which would reflect changes to the database made of late. OK, you want a different URL? Here's a different URL! On all of the embedded image URLs, I have added a "cachebuster" argument which is a pseudorandom value with no function other than ensuring each image request URL is unique. This definitely fixes the cache problem with Opera, which was consistently reproducible; we'll have to see it if is equally successful with other rogue caches which ignore the `Cache-Control` directive. (Reported by Robert Ewing.)

2007 May 4

If the calorie balance and desired weight change in the diet calculator page had different signs, the "**End date**" in the form would display "*Never*" in the JavaScript code, but if the user pressed the "Save" button, it would inanely show an end date before the start date. I added code to the diet calculator form generation to preset the display modes of the "**End date**" based on whether it is before or after the start date. Moving the mouse over a "*Never*" end date will open a date entry field—you still should be able to adjust the calorie balance to achieve a goal date even if the previous specification went in the wrong direction. (Reported by Robert Ewing.)

The initial chart displayed by the Chart Workshop was 800×600 pixels but the default selection for the "Chart size" item was 640×480, so if you just pressed the "Update" button without changing any of the selections, the chart displayed would shrink. Even more jarring, in a 800×600 chart with the default interval of one quarter, daily weights can be plotted as floats and sinkers, but at 640×480 they don't fit, so the grey line format is used. I changed the initial selection to 800×600 to agree with the default on initial display of the page.

If a user entered a weight entry on a monthly log in which there was no trend-carry forward (i.e. the first log in the database), which thus defined the trend as starting with the first weight entered, then entered subsequent weights, and finally went back and deleted the first weight in the log, the trend was not redefined as starting with what has then become the first weight in the log. This problem refers exclusively to the

JavaScript live update code—once the log was saved in the database, the trend would be computed and displayed correctly. I modified the trend re-propagation code in the `changeWeight` function to "propagate" a blank trend downward until the first nonblank weight is encountered, then define the trend as starting at that value. (Reported by Robert Ewing.)

When importing Excel CSV records, records which were not imported because they would overwrite records already in the database (when the "Overwrite" box is not checked) were handled as if they didn't parse and were passed down the chain to the Palm and native CSV import parsers. I added a flag to indicate a record was rejected due to a conflict with preexisting data which prevents this from happening.

2007 May 5

Excel CSV records with a blank or zero specification for the Flag field resulted in an "`Argument "" isn't numeric`" warning because missing values were set to the null string instead of `undef`—fixed.

Unified the cached log retrieval code in Excel CSV, Palm/`hdread` CSV, and XML import.

Commented out the "`Barfel`" and "`Garfel`" debug code in CSV and XML import parsers.

Uploading of CSV files with embedded ISO 8859-1 characters failed because standard input was put into UTF-8 mode prior to reading the POST data for the transaction request. I modified the CGI argument processor to check for the string "`enc=raw`" in the `QUERY_STRING` argument. If present, the POST arguments are read in "`raw`" mode, bypassing the UTF-8 input decoder. Note that one can specify a query string on a URL even if it is a form submission with multipart arguments in POST format.

Integrated the `bump` utility, used to increment the build number each time `hdiet.w` is extracted, into the main program web.

2007 May 6

Fixed a misplaced colon in the "Read only" item in the `session::describe` method.

Added check boxes to the bottom of the monthly log in administrator mode to dump the `user` and `session` objects as well as the `monthlog` object.

Added the ability for the administrator to dump objects from the monthly log form when accessing another user's account as well as within the administrator's own account.

Performed HTML quoting on the object dumps performed from the monthly log form. Added a "`unicode-bidi: bidi-override;`" CSS style specification to the `<pre>` block which encloses the object dumps so that right to left text does not befuddle the browser's layout of the object dump. A new `html::quoteHTMLFile` function is used to quote the object dumps, which are written to intermediate temporary files.

Improved formatting of the `describe` method output for the `monthlog` and `user` objects. These previously had some long lines which are now wrapped.

2007 May 7

A user who entered monthly log data infrequently could receive a warning from the JavaScript code, which considered a difference of ±6% between the weight entry and the current trend value indicative of a possible error in the weight. With infrequently entered weights, the trend update falls behind, which can lead to warnings for legitimate weight entries. Now, the first thing to note here is that this is just a warning—the user can accept the entered weight value simply by clicking "OK". Secondly, *The Hacker's Diet* encourages readers to record their weight daily if possible, not at infrequent intervals, because the more frequent the weight measurements, the more closely the trend will reflect the actual smoothed weight. I modified the plausibility check for weight entries in `hdiet.js` to first test for a discrepancy between the entry and the trend as before. If this indicates a possible error, we now look for a previous weight entry in this month and, if one is present, use that instead of the trend to compute the variance (since a previous weight entry can be presumed to either be in range or to have been previously confirmed as correct by the user). If no previous entry has been made for this month, we extrapolate the trend as having been evolved by a linear

change from the trend at the start of the month to the weight entered, then compare the weight entered against that simulated trend value. (Reported by Lorenzo Emilitri.)

2007 May 8

If the diet plan was outside the date range plotted in an historical chart and consequently reported −1 for the plan weight, it was still used to scale the weight axis in historical charts, squashing all of the weight entries toward the top of the chart. I modified the automatic scaling in `history::drawChart` to ignore these out of domain values and only include diet calculator values in the scale computation if they actually contribute data to be plotted in the chart. (Reported by Lorenzo Emilitri.)

2007 May 9

Deleted numerous long-commented-out diagnostic messages to `STDERR`.

Calculation of trend analysis for the caption of historical charts was incorrect for periods with sparse weight entries. Instead of replicating the trend for days without weight entries as `history::analyseTrend` does, the chart generator only released trend values to `addPoint` for days which had weight entries. Since the trend fitter assumes continuous data points, this resulted in a gross overestimate of the slope of the data, resulting in inflated weight and energy balance values in the chart caption. I modified the `history::getDays` function to fill in trend values for days with no weight entry as `analyseTrend` does. Values in chart captions now agree with those calculated by the trend analysis page for equivalent intervals. (Reported by Lorenzo Emilitri.)

