

Quancept and mrInterview

Scriptwriter's Manual

PUG050224WI

COPYRIGHT © 2005 BY SPSS MR LIMITED

All rights reserved as an unpublished work, and the existence of this notice shall not be construed as an admission or presumption that publication has occurred. No part of the materials may be used, reproduced, disclosed or transmitted to others in any form or by any means except under license by SPSS Ltd. or SPSS US Inc.

SPSS MR Limited
Handrail House
65 Maygrove Road
LONDON
NW6 2SP
England

Please address any comments or queries about this guide to the Support Department at the above address, or via e-mail to:
support-uk@spssmr.spss.com

All trademarks acknowledged.

All screenshots and examples in 'Creating an IVR script to receive Quancept data' on page 634 et seq. are
Copyright © 1995 - 2002 DBM of America, Inc. All rights reserved.

Contents

About this manual	21
Chapter overview	21
Typographic conventions	24
Differences between the Quancept and mrInterview variants	25
Glossary	26
1 Writing in the Quancept language	31
1.1 Components of a script	31
Questions and responses	31
Checking and verification statements	33
Routing	33
Loops	36
Screen layout and formatting	37
1.2 The character set	38
Uppercase versus lowercase letters	42
1.3 Quancept language statements	43
<i>Labels</i>	43
<i>Statements</i>	44
<i>Continuation lines</i>	44
Naming temporary variables	45
1.4 Spaces	45
1.5 New lines	46
Using a character other than @ for new lines	47
1.6 Putting comments in your script	48
1.7 The continue keyword	49
1.8 The end of the script	50
1.9 Sections in a script	50
1.10 Naming script files	52
2 Language summary	53
2.1 Keywords in scripts	53
2.2 Keywords used with set	62
set variable=value	63
set variable = special keyword	64
set variable=value	65
2.3 Callable functions	70
2.4 Variables set by mrInterview	73
2.5 Special responses in qtip	73
Standard responses	74
Entering verbatims and comments	74
Working with long response lists and Quancept databases	75
Snapping during an interview	75
Ending an interview early	75
Miscellaneous commands	76
Setting appointments	77
Quantime Telephony System commands	77
2.6 Working with external Unix databases	78
Shared memory commands for resp dbase questions	78
GNU database commands	78

3 Questions and answers	79
3.1 Naming questions	79
3.2 Question and response format	80
3.3 Response types	81
Single-punched and multipunched responses	81
Numeric responses — whole numbers	84
Numeric responses — real numbers	86
Open-ended responses	88
Open ends with Quancept Web and mrInterview	90
3.4 Dummy questions	91
3.5 Variable values in numeric ranges	92
3.6 Displaying more than one question on the screen	95
3.7 Sample script	97
4 Special responses	99
4.1 No answer	99
4.2 Don't know	101
4.3 Refusals	101
4.4 Other responses	102
Other as a quoted response	102
Specified other	103
Data allocation of specified other	104
4.5 Choosing your own words for other, null, dk and ref	104
In Quancept CAPI	105
In Quancept Web and mrInterview	106
4.6 Automatic open ends with sp and mp responses	107
Using promptoth	107
Using ^o	109
4.7 Prompt text for numeric, open-ended and other responses	111
4.8 Sample script	112
5 Keywords in response lists	113
5.1 Displaying responses in alphabetical order	113
Additional features for alphabetic sorting in multilingual mrInterview scripts	114
5.2 Echoing responses on the screen	117
5.3 Response lists referring to several questions	117
5.4 Routing in single-punched and numeric response lists	119
Routing with null, dk and ref	120
5.5 Number of responses chosen from a multipunched list	120
5.6 Asking questions without storing responses	121
5.7 Rotation and randomization of responses	122
Rotated response lists	123
Testing response patterns for rotated response lists	124
Randomized responses lists	125
Rot and ran with loops	126
Fixing a response's position in a list	127
Grouping responses in a list	128
Using rottranfix and rottransub in the same response list	129
5.8 Single-punched responses in a multipunched list	130
Using sp	130
Using ^s	131

5.9	Scaling factors for numeric responses scale	132
	Setting the scaling factor	132
	Find the data to be converted	133
	Apply the scaling factor to the data	134
5.10	Slide bars for numeric response lists	136
5.11	Automatically jumping to the next question	138
5.12	Unique IDs for responses	139
	Unique IDs in mrInterview	140
5.13	Sample scripts	143
	Script A	143
	Script B	145
6	Writing multimedia scripts	147
6.1	Displaying pictures in Quancept CAPI	148
	Positioning response text relative to a picture	150
	Defining a picture's size	151
6.2	Playing video clips	152
6.3	Playing sound recordings	153
	Playing audio files automatically	154
6.4	Recording open ends in audio format	154
	Setting a maximum recording time	155
	Setting the sample rate for recordings	156
6.5	Saving an open end as a picture	157
6.6	Displaying images in Quancept Web	158
6.7	Displaying images in mrInterview	159
6.8	QTS Record and Play	159
	Playing sound files	160
	Recording open ends	161
	Recording subsurveys	163
6.9	Sample script	164
7	Defined response lists	165
7.1	Defining response lists	165
7.2	Column allocation with defined lists	167
7.3	Using a master list as a response list	167
7.4	Null, dk and ref with defined lists	168
7.5	Specified other with defined lists	169
	Specified other on the defined list and resp statement	169
	Specified other on the defined list only	170
	Specified other on the resp statement only	170
7.6	Other keywords with defined lists	171
7.7	Reserving space for extending lists	172
7.8	Using sublists as response lists	173
7.9	Using responses not in a sublist	175
7.10	Using question names with lists	175
	Specified other with not question in list'	177
7.11	Finding the relevant sublist	178
	Displaying responses not chosen from a variable sublist	180
7.12	Blank responses in defined lists	181
7.13	Sample script	183

8	Displaying information on the screen	185
8.1	Displaying temporary text	185
	Displaying pictures	187
	Changing the appearance of the text	187
	A note about display statements and design mode	188
8.2	Displaying text in a message box	188
8.3	Controlling the display time for text	189
8.4	Protected texts	190
	Displaying the current question name in CAPI and mrInterview interviews	192
8.5	Reading texts from files	193
8.6	Reading single lines from look-up files	194
8.7	Displaying data in texts	196
8.8	Text substitution in response lists	198
8.9	Highlighting text during an interview	200
8.10	Scrolled response lists in interviews	201
8.11	Specifying the number of columns for response lists	202
	Setting the width of response columns	204
8.12	Drop-down lists and combo boxes	205
8.13	Sample script	208
	Look-up files	210
9	Routing and flow control statements	211
9.1	Routing in single-punched and numeric response lists	212
9.2	Routing outside response lists	213
9.3	Conditional routing	214
9.4	Routing dependent on keyboard input	215
9.5	Conditional questions (using if and else)	216
9.6	Rotating and randomizing questions	219
9.7	Rotating and randomizing blocks of questions	221
9.8	Testing for snapbacks	223
	Timeofday and snapbacks	225
	Routing around ifsnap assignments	226
9.9	Returning to the SMS menu	227
9.10	Actions for test interviews only (CATI)	229
9.11	Actions for test interviews only (mrInterview)	230
9.12	Sample scripts	232
	Script A	232
	Script B	235
10	Repetitive questions	237
10.1	How loops work	237
10.2	Writing loops	238
	Labelling for statements	240
	Loops with ranges	240
	Specified other and defined lists with loops	241
	Multiask in loops	242
	Nested loops	243
10.3	Routing with loops	244
	Terminating interviews with infinite loops	246
10.4	Rotation, randomization and alphabetical sorting	247
10.5	Using freeze with loops	248

10.6	Column allocation with loops	248
10.7	Displaying responses in grids	249
	Specified other in grids	251
	Setting column widths in response grids	251
	Numeric or real questions in grids	252
10.8	Respondent-specific texts in value lists	255
10.9	Sample scripts	257
	Script A	257
	Script B	258
11	Iteration and subscription	261
11.1	Counting loop iterations	261
	Iterations with rotated and randomized value lists	262
	Using the iteration value in the script	263
	Iteration in nested loops	263
11.2	Referring to answers given in loops	264
	Specified other in loops (negative subscripts)	267
	Subscription when assigning values to repeated questions	268
11.3	Subscription with sequential numeric for statements	270
11.4	Subscripts with nested loops	270
11.5	Subscription of temporary variables	272
11.6	Reducing the size of data maps with loops	273
	A few responses at random from a large list	273
	Restricted questions in nested loops	275
	Loops with all respondent-specific texts	277
11.7	Sample script	279
12	Assignment	281
12.1	Types of variables	281
12.2	The set statement	282
	Data types	282
12.3	Text assignments	283
12.4	Assigning responses to questions	284
	Response texts	285
	Response numbers	288
	Question labels	289
12.5	Assignments with null, dk and ref	292
12.6	Assigning values to subscripted variables	292
12.7	Special keywords with set	293
	Automatic null coding when no response is chosen	296
	Allowing blanks in numeric grids	297
	Customizable error messages in Quancept Web	297
12.8	Customizing messages in mrInterview	298
	Making message texts translatable — version 2.2 and later method	301
	Making message texts translatable — pre-version 2.2 method	302
12.9	Setting the language in multilingual mrInterview scripts	303
12.10	Saving copies of SQL statements that write data	304
12.11	Different texts for Web and WebCATI interviewing	305
	Defining the context in the script	307
	Defining the label type in the script	307
12.12	Other variables that can be set in mrInterview	308

12.13 The unset statement	309
12.14 Removing a single response from a multipunched selection	310
12.15 Checking for temporary variables	311
12.16 Combining responses and other values with longtext	312
12.17 Assigning a replacement value to a variable	315
12.18 Sample scripts	317
Script A	317
Script B	319
13 Logical expressions.....	323
13.1 Syntax of an expression	323
13.2 Symbols used with logical expressions	324
Symbols with multipunched responses	325
13.3 Logical operators used with logical expressions	327
The .and., .or. and .xor. operators	328
The .not. operator	329
Precedence with more than one operator	329
.not. with other operators	330
13.4 Assigning logical values to variables	331
13.5 Sample script	333
14 Checking non-numeric responses.....	335
14.1 Checking single-punched responses with nbit	335
14.2 Checking multipunched responses	337
At least one named response chosen	337
All named responses chosen	338
Only one named response chosen	339
Order of mention for multipunched responses	340
14.3 Testing for chosen responses by their position in the list	342
Differences between nbit and bit	343
14.4 Number of responses in a list	344
14.5 Number of responses chosen from a list	344
14.6 Find response text by position of response in list	345
Using bit and elm together	346
14.7 Displaying specified other answers	347
Using othertext	347
Negative subscription	349
14.8 Null, dk and ref	350
14.9 Checking text responses	350
14.10 Editing open ends and specified other responses	352
14.11 Testing for null-response and unanswered questions	354
Testing for questions with empty responses	355
Treating unanswered questions the same as questions with a null response	355
14.12 Sample script	357
15 Working with numeric data.....	361
15.1 Arithmetic statements	361
15.2 Variable types with arithmetic assignments	363
In Quancept CATI	364
In Quancept CAPI, Quancept Web and mrInterview	365
15.3 Arithmetic assignment with question variables	368

15.4	Arithmetic with text numbers	370
	Real numbers	371
	Integer numbers	372
	Converting text numbers to integers	372
15.5	Sample scripts	376
	Sample script A	376
	Sample script B	377
16	Ending the interview	379
16.1	Allowing/disallowing early completion	380
16.2	Ending an interview without going to 'end'	381
16.3	Stopping and restarting interviews	382
	Stopping an interview from the script	383
	Changing the names of the stopped data files	384
16.4	Forced termination of an interview	385
16.5	Saving data for manually terminated interviews	387
	Saving data for stopped interviews	388
	Saving data for terminated interviews	389
	Saving data for all uncompleted interviews	389
16.6	Sample scripts	391
	Script A	391
	Script B	393
17	Data format	397
17.1	Adding extra questions to the script	398
17.2	Forcing data onto a new card	402
17.3	Specifying the card and columns for one question	404
17.4	Specifying card and columns for blocks of questions	405
	Avoiding data map conflicts	406
17.5	More columns for serial number and card type	407
17.6	Data mapping for single-punched and multipunched responses	407
	Changing the data format for a block of questions	408
	Setting the data format for individual questions	409
	What does the data look like?	410
17.7	Inserting non-response data in the data file	413
	User-defined text	413
	Variables	413
	Text and variables together	414
	Inserting open ends in the precoded data file	415
17.8	Saving 'dirty' data	416
17.9	Set statements with snapbacks	417
17.10	Private status codes to collect accounting data	418
17.11	Suppressing blank cards	420
17.12	Sample script	421
	Script A	421
	Script B	422
18	Report files	425
18.1	Filenames	425
18.2	Opening a report file	426
18.3	Closing a report file	426

18.4	Writing to a report file	427
	Defining the output	428
	Using conversion characters in an output definition	429
	Example usage	432
18.5	Sample script	435
	Script A	435
	Script B	436
19	Very long response lists in CATI (response list databases)	437
19.1	Response list database format	438
19.2	Where to store response list databases	438
19.3	Displaying the response list database	439
	Allowing multiple selections from a database	441
19.4	Saving the record number in a variable	442
19.5	Selecting a field from a record	443
19.6	Selecting a record from a response list database	444
19.7	Making the response list database available to interviewers	446
19.8	Checking for response list databases in shared memory	447
19.9	Removing response list databases from shared memory	448
19.10	Sample scripts	449
	Script A	449
	Script B	451
20	ODBC databases in Quancept Web and mrInterview	453
20.1	What database formats can I use?	453
20.2	Making the database public	454
20.3	Connecting to the database	454
20.4	Closing the connection to the database	455
20.5	Querying the database	455
20.6	Selecting records from the database	456
	Extracting a single record from the temporary variable	457
	Extracting data from a field	458
20.7	Updating records in the database	458
20.8	Adding new records to the database	460
20.9	Number of script replays after an ODBC error	461
20.10	Setting the SQL cursor type	461
20.11	General notes about working with ODBC databases	463
20.12	Sample script	464
21	GNU GDBM relational databases in Quancept CATI	467
	About the GNU GDBM database manager	467
21.1	Defining a GNU database	468
	Key fields	468
	Creating original data files	469
21.2	Description file for variable field length (gdbm) database	469
	Example data	470
21.3	Description file for fixed field length (gdbmfix) databases	471
	Example data	472
21.4	Creating the GNU database	472
	Duplicate keys	473
	Other errors	473

Writing data back to the original text file	473
Viewing data in a GNU database	474
21.5 Using GNU keywords in scripts	474
21.6 Opening a database	475
21.7 Reading database records	475
21.8 Manually updating records	476
21.9 Automatically updating records	477
21.10 Sample scripts	479
Additional files required for Script A	479
Script A — Displaying records in GNU databases	479
Additional files required for Script B	480
Script B — Updating records in GNU databases	481
22 Screen management with graphical user interfaces	485
22.1 Formatting instructions in CAPI scripts	485
Bold, italics, underlining and strikethrough	486
Changing the font	487
Changing the text size	487
Changing the text color	488
Global text formats	490
Background colors and images	491
22.2 Screen design using HTML code	493
How to use HTML tags in scripts	494
Bold, italics and underlining	495
Line breaks and paragraphs	496
Color, typeface and character size changes in texts	496
Default characteristics for questions, responses, display and protect texts	498
Positioning text on the page	500
Tabular layouts	500
Indenting response lists	502
Choosing the background and default text colors	503
22.3 Input boxes for open ended and numeric responses	504
Multi-line boxes for open ends	504
22.4 One-line boxes for numerics and open-ends	506
22.5 Controlling the display of the Previous and Stop buttons	507
22.6 Sample scripts	509
Script A	509
Notes for Quancept Web users	511
Notes for mrInterview users	511
Script B	512
23 Using page layout templates in mrInterview	515
23.1 Templates — an overview	516
23.2 Line Breaks and Blank Space	518
23.3 Single questions and response lists	519
23.4 Multiask questions	522
23.5 Display and protect statements	524
23.6 Page titles	525
23.7 Navigation buttons	526
23.8 Automatic completion of data entry fields	528
23.9 Choosing a page template for a question	528

Using pagestyle	529
Using pgstyle and curpgstyle	530
23.10 Defining your own page titles	531
23.11 <meta> and <!doctype> tags in templates	533
23.12 Cascading style sheets	533
Format and location of a CSS	533
Tags for questionnaire items	534
Naming the CSS in the template	535
23.13 Editing the page's form definition	536
23.14 Accessing an interview while it is running	536
23.15 Sample script	537
CSS and template files	539
24 Templates for grids	541
24.1 Naming grid template files	541
24.2 Positioning the grid on the page	542
24.3 The grid template file	542
24.4 Starting a grid template	543
24.5 Levels in a grid template	543
24.6 Background color	545
24.7 Wrapping text within a column	547
24.8 Text alignment	547
24.9 Bold and italic text	549
24.10 Typeface and text size	549
24.11 Text color	551
24.12 Special text effects	553
24.13 Row height	553
24.14 Column widths	554
24.15 Row and column headings	556
Cell formatting	557
Repeating the row or column headings	557
24.16 Different formats for alternate rows or columns	558
24.17 Precedence between specifications at different levels	560
24.18 Overall characteristics of the grid table	562
Grid width	563
Grid alignment	563
Borders	564
Space between rows and columns	564
Space around cell contents	565
Example	565
25 Using multiple script files (subsurveys)	569
25.1 Loading library files	569
25.2 Running one script from inside another	570
Parsing subsurvey scripts	571
Storing subsurvey data	571
Scripts with more than one subsurvey	572
Recording subsurveys	573
Passing data between the main and subsurvey scripts	574
25.3 Sample script	577

26 Text manipulation	583
26.1 Appending one text to another	583
26.2 Extracting strings from texts and converting them to numbers	586
26.3 Extracting characters from a text and copying them into a variable	587
26.4 Extracting strings from texts	588
26.5 Sample scripts	591
Script A	591
Script B	593
27 Callfuncs	595
27.1 Date and time manipulation callfuncs	596
Date and time in operating system format	596
Converting an operating system date and time into a text	597
Breaking the date and time into separate sections	597
Setting dates and times within the script	599
Converting a text date and time into operating system format	600
Making appointments from within the script	601
27.2 Scripts for multilingual CATI interviews	602
Setting the language in which interviewing will take place	602
Setting the language for alphabetic sorting, punctuation and line breaks	603
27.3 Setting the respondent serial number inside the script	604
27.4 Finding the value of an environment variable	605
27.5 Automatic test mode	606
In Quancept CATI and Quancept CAPI	606
In mrInterview	607
27.6 Running an external program from within the script	608
27.7 Writing variables to a temporary file	608
27.8 Storing accounting information	611
Timing sections of the interview	611
Writing customized information to the qca file	613
27.9 Naming script sections for use by the supervisory program	615
27.10 Saving response data during an interview	617
27.11 Retaining original data for stopped interviews	619
27.12 Automated conjoint analysis	620
27.13 Switching QTS recording on and off	622
27.14 Hanging up a QTS call	623
27.15 Redialing a dropped QTS call	624
27.16 Interactive Voice Response callfuncs	624
Defining the initial IVR configuration	625
Dialing the IVR system	627
Transferring the respondent to IVR	633
Canceling an IVR call	634
Creating an IVR script to receive Quancept data	634
27.17 Sample scripts	640
Sample script A	640
Sample script B	644
28 Writing and using local callfuncs	647
28.1 Local callfuncs in Quancept CATI	647
Using local callfuncs	647
Naming local callfuncs	648