Added code to process the `HDiet_tzoffset` variable set by the JavaScript code to the user's local timezone offset with GMT. This code sets variables at the transaction global level which contain the local time (in Unix time format), and the civil wall clock time at the user's site. All direct links between pages now propagate this variable along with the session ID, so that any page which requires it should have access to this value. Note that if the user does not have JavaScript enabled or the has a ridiculous time zone setting, `HDiet_tzoffset` will be set to "`unknown`" and the local time variables will be set to UTC.

Changed all instances in which a UTC date was used to decide what to display (for example, the current month's log immediately after a sign in) to use the date in the user's time zone as determined above. This fixes the problem where, for example, a log is displayed for the the first day of a month while, in the user's time zone, it is still the last day of the previous month. (Reported by Robert Ewing.)

2007 May 10

The `countChange()` and `leaveDocument()` functions in the JavaScript code which warn a user about to navigate away from a monthly log page with unsaved changes continued to issue the warning if the user had reversed all the changes with the "Reset" button. I added an "`onclick`" event handler to the reset button which resets the `unsavedChanges` count to zero.

Created an icon for the dynamically generated pages in a new `winicon` directory and added a `link` reference to it in the standard XHTML header.

Changed the main Fourmilab logo in the page title from a GIF image to the PNG used on the site home page.

2007 May 11

Added computation of the trend minimum, maximum, and mean values to the `trendfit` object, and display of these quantities for each of the intervals in the Trend Analysis page. (Suggested by Rob Campbell.)

The selection fields for custom intervals in the Trend Analysis and Chart Workshop pages were preset to the first and last month in the database, but the day in these items was always set to 1. I modified them to show the first and last days in the respective months. I also unified the code which generates these fields for the trend and chart pages; it was almost identical before and is now common.

2007 May 12

The JavaScript `updateVariance` function, which updates the variance field when a weight entry is changed, had two problems. First, it did not round the variance to the usual one decimal place before deciding whether it was positive, negative, or zero; consequently, it could consider a value which displayed as "0.0" as signed and prefix a sign to the number. Second, if the variance was zero, it was displayed in red as opposed to black, as the CGI form generation does. Live variance updates from the JavaScript code are supposed to display in colour according to their sign. This works in competently implemented browsers such as Firefox and Opera, but not in Exploder. So far, I have not found any work-around to fix this, so, at the moment, live updates to the variance will always display in black in that regrettable browser.

A blank (as opposed to null) flag field in an Excel-format CSV import resulted in an "Argument isn't numeric" warning. I modified the Excel CSV import code to treat all-blank flag fields as if they were explicitly set to zero.

2007 May 18

Signs on variance items updated by the JavaScript code were always positive for nonzero numbers because the sign editing code tested the absolute value of the variance instead of the original signed quantity—fixed. (Reported by Robert Ewing.)

2007 May 23

Integrated the CPAN `Digest::Crc32` module as `HDiet:::Digest::Crc32`, providing access to it without the need to install it on the host server. This will be used for validity checking of cookies.

2007 May 30

Added a summary line to the administrator account manager page which shows the total number of accounts, how many of those accounts grant public access, and what percent of accounts allow public access.

2007 May 31

The `expandAbbreviatedWeight` JavaScript function failed with entries containing leading or trailing spaces because the pattern matches which were intended to elide them never stored the result back into the string variable—fixed.

Added logic to the JavaScript code to accept either a comma or period as the decimal character in all contexts where decimal numbers are permitted: weight entries in monthly logs, height in centimetres or inches in the account settings page, and weight items in the diet calculator.

Modified `monthlog::updateFromCGI` to accept either comma or period as the decimal character in monthly log weight entries.

Modified the `parseWeight` subroutine to accept either comma or period as the decimal character. This allows either decimal character to be used in static updates to the diet calculator when JavaScript is disabled.

Added code to the parsing of height in centimetres and inches in CGI account settings to accept either a comma or period as the decimal point character. This handles account setting changes when JavaScript is disabled.

2007 June 1

Replaced the four separate instances of the `in` function in the persistent objects with calls on a new "Read line from persistent object file" macro, which generates a custom `in` function for the object, taking its name as a macro argument (for error messages). The new `in` function has an optional second argument which supplies the default value to be returned if end of file is encountered when attempting to read the field from the stored object. If no second argument is given, the function will abort on end of file, as before.

Added a `decimal_character` field to the `user` object. This is set to "`.`" or "`,`" as the user's preference for the decimal separator character. A new `localiseNumber` method within the `user` object edits a number in the same fashion as the `canonicalNumber` function, but uses the user's preferred decimal character if the value contains decimal digits.

Modified the `canonicalNumber` function in the JavaScript code to accept an optional third argument which specifies the user's preferred decimal separator character. The separator is passed to the JavaScript code by a new `decimal_character` hidden field in the account settings form, and used by the dynamic update code for the centimetre and inch fields in the Height settings item to display decimal values with the configured separator.

Modified the JavaScript `expandAbbreviatedWeight` function to handle abbreviated entries with commas as well as periods as decimal characters. A comma can be used in any context where a period was used before, including by itself to copy the previous entry.

Added a hidden `decimalCharacter` argument to the JavaScript `editWeight` and `updateVariance` functions. The variable is set from a hidden form field "`dc`", which is passed to forms which need to edit decimal numbers (presently the Monthly Log and Diet Calculator pages).

When JavaScript was disabled, using more than one exercise rung abbreviation in a row caused all but the first to be set to 1 because the abbreviation was not stored back into the CGI arguments hash—fixed.

The exercise rung increment and decrement shortcuts (entering a plus or minus sign in the rung field to specify a rung one greater or less than the most recent entry) did not work when JavaScript was disabled—fixed.

Copying more than one previous comment in a monthly log when JavaScript was disabled failed for all but the first copied item—fixed.

Added a test to `validMailDomain` to handle domains with no `MX` record but which have a valid `A` record. This copes with ill-configured sites which run a mail exchanger but do not have an `MX` record pointing to itself.

At this point, comma should be accepted as an alternative decimal separator character to period in all contexts in which numbers with decimal places are entered, and the "Decimal character" setting should affect the editing of all numbers displayed on user pages. The closes the feature request by Jens Peter Bork for this item. Note that numbers in exported CSV and XML files always use period as decimal separator, and files imported must be in this format.

2007 June 5

Added a new `handheld` field to the session object. This will be set if the user specifies that the session is being conducted from a handheld device such as a PDA or mobile telephone.

Added columns to the Session Manager form to indicate whether a session is read-only or from a handheld device.

2007 June 10

Importing Excel CSV records with four-digit year fields failed because the test for two-digit years greater than 88 failed to also verify that the year was less than 100. Consequently a year of, say, 2006 would be deemed a specification of year 3906, which would be rejected by the sanity check. I rewrote the test for two-digit years and, in the process, simplified the code. I also made the Excel CSV diagnostic code, which previously had to be commented on or off, conditional on a new `$excelCSVdebug` variable, so simply changing this variable enables or disables all of the diagnostic output. (Reported by Jennie Koffman.)

2007 June 11

Added a "Handheld device" check box to the login form which, if checked, opens a session with the `handheld` attribute in the session object set. The state of this box is not in any way persistent across failed login attempt or other misadventures—this can be tweaked as the handheld support matures. It should be integrated with our "`handheld=y`" query string plans for direct access to a special handheld login form.

If handheld mode is set, the chart in the monthly log defaults to 320 × 240 pixels as opposed to twice that in each dimension for a regular desktop display.

2007 June 12

Implemented a "handheld=y" query string option on the login page. If this is set, the "Handheld device" checkbox is preset and the login form will be reformatted for a small handheld screen. Either this special string (intended to be set in a stored URL) or the `HDiet_handheld` form variable will trigger this mode, so it will persist on subsequent forms for failed login attempts.

Added a `$handheld` argument to `write_XHTML_prologue` which generates a streamlined prologue and links to the lightweight `hdiet_handheld.css` style sheet instead of the massive style sheet for regular screen presentation.

2007 June 13

If the historical chart size was set to 320 × 200 the actual chart would be 320 × 320 due to an incorrect test in the chart size sanity check—fixed.

When the width of an historical chart was set to less than 480 pixels, the first line of the caption was truncated. I added an abbreviated caption which is generated in this case. The body mass index caption barely fits, but there's no reason to abbreviate further unless we allow charts less than 320 pixels wide.

Changed the smallest historical chart size to 320 × 240. With the caption at the bottom, this leaves more room for the actual chart.

2007 June 15

Added a "Remember me" checkbox to the login form. This is persistent across failed login attempts.

Fixed permissions on `HDiet/Digest/Crc32.pm` which caused it not to be found when installed on the server.

Added a `cookie` field to the `session` object to record if this session was initiated by a cookie login. The status of this field is shown in a new column in the Session Manager.

Added a `jd_to_old_cookie_date` function to the `Julian` package which converts a Julian day to the eccentric date and time format used in old-style HTTP cookies. I also added an export for the `jd_to_RFC_3339_date` function which has been accidentally omitted.

2007 June 16

Added an option to the Monthly Log form when viewed with administrative privilege to dump the CGI arguments and environment string used to invoke the form.

2007 June 18

Added fields to the login history record to indicate whether the session is on a handheld device and if the login was done via a cookie.

The initial implementation of "Remember me" is now in place. The sign in form now includes a checkbox which selects "Remember me" mode, which (unless the user is logging into a read-only account), drops a cookie in the user's browser and stores a corresponding token in the "`RememberMe`" directory which records the user name, time of creation, and time of expiry. When about to display the sign in form, we check whether the browser sent a persistent login cookie. If so, and a token is stored with its code, we automatically sign in the user and proceed to the monthly log page without the need to enter a user name or password. When a sign in via cookie is performed, the cookie used is revoked and a new cookie is assigned. Thus, only the most recently cookie is valid; a previously intercepted and stored cookie is useless.

A sign out takes the user to a sign in page via a special "`newlogin`" transaction which bypasses the automatic cookie login. This allows the user to uncheck "Remember me" and revoke the cookie (for this browser).

When logged in via a cookie, the "Settings" page is inaccessible. This keeps a user who somehow manages to hijack a cookie from changing the user's password or disclosing identity information. A user who has logged in with a cookie can log out, log back in with their user name and password, and then access the settings page.

A new "Forget persistent logins" item in the Utilities page permits a user to delete all stored "Remember me" tokens. This will invalidate all cookies stored in browsers for this user.

A new administrator "Manage persistent logins" page shows all persistent login tokens. The administrator can delete any persistent login token by checking it and pressing the "Delete" button, specifying the administrator's password.

2007 June 20

Revised the `dist` target in the `Makefile` to Bowdlerise the source distribution and rebuild all derivative files from it. The PDF is now properly generated after running LaTeX to build the cross-references for the Bowdlerisied edition. We still need to verify that the complete application can be rebuilt in all circumstances from the files in the source distribution.

2007 June 21

Made display of the build number not conditional upon beta test mode. The item is sufficiently unobtrusive at the bottom of the utilities menu page and useful to knowledgeable folks checking for changes that it's worth leaving on in production.

2007 June 22

The `monthlog::updateFromCGI` method referenced undefined rung CGI variables when called from a form in which some or all of the rung fields were non-edit fields (as in a read-only or printer-friendly log page)—fixed.

Added "Printer friendly" and "Monochrome" checkboxes to the monthly log page. These set "`print`" and "`mono`" CGI arguments which are propagated to navigation links and the embedded chart. These choose CSS classes which re-format the monthly log for a printer and, if set, monochrome output. In printer friendly mode, the monthly log table is a non-editable static table with collapsed borders.

2007 June 23

Modified sorting of items in the administrator account, session, and persistent login manager forms to be case-insensitive (or at least follow the order which the Perl `lc()` function produces).