How the parser sees callfuncs	648
Making a special version of qtip	649
28.2 Using your own functions in mrInterview	650
Writing the function	651
Defining property variables	653
Calling a user-defined function	656
Checking whether the function ran successfully	656
Restarting interviews that use functions	658
29 Using information in sample records	659
29.1 Sample record format	660
29.2 Reading sample information into the script	661
29.3 Updating a sample record from the script	662
29.4 Updating sample records changed by the script	664
29.5 SMS database format	666
29.6 Reading data from the SMS database file	667
29.7 Writing data into the SMS database file	668
29.8 Sample script	671
30 Quota control.....	673
30.1 Types of quotas	673
30.2 How the CATI and Web interviewing programs deal with quotas	674
Decrementing quotas	674
Restarting stopped interviews	675
Reviewing completed interviews	675
Single-punched quotas	675
Multipunched quotas	676
Numeric quotas	676
null, dk and ref	676
Effect of snapbacks on quotas	677
Dealing with quota errors during interviews	677
30.3 Checking quotas in Quancept CATI and Quancept Web	677
Scripts with more than one quota	679
Controlling how interviews terminate	679
Combining responses in quotas	681
30.4 Additional facilities for Quancept CATI	682
Finding a respondent's quota cell	682
Checking for unfilled quota cells	682
Checking quota ratios	685
Requesting quota status information in the script	687
Creating a quota log file for debugging purposes	688
30.5 How the mrInterview quota system works	691
Keeping track of quotas	691
Quotas for categorical responses	694
Quotas for numeric and text responses	694
Quotas for null, dk, ref and specified other	695
Quotas for stopped or timed out interviews	695
Overquota and counter quotas	695
Quotas in projects that use Sample Management	696
Quota prioritization for multiple response categorical questions	698
Sharing quota files between projects	700

30.6	Checking quotas in mrInterview	701
	Controlling how interviews terminate	703
	Checking for missing quota specifications	704
	Working with quota controlled projects in DimensionNet	705
30.7	Sample scripts	707
	Script A	707
	Script B	710
	Script C	711
31	Quota maintenance	713
31.1	Quota files	713
31.2	The Quancept CATI quota program	714
	Setting up quotas for a new project	715
	Defining quota texts	716
	Defining quota targets	717
	Changing cell texts and targets	717
	Inspecting quotas	718
	Changing individual quotas	718
	Combining quota cells	719
	Changing the completed count	721
	Removing hung temp entries	721
31.3	The Quancept Web quota program	722
	Defining targets	723
	Checking and monitoring quotas	725
	Changing the count of interviews reached	727
	Deleting targets and counts	728
31.4	The mrInterview quota program	729
	Starting Quota Setup	730
	The Quota Setup window	730
	Docking/undocking the List and Details Panes	731
	Changing the layout of the List Pane	731
	Displaying all variable types in the List Pane	731
	Displaying system variables in the List Pane	731
	Displaying sample variables in the List Pane	732
	Hiding components of the Quota window	733
	Creating and opening quota files	733
	Undoing and redoing actions	734
	Choosing your working language	735
	Table Quotas	735
	Creating more than one Table Quota	738
	Deleting Table Quotas	738
	Expression Quotas	738
	Advanced expressions	748
	Specifying expressions using the advanced method	748
	Comparison operators in Expression Quotas	749
	Functions	750
	Logical operators in Expression Quotas	752
	Arithmetic operators in Expression Quotas	753
	Operator precedence in Expression Quotas	754
	Editing Expression Quotas	754
	Deleting Expression Quotas	754

31.5 Reusing a quota definition file in another project	754
31.6 Checking and changing quotas in mrInterview	755
32 Parsing and compiling scripts	757
32.1 Parsing scripts using qparse	757
Using predefined data map files	759
32.2 Parsing and compiling CAPI and Web scripts with QCompile	759
Starting QCompile	760
Selecting a project	761
Parsing and compiling a script	761
QCompile options	763
The Parse/Compile tab	763
The Run tab	765
The Special tab — additional runtime options	767
Generating Quantum axes and table specs	770
32.3 Reparsing a CAPI or Web script once coding has started	771
If you have edited the qdi file with Qwincode	771
The structure of the qdi file	771
Copying code frames between qdi files	775
32.4 Parsing and compiling mrInterview scripts with QCompile	778
Starting QCompile	779
Selecting, parsing and compiling a project	779
Activation	780
Activation process	790
Starting Quota Setup from QCompile	791
QCompile options	792
Checking language codes in the questionnaire definition file	794
32.5 Parser error messages	795
Additional QCompile parser error messages	801
33 Testing scripts with qtip	803
33.1 Steps in an interview	803
33.2 Before you start	804
33.3 Starting a session	804
33.4 The script	805
33.5 Single-punched answers	806
33.6 Multipunched response lists	806
33.7 Numeric answers	807
33.8 Open-ended responses	808
33.9 Other and specified other	809
33.10 Selecting answers from a response list database	809
33.11 Echoed responses	811
33.12 The end of the script	811
33.13 Interviews terminated by the script	812
Termination by quota control	812
Stopped interviews	812
33.14 No answer, don't know and refused	813
33.15 Forcing an open-ended response	814
33.16 Scrolled response lists	814
33.17 Changing the layout of long response lists	815
33.18 Early completion of an interview	815

33.19	Premature termination with data	816
33.20	Terminating an interview without data	816
33.21	Stopping interviews temporarily	817
33.22	Messages to the supervisor	817
33.23	Redrawing the screen	818
33.24	Checking answers to previous questions	818
33.25	Listing question names	818
33.26	Finding where to snap	819
33.27	Snapping backwards and forwards in an interview	820
33.28	Altering responses to questions you have snapped to	821
33.29	Snapping back and making no changes	821
33.30	Miscellaneous commands	821
33.31	Using SMS	823
	Busy and no answer	824
	Arrange appointment with respondent	824
	Result codes that prompt for information	826
	Returning to the SMS menu	826
33.32	qtip error messages	826
34	Miscellaneous CATI facilities	831
34.1	Automatic testing	831
34.2	Sharing a .qoc file	832
34.3	Reviewing completed interviews	833
34.4	Recording keystrokes of interviews	834
35	Testing CAPI scripts	837
35.1	Starting new interviews	837
	The interviewing window	838
	The toolbar	839
35.2	Usage summary	839
35.3	Stopping and restarting interviews	842
35.4	Automatic testing	843
36	Testing Quancept Web scripts	845
36.1	Files required for interviewing	845
36.2	Running the script	845
36.3	Stopping and restarting interviews	846
	Behind the scenes with the stop-start mechanism	847
36.4	Displaying the respondent's interview ID	848
	Variables the ini file	848
	Statements in the script	849
36.5	Other methods of restarting stopped interviews	849
	Always start a new interview	850
	Start interviews for new respondents only	850
	Always restart stopped interviews but never start new ones	851
	Prompt the user for an interview ID before looking for a stopped interview	851
37	Testing mrlInterview scripts	853
37.1	Testing your script	853
	Stopping and restarting interviews	853
	Running test interviews on active projects	854

37.2	Checking the data	854
	Data structure	854
	Looking at the data	856
38	File formats	859
38.1	Files created by the parser	859
	Intermediate files	861
	Directories	861
	The acl file	862
	The bqt file	863
	The bt8 file	863
	The err file	863
	The exe file	863
	The exp file	863
	The fr1 file	863
	The fr2 file	865
	The ipk file	866
	The lib file	866
	The lst file	866
	The map file	867
	The maq file	868
	The mdd file	868
	The mdm file	869
	The mqw file	869
	The psm file	869
	The ptf file	869
	The qc8 file	870
	The qdf file	870
	The qdi file	870
	The qdt file	871
	The qoc file	871
	The qtv file	871
	The quo and quiz files	872
	The ref file	873
	The sif file	875
	The sum file	875
	The tx0 file	876
	The txx file	877
	The xin file	877
38.2	Files created by the CATI, CAPI and Web interviewing programs	879
	The act file	880
	The auto directory	881
	The bitmap directory and bmp files	882
	The com file	882
	The comments directory	882
	The dat file	883
	The ddr directory	883
	The drs file	884
	The drx file	890
	The idm file	890
	The msg file	890

The qca file	891
The qsh file	893
The recordng.dir directory	893
The ser file	893
The stopped interview directory	894
The stopped directory and stp files	894
The tbl file	894
The tex file	896
The tim file	897
The tiperrs file	898
The wav files	898
38.3 SMS files	898
accounts.sms	898
serverlog.sms	899
38.4 The mrInterview case data file	899
The Variables table	900
The Responses table	902
The OtherData table	903
38.5 Other mrInterview files	903
The mqd file	903
39 Managing CATI projects	905
39.1 Storing your script on the computer	905
39.2 Environment variables	906
Setting environment variables	906
The environment file	907
General environment variables	908
SMS and QTS environment variables	917
39.3 Dialing with modems	918
Optional area codes	920
40 Managing Quancept Web projects.....	923
40.1 The server manager program	923
40.2 Making projects available on the Web	924
Respondent information limited to 128 characters in URL	925
Data security with Web interviews	925
40.3 The project initialization file	925
HTML commands	926
40.4 Navigation buttons or icons?	928
40.5 Sharing initialization settings between projects	929
40.6 dat and tex files as well as the drs file	930
40.7 Using interview data outside Quancept	930
Exporting the data	931
Defining the export form	932
Writing a postprocessor	933
40.8 Reusing respondent IDs	933
41 Managing mrInterview projects	935
41.1 Making projects available on the Web	935
Data security with Web interviews	935
41.2 Activating projects from the command line	936

The Activate Document	936
Activating projects	936
Example activation commands	938
Converting activation .ini files to .xml format	939
41.3 Backing up and deleting old projects	939
42 Quancept CAPI design mode	943
42.1 Working in design mode	944
Selecting objects on the interviewing screen	945
Moving objects	946
Resizing objects	946
Left-aligning objects	947
Adjusting the vertical distribution of objects	947
A Limits	949
B Keywords in questionnaires created with Build	953
B.1 Using the current version of the mdd file	953
B.2 Mapping for variable and response names	954
B.3 Questionnaires with pictures	954
B.4 Evaluating expressions	955

About this manual

The Quancept and mrInterview Scriptwriter's Manual is aimed at the person who develops questionnaires for computerized telephone, personal or Web-based interviewing. The manual also includes information that will be useful for interviewers and administrators.

This manual is not intended as a tutorial or teach-yourself document. Instead, it provides a complete and detailed description of the Quancept language . However, the manual has been designed with your needs in mind. If you are an experienced user, you will find the Quick Reference frames at the start of each section helpful as a reminder of syntax. If you are less experienced, you will probably prefer the more detailed explanations and examples in the main body of each section.

Chapter overview

Chapter 1, Writing in the Quancept language, introduces the components of a Quancept script and describes the rules for laying out scripts.

Chapter 2, Language summary, lists all the statements in the Quancept language with a short description of each one and a note of the Quancept and mrInterview variants in which it is valid.

Chapter 3, Questions and answers, describes how to define questions and answers in a script.

Chapter 4, Special responses, explains how to deal with no answer, don't know and refusals, and how to allow respondents to give 'other' answers that are not included in the predefined response lists.

Chapter 5, Keywords in response lists, introduces keywords for controlling the way response lists are managed and displayed. These include keywords for routing, altering the order in which responses are displayed, and overriding the default response type for the list.

Chapter 6, Writing multimedia scripts, describes how to display pictures, play sound files and record verbatim responses during interviews.

Chapter 7, Defined response lists, describes how to create and use defined response lists to enhance the functionality of your script.

Chapter 8, Displaying information on the screen, describes ways of displaying fixed and variable text during an interview, and of providing scrolling facilities for long response lists.

Chapter 9, Routing and flow control statements, explains how to control a respondent's path through the script, for example, by skipping questions that do not apply, and how to manipulate the order in which questions are asked.

Chapter 10, Repetitive questions, describes how to use loops to specify repeated questions efficiently and with the minimum of typing.

Chapter 11, Iteration and subscription, explains different ways of counting the number of times a repeated question has been asked and of extracting the responses chosen at a given repetition of a question.

Chapter 12, Assignment, explains how to assign values to variables, and shows how you can use special variables to control the way Quancept and mrInterview work.

Chapter 13, Logical expressions, describes how to test the value of a variable.

Chapter 14, Checking non-numeric responses, introduces keywords that can be used in logical expressions that test single-punched and multipunched variables.

Chapter 15, Working with numeric data, describes differences in the way the Quancept and mrInterview variants process arithmetic expressions and introduces ways of manipulating numeric data that is held in different formats.

Chapter 16, Ending the interview, describes different ways that interviews can be terminated or suspended from within the script.

Chapter 17, Data format, describes various methods of controlling the layout of the data files that Quancept creates. This includes how to add extra questions to a live script without invalidating the data already gathered, and how to insert non-question data into the data files.

Chapter 18, Report files, explains how to create formatted text files containing data and other information gathered during interviews.

Chapter 19, Very long response lists in CATI (response list databases), suggests ways of dealing with very long response lists that are efficient not only for scriptwriters but also for the interviewing program in terms of the resources that are required to run the script.

Chapter 20, ODBC databases in Quancept Web and mrInterview, describes how ODBC databases can be used for selecting information from an external database, and updating and adding to that database using information gathered during interviews.

Chapter 21, GNU GDBM relational databases in Quancept CATI, explains how to use GNU GDBM databases as response lists in Quancept CATI.

Chapter 22, Screen management with graphical user interfaces, explains how you can control the appearance of the CAPI, Web or mrInterview interview screen by inserting color, point size, typeface and font instructions in the script.

Chapter 23, Using page layout templates in mrInterview, describes how page templates and cascading stylesheets allow you to define script-independent page layouts by separating the formatting commands from the questionnaire content.

Chapter 24, Templates for grids, explains how to create grid templates that determine how grid questions will appear in the page.

Chapter 25, Using multiple script files (subsurveys), explains how library files allow you to share questions between projects and how to call a subsurvey script from within the main script.

Chapter 26, Text manipulation, discusses a number of text manipulation facilities including ways of extracting strings from longer texts.

Chapter 27, Callfuncs, describes Quancept's callable functions. Callable functions cover a wide range of features including date and time manipulation, running external programs during the interview, and switching recording facilities on and off when a project uses the Quancept Telephony System.

Chapter 28, Writing and using local callfuncs, gives a brief overview of the things to bear in mind when writing your own callable functions. It also explains how to call local functions from within the script.

Chapter 29, Using information in sample records, explains how to use information from the sample record in the interview and how to write new or updated information back to the sample record during or at the end of the interview.

Chapter 30, Quota control, describes how to implement quota control in CATI, Web and mrInterview scripts.

Chapter 31, Quota maintenance, describes the Quancept CATI quota program, the Quancept Web quotedit program, and the mrInterview Quota Setup program which you use to set up and inspect quotas.

Chapter 32, Parsing and compiling scripts, explains how to check your script for errors and how to create an interviewing program from your script.

Chapter 33, Testing scripts with qtip, explains how to test your script in a CATI scenario.

Chapter 34, Miscellaneous CATI facilities, describes how to use run batches of test interviews automatically, how to record interview keystrokes, how to review completed interviews and how to share a qoc file.

Chapter 35, Testing CAPI scripts, explains how to test your script using Quancept CAPI.

Chapter 36, Testing Quancept Web scripts, explains how to test your script using Quancept Web.

Chapter 37, Testing mrInterview scripts, describes how to test your script using mrInterview.

Chapter 38, File formats, describes the content of all files created when you parse and compile a script, as well as those created during interviewing.

Chapter 39, Managing CATI projects, discusses the layout of a Quancept CATI project directory and describes the environment variables that can be set to control the way in which a project runs.

Chapter 40, Managing Quancept Web projects, explains how to use the server management program and how to make projects available on the server. It also describes how you can use the project initialization file to customize the appearance of the interviewing screens using HTML code, and control how an interview runs.

Chapter 41, Managing mrInterview projects, tells you how to make projects available on your web site, how to run the activate wizard from the command line, and how to back up and delete old projects.

Chapter 42, Quancept CAPI design mode, explains how to use design mode to improve the layout of objects on the CAPI interviewing screens.

Appendix A, Limits, lists restrictions related to the use of certain statements in the language.

Appendix B, Keywords in questionnaires created with Build, documents keywords that the Quancept activity inserts in Quancept scripts that it creates from Build questionnaires.

Typographic conventions

In this manual, we have used the following typographical conventions.

Convention	Meaning
<i>bold italics</i>	In text, we use this to introduce terms that are specific to Quancept or mrInterview, or that may be new or unfamiliar, for example, <i>callfunc</i> .
<i>ordinary italics</i>	In text, we use this to refer to words that have special meaning in Quancept or mrInterview, such as commands or script keywords, for example, <i>ask</i> .
	In a syntax statement, we use this to describe information that you must provide, for example, <i>project</i> to refer to the name you give your current campaign, survey or project.
bold text	In a syntax statement, this indicates information you must enter exactly as shown.
courier font	Shows examples of scripts or commands exactly as you would type them.

Convention	Meaning
[]	In a syntax statement, square brackets almost always indicate optional information. A syntax statement looks like this: <code>label ask 'question text' resp resp_type [null] [ref] [dk]</code>
	where <i>resp_type</i> can be sp or mp . You enter the words ask , resp and the ' (single quote marks) exactly as shown. You must pick one <i>resp_type</i> from the choices given, either sp or mp , but not both, and enter it exactly as written. You change the words <i>label</i> and <i>question text</i> to whatever you like, subject to ordinary limitations. (Limitations would be described separately nearby; for example, text following the syntax statement would explain that <i>label</i> must be eight characters or less.)
	The words [null] [ref] and [dk] are optional because they are enclosed in square brackets. If you like, you can use one, two or all three of them. If you use any of them, you must type them exactly as shown but do not include the square brackets.
☞	Occasionally, square brackets are required as part of the syntax statement; this is clearly indicated in the text nearby.
☞	Directs your attention to another part of this manual or to another manual.
»	Highlights important information.

Differences between the Quancept and mrInterview variants

Although the majority of the Quancept language is common to Quancept CATI, Quancept CAPI, Quancept Web and mrInterview, there are some differences either in syntax or in functionality between the variants. This manual identifies areas where these differences occur using the following symbols:

Symbol	Marks
	Text for Quancept CATI only
	Text for Quancept CAPI only
	Text for Quancept Web only
	Text for mrInterview only

If a symbol appears in the margin, it applies to the whole section. If a symbol appears at the start of a paragraph, the paragraph is indented and the symbol applies to that paragraph. If a block of paragraphs is indented, the symbol applies to the whole block but normally appears on the first paragraph in the block only.