The `monthlog::updateFromCGI` method would clear existing flag and comment items when processing a log in which the edit fields were not defined (for example, one with protected days or a printer-friendly table). I modified the code to skip all updates if the day's rung or comment field was not defined in the CGI arguments.

Added "Printer friendly" and "Monochrome" checkboxes to the Chart Workshop and passed them on to `history::drawChart` to apply those modes to the historical chart.

Wrapped the "Custom" radio button in the Chart Workshop and Trend Analysis pages in a `<label>` tag with its label. This allows clicking the label as well as the button and assists non-visual browsers in identifying the control.

Added `<label>` tags to the radio buttons in the Export Log Database form.

Added `<label>` wrappers to the administrator object dump checkboxes at the bottom of the Monthly Log form.

2007 June 25

Replaced the two separate instances of generation of the "cachebuster" argument for embedded monthly log and historical charts with a common macro and added documentation as to why it is needed.

2007 June 28

Made the feedback form not conditional upon beta test (but left the test in the code, commented out). We'll leave the feedback form in for the nonce as we transition from beta to production.

Built a test version with beta test set to zero.

Changed working version number to 1.0 for production release.

2007 June 30

A Palm Eat Watch CSV record which included leading or trailing spaces in the Date, Weight, Rung, or Flag fields would be ignored or, in the case of the Flag field, interpreted incorrectly. I added code to discard all white space in these fields, as none should be present. (Reported by Reed Lipman.)

2007 July 1

Added code to disallow browsing of publicly-visible account by users logged into read-only demonstration accounts. This restricts access to public accounts to those users who have gone to the trouble of creating an account of their own. This is enforced not only by removing the "Browse public user accounts" item from the Utilities menu for read-only accounts, but also aborting transactions ginned up from a read-only login with the transaction codes for public account access.

2007 July 2

When creating a new monthly log, the code which fills in the trend carry-forward from the last trend value in the most recent existing log failed to convert the trend value from the log unit of that log to the log unit of the new one. This resulted in wild variances if a user changed the log unit and then entered data in a new log. I added unit conversion for this case, which was already handled correctly for the case of complete recalculation of trend carry-forwards (and hence can be used to correct any existing problems due to this bug). (Reported by Eric Carr.)

History records for CSV/XML import transactions were not being generated because the test for a read-only session was backwards—fixed.

2007 July 21

Completed implementation and began production test of cluster file system synchronisation support for server farm architectures such as Fourmilab's. Cluster support is implemented in the new `Cluster` module, through functions such as `clusterCopy`, `clusterDelete`, `clusterMkdir`, etc. When a database file or directory is modified, immediately after the modification is made, (for example, after the `close()` when writing back a file), the corresponding cluster function is called with the full path name of the modified file. This then calls `enqueueClusterTransaction` with the specified operation and path name, which creates one or more synchronisation transaction files in the `ClusterSync` directory, within subdirectories bearing the names of the servers defined in "Cluster Member Hosts". (Transactions are never queued for the server executing the transaction, nor for servers named as cluster members for which no server subdirectory exists. This allows you to have identical directory structures on all servers, or to exercise fine-grained control over which servers are updated automaticallly [for example, if you wish to reserve one server for testing new releases and not have changes made on it propagated back to the production server]).

Synchronisation transaction files are named with the current date and time to the microsecond, a journal sequence number which is incremented for each transaction generated during a given execution of the CGI application (to preserve transaction order in case the time does not advance between two consecutive transactions), and for easy examination of the synchronisation directory, the operation and path name, the latter with slashes translated to underscores. The contents of the transaction file is a version number, the operation, and the full path name.

Actual synchronisation is accomplished by a separate, stand-alone program, `ClusterSync.pl`, which runs under group and user `apache`, which is the owner of the `ClusterSync` transaction directory and its contents.

This program is started automatically from the `init` script and runs as a daemon, saving its process ID in a `ClusterSync.pid` file in the `ClusterSync` directory.

When a synchronisation transaction is queued, the CGI program sends a `SIGUSR1` signal to the `ClusterSync.pl` process, which then traverses the server subdirectories, sorting the transactions into time and journal number order, and attempts to perform the operations they request. Synchronisation operations are performed by executing `scp` and `ssh` commands directed at the designated cluster host, which must be configured to permit public key access by user `apache` without a password. If the synchronisation operation fails with a status indicating that the destination host is down or unreachable, the host is placed in a `%failed_hosts` hash with a timeout value of ten minutes from the time of failure. Synchronisation operations for that host will not be attempted until the timeout has expired, which prevents flailing away in vain trying to contact a down host over and over, possibly delaying synchronisation of other cluster members which are accessible. In the absence of a signal indicating newly-queued transactions, `ClusterSync.pl` sweeps the transaction directory every five minutes to check for transactions queued for failed hosts which should now be retried due to expiry of the timeout.

All of the directory names, signal, and timeout values given above are specified by items in the "Host System Properties" section of the configuration; I have given the default settings, which should be suitable in most circumstances.

You can check whether two cluster hosts are synchronised by logging into one host, say `server1`, and then running a command like:

```
rdist -overify -P /usr/bin/ssh -c /server/pub/hackdiet \
    server0:/server/pub/hackdiet
```

This will report any discrepancies between the database directory trees on the two servers. If the servers are synchronised, you should see only a "need to update" message for the `ClusterSync/ClusterSync.pid`, plus any synchronisation transactions queued for failed servers awaiting retry. This operation is non-destructive and requires only read access to the database directory.

2007 July 22

Added date and time to the first line of the log items written to standard output by `ClusterSync.pl` when `$verbose` is set and set standard output to "autoflush" mode so log items are written immediately regardless of redirection.

Added configuration parameters which allow `ClusterSync.pl`, if started as super-user, to change to a designated group and user identity. Running a Perl program under an assumed identity turns on the "taint" mechanism, so input from the transaction directory and the files within it is sanitised before being used in potentially dangerous ways (even though it should, in fact, only be coming from the CGI application, never the "outside").