The rules for the use of unique identifiers for responses differ between Quancept and mrInterview. In Quancept they are optional, whereas in mrInterview they are optional but highly recommended. Example scripts that refer only to mrInterview have unique IDs for responses; example scripts that refer only to Quancept or that refer to both Quancept and mrInterview do not have unique IDs.

Glossary

Like most computer programs, Quancept and mrInterview have their own jargon. Some words are unique to the Quancept language, while others are applicable to computer-assisted interviewing systems in general. Other words in this glossary are standard market research terms used by Quancept and mrInterview, which we have included for users who are new to the industry.

Term	Meaning
Activate	To prepare an mrInterview script for interviewing. Once you have created an interviewing program from your script you run the activation wizard which copies all the necessary files onto your web server and creates a database file in which to store the interview data. See also <i>parse</i> , <i>QCompile</i> , <i>interviewing program</i> .
Array	A single variable that is broken down into a number of cells each of which may store a different value. For example, the first cell of the brand variable, brand(1), may contain the name of the first brand chosen, while the second cell of that variable, brand(2), may contain the name of the second brand chosen.
Callfunc	A callable function — a special type of statement in the Quancept language. It is used to execute routines written in the C programming language which allow you to perform more complex actions than would be possible with the standard statements in the Quancept language.
Categoric	A response type that refers to questions where the respondent may select one or more answers from a predefined list. See also <i>single punched</i> , <i>multipunched</i> .
Dummy question	A question that is never directly asked of a respondent. Its response is set by statements in the script. Dummy questions are often used to build up a list of responses given to a number of similar questions. For example, in a brand awareness study you may have separate questions for aided and unaided awareness but may wish to merge the two sets of answers so you have a single question variable containing all the brands the respondent knows. This question is the dummy question.

Term	Meaning
Environment variable	A variable whose value is defined outside Quancept but which determines how a particular aspect of Quancept will work. For example, if the environment variable QCNONULL has a value of 1, 'No answer' responses entered with the special response code <i>null</i> will be coded as blanks rather than with – signs. If the variable is set to zero or is not defined, these responses will always be coded as –. All environment variables have uppercase names.
Frozen response list	A list of responses which applies to a set of consecutive questions. The list is defined for the first question only and is then 'frozen' on the screen for all other questions which use it.
Integer	A whole number.
Interview path	The route the respondent has taken through the script. All questions asked and the names of all variables to which values have been assigned are stored and are said to be on the interview path. Changing the value of a question or variable on the interview path may affect the route through the script from that point onwards.
Interviewing program	The program that you run to conduct an interview. In Quancept CATI the interviewing program is called qtip . After you <i>parse</i> your script without errors, you type qtip <i>scriptname</i> at the command line or system prompt to run the script. This displays the script on the screen so that you can conduct an interview.
	In Quancept CAPI, Quancept Web and mrInterview, the interviewing program is produced when you <i>parse</i> and <i>compile</i> your script with no errors. This creates an executable file runs the interview. If you are using Quancept CAPI, the executable file is called <i>script.exe</i> and can be run from QCShell or QCompile. If you are using Quancept Web or mrInterview, the file is called <i>script.sif</i> and can be run via your web server.
Label	A name of eight or fewer characters that uniquely identifies a question or statement in a script. Labels on statements other than questions and defined lists act as markers for routing and similar statements which allow you to move from one part of the script to another.
Library file	A file of questions located in a central directory which may be included in any script which needs them.

Term	Meaning
Logical expression	A statement used for testing whether or not a given set of conditions is true. If the conditions are true, the expression is true, if the conditions are false, the expression is false. For example, if it is important that the respondent is over 18 years of age, you can collect the respondent's age in a question variable and then test the value of that variable to see whether or not it is greater than 18. The script can then be designed to follow different routes depending on whether the test is true or false.
Loop	A series of statements that are to be executed more than once.
Multipunched	A response type indicating that one or more answers may be chosen from the list of permissible answers.
Nesting	This applies to loops and <i>if</i> statements, and means placing one loop or <i>if</i> statement inside another. For example, if you have a number of questions to be asked about travel by trains, buses and taxis you could set these up as a loop. If you had a further set of questions about the three most recent journeys made using each method of transport, you might set these up as a separate loop inside the main loop. The inner loop is called a 'nested loop'.
Netting	Merging the values of two or more variables (usually question variables) into a single variable. For example, if q1 is red and q2 is green, and the two questions are netted into a variable called color, the value of color will be red; green.
Open-ended	A type of response indicating that the respondent is not restricted in the answer that may be given. The interviewing program stores the respondent's exact words so that they can be reviewed and coded later. This is also called a <i>Verbatim</i> response or a coded response.
Other specify	See <i>specified other</i> .
Parse	A computer term that means 'to check'. When you parse your script, the program qparse (for Quancept CATI) or QCompile (for Quancept CAPI, Quancept Web and mrInterview) evaluates it for certain errors and inconsistencies and issues warnings and error messages if any are found. You can run a parsed script that contains warnings (although you should be sure that you understand what the warnings mean), but if it contains errors you will not be able to run the script until the errors are corrected. You cannot run a script unless you parse it first. See also <i>interviewing program</i> .
Project directory	The directory in which all files to do with a project are created. Each project has its own project directory. Projects are sometimes called studies or jobs.

Term	Meaning
QCompile	The Quancept CAPI, Quancept Web and mrInterview program that parses your script into a usable file with which you can conduct interviews. See also <i>parse</i> .
Questionnaire definition	Defines a project in terms of the texts of the questions and their order, layout and routing, and the structure of the variables that hold the responses. The questionnaire definition can also hold variations of each text for use in different contexts (for example, interviewing and analysis) and in different interviewing environments (for example, CATI, paper and Web), and translations of the texts for use in multilingual studies. After a project is <i>activated</i> , the questionnaire definition stores details of the database in which the case (interview) data is stored.
Quota control	A method of ensuring that you interview a fixed number of respondents with given sets of characteristics; for example, 100 women and 90 men, or 250 sports car owners and 510 convertible car owners.
Real	A number containing a decimal point.
Routing	Statements that allow non-consecutive movement through the script; for example, if the respondent does not own a car, skip the questions about car insurance.
Sample	Records in a text file or database that contain information about people who are eligible to take part in the survey.
Sample database	In mrInterview, the database that holds sample records for all surveys.
Sample record	A record containing information about an individual who may be interviewed for a survey.
Sample management system	A program or suite of programs that controls how sample is used and how information is passed to and received from the interviewing program.
Sample table	In mrInterview, a table within the sample database that holds the sample records associated with a specific survey or group of surveys.
Script	The questionnaire written in the Quancept language.
Single punched	A response type indicating that only one answer may be chosen from the list of permissible answers.
Snapbacks	The action of going back to a question which has already been asked in order to check or change the answer given. Having snapped back, the interviewer or respondent may often be able to snap forward again to the original place in the script.

Term	Meaning
Specified other	A response that may be selected when the respondent gives an answer that is not in the list of permissible answers. Quancept CATI issues a secondary question asking the interviewer to type in the respondent's answer, whereas Quancept CAPI, Quancept Web and mrInterview display a box in which the interviewer or respondent may type the answer.
Stopped data directory	An interview need not run uninterrupted from start to finish: it can be stopped part way through and then restarted from that point at a later date. This might happen when you want respondents to try a product and then report back on their opinion of it, or when a respondent is unable to continue the interview now but will do so later on. When interviews are stopped, Quancept CATI, Quancept CAPI and Quancept Web store all the data gathered so far in a file unique to that respondent. These files are located in subdirectories of the stopped data directory which is itself part of the overall project directory.
Subsurvey	A complete script which is run as part of another script, for example, in an omnibus study. Also called a <i>secondary script</i> .
Variable	A ‘storage box’ for holding a value used in the script. Quancept and mrInterview have two main types of variables. A <i>question variable</i> holds the answer or answers to a question. A <i>temporary variable</i> holds a value that is assigned to it, usually using the <i>set</i> keyword in the script.
Verbatim	See <i>Open-ended</i> .

1 Writing in the Quancept language

Quancept telephone, personal and Web interviewing is controlled by a script. Like printed questionnaires, the script contains the questions to be asked, the acceptable responses for each question, and other statements for checking answers and determining exactly which questions will be asked of each respondent. However, because it is a computer program, the script is written in the special Quancept language.

You must be very precise when writing scripts. People are able to make judgments when using paper questionnaires but the computer interprets each statement literally. Before you can start writing your script, you need to understand the basic concepts of the Quancept language. This chapter describes how to lay out a script and introduces the following keywords:

comment	comments
continue	a place holder statement
end	the end of the script
main	the main part of the script
newline	defines a character for forced line breaks in text
screener	the screening part of the script

Other keywords appear in the examples. These are described in later chapters.

1.1 Components of a script

A script is a collection of text to be presented by the interviewing program and special Quancept words called **keywords**. Each keyword performs a separate task.

Questions and responses

Questions and responses are the basis of your script. A question is a special statement in the Quancept language for two reasons. First, it is one of only three statements that must have a label (the others are *define* and *fix*). Second, it is the only statement that has two distinct parts, the question text and the response list. A question and its response list consist of the following parts:

- The label that names the question.
- The keyword **ask**.
- The question text enclosed in single quotes.
- The keyword **resp**.
- The main response type. This identifies whether the response is numeric (integer or real), open ended or precoded.

- The list of acceptable responses, with each precoded response enclosed in single quotes. If the response type is *coded* (open ended), no list is needed because respondents can say what they like. However, you may indicate the number of codes to be reserved for later recoding of such responses.
- Subsidiary allowable responses such as ‘specified other’ (for when none of the precoded answers applies), no answer, don’t know or refused.
- Optional keywords that modify the standard behavior, appearance or coding of the question and its responses.

Here is an example of a question and its response list written in the Quancept language:

```
fveg1    ask 'Which other frozen vegetables do you buy?'
        resp coded (39)
```

In this example, *fveg1* is the label. As you can see, the label refers to the text of the question. *ask* is the keyword that tells Quancept and mrInterview that this is a question, and ‘Which other frozen vegetables do you buy?’ is the question to be asked. Notice how the whole question, including the question mark, is enclosed in single quotes.

The response list for this question starts on the next line. It does not have a label of its own but shares the label of the question to which it refers. This response list has the response type *coded* meaning that respondents may say whichever frozen vegetables they buy. Respondents are not restricted in their replies in any way. The number 39 in parentheses indicates the maximum number of codes which may be used when open ends are coded.

Here are some variations of this question and response list which allow different types of responses.

```
fveg2    ask 'Which of these frozen vegetables do you buy?'
        resp mp 'Peas' / 'Carrots' / 'Beans' / 'Sprouts' / 'Mixed'
```

Here, the respondent is being asked to say which of the given frozen vegetables he/she buys. He/she is allowed to specify more than one type because the response is defined as *mp* (multipunched or multicoded), but he/she may only choose from the vegetables shown in the list. Notice that each response is enclosed in single quotes and separated from its neighbors by a slash.

```
fveg3    ask 'Which frozen vegetable do you use most often?'
        resp sp 'Peas' / 'Beans' / 'Carrots' / 'Sprouts' / 'Mixed'
```

In this example, respondents must indicate which type of frozen vegetable they use most frequently. They must choose one type from the given list because the response is defined as *sp* (single punched or single coded).

The response list to the final question, shown below, is designed to accept any type of response. It is primarily a single-punched list where the respondent may choose one answer from the list of vegetables. If the respondent mentions a vegetable which does not appear in the list, or says he/she uses two vegetables equally, the interviewer may select *other* and enter the respondent’s exact words when prompted to do so. This response will then be coded and added to the data with other open ends later in the project cycle.

If a respondent gives no answer (for example, he/she never buys frozen vegetables), the interviewer may select the special *null* response representing no answer. If the respondent does not know which answer to give, the interviewer may type *dk*. If the respondent refuses to answer, the interviewer may type *ref*:

```
fveg4 ask 'Which frozen vegetable do you use most often?'
resp sp 'Peas' / 'Beans' / 'Carrots' / 'Sprouts' / 'Mixed'
      other null dk ref
```

☞ Questions and responses are discussed further in chapters 4 to 7.

Checking and verification statements

You can write statements that check or verify responses given to questions during the course of an interview. If a respondent gives inconsistent answers to certain questions, the interviewer can be alerted by the interviewing program and, if you choose, can be routed back to check those answers. With care, it is possible to write a script so that data is collected 100% clean and verified.

The way to do this is to use statements which relate one answer to another. You may, for example, want to check whether a certain response or set of responses was made to a given question. Here is an example of such a statement:

```
set pbc = or(fveg2 / 'peas' / 'beans' / 'carrots')
```

This checks whether the response to the question called fveg2 includes any of peas, beans or carrots. If this is so, the variable pbc is set to true. If the response does not include peas, beans and/or carrots, then pbc is false.

You will be able to recognize this type of statement because it will contain the special word **set**.

☞ A full description of these statements, with examples of how to use them, is found in chapter 12, Assignment.

Routing

Routing generally occurs when one or more questions do not apply to a certain respondent. For example, if a respondent does not use Brand X it does not make sense to ask more questions about it. There are two routing statements in the Quancept language: one that is obeyed every time the statement is read, and one that is obeyed only if certain conditions are true — for example, if the respondent is female.

Routing that must always be obeyed consists of the following parts:

- The keyword **goto**.
- The label which is the routing destination.

You will probably use this type of statement when you have different sections of the script for different types of respondent, and you want to jump from the end of each section to the start of a general section containing questions which must be answered by all respondents. Here is an example:

```
comment Questions for people who have seen the advertising
      (statements go here)
      goto collect
comment Questions for people who have not seen the advertising
      (statements go here)
comment Questions for all respondents
collect  continue
```

Most routing statements you write will be conditional; that is, they will be executed only if a certain set of conditions is true. There are two ways of writing this type of routing. The first is to include the routing command as part of the response list, when it will then consist of the following parts:

- A response text.
- The word **go**.
- The label which marks the routing destination.

For example:

```
advert  ask 'Have you seen the recent advertising for Brown''s
frozen peas?'
      resp sp 'Yes' / 'No' go ctn1
```

As part of the script, you want to talk about people's reactions to a recent advertising campaign for Brown's frozen peas. People who have not seen the advertisement cannot answer questions about it so we tell the interviewing program that if the reply is 'no', then we always want to go straight to the statement labeled ctn1.

Before continuing, look again at the question text. You will notice that Brown's has two single quotes (not a double quote) instead of just one. This is because Quancept and mrInterview associate a single quote with the beginning and end of a text to be displayed. If you had used only one single quote, Quancept and mrInterview would assume that the question ended with Brown. By using two single quotes you have shown that you mean Brown's.

The second way of writing conditional routing is to write a completely separate statement which defines the condition which must be true in order for the routing to be executed. Statements of this type consist of the following parts:

- The keyword **route**.
- A logical test enclosed in parentheses, defining the conditions under which the routing will be followed.
- The keyword **go**.

- The label which marks the routing destination.

Here is an example of routing written in this way:

```
route (child$='No') go exit
```

The statement enclosed in parentheses is the logical test. It asks whether the variable child\$ has the answer 'No'. If so, the interviewing program will route to the statement named 'exit'. If the variable has any other value, the interviewing program will continue with the next statement in the script.

You would generally use this type of routing when the next question to be asked depends on the answer to a previous question. Let us look at this statement in its proper context. Suppose you are conducting a survey about family life and one of the first questions you ask is whether there are children in the family. You then ask other basic demographic questions to build up a picture of the family before asking questions specifically related to children. If there are no children in a household, you would route to the end of the script. Your script could look like this:

```
name      ask 'What is your name?'
          resp coded
child     ask 'Are there any children in your household?'
          resp sp 'Yes' / 'No'
comment   Ask about children - age / gender / education
comment   If no children skip to end of script
          route (child = 'No') go exit
nchild    ask 'How many children do you have?'
          resp num 1+
          (statements go here)
exit      end
```

 Routing is discussed in chapter 9, Routing and flow control statements.

Routing allows you to skip parts of the script which do not apply to the current respondent. Another way of entering statements which apply to certain respondents only is to use an *if* statement. This has the following parts:

- The keyword **if**.
- A logical test enclosed in parentheses, defining the conditions under which the statements defined with *if* will be executed.
- The { character.
- The statements to be executed if the logical test is true.
- The } character.

For example:

```
if (age > 18) {  
school    ask 'At what age did you leave school?'  
          resp num 16 to 18  
univ     ask 'Did you go on to university?'  
          resp sp 'yes' / 'no'  
}  
ctn      continue
```

Here, the questions inside the { and } signs will be asked only if the respondent is over 18 years old. Respondents aged 18 or younger skip this section and go straight to the *continue* statement.

Sometimes you will want to perform one set of actions if the question or variable has the given value and another set of actions if it does not. You define this second set of actions with an **else** clause, as follows:

- The keyword **else**.
- The { character.
- Statements to be executed if the logical test defined with *if* is false.
- The } character.

Here is an example with both *if* and *else*:

```
comment pbc has a true or false value so we can check it by  
comment typing its name as a logical test by itself  
if (pbc) {  
d1      display 'Peas, beans and/or carrots used most often'  
}  
else   {  
d2      display 'Peas, beans and/or carrots not used most often'  
}
```

In both examples the statements inside the braces are indented. Indentation makes it easy to see where each *if/else* statement begins and ends but it is not obligatory.

Loops

You use loops when you want to ask a series of questions more than once. For example, you might want to ask a respondent the same set of questions about three different brands of frozen peas. Instead of typing the same set of questions three times you can put them into a loop so that each question is asked automatically for each of the three brands of peas.

A loop must start with the word **for** and finish with the word **next**. *for* tells you how many times the questions are to be asked and *next* sends you back to the start of the loop ready to ask the questions again. For example:

```
f1      for brand = 'Brown''s' / 'Smith''s' / 'Jones''
shop    ask 'Where do you usually buy '+brand+' brand of frozen peas?'
        resp sp 'Tesco' / 'Sainsbury' / 'Co-op' / 'Safeway'
size    ask 'Which size packet do you usually buy?'
        resp sp 'Family' / 'Large' / 'Medium' / 'Small'
next
```

In this example, the questions shop and size will be asked for each of the three brands of frozen peas. 'brand' is a pointer that remembers which brand you are asking about; first it points to Brown's, then to Smith's, and finally to Jones'. When the last question has been asked for the last brand, the interviewing program will go on to the question following the loop.

A question label can contain the answer to only one question. Therefore, when questions appear in a loop each question name is divided according to the number of times the question is asked. In the example, shop is asked three times so the three responses would be stored as shop(1), shop(2) and shop(3). Similarly, the responses to size would be stored as size(1), size(2) and size(3). This method of identifying individual responses is called **subscription**.

The text of the shop question also illustrates how to insert a variable text, 'brand', into the fixed text of a question.

☞ Displaying responses as part of a text is described in chapter 8, Displaying information on the screen.

To find out more about loops and subscription, see chapters 10 and 11.

Screen layout and formatting



Quancept CAPI, Quancept Web and mrInterview have special screen formatting capabilities that give you precise control over how text appears on the screen. You can use codes in the script to control the color, style, size, and font of the text. In the following example, written for CAPI, the first line will display in bold type, the second line will display in red, and the third line will display in 18-point type.

```
display '<b>Thank you for your help. Goodbye.'
display '<c:red>Thank you for your help. Goodbye.'
display '<p:18>Thank you for your help. Goodbye.'
```

Text formatting in Quancept Web and mrInterview is based on standard HTML commands. Here is how you would write the previous examples for Quancept Web and mrInterview:

```
display '<b>Thank you for your help. Goodbye.'
display '<font color="red">Thank you for your help. Goodbye.'
display '<font size="5">Thank you for your help. Goodbye.'
```

- ☞ mrInterview supports cascading stylesheets, so if you want all display statements to look the same you can specify the formatting in the stylesheet rather than separately for each statement in each script.
 - ☞ See section 23.12, Cascading style sheets, for details.
-

Font sizing in HTML is not based on point size. Instead, you define the text size relative to the default size 3 text. Possible values are between 1 and 7, with values less than 3 giving smaller text and values greater than 3 giving larger text.

Your company may develop standards for the way different types of text are presented. For example, it may be the standard to show all text in an 11-point, blue, Arial font and for question texts to be shown in bold. Rather than typing these instructions at the start of every text, you can define them as defaults either at the start of the script or in a separate file. This has the advantage of removing the formatting instructions from the text and means that any changes to the formatting instructions must be made in one place only.

Additionally, QCompile has a design mode for CAPI interviews that lets you control the layout of objects such as text, buttons, and icons on your screen.

-
- ☞ For more information about design mode, see chapter 42, Quancept CAPI design mode.
-

1.2 The character set

The characters and symbols that you may use in Quancept and mrInterview scripts are:

- The uppercase letters A to Z
 - The lowercase letters a to z
 - The digits 0 to 9
 - The space (blank) character
 - The tab character (when you press the TAB key)
 - The special symbols + - * / \ = < > () { } [] ' : " ? ; @ # ! ^ % , . £ \$
-  In Quancept Web and mrInterview, characters with accents or other diacritical marks, such as à, î and ç, all have special HTML codes. You'll find a full list of codes for special characters in any good HTML reference guide.

The exact meaning of many of the special symbols depends on the context in which they are used. The following list summarizes this usage. For further information, see the appropriate sections of this manual.

Symbol	Used for
+	An addition sign in arithmetic statements. In text, a pair of + signs encloses variables whose values are to be substituted in that text. In Quancept CATI interviewers use + to scroll forwards through a database or response list.
-	A subtraction sign in arithmetic statements. In Quancept CATI interviewers use – to scroll backwards through a database or response list.
*	A multiplication sign in arithmetic statements. In loops, a variable name followed by an asterisk in parentheses refers to the value of that variable for the current iteration of the loop. For instance, if the loop is being repeated for the fourth time, qname(*) refers to the answer given to qname the fourth time it was asked. With the <i>strngchk</i> function, used as part of a string pattern to match any character in a given position in the string.
/	A division sign in arithmetic statements. Separates responses in a single-punched, multipunched or numeric response list and in defined lists. Separates path and file names in Unix.
\	Removes the special meaning from certain symbols so that they can be displayed on the interview screen. To make a backslash appear in text, type two consecutive backslashes for Quancept CATI, Quancept Web and mrInterview, and four consecutive backslashes for Quancept CAPI.
=	Assigns a value to a variable. In logical tests, checks whether a variable has a given value. In Quancept CATI an interviewer may, when prompted to append verbatims (open ends) at the end of the interview, start the text with = to overwrite the existing text rather than appending to it. With response list databases, interviewers may type =key to select a record using its record key.
<	A logical operator meaning less than. In logical tests with multipunched texts, checks whether the response was chosen with any responses which come before it in the response list. In Quancept CATI, used as a special response by interviewers to snap back to the previous question. The notation << snaps back to the first question asked. When used in text in Quancept CAPI, this symbol truncates the text at that point and does not display on screen unless it is entered as < May not display in text in Quancept Web and mrInterview scripts unless entered as <

Symbol	Used for
>	<p>A logical operator meaning greater than.</p> <p>In logical tests with multipunched texts, checks whether the response was chosen with any responses which come after it in the response list.</p> <p>In Quancept CATI, used as a special response by interviewers to snap forward to the next question. The notation >> snaps forward as far as possible.</p> <p>In Quancept CAPI it may be necessary to type \> in order to display this character in text.</p> <p>May not display in text in Quancept Web and mrInterview scripts unless entered as &gt;;</p>
<>	<p>A logical operator meaning not equal to.</p> <p>In logical tests with multipunched texts, checks whether the response was chosen by itself.</p> <p>In Quancept CATI, used by interviewers to check which questions can be snapped to.</p>
()	<p>Enclose logical expressions in routing or <i>if</i> statements.</p> <p>In single-punched responses lists, enclose responses with common routing.</p> <p>In open-ended response lists, enclose the number of codes to be allocated for coding responses.</p> <p>In loops, enclose an asterisk to refer to a variable's subscript.</p> <p>In defined lists, enclose the number of codes to be reserved for responses in the list.</p> <p>In Quancept CATI, used by interviewers to enclose open-ended responses entered as part of a single-punched or multipunched response.</p>
{ }	Enclose statements forming an <i>if</i> or <i>else</i> clause.
[]	<p>Enclose a question variable whose current value is to be substituted in a response list.</p> <p>In Quancept CATI, enclose screen highlighting characters (for reverse video displays, and so on.).</p>
'	<p>A pair of single quotes encloses text such as question text or response text.</p> <p>Two consecutive single quotes inside a quoted text represent an apostrophe in the displayed text</p>
<hr/> <p>❖ Do not use the typographic " character in place of two single quotes as this will cause an error when you parse your script. Also, Quancept and mrInterview do not recognize the back-quote or ' character. If your text processor supports 'curly' or 'smart quotes', we recommend you switch off this facility as it may insert a ' character into your script.</p>	

Symbol	Used for
"	In Quancept CATI, used by interviewers to keep the current response to a question. No special meaning in the script, but see the note about the ' character above. May not display in text in Quancept Web and mrInterview scripts unless entered as "
:	In Quancept CATI, used by interviewers to check a response to a question. No special meaning in the script.
?	In Quancept CATI, used by interviewers to check which questions can be snapped to. In Quancept CATI, two consecutive question marks report the names and current values of snappable questions. No special meaning in the script.
;	In Quancept CATI, used by interviewers to obtain a serial number for a handwritten open-end. No special meaning in the script.
@	Forces a new line in displayed text on the screen but is not otherwise displayed. (You may define a different newline character if you need to display @ as part of the text.) In Quancept CATI, used by interviewers to obtain a serial number for a handwritten open-end.
!	In Quancept CATI, interviewers may type <i>text</i> ! to filter the responses displayed by a database response list, or ! by itself to cancel a filter applied to a database response list. In Quancept CATI, interviewers may type ! followed by a response code to cancel that response from the list of responses chosen for the current question. No special meaning in the script.
^	Followed by o to associate a specified other response with a response in a single-punched list. Followed by s to signal a single-punched response in an otherwise multipunched list. Followed by v to associate a scaling factor with the current response in a single-punched list. If used in text, ^ truncates the text at that point and is not displayed on the screen.
#	With the <i>strngchk</i> function, used as part of a string pattern to match any digit in a given position in the string.
%	With the <i>strngchk</i> function, used as part of a string pattern to match any letter in a given position in the string.
&	No special meaning in Quancept and mrInterview. May not display in text in Quancept Web and mrInterview scripts unless entered as &

Symbol	Used for
,	(comma). No special meaning in Quancept and mrInterview.
.	(period). Represents a decimal point regardless of your computer's locale setting.
£	(British keyboards only) No special meaning in Quancept and mrInterview. Does not display in text in Quancept Web and mrInterview scripts unless entered as &pound;
\$	In Quancept CATI, interviewers may type \$ to enter 'null' on a response list database question. No special meaning in mrInterview.

Uppercase versus lowercase letters

The interviewing program ignores differences between uppercase and lowercase letters in variable names and label names. For example, if the variable 'STATUS' is set to Married, the following comparison is still true:

```
status='Married'
```

The interviewing program is also case insensitive when testing the values of variables. For example, if you set the variable 'status' to Married:

```
set status = 'Married'
```

the tests status='married' and status='MARRIED' will both be true.

However, differences between uppercase and lowercase characters are recognized in any texts that are displayed. If you type a text in a combination of uppercase and lowercase, it will be displayed exactly as you type it in. For example:

```
ThE cAt SAT on the MAT
```

would be displayed exactly as it appears here.

-
- ☞ Although case is not important in text comparisons, spacing and punctuation are important. Two texts are considered a match only if they have identical spacing and punctuation.
 - ☞ See also the discussion about case and pattern matching between sublists and master lists in section 7.8, Using sublists as response lists.
-

1.3 Quancept language statements

Quick Reference

The general format of any statement is:

[label] keyword [other text or parameters]

Every statement contains a Quancept keyword that tells the parser (the syntax checker) what else it should expect to find in the statement. For example, the keyword *ask* is associated with question texts so when the parser reads an *ask* keyword it knows that the next thing it reads should be the question text. When it reaches the end of what it expects to read in the current statement (for example, the end of the question text), the parser assumes it has reached the start of the next statement.

Labels

Some statements have names, which are called ***labels***. Some statements must have labels and the parser will issue error messages if the label is missing. Labels on other statements act as markers for routing and similar statements which allow you to move from one part of the script to another, and are optional. In Quancept CATI, Quancept CAPI and Quancept Web all questions, defined lists and *fix* statements must have labels. Labels on other statements are optional. In mrInterview, you must label all statements that write information to the questionnaire definition (mdd) file — that is, all questions, defined lists, *fix* statements, *display* statements and *for* statements.

Labels consist of up to eight letters and digits and must start with a letter. They are case insensitive, so the labels 'name' and 'NAME' are seen as identical. Labels must not be the same as any of the keywords listed in chapter 2, 'Language summary'.

If you are writing scripts for Quancept CAPI, Quancept Web or mrInterview, do not use words that have special meanings in the C++ programming language. For mrInterview, you must also avoid words that are SQL keywords. You can find a full list of SQL keywords in the MSDN article entitled *Reserved Keywords*, available at <ms-help://MS.VSCC/MS.MSDNVS/tsqlref/ts_ra-rz_9oj1.htm>. Alternatively, open MSDN help and search for "odbc reserved keywords". The SQL keywords that are most likely to cause problems if you use them as labels are those used by the SPSS MR Data Model. These are as follows:

Add	Alter	As	Asc	Boolean	By
Categorical	Column	Create	Date	Default	Delete
Desc	Double	Drop	Exec	From	Group
Having	Insert	Into	Level	Long	Object
On	Order	Select	Set	Table	Text
Truncate	Update	Values	Where	With	

If in doubt, avoid using proper words as label names and use codes such as q1 instead.

A label must start in column 1 and it must be unique. If you use the same label again later in the script, it must refer back to the original statement to which it was attached. For example, if you have a question called *name* and you refer to that label in another statement later in the script, the interviewing program expects you to be referring to the question which was called *name*.

Statements

Most statements must start in columns other than column one. There are three exceptions:

- Label names.
- Continuation text of questions and other display text.
- Comments beginning with the keyword *comment*.

Here is an example that illustrates these points:

```
comment An example of script layout
        display 'Now I''m going to ask you some questions
about supermarket shopping...@@'
f1          for i=1 to 10
shop         ask 'Which supermarket did you shop at next? '
            resp sp 'Sainsbury' / 'Tesco' / 'Co-op' null go exit
buy          ask ' What did you buy there? '
            resp coded
next
include 'extras'
exit         display 'Thank you for your help'
```

Continuation lines

Long statements such as questions or response lists may spread over several lines. In these cases, you may enter question or response text in column one as long as the parser is expecting to read a continuation of the current text. All texts are enclosed in single quotes; as long as the text in column one is part of a quoted text, or the character in column one is a single quote, Quancept and mrInterview will accept this.

(All examples in the rest of this manual show continuation lines starting in column 2 or later. This is to ensure that if you copy the example it will work, without you needing to remember to type a space at the end of the previous line.)

Naming temporary variables

Temporary variables hold information that you want to use or manipulate during the course of the interview but which does not automatically become part of the data for an interview. There are numerous examples of temporary variables in the sample questionnaires and the script snippets throughout this manual, but here are a couple of common examples to start with.

- For arithmetic calculations. You can assign the result of calculation to a variable so that it can be used elsewhere in the script.
- As a counter. For instance, if you want to ask a question three times (for example, about three different products) you can increment the value of a temporary variable by 1 each time the question is asked.
- To hold the return value of a function. Statements that perform arithmetic or logical tests need somewhere to place the result of the test. You use temporary variables for this.

Every temporary variable in a script must have a unique name. Like labels, variable names may consist of up to eight letters and digits and must start with a letter. They must not match the name of a Quancept keyword or any word that has a special meaning in the C++ or SQL languages. Once you have named a temporary variable in the script, any subsequent occurrences of that name are assumed to refer to the original variable. If you do not want to lose the original value of the variable, create a new temporary variable instead.

1.4 Spaces

Spaces are allowed anywhere in a script except within names and keywords. Keywords are listed in Chapter 2. If you type in the text of a question leaving more than one space between each word, the question will be displayed on the screen exactly as you typed it. For example, if you type:

```
q6      ask 'Did your      cat      sit      on the      mat?'
```

the question will be displayed on your screen as:

```
Did your      cat      sit      on the      mat?
```

In statements other than those that display text on the screen, extra spaces are ignored.

```
set      tran      =      t (5 )
```

is the same as:

```
set tran = t(5)      or      set tran=t(5)
```

Spaces in response texts are important if your script includes statements that test the response given to a question. For example:

```
color  ask 'Which color combination did you like best?'
      resp sp 'Red and yellow' / 'Blue and green'
      if (color = 'Red and yellow') {
why       ask 'Why was red and yellow best?'
      resp coded
    }
```

Here, the two occurrences of 'Red and yellow' have the same number of spaces in the same places, so if the respondent chooses 'Red and yellow' at the color question Quancept and mrInterview will ask why. If the two texts had different numbers of spaces, they would no longer match and the second question would never be asked even if the respondent chose 'Red and yellow'.

Tabs (that is, pressing the TAB key) are valid in the same places as spaces. We recommend that you use them for setting a constant indentation for statements because this increases the clarity of your script, making it easy to see where loops or *if/else* statements begin and end.

1.5 New lines

Quick Reference

Use the @ character to force a line break in text:

'text@text'

Displayed text (such as questions) may spread over several lines. If your text is typed on several lines, the interviewing program will ignore the line breaks and will fit as many whole words on a line as possible before starting the next line. The interviewing program does not split words between two lines. The text:

The cat sat on the mat

is printed as a single line as long as there is room on the current line. If not, it is split in between two words. For example:

.....The cat sat
on the mat.....

To split the sentence between the words cat and sat, you should use the @ character. The text:

The cat@sat on the mat

will generate the two lines:

The cat
sat on the mat

When you type a @ symbol in a text, there is no need to start a new line unless you want to. Some people find it easier to see how the text will appear on the screen if they always start a new line after @.

If you have to type your text on more than one line, remember to include a space either at the end of every continued line or at the start of all continuation lines. If you forget these spaces, Quancept and mrInterview will display the word at the start of the continuation line immediately after the word from the end of the previous line, with no intermediate spaces. For example:

```
col 1
q1 ask 'Why did your cat
sit on the mat?'
```

would be displayed as ‘Why did your catsit on the mat?’ (assuming that the first line ends with the ‘t’ in cat), whereas

```
col 1
q1 ask 'Why did your cat
sit on the mat?'
```

with a space in column 1 of the continuation line causes the question to be displayed as ‘Why did your cat sit on the mat?’. An alternative is to type a space at the end of the first line, as indicated by the underscore character in the example below:

```
col 1
q1 ask 'Why did your cat_
sit on the mat?'
```

Using a character other than @ for new lines

Quick Reference

To define a character other than @ as the new-line character, type:

CAPI: **setfmt newline= 'character(s)'**

CATI/Web/mrInterview: **set newline = 'character'**

To return to the default, type:

CAPI: **setfmt newline='@'**

CATI/Web/mrInterview: **unset newline**

-  For example, in Quancept CAPI you could use the characters #% to represent a line break, as follows:

```
setfmt newline = '#%'
d1 display 'Please address any comments about this interview to:#%
doc@spssmr.com
```

-
-  Not all character combinations work as new-line variables. We recommend you use #% as it is unlikely to occur as a true character combination in a script.
-

-  In Quancept CATI, Quancept Web and mrInterview you could use the \ character to represent a line break, as follows:

```
set newline = '\'
d1 display 'Please address any comments about this interview to:\\
doc@spssmr.com
```

Notice that the \ character is typed twice when it appears in the text. This is the normal procedure whenever you want to type \ as part of a text.

1.6 Putting comments in your script

Quick Reference

To enter a comment in the script, type:

comment *text*

To make your script easier to read and understand, Quancept and mrInterview allow the use of comment statements anywhere except between:

- a question and its response list
- the closing } of an *if* clause and the *else* statement that follows it.

They do not affect the running of the program; they just give you information about it. Any line beginning with the word **comment** is ignored by Quancept and mrInterview. You may add text to the comment line, but it must be separated from the keyword *comment* by at least a single space:

```
comment This is an example of a comment line
```

If the comment text requires more than one line, each line must start with *comment*.

You can use comments to remind you who the job is for or when the script was written, and to explain what the more complex parts of the script are designed to do. Such comments are helpful if you take over a job from someone else. Anything special that you need to know about the script can be stored in a comment statement.

-
- ❖ Although comments are not essential, we strongly recommend that you take the extra time to write them. They may save you a lot of time in the future if you or someone else needs to modify the script.
-

1.7 The continue keyword

Quick Reference

To insert a label in a script as a routing destination, type:

label continue

continue is a place holder keyword which has no effect on the way the script is read; the interviewing program reads it and then reads and executes the following statement. You would normally use *continue* when you need to insert a label in a script as a routing destination, but you do not want to carry out any action at that point. One example is a script containing a number of sections (for example, for different products tried) with a general section at the end to which all respondents are eventually routed. Instead of routing everyone direct to the first general question, you might insert a labeled *continue* statement before that question to gather all respondents ready for the general section:

```

qa      ask 'Which country did you visit?'
        resp sp 'England' / 'France' go f1
comment England questions here
        (statements go here)
        goto all
comment France questions here
f1      ask 'question about France'
        (statements go here)

all      continue
comment General questions here

```

Anyone answering England at question qa answers questions about it and then skips directly to the statement labeled 'all', missing out the questions about France. They then continue through to the general questions. Anyone answering 'France' at question qa is routed over the questions about England to question f1. At the end of that section they read the *continue* which takes them on to the general questions answered by all respondents.