2007 July 23

Added much more stringent validation to `ClusterSync.pl` transaction processing. Every file name submitted must begin with the "Database Directory" path name, and may not contain abusive (shell-interpreted) characters or sequences such as "`..`". In addition, all input from transaction files is single quoted when used on `system()` commands to prevent attack by overlooked shell escapes. Finally, an invalid transaction type in a transaction file causes an immediate abort. Now, since we're basically using the transaction directory as an interprocess communication channel, this might be deemed paranoia, but "you can't be too careful". Besides, one can imagine an attack where somebody manages to hijack another CGI application and trick it into adding bogus transactions to the directory which cause `ClusterSync` to do its dirty work for it.

Added an SHA1 signature as an additional line in cluster synchronisation transaction files. This signature incorporates the content of the transaction as well as our site-secret "Confirmation signature encoding suffix", without which it is unlikely in the extreme an attacker will be able to spoof transactions. Signature failure

crashes ClusterSync, alerting the administrator that something untoward is underway and thwarting an attacker who contemplates a brute-force search for the suffix.

2007 July 24

If a log had an unspecified trend carry-forward and the previous log in the database was present but had no weight entries whatsoever, "Fill in trend carry-forward from most recent previous log, if required" would hang in a CPU loop due to a backwards-coded loop termination test. If there were any log entries, the loop would bail out due to a last, but for an empty log the termination when the beginning of the log was reached would never occur and the program would crash when the CGI time limit expired. I corrected the loop termination test and verified that the hang no longer occurs for a blank previous log. (Reported by Andres Kievsky.)

Added a handler for the INT signal which, when received, prints a stack trace to STDERR (which will thus appear in the HTTP server error log) and terminates. This simplifies the task of debugging CPU hang or other problems which lead to a CGI program timeout and the resulting 500 response to the requester.

The "Julian date constant definitions" macro had an incorrect name. Its name had accidentally been left the same as the support functions macro, which worked fine since the references to the two macros are consecutive. I corrected the name to get rid of a harmless warning message.

Modified the Makefile publish and production targets to install the cluster synchronisation program as an executable named ClusterSync in the server*n*/bin/hackdiet directory. This allows it to work without modification with our standard /server/init mechanism, in particular a new /server/init/hackdiet script which starts and stops the cluster synchronisation process.

Moved the process ID file for the cluster synchronisation process to /server/run/ClusterSync/ClusterSync.pid to conform with our standard structure in the /server partition.

Implemented a proper log file for ClusterSync.pl. The full path name for the log file is configured with "Cluster Synchronisation Log File". If the null string, logging is disabled. Otherwise, the specified file is opened for appending, and items are appended for each transaction. When logging is active, the program listens for the HUP signal and, upon receiving it, closes and re-opens the log file to permit it to be rotated by renaming it and then sending the signal. The format of the log file identical to the information written to STDOUT when $verbose is nonzero. The default location for the log file is /server/log/hackdiet/ClusterSync.log.

2007 July 25

Replaced all of the parallel calls in ClusterSync.pl to write output to standard output in verbose mode and to the log file when logging with calls on a new logmsg function which writes its arguments to the appropriate destinations according to the global option variables.

If the start or end date of a diet plan in the diet calculator were outside the union of the range of years in the database and the current year plus one, the start and/or end year of the diet plan would not be included in the start and end date selection boxes, resulting in an incorrect date appearing in the form. This most often manifested itself when a long-term diet extends past the end of the year after the present. I added logic to make sure that the range of years included in the start and end date boxes includes the least of the current year and the first year of the diet plan minus one and the greatest of the next year and the last year of the diet plan plus one. (Reported by Jim Hollcraft.)

2007 July 27

Commented out the etime() utility function which is presently used nowhere.

Wrapped the checkboxes and labels for the "Allow overwrite" and "List imported records" options in the Import CSV/XML page and the "Plot plan in chart" item in the Diet Calculator with <label> containers so that the labels as well as the checkboxes can be clicked.

Arghhh! Perl 5.8 relies upon the underlying C library's gmtime function for the Perl gmtime function. This means that on a 32-bit platform the Perl function is limited to dates between the start of 1970 and

"doomsday", 2038-01-19. (I understand that this problem does not exist on native 64-bit platforms and will be fixed in Perl 6.) Even though we have some time to go until the tick of doom, it is easily possible to generate dates beyond 2038 by entering small calorie balance values in the diet calculator. I added a new `Julian::unix_time_to_civil_date_time` function which uses the Julian day functions to convert a Unix `time()` value to a list of year, month, day of month, hour, minute, seconds (actual values, not the crazy offsets returned by `gmtime`, so this is not a drop-in replacement). I replaced all references to `gmtime` in the program to calls on `unix_time_to_civil_date_time`, which corrects the original problem reported in the diet calculator. There are a few calls on `localtime` left in the code, but these are all in `describe` methods for various objects (used only for administrator debugging output, and all representing times close to the present) and in the generation of log entries, which are also obviously in the present. Since these won't break for more than thirty years, it's likely we'll be on a version of Perl with the truncation fixed before then or, failing that, there's plenty of time to fix them before the dawn of the dreaded day. (Reported by Jim Hollcraft.)

The JavaScript live update for the diet calculator rounded the diet duration in weeks differently from the Perl code: JavaScript truncated to the next lower integer, while Perl rounded to the nearest integer. This could result in a one-week discrepancy in diet duration between the value shown immediately and that which appeared after the user saved the diet calculator results. I modified the JavaScript code to round the same way as the Perl code does. (Reported by Jim Hollcraft.)

2007 July 28

Import of an XML database set the weight unit of the log only from the `log-unit` in the preferences, and did not allow the `weight-unit` in an individual monthly log (which might be different) to override the default. I added code to set the unit for a given month from its own `weight-unit`. Note that logs created in the process of importing CSV or XML data are always created using the user's current log unit setting; the log unit in the data imported is used to convert weight values (if necessary) from the unit in the imported log.