1.8 The end of the script

Quick Reference

To mark the end of the script, type:

end

All Quancept and mrInterview scripts must finish with the keyword **end**. If you want to route to the end of a script, you may precede *end* with a label and then use *go* or *goto* to route to that label:

```
q1      ask 'What did you do when you left school?'
        resp sp 'Joined the Youth Opportunities Program' /
                'Took a full-time job' / 'Took a part-time job' /
                'Did voluntary work' / 'Nothing' go exit
        more statements
exit    end
```

If you want to have a message displayed on the screen at the end of the interview, the message must precede the *end* statement:

```
q21     ask 'What is your opinion of the service offered?'
        resp coded (14)
d1      display 'Thank you for your help'
        pause
        end
```

1.9 Sections in a script



Quick Reference

Mark the start of the screening section of a script with:

screener

Mark the start of the main section of the script with:

main

Some scripts may contain screening questions before the main part of the script. If you want to differentiate between these parts of the script in the accounting information generated by the interviewing program, insert the keywords **screener** and **main** at the appropriate places in the script. For instance, you may find it useful to know how much of the total interview time was spent in the screener section of the script, or to know whether the interview was terminated in the screener or the main section of the script.

-
- ☞ To find out more about the accounting information that qtip stores, see 'The act file' in chapter 38, File formats.
-

If neither keyword is used, the interviewing program assumes that the whole script is *main*. If both keywords are used, each one must appear once only and *screener* must appear before *main*. For example:

```
comment Find eligible respondent
      screener
age      ask 'How old are you?'
      resp num 18+
      route (age < 20 .xor. age > 30) go exit
school   ask 'Did you leave school before you were 18?'
      resp sp 'Yes' / 'No' go exit

comment Only 20-30 year-olds who left school before they were 18
m1      main
q1      ask 'first main question text'
      resp responses
      (statements go here)
exit    end
```

If you are working on a long or complex script, you may wish to break the script down into more than just the screening and main sections. For example, you may create sections for screening questions, demographics, pets owned, shopping habits, leisure activities, and so on. This is useful when you want interviewing supervisors to see which section of the script each interviewer is currently in.

The Quancept language provides the callable function **showinf** for this sort of division. The following example gives you an idea of how a script which uses this facility would look.

```
comment Screener section
      set sectname = 'screen'
      callfunc('showinf',sectname)
sql      ask 'Are you or is anyone in your immediate family employed in
the market research business?'
      resp sp 'Yes' go exit / 'No'
      more statements
comment Demographic section
      set sectname = 'demog'
      callfunc('showinf',sectname)
dq1      ask 'How old are you?'
      resp num 18 to 99
      more statements
exit    end
```

-
- ☞ A full explanation of *showinf* is given in section 27.9, Naming script sections for use by the supervisory program.
-

1.10 Naming script files

- ❑❑❑ For easy management, script files are usually given the same name as the project. A project called cars, for instance, will have a script called cars and associated files called cars.xxx, where xxx is a three-character extension that identifies the file's function and content.

You can give your scripts (and projects) names of up to 16 characters, as long as the name starts with a single-byte letter. The rest of the name can consist of any alphanumeric characters and the @, \$ and _ symbols. Spaces are not allowed in filenames.

☞ Japanese, Hebrew and Greek letters are not single-byte letters.

You may not choose filenames that are the names of any of the SQL databases, such as master, model, tempdb or msdb.

When QCompile processes a script with a long name, it copies the script into a temporary file and then parses and compiles the temporary file and creates the metadata before copying the file back to its original name.

If you are creating projects for use with DimensionNet, you are strongly advised to use only the characters, A–Z, a–z, and 0–9 in all names as other characters may not be passed correctly when present in a URL.

2 Language summary

This chapter is a quick reference guide to writing scripts, which you may like to copy and keep handy once you become more familiar with Quancept or mrInterview. It covers the following topics:

- Keywords in scripts
- Keywords used with *set*
- Callable functions
- Special responses in qtip
- Shared memory commands for *dbase* questions
- External Unix databases

2.1 Keywords in scripts

The following table lists all Quancept language keywords except those that are used as part of *set* statements and those that are the names of callable functions. These keywords are listed in separate sections later on.

A tick in a column means that the keyword works in that variant of Quancept or mrInterview; a cross means that it does not. The letter C indicates that the keyword parses and compiles in that variant but has no effect in the interview. The letter W indicates that the parser issues a warning when it encounters this keyword and lets you choose whether to continue. If you choose to continue the keyword will have no effect on the interview.

Keyword	Function	CATI	CAPI	Web	mrInt.
*	Automatic calculation of variable subscripts	✓	✓	✓	✓
@	Forces a new line in displayed text	✓	✓	✓	✓
[\$eval(...)\$]	Evaluates an expression and substitutes its value in a text	✗	✗	✗	✓
acczero	Prints accounting information on card zero	✓	✗	✗	✗
allow	Permits early completion of an interview	✓	W	W	✗
.and.	Expression is true if both/all subexpressions are true	✓	✓	✓	✓
arrays	Maximum number of subscripts for temporary variables	✓	C	C	C

Keyword	Function	CATI	CAPI	Web	mrInt.
ask	Question keyword	✓	✓	✓	✓
atoz	Displays the response list in alphabetical order	✓	✓	✓	✓
audio	Allows an open-ended response to be recorded	✗	✓	✗	C
#audio=	Plays a sound file or record an open-ended response (only when using the QTS)	✓	✗	✗	✗
autojump	Automatically jumps to the next question when a response is chosen from a single-punched response list	✗	✓	C	✗
axis	Obsolete keyword, do not use as a label name	✓	✗	✗	✗
bkcolor	Sets the screen background color	✗	✓	C	C
bkground	Sets the screen background image	✗	✓	C	C
bkstyle	Defines how to display the screen background image	✗	✓	C	C
call	Calls a nested block. Not implemented.	✓	✗	✗	✗
callfunc	Calls a function	✓	✓	✓	✓
cardcols	Location of card number in the .dat file	✗	✓	✓	C
cents	Currency response type. Not implemented	✓	✗	✗	✗
chgud	Reserved word, do not use as a label	✓	✗	✗	✗
coded	Response type for open-ended responses	✓	✓	✓	✓
codscheme	Defines the format in which single-punched and multipunched data is written to the .dat file	✓	W	W	W
col	Defines the card and column to be allocated to a question	✓	✗	✗	✗
colgrid	Presents all iterations of a question on one screen with the loop control values as the columns of the grid	W	✓	✓	✓
column 100	Resets (switches off) card and column mapping set with <i>col</i> and <i>column</i>	✓	✗	✗	✗
column	Defines the card and column to be allocated to the following and subsequent questions	✓	✗	✗	✗
comment	Comments	✓	✓	✓	✓
connect	Connects to an ODBC database	✗	✗	✓	✓

Keyword	Function	CATI	CAPI	Web	mrInt.
continue	A place-holder statement generally used for routing	✓	✓	✓	✓
control	Determines whether response lists will be displayed as drop-down lists or combo (list) boxes	✗	C	✓	✓
currency	Currency response. Not implemented	✓	✗	✗	✗
currencybig	Currency response with two decimal places. Not implemented	✓	✗	✗	✗
datemap	Defines data layout for the dat file	✗	✓	✓	C
dbase	Response type for a single response chosen from a database	✓	✓	C	✗
connect	Establishes a connection to an ODBC database	✗	✗	✗	✓
dbclose	Closes a connection to an ODBC database	✗	✗	✓	✓
decrm	Decrement quotas in named file	✓	✗	✗	✗
define	Defines a list of responses	✓	✓	✓	✓
disallow	Disables early completion	✓	✗	✗	✗
display	Displays a text temporarily on the screen	✓	✓	✓	✓
dk	A response meaning 'don't know'	✓	✓	✓	✓
dollars	Currency response with two decimal places. Not implemented	✓	✗	✗	✗
dummyask	Defines a dummy question	✓	✓	✓	✓
dummyother	Defines a dummy specified other response	✓	✗	✗	✗
echo	Echoes responses on the screen during interviews	✓	✓	C	✗
editopen	Forces editing of an open end	✓	✓	✗	✗
eisrec	Record for the EIS dialer. Obsolete	✓	✗	✗	✗
else	Statements to be executed when <i>if</i> condition is false	✓	✓	✓	✓
empty	Tests whether a question is unanswered.	✗	✗	✗	✓
end	End of the questionnaire (main script) or end of subsurvey	✓	✓	✓	✓

Keyword	Function	CATI	CAPI	Web	mrInt.
eximport	Connects to a GNU database file	✓	✗	✗	✗
export	Overwrites or creates a record in a GNU database	✓	✗	✗	✗
file	Names the quota file to read	✓	C	✓	✗
fix	Copies text and variables into the data	✓	✓	✓	✓
fixall	Copies text and variables into the same location on all cards in a record. Not implemented	✗	✗	✗	✗
for	Starts a loop	✓	✓	✓	✓
force	Codes extra questions at the end of the record	✓	✓	✓	✗
formhead	Defines the header for a multi-question form. Not implemented	✓	✗	✗	✗
formquest	Adds a question to a multi-question form. Not implemented	✓	✗	✗	✗
freeze	Response list common to several questions	✓	✓	✓	✓
getud	Reserved word, do not use as a label	✓	✗	✗	✗
go	Routing within a response list or with <i>route</i>	✓	✓	✓	✓
gosub	Routing with saved return. Not implemented	✓	✗	✗	✗
goto	Routing outside response lists	✓	✓	✓	✓
gotosms	Returns to the SMS menu	✓	✗	✗	✗
gototnms	Obsolete keyword	✓	✗	✗	✗
grouped	With loops, assigns columns in the order questions are asked	✓	✓	✓	C
help	Displays help. Not implemented	✓	✗	✗	✗
highlight	Reserved word, do not use as a label	✗	✗	✗	✗
hsetcols	Lists responses across the screen with responses above the response button	✗	✓	✓	✓
if	Starts a group of conditional statements	✓	✓	✓	✓
import	Imports and views a record from a GNU database	✓	✗	✗	✗
impupdate	Imports, views and updates existing records, or creates new records in a GNU database	✓	✗	✗	✗

Keyword	Function	CATI	CAPI	Web	mrInt.
in	Associates a sublist of responses with a master list	✓	✓	✓	✓
include	Loads the contents of a named file	✓	✓	✓	✓
layout	Reserved word, do not use as a label	✗	✓	✗	✗
list	Response type for defined response lists	✓	✓	✓	✓
main	Starts the main section of the questionnaire	✓	W	W	W
mapothzero	Codes specified other as zero in first column of field	✓	✓	✓	C
maxrecord	Sets the maximum recording time for an open-ended response	✗	✓	C	✗
maxtime	Displays an item for up to a maximum time limit and then continues automatically	✗	✓	C	✗
maxvideo	Plays a video recording full screen	✗	✓	C	✗
mcodscheme	Defines the format in which single-punched and multipunched data is written to the dat file	✓	W	W	W
messagebox	Displays text in a message box	✗	✓	C	✗
mintime	Displays an item for at least a minimum time limit	✗	✓	C	✗
mmaudio	Plays a sound recording	✗	✓	C	✗
mmcaption	Defines where to display response texts for responses with <i>mmpicture</i>	✗	✓	C	C
mmpicsize	Specifies the size at which to display a picture	✗	✓	C	C
mmpicture	Displays a picture file	✗	✓	C	C
mmpicunit	Defines the unit of measurement for <i>mmpicsize</i>	✗	✓	C	C
mmvideo	Plays a video recording	✗	✓	C	✗
mp	Response type for multipunched responses	✓	✓	✓	✓
mp #	Limits the number of responses chosen from a multipunched response list	✓	✓	✓	✓
multitask	Displays more than one question on the screen	✗	✓	✓	✓
multidbase	Allow multiselection from a response database	✗	✓	C	✗
newcard	Starts a new card	✓	✓	✓	✓

Keyword	Function	CATI	CAPI	Web	mrInt.
newline	Defines the newline character	✓	C	✓	✓
newpage	Starts a new page in the printed questionnaire	✓	W	W	W
newran	Marks the start of a statement group in a block of randomized questions	✗	✓	✓	✓
newrot	Mark the start of a statement group in a block of rotated questions	✗	✓	✓	✓
next	Ends a loop	✓	✓	✓	✓
noblank	Prevents blank cards from being written. (This is the default behavior for Quancept CAPI and Quancept Web.)	✓	✗	✗	W
noclean	Does not clean data after changes to routing	✓	W	W	W
nodata	Reserves no columns for this question	✓	✓	✓	✓
not	Uses responses not in the named sublist or question	✓	✓	✓	✓
.not.	Negates a logical expression	✓	✓	✓	✓
notemp	Parser to fail when temporary variables encountered	✓	✓	✓	✓
null	A response meaning ‘no answer’	✓	✓	✓	✓
num	Response type for integer responses	✓	✓	✓	✓
^o	Automatic specified other prompt for <i>sp/mp</i> responses	✓	✓	✓	✓
onresp	Global conditional routing	✓	✓	C	✗
.or.	Expression is true if at least one subexpression is true	✓	✓	✓	✓
other	Specified other — an automatic ‘please specify’ question is issued	✓	✓	✓	✓
othzero	Defines how specified other will be coded in the dat file	✗	✓	✓	✓
parsesmall	Obsolete keyword, do not use as a label	✓	C	C	C
pass	Names a variable to indicate whether quota cell is full	✓	C	✓	✓
pause	Causes display text to remain static	✓	✓	✓	C

Keyword	Function	CATI	CAPI	Web	mrInt.
peekcoded	Returns the value of the next text serial number. Not implemented	✓	✗	✗	✗
pence	Currency response. Not implemented	✓	✗	✗	✗
pgstyle()	Associates a page template with a script	✗	✗	✗	✓
ph1error	Fail the parser with one error. Not implemented	✓	✗	✗	✗
poolable	Overlays identical texts. Not implemented	✓	✗	✗	✗
pounds	Currency response with two decimal places. Not implemented	✓	✗	✗	✗
print	Prints text on a questionnaire	✓	W	W	W
privsig	Defines private status code	✓	W	W	W
promptoth	Automatic specified other prompt for <i>sp/mp</i> responses	✓	✗	✗	✓
protect	Displays text on the screen until specifically removed	✓	✓	✓	✓
putud	Reserved word, do not use as a label	✓	✗	✗	✗
qexit	Terminates the interview as if <i>end</i> had been reached	✓	C	C	C
qname	Displays the name of the current question as it appears in the script.	W	✓	✗	✓
qnameful	Displays the full name of the current question, including the iteration number, if any	✗	✗	✗	✓
quota	Tests quotas	✓	C	✓	✓
ran	Randomizes responses in a response list	✓	✓	✓	✓
ranend	Ends a randomized question block	✓	✓	✓	✓
ranstart	Starts a randomized question block	✓	✓	✓	✓
real	Response type for real responses	✓	✓	✓	✓
receive	Reserved word, do not use as a label	✓	✗	✗	✗
ref	A response meaning 'refused'	✓	✓	✓	✓
remove	Removes a response from those chosen from a multipunched list	✗	✓	✓	✓
resp	Response list	✓	✓	✓	✓

Keyword	Function	CATI	CAPI	Web	mrInt.
respvalue	Assigns a response value. Not implemented	✓	✗	✗	✗
rfclose	Closes a report file	✗	✓	✗	✗
rfopen	Opens a report file	✗	✓	✗	✗
rfprint	Prints to a report file	✗	✓	✗	✗
rlayout	Reserved word, do not use as a label	✗	✓	✗	✗
rot	Rotates responses in a list	✓	✓	✓	✓
rotend	Ends a rotated question block	✓	✓	✓	✓
rotranfix	Fixes the position of a response in a randomized or rotated list	✓	✓	✗	✓
rottransub	Groups a set of responses in a randomized or rotated list	✓	✓	✗	✓
rotstart	Starts a rotated question block	✓	✓	✓	✓
route	Conditional routing	✓	✓	✓	✓
rowgrid	Presents all iterations of a question on one screen with the response texts as the columns of the grid	W	✓	✓	✓
rtgosub	Return from a <i>gosub</i> block. Not implemented	✓	✗	✗	✗
scodscheme	Defines the format in which single-punched and multipunched data is written to the dat file	✓	W	W	W
screener	Screening section of the questionnaire	✓	✗	✗	✗
scribble	Allows open-ended responses to be written on screen	✗	✓	✓	✗
serialcols	Defines the location of the serial number in the dat file	✗	✓	✓	C
set	Assigns a value to a variable	✓	✓	✓	✓
setcols	Defines the number of columns to use for displaying responses	✗	✓	✓	✓
setfmt	Defines text layout and appearance	✗	✓	✓	✓
setstr()	Extracts characters from a text and copies them into a variable	✓	✗	✗	✗
showform	Reserved word, do not use as a label	✓	✗	✗	✗
signal	Forces interview to terminate in a given state	✓	✓	✓	✓

Keyword	Function	CATI	CAPI	Web	mrInt.
slider	Displays a numeric response list as a slide bar	X	✓	C	X
source	Included file name. Not implemented	✓	X	X	X
sp	Response type for a single-punched response list; also used to flag a single-punched response in a multipunched response list	✓	✓	✓	✓
^s	Single-punched response in a multipunched response list	✓	✓	✓	✓
start	Point at which to restart a stopped interview (no longer needed)	✓	X	X	X
step	Incrementation by more than 1 in a numeric loop	✓	✓	✓	✓
stop	Stops an interview temporarily	✓	✓	C	✓
ststest	Returns True if this is a test interview	X	X	X	✓
#SUB	Executes a subsidiary script	✓	✓	C	C
substitute	Assigns a value or a command to a variable	W	✓	✓	✓
substr	Extracts characters from a text	✓	X	X	X
testblock	Callable block of statements. Not implemented	✓	X	X	X
text	Reserved word do not use as a label	✓	X	X	X
textarea	Defines the size of a multi-line text box	X	C	✓	✓
textline	Defines the length of a single-line text or numeric box	X	C	✓	✓
tnms	Obsolete keyword, do not use as a label	✓	X	X	X
to	Ranges in loops and numeric response lists	✓	✓	✓	✓
transmit	Reserved word, so not use as a label	✓	X	X	X
unfreeze	Deletes a frozen response list from the screen	✓	✓	✓	✓
unpoolable	Stores identical texts individually. Not implemented	✓	X	X	X
unprotect	Deletes protected text from the screen	✓	✓	✓	✓
unset	Removes a variable's value	✓	✓	✓	✓
usecurrmdd	Causes the compiler to use the current version of the MDM document and not to add another version to the mdd file.	X	X	X	✓

Keyword	Function	CATI	CAPI	Web	mrInt.
useform	Joins questions to a multi-question form. Not implemented	✓	✗	✗	✗
usefile	Names a file to be used by the conv8 preprocessor. Not implemented	✓	✗	✗	✗
usemapfile	Causes the compiler to use the <i>script.qqc.map</i> file to map variable and response names when creating metadata	✗	✗	✗	✓
^v	Defines a scaling factor for a response	✓	✗	✗	✗
vsetcols	Lists responses down the screen in the specified number of columns with response text appearing above the response button	✗	✓	✓	✓
warntemp	Warns when temporary variables are used	✓	✓	✓	✓
windfmt	Default format for <i>display</i> text	✗	✓	C	C
winfmt	Default format for all texts	✗	✓	C	C
winpfmt	Default format for <i>protect</i> text	✗	✓	C	C
winqfmt	Default format for questions	✗	✓	C	C
winrfmt	Default format for responses	✗	✓	C	C
xdisplay	Displays text from a file temporarily	✓	✗	✗	✗
xor	Requires only one response from a list to be given	✓	✓	✓	✓
.xor.	Expression is true if only one subexpression is true	✓	✗	✗	✗
xprotect	Displays text from an external file until it is explicitly removed	✓	✗	✗	✗

2.2 Keywords used with *set*

This section lists keywords and other values that you use with *set*. The first table shows values you can assign; the second table shows some special keywords you can use with *set* to return certain values, such as date and time; and the third table shows keywords whose values can be changed with *set*.

set variable=value

Value	Description	CATI	CAPI	Web	mrInt.
'text'	Assigns a value to a variable	✓	✓	✓	✓
and(...)	Checks whether all responses in a given set were chosen from a multipunched response list	✓	✓	✓	✓
bit(...)	Checks whether a particular response number was chosen from a single-punched or a multipunched list	✓	✓	✓	✓
dbquery(...)	Defines a query to be passed to an ODBC database	✗	✗	✓	✓
dbrecono(...)	Saves a database record number in a variable	✓	✓	C	✗
elm(...)	Finds the text of a response in a single-punched or a multipunched list according to its position in the list	✓	✓	✓	✓
<i>integer</i>	Assigns an integer value to a variable	✓	✓	✓	✓
logical(...)	Assigns a true or false value to a variable according to the value of a logical expression	✓	✓	✓	✓
mention(...)	Stores the order in which responses are selected from a multipunched response list	✓	✓	✓	✓
nbit(...)	Checks whether a particular response was chosen from a single-punched response list	✓	✓	✓	✓
nselrec(...)	Selects a record from a database based on the line number	✓	✓	✗	✓
nsubstr	Extracts characters from a text and converts them to a number	✓	W	W	✗
numb(...)	Finds the number of responses chosen from a single-punched or multipunched list	✓	✓	✓	✓
numv(...)	Finds the number of responses available in a single-punched or multipunched list	✓	✓	✓	✓
or(...)	Checks whether at least one of the given responses was chosen from a multipunched response list	✓	✓	✓	✓
othertext(...)	Extracts individual texts from a list of open ends entered for o responses	✓	✓	✓	✓

Value	Description	CATI	CAPI	Web	mrInt.
selfld(...)	Copies a field from a response list database record or from an ODBC database record into a variable	✓	✓	✓	✓
selrec(...)	Selects a record from a response list database according to the text in the first field of the record. Selects a record from an ODBC database that contains a given value in a given field.	✓	✓	✓	✓
substr(...)	Extracts characters from a text	✓	✗	✗	✗
xor(...)	Checks whether only one of a set of responses was chosen from a multipunched list	✓	✓	✓	✓

set variable = *special keyword*

Keyword	Returns	CATI	CAPI	Web	mrInt.
cell	The quota cell a respondent belongs in for the most recently executed <i>quota</i> statement	✓	✗	✗	✗
date	The date on which the interview was started	✓	✓	✓	✓
dayofweekn	The day of the week as a number	✓	✓	✓	✓
dayofweeks	The day of the week as a three-character string	✓	✓	✓	✓
heaptop	The number of characters of heap space that have been used	✓	W	W	W
ifsnap	True if the interviewer has reached this statement by snapping back, or false if not	✓	C	C	✓
intname	The interviewer's name in CATI and CAPI, or the interview ID in Quancept Web and mrInterview	✓	✓	✓	✓
iteration	The number of times a loop has been executed	✓	✓	✓	✓
login	The system login name of interviewer as in the Unix password file	✓	✓	C	✗
partofday	The part of day local to the interviewer	✓	✓	✓	✓
partofdayr	The part of day local to the respondent	✓	✓	✓	✓
qcproject	The script name	✓	✓	✓	✓
random	The seed for randomization	✓	✗	✗	✗
respondent	The current respondent's serial number	✓	C	C	✓

Keyword	Returns	CATI	CAPI	Web	mrInt.
rotation	The position in which the first response in the original list appears in the current rotation of the list	✓	✓	✓	✓
subscript	The number of times a question in a loop has been asked	✓	✓	✓	✓
tcomment	Obsolete keyword, do not use as a label	✓	✗	✗	✗
telephone	Obsolete keyword, do not use as a label	✓	✗	✗	✗
teltext	Obsolete keyword, do not use as a label	✓	✗	✗	✗
testingrun	True if qtip is running in automatic test mode, or false if it is not	✓	C	C	✗
time	The time at which this statement was executed	✓	✓	✓	✓
timeofday	The time of day local to the interviewer	✓	✓	✓	✓
timeofdayr	The time of day local to the respondent	✓	✓	✓	✓
userinfo	The system user number of the interviewer as in the /ip/users file	✓	C	C	✗

set variable=value

Keyword	Description	CATI	CAPI	Web	mrInt.
atozflgs	Specifies rules for alphabetic sorting of response texts.	✗	✗	✗	✓
atozlang	Specifies the language to use for sorting response texts in multilingual scripts.	✗	✗	✗	✓
atozmode	Specifies the type of alphabetic sorting to use.	✗	✗	✗	✓
atozsrt	Specifies the alphabetic sorting order for multilingual scripts.	✗	✗	✗	✓
context	Specifies the context from which question and displayable text is to be selected.	✗	✗	✗	✓
curpgstyle	Names the page template to use	✗	✗	✗	✓
dsplmax	Maximum number of <i>display</i> and <i>protect</i> statements per page.	✗	✗	✗	✓
dbsqlcrs	Sets the SQL cursor type	✗	✗	✗	✓

Keyword	Description	CATI	CAPI	Web	mrInt.
dbsqlold	Reserved word. Do not use as a label.	X	X	X	✓
dfmt	Defines format for displayed and protected texts	X	C	✓	✓
emptyok	Switches on/turns off automatic null coding when respondents leave blank cells in a numeric grid.	X	X	X	✓
hideprev	Hides/displays the Previous button	C	C	✓	✓
hidestop	Hides/displays the Stop button	C	C	✓	✓
htmlfmt	Defines the characteristics of open-end or numeric input boxes	X	C	✓	✓
labeltyp	Specifies the label type from which question and displayable text is to be selected.	X	X	X	✓
language	Sets the language for the current interview	X	X	X	✓
logsql	Save copies of all SQL statements that write data	X	X	X	✓
msgalrcmp	Defines text to display when a respondent attempts to restart a completed interview	C	C	C	✓
msgansp	Defines the text that prompts for a numeric or open-ended answer	C	C	✓	✓
msgbrbut	Defines text to display if the respondent uses the browser's navigation buttons	C	C	✓	✓
msgcntac	Defines text to display when too many responses chosen from an <i>mp</i> list in a column grid.	X	X	X	✓
msgcntam	Defines text to display when too many responses chosen from an <i>mp</i> list in a multitask.	X	X	X	✓
msgcntan	Defines text to display when too many responses chosen from an <i>mp</i> list in an ordinary question.	X	X	X	✓
msgcntar	Defines text to display when too many responses chosen from an <i>mp</i> list in a row grid.	X	X	X	✓
msgdk	Defines text to display for Don't know	C	C	✓	✓
msgend	Defines the message to display at the end of the interview	C	C	C	✓
msginvan	Defines text to display when invalid responses are given	C	C	✓	C
msglang	Defines the language for error messages	X	X	X	✓

Keyword	Description	CATI	CAPI	Web	mrInt.
msglangd	Defines the default language for error messages	X	X	X	✓
msgmdidx	Sets the index number for error messages	X	X	X	✓
msgmdord	Determines the order in which files are searched for error messages	X	X	X	✓
msgmisac	Defines text to display when answers are missing from a column grid	X	X	X	✓
msgmisam	Defines text to display when answers are missing from a multiask	X	X	X	✓
msgmisan	Defines text to display when answers are missing (mrInterview — ordinary questions only)	X	X	✓	✓
msgmisar	Defines text to display when answers are missing from a row grid	X	X	X	✓
msgnull	Defines text to display for No answer	C	C	✓	✓
msgnumac	Define text to display when non-numeric answer is given in a column grid.	X	X	X	✓
msgnumam	Define text to display when non-numeric answer is given in a multiask.	X	X	X	✓
msgnuman	Define text to display when non-numeric answer is given in an ordinary question.	X	X	X	✓
msgnumar	Define text to display when non-numeric answer is given in a row grid.	X	X	X	✓
msgoth	Defines text to display for specified other	X	X	X	✓
msgothal	Causes <i>msgoth</i> text to be used for o texts too.	X	X	X	✓
msgotiam	Defines text to display when an open-end is entered in a multiask but <i>other</i> is not selected	X	X	X	✓
msgotian	Defines text to display when an open-end is entered in an ordinary question but <i>other</i> is not selected.	X	X	X	✓
msgotmam	Defines text to display when <i>other</i> is chosen in a multiask but no open-end text is entered.	X	X	X	✓
msgotman	Defines text to display when <i>other</i> is chosen in an ordinary question but no open-end text is entered.	X	X	X	✓

Keyword	Description	CATI	CAPI	Web	mrInt.
msgprefix	Defines a prefix for error message variable names	X	X	X	✓
msgref	Defines text to display for Refused	C	C	✓	✓
msgrngac	Defines text to display when an out-of-range numeric response is given in a column grid.	X	X	X	✓
msgrgto	Defines the character to use for separating the minimum and maximum values in a range for the <i>msgrng</i> messages	C	C	C	✓
msgrgor	Defines the character to use for separating ranges for the <i>msgrng</i> messages	C	C	C	✓
msgrngam	Defines text to display when an out-of-range numeric response is given in a multiask.	X	X	X	✓
msgrngan	Defines text to display when an out-of-range numeric response is given in an ordinary question.	X	X	X	✓
msgrngar	Defines text to display when an out-of-range numeric response is given in a row grid.	X	X	X	✓
msgsgac	Defines text to display when more than one answer is chosen from an <i>sp</i> list in a column grid.	X	X	X	✓
msgsgam	Defines text to display when more than one answer is chosen from an <i>sp</i> list in a multiask.	X	X	X	✓
msgsgan	Defines text to display when more than one answer is chosen from an <i>sp</i> list in an ordinary question.	X	X	X	✓
msgsgar	Defines text to display when more than one answer is chosen from an <i>sp</i> list in a row grid.	X	X	X	✓
msgsolac	Defines text to display when an <i>sp</i> response is chosen with other responses in a column grid.	X	X	X	✓
msgsolam	Defines text to display when an <i>sp</i> response is chosen with other responses in a column grid.	X	X	X	✓
msgsolan	Defines text to display when an <i>sp</i> response is chosen with other responses in a multiask.	X	X	X	✓
msgsolar	Defines text to display when an <i>sp</i> response is chosen with other responses in an ordinary question.	X	X	X	✓

Keyword	Description	CATI	CAPI	Web	mrInt.
noaneqnl	Treats unanswered questions and valueless variables as if they had a null response or value.	X	X	X	✓
nolimoth	In statements of the form <i>resp sp not q1 in list2</i> , prevents <i>not</i> from excluding <i>other</i> from the current response list if it was chosen at q1	X	X	X	✓
ofmt	Defines format for <i>other</i> in response list	X	X	X	✓
ofpathem	Makes off-path questions appear blank.	X	X	X	✓
ofpathmd	Whether to save data for off-path questions.	X	X	X	✓
pagstyle	Names the page style for the current page.	X	X	X	✓
pfmt	Defines default formatting for <i>protected</i> texts.	X	X	✓	✓
pgtitle	Defines the page title for the current page.	X	X	X	✓
pictfmt	Used in scripts generated from InterviewBuilder questionnaires. Cannot be used as a label.	X	X	X	✓
pictfmta	Used in scripts generated from InterviewBuilder questionnaires. Cannot be used as a label.	X	X	X	✓
pictfmtb	Used in scripts generated from InterviewBuilder questionnaires. Cannot be used as a label.	X	X	X	✓
pictfmtl	Used in scripts generated from InterviewBuilder questionnaires. Cannot be used as a label.	X	X	X	✓
pictfmtr	Used in scripts generated from InterviewBuilder questionnaires. Cannot be used as a label.	X	X	X	✓
picthttp	Used in scripts generated from InterviewBuilder questionnaires. Cannot be used as a label.	X	X	X	✓
pictsra	Used in scripts generated from InterviewBuilder questionnaires. Cannot be used as a label.	X	X	X	✓
proprefix	Defines the prefix for property variables and the name of the property collection to which those variables belong.	X	X	X	✓
qfmt	Defines the question text format	X	C	✓	✓
rfmt	Defines response text format	X	C	✓	✓
rotnwsed	Specifies that the same rotation pattern should be used for all rotations in an interview.	X	X	X	✓

Keyword	Description	CATI	CAPI	Web	mrInt.
rplaycnt	Defines how many times a script should be replayed after an ODBC error.	X	X	X	✓
scrtmout	Specifies the time-out delay for infinite loops	X	X	X	✓
sfmt	Defines format for <i>null/dk/ref</i> texts in response lists	X	X	X	✓
silentnl	Switches on/turns off automatic null coding when respondents press Next without selecting a response.	X	X	X	✓
usepictf	Causes the compiler to check the metadata for information about images to display for questions and responses.	X	X	X	✓

2.3 Callable functions

Callable function	Description	CATI	CAPI	Web	mrInt.
abendivr	Cancel a connection to the IVR system	✓	X	X	X
aca	Calls an ACA job from within a script	✓	✓	X	X
acctinfo	Writes information to a customized accounting field in the qca file	✓	W	W	X
adddatepart	Increments/decrements a section in the date/time string	✓	X	X	X
alphdate	Converts a text date/time into operating system format	✓	X	X	X
append	Joins two texts	✓	✓	✓	✓
autostop	Saves response data periodically during an interview	✓	X	X	X
chgstop	Changes the name of an interview's stop file	✓	X	X	X
commitvr	Transfer the current call to the IVR system	✓	X	X	X
convdata	Applies scaling factors to numeric data	✓	X	X	X
datepart	Copies a section from the date/time string into a variable	✓	X	X	X

Callable function	Description	CATI	CAPI	Web	mrInt.
datetext	Converts the operating system date/time string to a text	✓	✓	✗	✗
datetime	Returns date/time in operating system format	✓	✓	✗	✗
disable_audio_record	Disables audio recording with the QTS	✓	✗	✗	✗
enable_audio_record	Enables audio recording with the QTS	✓	✗	✗	✗
getdbvar	Reads value from SMS database for current record	✓	✗	✗	✗
getenv	Returns the value of an environment variable	✓	W	W	W
getsmvar	Reads information from an SMS sample record into the script	✓	✓	✗	✓
keepstopped	Keeps data for all questions asked in stopped interviews	✓	W	W	✗
limitresp	Variable limits for numeric responses	✓	✓	✓	✓
local	Calls a user-defined function	✓	✗	✗	✗
longmath	Arithmetic with integers	✓	✗	✗	✗
longtext	Stores responses to several questions under one variable name	✓	✗	✗	✗
makeappt	Saves an appointment set within the script	✓	✗	✗	✗
prepareivr	Dial the IVR system	✓	✗	✗	✗
putdbvar	Copies data into the SMS database	✓	✗	✗	✗
putsmvar	Copies data into the sample record	✓	✓	✗	✓
qc_hangup	Hangs up a call without terminating the interview	✓	W	W	✗
querysms	Sends a query to the SMS server	✓	✗	✗	✗
quitdata	Whether to save data for quit/abandoned interviews	✓	W	W	W
quorat	Checks quota ratios	✓	✗	✗	✗
quotacell	Tests for unfilled quota cells	✓	✗	✗	✗
quotacnt	Obsolete keyword, do not use as a label	✓	✗	✗	✗

Callable function	Description	CATI	CAPI	Web	mrInt.
quotamap	Takes quota snapshot for use with <i>quotacell</i>	✓	✗	✗	✗
quotaval	Obsolete function, do not use as a label	✓	✗	✗	✗
quoval	Displays quota cell information	✓	✗	✗	✗
readfile	Retrieves a line of information from a file	✓	✓	✗	✗
realmath	Arithmetic with real numbers	✓	✗	✗	✗
redial	Reconnects a disconnected call	✓	✗	✗	✗
rfclose	Closes a report file	✓	✗	✗	✗
rfopen	Opens a report file	✓	✗	✗	✗
rfprint	Writes to a report file	✓	✗	✗	✗
setcols	Displays responses in a given number of columns	✓	✗	✗	✗
setdata	Whether to save data for incomplete interviews	✓	W	W	W
setdataser	Sets the respondent serial number	✓	✓	✓	✓
setdatepart	Sets the value of a section in the date/time string	✓	✗	✗	✗
setinteger	Assigns a real value to an integer question variable	✓	✓	✓	✓
setivr	Define initial configuration settings for the IVR system	✓	✗	✗	✗
setlang	Sets the language for an interview using a multilingual script	✓	✗	✗	✗
setlocale	Sets the language for an interview with regard to alphabetic sorting, punctuation, and word wrap	✓	✗	✗	✗
setreal	Assigns an integer value to a real question variable	✓	✓	✓	✓
showinf	Saves timings for script sections	✓	✗	✗	✗
smscript	Calls a function defined in the Sample Management script from the questionnaire.	✗	✗	✗	✓
stopdata	Whether to save data for stopped interviews	✓	✓	✓	C
strngchk	Compares a text string against a mask	✓	✓	✓	✓

Callable function	Description	CATI	CAPI	Web	mrInt.
subprog	Runs an external program without leaving the interviewing program	✓	✓	C	✗
substr	Extracts characters from a text	✓	✓	✗	✓
testmode	Switches into/out of automatic test mode	✓	C	C	C
timesect	Times sections in script	✓	W	W	✗
valstring	Converts a number in a text variable to a numeric value	✓	✓	✓	✓
varlist	Writes variables to a temporary file	✓	✗	✗	✗

2.4 Variables set by mrInterview

Some features in mrInterview place information in variables with predefined names. These variables are as follows:

Variable	Description
quotrtxt	Set to the text for the HRESULT of the quota statement.
quotrval	Set to the HRESULT of the quota statement.
scrtmpas	Set to the time since the previous page was displayed.
smscrres	If the <i>smscript</i> callfunc was successful, <i>smscrres</i> is set to the function result of the called function; otherwise it is not set.
smsctxt	Set to the text for the HRESULT of the <i>smscript</i> callfunc.
smscrval	Set to the HRESULT of the <i>smscript</i> callfunc.
ststest	Set to 1 if this is a test interview.