The synthetic listing generated for log items imported from an XML file did not show decimal places due to an incorrect format code in the `sprintf` which generated the output. (The records were imported correctly; only the listing was affected.) I fixed the format code.

Added a new `<decimal-character>` container to the preferences section of the XML output format whose content is the user's choice for decimal separator character (period or comma). Note that regardless of this setting, decimal numbers within the XML file itself always use period as the decimal separator. Appropriate declarations for this item were added to the XML Document Type Definition and CSS style sheet of this `DOCTYPE`.

Added a sixth field at the end of the "`Preferences`" header item in our native CSV export for the decimal character. Natually, this field will be quoted if the decimal character is set to comma.

When parsing the first ("`Epoch`") line of a native CSV database import, two warning messages would be generated because this record contains only two fields, while the code which deletes embedded blanks for log entry records assumed all records to have four fields or more. I made each of these statements conditional upon the field's being defined.

2007 August 11

Completed the initial implementation of the `Aggregator` object, which allows retrieval of log items in a specified date range across all accounts, public accounts only, or a list of specific accounts. This will be used for application-wide statistics of various sorts.

2007 August 14

Added a "Global Statistics" report available for administrator logins. The report summarises, for all accounts and public accounts only, the number of open accounts, active and inactive accounts (with an active account defined as one in a weight log entry has been made in the last 30 days), the mean weight gain or loss across

all active accounts, users with the fastest rate of weight loss and gain, and a histogram of the frequency with which users update their weight logs.

2007 August 16

The `monthlog::updateFromCGI` method failed to clear flag fields which the user had once checked and then subsequently unchecked. This was because the browser does not send CGI arguments for `checkbox` fields which are not checked, and the code was assuming these fields would be sent, but with a null value. I added code which tests, if a flag field is not defined in the CGI arguments, whether the database field for the flag is set and, if so, turns it off. (Reported by Michael Kiesel.)

2007 August 17

The diet calculator update code invoked when the user presses the "Save" button did not handle values in the Initial and Goal weight fields with comma as the decimal character—fixed.

The diet calculator update code failed to round decimal values specified in the Daily balance field to integral values of calories or kilojoules, and also did not accept comma as the decimal character—fixed.

Although daily energy balance figures in the diet calculator are intended to be integral values of calories or kilojoules, we do, in fact, allow the user to specify decimal values, which are rounded to integers when displayed. The JavaScript live update code accepted decimal values in this field but did not accept entries with a comma as the decimal character, parsing them as a NaN and wrecking all of the derived values. I added code to accept energy balance values with comma as the decimal character.

The diet calculator static update performed when JavaScript is disabled did not regenerate the table of years used in the selection boxes for Start and End dates. As a consequence, if the user adjusted the calorie balance or other parameters which caused the end date to extend to a year beyond the previous end year plus one, no year would be selected in the End date box, defaulting to the first year in the list. I added a call to "Generate array of years for diet calculator selection" at the completion of a static update in which one or more fields were changed.

2007 August 19

Updated documentation for `history::analyseTrend` to reflect the addition of minimum, maximum, and mean values to the list of slopes returned.

2007 August 21

Completed implementation of "Web badges", which allow users to display their most recent weight log entry and the energy balance and rate of gain/loss for a specified trend interval. A new badge configuration page, accessible from the main utility menu, allows enabling the badge and selecting the trend interval, which is kept in a new `badge_trend` field in the `user` object. When this field is nonzero, any operation which modifies a log entry calls `history::drawBadgeImage` to update the `BadgeImage.png` file in the user's directory. (This file is swapped into place with a `mv` command to avoid race conditions if it is being retrieved at the time an update is in progress.)

The badge configuration page takes the user to a confirmation page which, if badge generation is enabled, shows XHTML code the user can copy and paste into a Web page to display the badge. This code invokes a new stand-alone lightweight CGI program named `HackDietBadge`, which is called with an opaque argument which is the user file name of the owner of the badge salted and encrypted with the application's master key using AES in CBC mode. The `HackDietBadge` program is separate so as to avoid having to load the full application and all of the modules it requires just to display a badge image on a Web page which may be hit far more frequently than full-fledged application transactions. The `HackDietBadge` program decrypts and validates the argument and, if all is well, copies the badge image for the specified user to standard output hacing specified a `Content-type` of `image/png`. If the argument is in error, or the specified user has disabled badge generation, a canned "Invalid request" image is returned instead.

Badge generation mis-handled the case where the log unit and display unit were set differently—fixed.

Eliminated a redundant ampersand in the URL submitted when the "Configure Web page badge image" item is clicked in the Utilities menu.

Deleted a redundant definition of the array of labels for trend analysis durations in the `update_badge` transaction handler.

Added validation of term duration specifications in the `update_badge` transaction handler. If a user cobbles up a URL with a bogus `badge_term` argument, it will be silently converted into a disable badge selection.

Added newly-referenced CPAN modules to the list of library modules we require in the documentation. Each is linked to its documentation on the CPAN site.

Modified the `publish` and `production` targets in the `Makefile` to install the executable Perl components (`HackDiet`, `HackDietBadge`, and `ClusterSync`) by copying them to the destination directories with an extension of `.NEW` and then renaming them to the destination name. This avoids the possible race condition when a request arrives while the file is being copied to the server and the CGI process attempts to read the Perl program before it has been entirely copied. Note that we still have a potential race condition for the modules in `HDiet`, but as these files are much smaller, the odds of encountering it are much less than with the large main program. I will eventually change these to install with a copy and move strategy as well, but that will require more work since they are installed with a recursive copy rather than a simple file copy.

2007 August 31

If a user makes log entries in a month in the future (for example, to add "to do" items in the comment field), the future month would be assigned a trend carry-forward at the time it was created, but the carry-forward would not be updated when weight entries were made in the current month because trend propagation was triggered only for entries in months prior to the "current month" in the user's time zone. This was an example of the sinfulness of premature optimisation—compared to the cost of updating the chart for an entry in a monthly log, checking the user directory for subsequent months, even if in the future, to which the trend should be propagated is negligible. I removed the unwarranted "optimisation" from "Write updated log item back to database", causing a check for trend propagation to be performed for all weight changes in monthly logs. (Reported by Anna E. Sage.)