2.5 Special responses in qtip

You can enter the commands given below at any prompt while you run qtip to test your scripts. The commands are grouped according to the type of task for which they are used:

- Standard responses
- Ending an interview early
- Quancept Telephony System commands

- Entering verbatims and comments
- Working with long response lists and Quancept databases
- Setting appointments (Unix only)
- Snapping during an interview

Standard responses

Type	To enter this response
!code	Cancel an incorrect response code in a list of codes chosen from a multipunched response list
dk	Don't know (respondent doesn't know the answer)
null	Null/no answer
ref	Refused
*dk	Forced don't know (when dk is not available)
*null	Forced null/no answer (when null is not available)

Entering verbatims and comments

Type	To
;	Obtain a serial number for a handwritten open end
@	Obtain a serial number for a handwritten open end
()	Enclose an open-ended response given as part of single-punched or multipunched response
= text	Overwrite an existing open end with new text at 'append/check verbatims' prompt
*e	Use an editor to append verbatims
*msg	Activate a prompt to send a message to the supervisor
*note	Enter a forced open end
ser	Obtain a serial number for a handwritten open end

Working with long response lists and Quancept databases

Type	To
+	Scroll forwards through a Quancept database or long response list
-	Scroll backwards through a Quancept database or long response list
\$	Enter 'null' on a Quancept database question.
<i>filter!</i>	Apply a filter to a Quancept database question
!	Cancel a filter that has been applied to a Quancept database.
<i>line_number</i>	Select the entry on the given line from a Quancept database.
= <i>key</i>	Select the entry with the given record key from a Quancept database. (The key is the value before the first space or semicolon.)

Snapping during an interview

Type	To
<	Snap back to the previous question
< <i>qlabel</i>	Snap back to a specific question
<<	Snap back to the first question asked
>	Snap forward to the next question
> <i>qlabel</i>	Snap forward to a specific question
>>	Snap forward as far as possible
?	Check which questions may be snapped to
??	Display 'snappable' questions and their current responses

Ending an interview early

Type	To
stop	Stop the interview, optionally saving data and arranging for a callback
quit	Terminate the interview early and save the data
abandon	Terminate the interview without saving the data

Type	To
complete	Terminate the interview early, save the data and flag the interview as successfully completed

Miscellaneous commands

Type	To
/	List the names of all questions on the interview path
:qname	Check the response to specified question
"	Keep the original answer to the current question
across	Display single-punched or multipunched response lists in rows across the screen
*comment	Display comments for the current interview
*date	Display the current date
*debug	Enter the script debugger (requires programming knowledge)
down	Display single-punched or multipunched response lists in columns down the screen.
*ln	Select the language for this interview in a multilingual script
*pause	Forces qtip to ask 'Another interview?'. Overrides QCANOTH.
*netwk	Displays network information (may be useful for debugging)
regn	Redraw the screen (clear screen of line noise characters)
*rot	Returns the value of the rotation sequence (may be useful for debugging)
*seed	Returns the seed of the random number generator used for randomizing responses (may be useful for debugging)
*sms	Forces a return to the SMS menu (same as <i>gotosms</i> in the script)
*time	Display the current time
*vers	Display version information

Setting appointments

When used in a script, the callfunc *makeappt* activates a prompt for interviewers to enter appointments. The following formats are available for entering the required appointment time.

Type	To set appointments for
now [+n] [+nh] [+nd] [+nw] [+nm]	Times relative to now: now (that is, in a few minutes) [plus <i>n</i> minutes] [plus <i>n</i> days] [plus <i>n</i> hours] [plus <i>n</i> weeks] [plus <i>n</i> months]
tomorrow or to or [Mon] [Tue] [Wed] [Thu] [Fri] [Sat] [Sun]	Days of the week
<i>mm/dd</i> or <i>dd/mm</i> <i>or</i> [<i>yy</i>] <i>yy/mm/dd</i>	Specific dates
<i>nn:nn</i>	Times, 24-hour clock
<i>nnam</i>	Times, 1am–noon
<i>nnpm</i>	Times, 1pm–midnight
<i>n:nnam</i>	Times, 12:00am – 11:59 am
<i>n:nnpm</i>	Times, 12:00pm – 11:59 pm

Quantime Telephony System commands

Type	To
*hangup	Hang up the telephone without terminating the interview script
*redial	Redial the last respondent immediately (to reconnect in case of accidental hang up)

2.6 Working with external Unix databases

The commands in this section are not entered in scripts, but at the command line or system prompt.

Shared memory commands for resp dbase questions

-
- ↖ Shared memory commands for dbase responses are used only in a multiuser Unix environment.
-

Type	To
shm <i>database</i>	Load a database into shared memory
listshm	List the contents of shared memory
rmshm <i>database</i>	Remove the named database from shared memory
rmshm id <i>n</i>	Remove the database with ID <i>n</i> from shared memory

GNU database commands

Type	To
setupdbm <i>textfile</i>	Read a text file and write out a database file which can be interpreted by qtip
listdbm <i>gnudatabase</i>	Display a GNU database file
listdbm <i>gnudatabase</i> > <i>textfile</i>	Write data from a GNU database to a text file

3 Questions and answers

Questions, and the answers to those questions, are the basis of any questionnaire. This chapter discusses some of the keywords associated with writing questions and answers in the Quancept language. These are:

ask	the question text
dummyask	a dummy question
limitresp	variable limits for numeric responses
multitask	display several questions on the same screen
resp	the response list

3.1 Naming questions

A question and its response list are treated as a single entry in the script even though they are two statements. A question must always have a response list, and a response list cannot exist without a question.

Each question and its response list have a unique label of up to eight characters. The label must start with a letter, but the rest may consist of a mixture of unaccented letters and numbers. You cannot use a Quancept keyword as a label. Here are some examples of question labels:

valid labels:	AGE	q1	ending	Q10	xyz
invalid labels:	1234	end	temporary	q_1	içî

The table below explains why these labels are invalid:

The label	Is invalid because it
1234	Starts with a number, not a letter
end	Is a keyword
temporary	Is greater than eight characters
q_1	Contains an underscore
içî	Contains an accented character

3.2 Question and response format

Quick Reference

To define a question and its response list, type:

```
label  ask 'question_text'  
      resp resp_type [resp_list]
```

In the question part of the statement, *label* is the unique name for this question. The interviewing program uses it as the label for the responses as well. *ask* is the Quancept keyword which introduces a question, and *question_text* is the text of the question. This may be continued over as many lines as necessary.

The quotes around the question text mark the start and end of the text. When the question is displayed during the interview the quotes are ignored.

- ⌚ In Quancept Web, question texts may be up to 2091 characters long. Texts longer than this cause the compilation to fail with a GPF and an invalid page fault.

In the response part of the statement, *resp* is the keyword which introduces a response list, *resp_type* is the response type (for example, single-punched), and *resp_list* is the list of permitted responses. The example below defines a question with a single-punched response list from which the respondent may choose one day of the week:

```
day  ask 'What day did you go?'  
      resp sp 'monday' / 'tuesday' / 'wednesday' / 'thursday' /  
             'friday' / 'saturday' / 'sunday'
```

During interviews, each response is usually displayed on a separate line. Long response texts are split across two or more lines and break at the exact point at which the screen line ends. There is no intelligent splitting of text between words. If you want to split text between words, run a test interview to see where line breaks are required and then insert as many spaces after the last full word on a line as are required to force the next word onto the start of a new line.

If there are more responses than will fit into one column, the interviewing program starts a second column. When this happens, long response texts are truncated so that they fit into the available column space.

3.3 Response types

All responses in a list must be of the same type, and that type must match the type specified by the response type. For example, if your response type is *sp*, then the responses in the list must all be texts enclosed in single quotes.

The way in which responses are listed depends on the response type. This may be:

sp	single-punched responses (one response only)
mp	multipunched responses (one or more responses)
num	numeric responses that are whole numbers
real	numeric responses that are decimal numbers
coded	open-ended (verbatim) responses
dbase	responses chosen from a Quancept database
list	responses are the names of defined response lists

☞ See chapter 19, Very long response lists in CATI (response list databases), for more information about the *dbase* response type, and chapter 7, Defined response lists, for more information about *list*.

Single-punched and multipunched responses

Quick Reference

To define a single-punched response list, type:

resp sp '*resp1*' / '*resp2*' / / '*respN*'

To define a multipunched response list, type:

resp mp '*resp1*' / '*resp2*' / / '*respN*'

Single-punched responses are identified by the letters **sp**. This is used when the respondent must choose only one response from a list of permitted responses. Each response is separated by slashes and enclosed in single quotes.

```
meal      ask 'The last time you went out for a meal, what type of
           restaurant did you visit?'
           resp sp 'indian' / 'chinese' / 'greek' / 'italian' /
           'english' / 'west indian' / 'french' / 'thai' /
           'japanese' / 'spanish' / 'turkish' / 'other'
```

Multipunched responses are similar to single-punched answers except that they indicate that a combination of responses from the list is acceptable. Multipunched answers are represented by the letters **mp**. As is the case with single-punched responses, all choices following *mp* must be enclosed in single quotes and separated by slashes. Here is an example:

```
like    ask 'What types of food do you like?'
resp mp 'indian' / 'chinese' / 'greek' / 'italian' /
      'english' / 'west indian' / 'french' / 'thai' /
      'japanese' / 'spanish' / 'turkish' / 'other'
```

In this example, the respondent may choose as many of the given responses as he/she likes. It is never wrong to have a single answer where multipunched responses are allowed. If the respondent only likes English food, this is just as correct as if he or she likes English, Indian, Italian, and Chinese food.

Data file format for Quancept CATI, Quancept CAPI and Quancept Web



When the parser allocates space in the data file for single-punched and multipunched responses, it places the first 9 responses as codes 1 to 9 in the first column and the next 10 responses as codes 0 to 9 in the second column, and so on. Code 0 in the first column is unused unless your script contains the keyword *mapothzero*. This causes a specified other to be coded as zero in the first column. Codes – and & in the first column are reserved for the special keywords *null* and *dk*. These codes are unused in the second and subsequent columns of a field.

-
- ☞ See chapter 5, ‘Keywords in response lists’ for further details on *null* and *dk*, and for information about *mapothzero*.
-

Let's look at the examples again to see how the data will be stored for each one. The first question:

```
meal    ask 'The last time you went out for a meal, what type of
           restaurant did you visit?'
resp sp 'indian' / 'chinese' / 'greek' / 'italian' /
       'english' / 'west indian' / 'french' / 'thai' /
       'japanese' / 'spanish' / 'turkish' / 'other'
```

has 12 responses that will be coded 1 to 9 in the first column and 0 to 2 in the second column. If the parser allocates columns 15 and 16 to this question and the respondent visited an Italian restaurant, the data will be stored as follows (the ruler shows the column numbers, where 1 is 10, 2 is 20, and so on):

```
-----1-----2
0012301      4
```

00123 is the respondent number and 01 is the card number within this record. If the respondent visited a Turkish restaurant instead, the record will show:

```
-----1-----+1---2
0012301      2
```

Now let's look at a question with the multipunched response list:

```
like    ask 'What types of food do you like?'
resp mp 'indian' / 'chinese' / 'greek' / 'italian' /
       'english' / 'west indian' / 'french' / 'thai' /
       'japanese' / 'spanish' / 'turkish' / 'other'
```

This is the same response list as the previous example, but it allows more than one answer to be selected. If the columns for this question are 21 and 22 and the respondent likes Indian, Greek, English, Spanish and Turkish food, the data will be stored as:

```
-----1-----2-----
0012301      10
              31
              5
```

If the respondent likes Mexican food only, this will be coded as 'other' and the data will be:

```
-----1-----2-----
0012301      2
```

The first seven columns of each card are reserved for the respondent serial number and card number. This leaves 73 columns for data, which means that each question can have a maximum of 729 responses, excluding specified other. The first nine responses and specified other are coded in the first column, followed by 72 columns of ten codes each.

Case data format for mrInterview

mrInterview stores interview (case) data in a relational database. Each single-punched and multipunched question in a script generates a categoric variable in the database. The variables' names match the question names in the script. Every single-punched or multipunched response in a script has a unique code that is assigned to it when the questionnaire definition (.mdd) file is created. The categoric variables in the database store the codes of the responses chosen at each question. For example, if the first question in the script is:

```
like    ask 'What types of food do you like?'
resp mp 'indian' ('ind') / 'chinese' ('chin') / 'greek' ('greek') /
       'italian' ('ital') / 'english' ('eng') / 'french' ('french') /
       'west indian' ('westind') / 'thai' ('thai') / 'japanese' ('jap') /
       'spanish' ('span') / 'turkish' ('turk') / 'other' ('other')
```

the case data will contain a categoric variable called 'like' with up to 12 unique codes representing the responses given.

Although mrInterview uses a relational database, it uses the basic Quancept formula for calculating the maximum number of categories that a question may have. However, the formula has been extended to allow up to 993 category columns which generally allows for 9929 categories per question (depending on the complexity of the questionnaire, the exact number of categories may be different).

Questions with more than 700 categories may cause mrInterview and the Data Model tools to run more slowly than you would like. The compiler therefore issues a warning to this effect whenever it encounters a question with more than 700 categories.

-
- ☞ For further information about the case data file, see section 38.4, The mrInterview case data file.
-

Numeric responses — whole numbers

Quick Reference

To define a response list that allows a range of integer (whole number) responses, type:

resp num lowest [to highest]

Use *n-* and *n+* for open-ended ranges.

Enter lists of single values or ranges separated by slashes as for single-punched responses.

Numeric responses that are whole numbers (integers) are identified by the response type **num**. This is followed by a list of numbers which may be given as a response. All numbers must be whole numbers in the range -999,999,999 to +999,999,999 inclusive.

Negative numbers must always be preceded by minus signs, but positive numbers must never be preceded by plus signs. Here are some examples of valid numbers:

14536 -90 56 -27757 13297767

More often than not, you will want to specify the permissible response as a range of numbers. For example:

```
people    ask 'How many people were there in your party?'
            resp num 1 to 20
```

This tells the interviewing program to accept any number between 1 and 20 inclusive as the response to this question.

If one number is negative, this counts as the lower end of the range and must come first. If both values are negative, we would write, for instance:

```
resp num -100 to -52
```

To allow for ranges where the highest value is unspecified (for example, it could be the highest value Quancept allows), use the notation $n+$, where n is the lowest value you will accept. Similarly, for ranges where the lowest value is unspecified, use the notation $n-$. Do not use the mathematical signs < and > as Quancept will not recognize them. For example:

```
people    ask 'How many people were there in your party?'
          resp num 1+
```

Remember that the number given on the *resp* statement is always assumed to be a valid response, so you would enter the response 'less than 23' as:

```
resp num 22-
```

In response lists containing more than one range, each range is separated by a slash:

```
resp num 10- / 11 to 20 / 30 to 40 / 50+
```

Any number that does not fall within one of the specified ranges is invalid. In this example, the numbers 21 to 29 and 41 to 49 are invalid as responses.

Sometimes, the acceptable responses will be unrelated values rather than numbers in a range. In this case, enter each number separately separated by slashes:

```
tax      ask 'When you tax your car, how many months do you pay for
           at a time?'
           resp num 6 / 12
```

☞ If you have a series of questions with numeric responses and the sum of those responses must fall within a fixed range (for example, percentages that must not exceed 100%) place a *limitresp* statement just before each question defining the maximum and minimum values for this interview.

☞ See section 3.5, 'Variable values in numeric ranges', for more information about *limitresp*.

Numeric responses in the CATI, CAPI and Web data file



For all numerics, the parser allocates as many columns in the data file as are necessary to store the value with the largest number of digits. With negative numbers, the minus sign is counted as a digit since it requires a column of its own in the data. Thus, the maximum positive number that Quancept accepts (999,999,999) will be allocated nine columns, whereas the corresponding negative value will receive ten columns. Numbers with fewer than the maximum number of digits are right-justified in the field and padded on the left with zeroes; the minus sign is always placed in the left-most column of the field. For example:

```
resp num -10 to 30
```

is assigned a three-column field that might contain:

response	data
-10	-10
-5	-05
30	030

Open-ended ranges are always allocated nine columns if they allow only positive numbers, or ten if negative values are valid.

Numeric responses in the mrInterview case data

- mrInterview creates a long variable for integer responses and a double variable for real responses. Variables have the same names as the questions in the script.

-
- ☞ For further information about the case data file, see section 38.4, The mrInterview case data file.
-

Numeric responses — real numbers

Quick Reference

To define a response list that allows real (decimal) responses, type:

resp real lowest [to highest]

Enter lists of single values or ranges separated by slashes as for single-punched responses.

Numeric responses that are decimal numbers (reals) have the response type **real**. This recognizes the decimal point as a valid character in an otherwise numeric string, and may be used to collect currency or other similar types of response where real numbers are common. For example, if the question is:

```
total    ask 'What was the total price of the meal, excluding  
the service charge or tip?'  
        resp real 0 to 500.00
```

the interviewing program will accept any value between 0 and 500.00. When the maximum or minimum values in a real range are whole numbers, you may omit the zero decimal places if you wish. In this example the interviewing program will accept 0 to 500 as the response range.

The parser allocates columns in the data file to real responses in the same way it does for integer responses, with the additional rule that real responses are written with three decimal places. In the example above, the largest response that the response list allows is 500.000 so the parser will allocate seven columns to this question. If the respondent paid 98.56 for the meal, the interviewing

program will write it to the data file as 098.560. Values with more than three decimal places will be rounded to the nearest 10 immediately after the third place, so 123.4567 will be stored as 123.457.

Real responses may be positive or negative. Open-ended ranges of the form 1+ and 1– are not allowed.

Real responses in CATI, CAPI and Web interviews and data files



When dealing with real numbers, the interviewing program maintains accuracy to seven digits only, counting from the leftmost digit. By this we mean that when the interviewing program encounters a real number containing seven or fewer digits it will be able to display it and write it to the data file exactly as it is defined in the script or entered during the interview. If it encounters a real value with more than seven digits, the interviewing program will keep the leftmost seven digits exactly as they were but may change any other digits.

For example, if your script contains the statement:

```
resp real -123456789 to 987654321
```

the interviewing program will display the range as:

```
-123456792 to 987654336
```

The first seven digits in each number are the same as the definition, but the last two in each number are different.

Similarly, if you enter your response as:

```
654321912.123
```

the interviewing program will display it and write it to the data file as:

```
654321920.000
```

-
- ☒ This loss of accuracy is caused by the way computers handle real numbers: it is not a function of Quancept. You may find that the exact value of a real number with more than seven digits will vary between different makes of computers and different operating systems.
-

Real responses in mrlInterview interviews and case data



The interviewing program stores data for questions with real number responses in a variable of type double; that is, in binary 64-bit floating point notation. A double has approximately 15 significant digits of precision, so the effective range for real numbers is roughly +/-1.7e308. This is rounded to 12 digits to ensure that decimal values that are not representable in binary are still handled correctly. Users can enter real values using scientific notation so the number of decimal places is not fixed.