2007 September 4

The `ClusterSync.pl` program could go into an infinite loop if given a transaction which requested the deletion of a file which was not present on the destination cluster host. This situation could occur due to race conditions in which a RememberMe file was created and replaced almost instantaneously. I added code which detects this case and considers the deletion transaction as having been completed successfully if the file is found not to exist on the destination host. The error handling has been restructured to allow other such special cases to be handled should they arise.

Corrected the description of the global statistics table section in the `hdiet.css` style sheet and removed a reference to a nonexistent macro for synthetic data generation style definitions.

2007 September 5

Added a line to the Open Accounts summary in the Global Statistics page which shows the number of accounts which have Web badge generation enabled.

Propagation from an initial month in the database with no weight entries to subsequent months would reference an undefined trend value for the month. I added code to set the trend carry-forward to zero in this case, as a trend of zero is our indication that no trend carry-forward exists. Since the undefined trend value would be treated as zero, this caused no problems but produced a warning message in the error log.

Global statistics computation became confused when presented with a user account which had a database entry for the first month in the statistics computation interval (currently 30 days), but in which the first weight entry was after the start of the interval. The code which computes the user's trend slope would

pass undefined trend items to the fitter, generating a snowdrift of (otherwise harmless) warning messages. I added a test for undefined trend values returned by the aggregator, which causes the coverage of such users' logs to be deemed incomplete and thus excluded from the summary trend analysis.

2007 September 17

Cluster synchronisation transaction files were written in UTF-8, but `ClusterSync.pl` failed to open them in this mode. This caused file names which contained ISO-8859 characters above the 7 bit ASCII range which we do not escape to be misinterpreted when the transaction was read, resulting in a signature verification failure for the transaction. I added a "`:utf8`" specification to the open of the transaction file so it will be read correctly. I also set `STDOUT` to UTF-8 mode so that error messages are printed in that mode. The log file remains in ISO-8859 mode, as that will handle all characters which we do not escape.

Cluster synchronisation could loop with a failed copy transaction when, while processing a backlog of synchronisation transactions, a copy transaction for a session (`.hds`), active session (`.hda`), or remember me (`.hdr`) file was executed after the file in question had been deleted at the close of the session. I added code, similar to the September 4 fix for deletion transactions, which considers copy transactions which fail due to nonexistence of the source file as having completed normally.

2007 November 13

If a user created one or more non-void monthly logs with no weight entries (for example, containing only exercise rung and/or comment fields), the administrator global statistics report would fail with a division by zero when it attempted to compute the "coverage" of the time period by weight log entries. I modified `receive_aggregated_statistics_records` to ignore returned records with undefined weight fields, as only such records are relevant to the global statistics.

2007 November 17

If an Excel-format CSV record contained a space before a single-digit exercise rung field, the record would be skipped as not parsable. I modified the test pattern to allow leading (but not trailing) spaces. Records of this type are created by the Palm `HDread` program when a day has an exercise rung between 1 and 9 and the `-o` option is used to generate Excel-format CSV. (Reported by Ömer Ay.)

Historical chart generation with multiple days per pixel could report different values for the trend analysis and flag fraction in the caption depending upon the chart size (and hence the number of days aggregated into each horizontal pixel). This was because `getDays` is not guaranteed to examine every day in the interval, particularly in the case of long intervals and small charts. I added code to `drawChart` in `history.pm` to call `analyseTrend` for the entire interval to perform the analysis and use the values it computes for the caption, instead of those computed on the fly by `getDays` which may depend upon the chart scale. (Reported by Jim Hollcraft.)

2008 January 10

Completed the implementation of a facility for generating and printing paper log forms for people who wish to log offline and then transcribe the data to the application later. A new "Print paper log forms" item on the Utilities menu displays a form which allows the user to select the first and last month and year (the form is preset to the default of all months in the current year). The current, previous, and next years may be selected. (If the user sets the end date before the start date, they are silently swapped.) When the "Generate" button is pressed, a new window opens with the log document in it, and after a one second delay to allow the page to render, a print command is queued (these features require JavaScript to function; if it is absent, the log document opens in the same window as the request form and the user must print it manually and return to the application with the "Back" button). A paged media style sheet is used to insert page breaks so that each monthly log prints on its own page.

2008 March 9

Added code to set automatic buffer flush for the ClusterSync log file when it is initially opened and cycled after receipt of the `HUP` signal. This allows those who monitor the log file with, for example, `tail -f` to see the complete log item for a transaction without waiting for the buffer to be flushed.

When the cluster synchronisation log file was cycled after receipt of a `HUP` signal, the cycling was erroneously performed both in the signal processing code and in the main loop code which responds to the signal. I removed the file cycling from the signal handler, where it is vulnerable to race conditions within Perl and the underlying C library.

Implemented recovery from transient and permanent cluster synchronisation transaction failures. Previously, if any error occurred reading, verifying, or executing a cluster synchronisation transaction, the `ClusterSync.pl` program would crash, suspending cluster synchronisation until it was restarted. Unfortunately, there were a number of circumstances in which such errors could occur, the most common being cases where a race condition between queueing the transaction and `ClusterSync`'s processing of it caused an incomplete file to be read (transient), and those where a crash of the process queueing the transaction caused an incomplete file to be written to the transaction directory (persistent).

When a cluster sync transaction fails, for whatever reason, it is placed into a failed transaction hash whose key is the transaction file name and whose value is an array containing the number of times the transaction has been tried and the next time the transaction should be retried. On subsequent passes through the transaction directory, failed transactions are skipped unless their retry time has arrived, whereupon they are retried and, if they fail, their try count is incremented and the next attempt count updated.

If the transaction eventually succeeds, it is closed out normally and removed from the failed transaction hash. If the transaction fails again, its try count is increment and if it has reached the limit, the transaction is deleted from the transaction directory and the failed transaction hash. Failure to delete the transaction from the transaction directory remains fatal to the `ClusterSync` program.

The intervals between retries of a failed transaction and the number of failures which cause a transaction to be abandoned are set by configuration parameters.

2008 March 10

Modified the cluster synchronisation log file generation to skip the "Results:" line if the command produced no output.

2008 June 3

It was possible by using quoted fields to import a native mode CSV file whose exercise rung field included embedded spaces. This would cause a native database file to be written which, when `monthlog::load` attempted to load it, would cause the parser to abort. I added code to `monthlog::importCSV` to delete any spaces in an exercise rung field.

To cope with existing database entries with spaces in exercise rung fields, I added code to `monthlog::load` to delete all spaces from the record. An entire database can be cleaned up by performing a trend recalculation. (Reported by Tom Gunter.)

2009 January 15

Updated configuration to accommodate the switch from Server1 to Server0 as the primary production server. We now install test versions on Server1 and production versions on Server0, and permit non-`https` logins to Server1 without the security warning.

A change in the handling of the `decode_utf8` function between Perl 5.8.5 and 5.8.8 broke decoding of CGI arguments containing ISO-8859 and Unicode characters when received as POST arguments. Whereas before we needed to read POST arguments with "`:utf8`", now it appears we need to use "`:raw`" unconditionally. How much would you like to bet we'll be changing this back somewhere down the road?

Added a "Month" option for Web badge generation. This is something intended from the start, but omitted due to scatterbrained developer. (Reported by Kees Huyser.)

2009 February 8

Due to race conditions (for example, processing cluster synchronisation transactions while a global server synchronisation is underway), it is possible for an `mkdir` or `rmdir` transaction to fail because the directory in question already exists or has already been removed. To avoid a possible loop retrying such transactions, I added tests for these cases which deem the transaction successful if its intended effect has already taken place.

2009 March 7

When displaying the current (or most recent) monthly log, the trend analysis displayed beneath the chart was based on the data plotted in the chart. At the start of a month, when there were only a few data points, this could result in day-to-day instability in the trend analysis. I added code to test whether the most recent log is being displayed and, if so, a trend for the last seven days is computed using a `history` object, even if that requires retrieving days from the previous month. This also ensures that the trend reported on the monthly log page will always be identical to that shown for the last week in the Trend Analysis page.

If only one weight was present in the most recent log and no entries existed in the previous month's log for the preceding week, the `trendfit::fitSlope()` method would divide by zero because there were insufficient points to fit a linear trend. I added code to the method which reports a zero trend slope when insufficient points are available to fit a slope. Note that while this was discovered testing the new method of computing the trend for the current month, the bug was present prior to the change.

2009 April 18

Added Server2 to the list of cluster member hosts.

2009 August 8

With more than 2500 public accounts, it takes almost forever for the list of public accounts to load. This is very irritating to people who regularly check on their friends' progress. What I'd like to do is add the ability, when viewing a public account, to check a box to make it (or remove it from being) a "friend". In the Utilities page, a drop-down list of friends will be displayed, from which you can select a public account to access with a single click. To address the immediate problem, until I manage to put all of this machinery in place, I've added a simple text box and "View" button below the "Browse public user accounts" item in the Utilities menu. The user can simply enter the name of the public account in the text field and press the button to go directly to the account. This isn't as convenient as a list of friends, but it's a lot better than waiting for the monster form to load. The direct name access form is also included in the Utilities page displayed when viewing a public account, allowing direct transfer from one public account to another.

2010 March 31

The error messages generated when the administrator attempts to purge the logs or delete a nonexistent account name were missing a space before the name of the account—fixed.

If the administrator attempted to access (view) a nonexistent account, the application would exit with an error and yield a blank screen. With the original account manager, this could happen only if the user deleted the account between the time the list was displayed and when the administrator attempted access, but with the direct access facility, the error would occur whenever the administrator entered an invalid account. I added an explicit error message for this circumstance which includes the invalid account name.

To expedite display of public accounts, I added a drop-down box to the "Browse public user accounts" item on the Utilities page which allows the user to select active, inactive, or all accounts with active the default. An active account is defined as one with a transaction within the last 30 days. In addition, the user can switch between the display of active, inactive, or all public accounts on the Browse Public Accounts page.

Under Administrator Functions on the Utilities page, the administrator may choose, when managing user accounts, to display active, inactive, or all user accounts (as for public accounts, active means a transaction

within the last 30 days), and may switch selections from the Account Manager page. In addition, the administrator can access a user account directly by name to view, purge logs, or delete. For the latter two functions, the administrator password must be entered as a confirmation before the button is pressed.

When enumerating public accounts for the browse public accounts page, or all user accounts, open sessions, or persistent login tokens for the administrator, we unnecessarily sorted the names of the files in the respective directories before retrieving them into the hash used to build the displayed table. Since the table generation code sorts the hash keys, there is no need to sort the file names, which can be very time consuming when these directories get large. If these directories get very much larger, it may make sense to read them serially and perform the `grep()` on them within the loop rather than bringing the directory into memory and using the `grep()` function as presently done.

The `cookie::storeCookie` and `cookie:testCookiePresent` methods failed to set UTF-8 mode when reading and writing the cookie token file for persistent logins. This caused warning messages when sorting cookie user names in the administrator Persistent Login Manager page. This change will force all users with non-ASCII login names to log back in, as their cookies will not match those stored with the incorrect file encoding. As it happens, there were only two such and both were inactive accounts, so I just purged the persistent logins for them myself.

2011 July 27

When generating a "printer friendly" monthly log display, values in the "Flag" column were not encoded as `hidden` items in the result page. If the user then did an update from this page, all checked flags would be lost. I added code to embed `hidden` items for checked flags, which will propagate them back on an update. Note that due to the way we integrate the rung and flag fields on updates there is no need to embed their values as hidden fields. It's only due to the odd design of HTML/CGI which doesn't allow you distinguish the absence of an editable field from a check box not checked that we require this work-around. (Reported by user "rhittom".)