In the Data Model, real values are passed as double values. In the Quantum DSC, values are formatted in simple floating-point notation (xxx.yyy) and are rounded if necessary to fit into the number of columns allocated in the card/column specifications. The number of columns includes the decimal point and a leading minus sign if the value is negative. If the formatted number cannot be made to fit by removing all digits after the decimal point, an error occurs, and the record containing that value is not written.

Open-ended responses

Quick Reference

To write a response list that allows open-ended (verbatim) responses, type:

mrInterview: **resp coded**

CATI/CAPI/Web: **resp coded [(n)]**

For CATI, CAPI and Quancept Web, the default is to allocate 29 codes (3 columns) for coding of the open-ended texts. To request more or fewer codes, enter the number of codes you require in the parentheses.

Open-ended or verbatim responses are responses where the respondent is unrestricted in his/her reply. For example, you may ask for a respondent's name and place of birth. There can be no preset response list for such questions so you use the response type **coded** to signal to the interviewing program that the interviewer is to enter whatever the respondent says. You can also use this type of response when you want to gather respondents' opinions or reactions to a product in detail, rather than asking them to choose from a predefined response list.

The easiest way to define an open end is simply to type the question text followed by the words *resp coded*. For example:

```
cook    ask 'When you invite friends to a meal at your house, what
           sort of meals do you prepare?'
           resp coded
```

How Quancept CATI, Quancept CAPI and Quancept Web store open ends

With CATI, CAPI and Quancept Web, *resp coded* allocates three columns in the data file for later coding of the responses given to this question. The parser allocates nine codes to the first column in a field and 10 codes to all subsequent columns, so you will have 29 codes available for this coding.

If you know how many codes you want to use for coding, you can type the number enclosed in parentheses at the end of the response list:

```
cook    ask 'When you invite friends to a meal at your house, what
sort of meals do you prepare?'
      resp coded(17)
```

This example asks for 17 codes which is equivalent to two columns in the data.

Suppose you do not know exactly how many codes are required, but you do know how many columns you want to allocate to the question. Instead of entering the actual number of codes, enter the maximum number of codes that will fill the number of columns you wish to reserve. Suppose, in our example above, that we want to reserve two columns for question 7 but we are not sure how many codes will be required. We define the response list as:

```
resp coded (19)
```

because two columns can contain a maximum of 19 codes.

For some open ends you will want to have access to the responses given but will not want to allocate columns in the data file for coding them (for example, you may want to store the respondent's name and address in the open-end text file even though you will not want to code it into the data file). To do this, enter the number of codes as zero:

```
resp coded(0)
```

The response may then be used during the interview and will be stored in the open-end text file, *project.tex*, even though the data map in *project.map* will show that no columns have been reserved for recoding that response. If you do not want to save the text at all, use the option *nodata* at the end of the response list instead:

```
resp coded nodata
```

This makes the open-ended response available for the duration of the interview, but does not write the text to the tex file and does not allocate any columns in the data file for that question either.

-
- ☞ To find out more about suppressing columns with *nodata*, see section 5.6, Asking questions without storing responses.
-

If you use any of the special response *null*, *dk* and *ref* with *resp coded*, you must request at least one code in the data file to allow these responses to be written out. If you use *resp coded(0)* with any of these responses, the parser will issue the error message 'Special responses with coded(0)'.

-
- ☞ For further details these special responses, see section 4.1, No answer, section 4.2, Don't know, and section 4.3, Refusals.
-

Open-ended responses are given serial numbers which are stored with the text in the tex file. These responses can be coded using one of the coding programs.

-
- ☞ For further information about the coding programs see the *Quancept Coding Open Ends Manual*.
-

How mrInterview stores open ends

mrInterview creates a text variable in the case data with the same name as the question to store the respondent's answer, and a variable called *qname*.Codes to store the codes assigned to this response when it is coded. If you do not want to save the response text, use the option *nodata* at the end of the response list instead:

```
resp coded nodata
```

This makes the open-ended response available for the duration of the interview, but create a *qname* variable in which to store the response. The *qname*.Codes variable is still created but remains empty.

-
- ☞ To find out more about suppressing columns with *nodata*, see section 5.6, Asking questions without storing responses.
 - For further information about the case data file, see section 38.4, The mrInterview case data file.
-

Open ends with Quancept Web and mrInterview



- When Quancept Web and mrInterview encounter a question with an open-ended response list, they display a box that is 40 columns wide by six rows deep, in which the respondent enters an answer.
- Text is wrapped onto the next line as soon as it reaches the end of the current line, so there is no need for the respondent to press Return at the end of each line. Also, the display scrolls if the respondent fills the box so that there is always a free line left on which to type.

You may find that the size of the box limits the amount of text that respondents will enter, and you'll want to increase the size of the box to encourage them to write in more detail. You can do this using the *htmlfmt=textarea* assignment statement.

In Quancept Web, the amount of text that can be entered per open end is browser specific. Typically, blocks of up to 32000 characters should be accepted at any one time. In mrInterview the limit is 2000 characters per open end.

-
- ☞ For more information about setting the size of the answer box for Quancept Web and mrInterview, see section 22.3, Input boxes for open ended and numeric responses.
-

3.4 Dummy questions

Quick Reference

To define a dummy question that is not asked but may have answers assigned to it, type:

```
label  dummyask 'question text'
      resp resp_type resp_list
```

Dummy questions are questions whose answers are assigned by statements in the script rather than by the interviewer entering a response given by a respondent. Therefore, dummy questions never appear on the interviewer's screen during an interview.

Dummy questions are useful tools for manipulating answers to other questions in the script. You may, for instance, merge answers to two separate questions into a dummy question for use later in the script. A typical example of this would be a brand awareness study where the script asks respondents to list the brands they know, and then prompts for names which are not spontaneously mentioned. If you need to refer to all brands known together, you may combine the spontaneous and prompted responses in a dummy question, and then refer to that question whenever you need all brands known.

You should never write statements that change the answers given to any question by a respondent. Instead, create a dummy question with the same response list as the question whose answer you wish to change, assign the response to that question, and then change the value of the dummy question as necessary.

The syntax for a dummy question is the same as for an ordinary question except that you use the keyword **dummyask** instead of *ask*. As the question will not be displayed during the interview, you may abbreviate the question text to a short note describing the function of the question.

You will find many examples of dummy questions throughout this manual. The next example shows how you can obtain a total value by adding together the answers to two questions with numeric response lists:

```
paid    dummyask 'Total cost including service/tip'
        resp real 0 to 700
total   ask 'What was the total price of the meal, excluding the service
charge or tip?'
        resp real 0 to 500.00
serv    ask 'How much did you pay in service charges or tips?'
        resp real 0 to 200
comment Add the total cost and the service charge and save in
comment dummy question 'paid'
        set paid = total + serv
```

When the interviewing program runs this script segment, it reads the question ‘paid’ but does not display it because it is a dummy question. The ‘total’ and ‘serv’ questions are asked and the responses are written to the data file in the normal way. The *set* statement tells the interviewing program to add the answer to the serv question to the answer to the total question and to save the result as the answer to the paid question. The sum of the two values will appear in the data file as if the interviewing program had displayed the paid question on the screen.

Here are three snapshots of the data as the interviewing program executes this part of the script. The parser allocates columns to questions in the order they appear in the script, not the order in which they are asked or are assigned values, so let’s assume we have ‘paid’ in columns 11 to 17, ‘total’ in columns 18 to 24, and ‘serv’ in columns 25 to 31. If the respondent paid 98.56 for the meal, the data will be:

```
----+---1---+---2---+---3---  
0012301      098.560
```

If the service charge was 9.86, the data becomes:

```
----+---1---+---2---+---3---  
0012301      098.56009.860
```

When the two answers are added together, they are written to the data file in the columns reserved for the dummy question:

```
----+---1---+---2---+---3---  
0012301      108.420098.560009.860
```

3.5 Variable values in numeric ranges

Quick Reference

To define a respondent-specific range for a numeric response, type:

callfunc('limitresp', *minimum*, *maximum*)

where *minimum* and *maximum* are the lowest and highest values that will be acceptable.

There may be times when you will want to alter the range of values accepted by a *num* or *real* response list to vary between interviews. A common example is when you ask a general question to get a total and then follow it with more specific questions designed to find out how the total was made up. Ideally, what you would like to do is define each response list such that the maximum value in each range could never cause the sum of the values to exceed the total. For instance, if you have three questions:

```

total    ask 'What was the total price of the meal?'
        resp real 1.00 to 100.00
food     ask 'How much of that was for food?'
        resp real 1.00 to 100.00
drink    ask 'And how much of that was for drink?'
        resp real 1.00 to 100.00

```

and the cost of the meal is 50.00, it is still possible to accept answers greater than that for both the food and drink questions.

An easy way to deal with this is to use a **limitresp** statement before the food and drink questions which will define the maximum value possible based on answers given to previous questions. This allows you to issue the range for food as 1.00 to whatever was the cost of the meal, and then to adjust the range for drink to take into account both the cost of the meal and the amount spent on food.

When defining the range limits, you may type numeric values or the names of integer or real variables for one or both these values. The values on the *resp* statement define the absolute minimum and maximum values which are acceptable as responses. If *limitresp* defines values outside this range (for example, 1 to 100 when the *resp* statement shows 1 to 50), the interviewing program will use the *resp* range instead.

- ☞ The interviewing program does not insist that the data types of the variables used with *limitresp* match the data type of the question to which they refer. This means that if the response list is defined as *resp real*, the interviewing program will still accept whole numbers as part of *limitresp*. Similarly, if the response list is defined as *resp num*, you can type real numbers as part of the *limitresp* statement. In this case, however, the interviewing program will truncate the real values at the decimal point.
- ☞ To find out more about the differences between integers and reals, and what happens when you mix the two together, see section 15.2, Variable types with arithmetic assignments.

When you write a script that uses *limitresp*, define the response lists as you normally would but then insert a *limitresp* statement before each question whose response range you wish to change. Our three questions would be defined as follows:

```

total    ask 'What was the total price of the meal?'
        resp real 1.00 to 100.00
comment Save total spent in temporary variable so we can change it
comment without altering the value which will be written to the
comment data file
        set maxcost = total
comment Now set the meal total to be the maximum amount possible
comment for food
        callfunc('limitresp',0,maxcost)
food     ask 'How much of that was for food?'
        resp real 1.00 to 100.00

```

```
comment Subtract cost of food from total cost so we have maximum
comment possible for drink
    set maxcost = maxcost - food

comment Set new maximum cost as maximum for drinks bill
    callfunc('limitresp',0,maxcost)
drink ask 'And how much was for drink?'
    resp real 1.00 to 100.00
```

The comments explain what each statement is doing. Notice how the response lists set a range of between 1 and 100, whereas the *limitresp* statements define it between 0 and maxcost.

If the respondent pays £50.00 for the meal, the response list for food will show a response range of 0 to 50.00. If the food cost £31.65, the range for drink will be 0 to 19.35.

limitresp is also useful within loops where you want to ask the same question but where the maximum and/or minimum value changes each time the question is asked. There is an example of a loop below. It uses several statements you have not met yet, but it follows the same principle as the simple example you have just seen. If you would rather skip this example until you know more about Quancept, then you may do so.

```
numhr ask 'How long, to the nearest quarter hour, did you spend
watching TV last week?'
    resp real 0 go ctn / 0.25 to 168

comment Save the total time given so we can alter it
    set timchk = numhr

comment These are the things we are going to ask the question about
comment ptype will point to each one in turn.
11     for ptype = 'films' / 'news programs' / 'documentaries' /
          'situation comedies' / 'other programs'
comment Define limits for the next numeric question
    callfunc('limitresp',0,timchk)

comment The interviewing program will substitute a program type
comment in place of ptype
ptime ask 'How much of that time was spent watching '
    +ptype+'?'
    resp real 0 to 168
comment Subtract hours for current program from the total to
comment calculate the new maximum for the next iteration.
comment The (*) is used because ptime is a question in a loop
    set timchk = timchk - ptime(*)
    next
ctn continue
```

Here, it is the question ptime whose response list will vary with each iteration of the loop. The maximum and minimum values for each iteration are defined by the *limitresp* statement. All questions must have a response list, so we define the one for ptime by giving the minimum and maximum values which will ever be acceptable. Before entering the loop, we set the initial maximum value for this question to whatever answer was given to the numhr question. Since we will want to change this, we assign this value to a temporary variable rather than using the question variable itself.

Each time ptime is asked, the interviewing program subtracts its response from the current maximum allowed to arrive at a new maximum for the next iteration of the loop.

Suppose the respondent watched 29 hours of TV last week. The response list for time spent watching films (the first item in the value list) will be shown as 0.0 to 29.0. If the respondent watched 5.25 hours of films, the response list for news programs will be shown as 0.0 to 23.75. And so it continues.

In this example *limitresp* does not check whether the sum of the individual values matches the total number of hours watched; it only prevents it from exceeding the limit. If you want to force the sum of individual values to match the total, you must control the minimum acceptable value as well, so that on the last iteration the response list contains just the value that brings the sum up to the total.

-
- ☞ For information on loops see chapter 10, Repetitive questions, and chapter 11, Iteration and subscription.
 - For information on substituting words in question texts, see section 8.7, Displaying data in texts.
-

3.6 Displaying more than one question on the screen

Quick Reference

To display more than one question at a time on the interviewing screen, define the questions and their response lists as dummy questions using *dummyask* and then type:

label multiask '[text]' dumq1 / dumq2 /

where *dumq1* and *dumq2* are the labels of dummy questions that have been defined earlier in the script.

The interviewing program normally displays each question on a new screen. If you have a series of related questions, you may want to display some or all of them on the same screen and wait until they have all been answered before allowing the respondent to continue. This is good for questions such as demographics that all have reasonably short question texts and response lists.

To display several questions on the same screen, first define those questions as dummy questions with *sp*, *mp*, *num*, *real* or *list* response types. Respondents must answer all questions in a multiask group, so you may not use the *go* keyword to specify routing for single-punched responses that are part of a multiask group. Then use a **multiask** statement to name the dummy questions and to define an optional introductory text. For example:

```
eatout    dummyask 'How often do you eat out?'
          resp sp 'At least once a week' / 'Two or three times a month'
                  'Once a month' / 'Less than once a month'
enjoy     dummyask 'Could you tell me what you enjoy about eating out?'
          resp coded
both      multiask 'Now I''d like to ask you some questions about
          eating out' eatout/enjoy
```

The interviewing program displays multiask questions one below the other. If the response lists are long, this may mean that the respondent has to scroll down the screen to see the end of the last response list.

- ▀ With Quancept CAPI you can save space by using *hsetcols* or *vsetcols* to display a question's responses across the screen rather than one below the other in a single column.
- ▀ You can do something similar in Quancept Web and mrInterview, although you may prefer to use HTML tags to display the questions side by side as a two-column table.

-
- ☞ For further information on *hsetcols* and *vsetcols* see section 8.11, Specifying the number of columns for response lists.
For further information on displaying questions as a two-column table, see 'Tabular layouts' in chapter 22, Screen management with graphical user interfaces, and chapter 23.4, Multiask questions.
-

You can use *multiask* in a loop, but remember the following points:

- The dummy questions must appear in the loop.
- The references to the dummy questions on the *multiask* statement must be subscripted.

The following example uses *multiask* in a loop:

```
chans   for channel = 'BBC1' / 'BBC2' / 'ITV' / 'Channel 4'
watch    dummyask 'Do you watch '+channel+'?'
          resp sp 'Yes' / 'No'
hours    dummyask 'Why do you watch/not watch '+channel+'?'
          resp coded
both     multiask 'Thinking now of '+channel+' ....'
          watch(*) / hours(*)
next
```

3.7 Sample script

```
comment Sample script for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 3, Questions and answers

comment Multipunched response list
like      ask 'What types of food do you like?'
          resp mp 'indian' / 'chinese' / 'greek' / 'italian' / 'english' /
                 'west indian' / 'other'

comment CAPI/Web/mrInterview only: Two questions displayed on the same screen
comment For CATI: Replace dummyask with ask and delete the multiask
eatout    dummyask 'How often do you eat out?'
          resp sp 'At least once a week' / 'Two or three times a month'/
                 'Once a month' / 'Less than once a month'
enjoy     dummyask 'Could you tell me what you enjoy about eating out?'
          resp coded
both      multiask 'I''d like to ask you some questions about eating out'
eatout/enjoy

comment Single-punched response list
meal      ask 'The last time you went out for a meal, what type of
           restaurant did you visit?'
          resp sp 'indian' / 'chinese' / 'greek' / 'italian' / 'english' /
                 'west indian' / 'other'
day       ask 'What day did you go?'
          resp sp 'monday' / 'tuesday' / 'wednesday' / 'thursday' /
                 'friday' / 'saturday' / 'sunday'

comment Integer response list
people    ask 'How many people were there in your party?'
          resp num 1 to 20

comment Real response list
total     ask 'What was the total price of the meal, excluding the service
           charge or tip?'
          resp real 0 to 500.00
          set maxcost = total

comment Variable ranges in numeric response lists
          callfunc('limitresp',0,maxcost)
food      ask 'How much of that was for food?'
          resp real 0 to 500.00
          set maxcost = maxcost - food
          callfunc('limitresp',0,maxcost)
drink     ask 'And how much was for drink?'
          resp real 0 to 500.00

comment Dummy question whose value is set later in the script
paid      dummyask 'Total cost including service/tip'
          resp real 0 to 700
```

Quancept and mrInterview Scriptwriter's Manual

```
serv      ask 'How much did you pay in service charges or tips?'
          resp real 0 to 200
comment Assign value to dummy question
          set paid = total + serv

comment Open-ended response list
cook      ask 'When you invite friends to a meal at your house, what sort
          of meals do you prepare?'
          resp coded

end
```

4 Special responses

This chapter deals with special keywords you can use in a response list to represent certain general answers such as don't know, no answer and refused. It also introduces the *mapothzero* keyword that may be used to alter the way in which certain data is coded. Keywords in this chapter are:

datemap othzero	code specified other with code zero
dk	don't know
dummyoth	define a dummy specified other response
mapothzero	code specified other with code zero
msgansp	prompt text for numeric and open-ended responses
msgdk	text to display for <i>dk</i>
msgnull	text to display for <i>null</i>
msgoth	text to display for <i>other</i>
msgref	text to display for <i>ref</i>
null	no answer
^o	automatic 'other specify' with selected sp/mp responses
other	specified other
promptoth	automatic 'other specify' with selected sp/mp responses
ref	refused

4.1 No answer

Quick Reference

To allow 'no answer' as a response, place the keyword **null** at the end of the response list:

resp resp_type resp_list null

The keyword **null** in a response list represents a No Answer response; that is, where none of the answers in the list is applicable to the respondent. Since *null* is a keyword it is not enclosed in single quotes like other responses. If used, it must come at the end of the response list separated from the other responses by a space. For example,

```
machine ask 'What types of laundry equipment do you have?'
    resp mp 'automatic washing machine' / 'twin-tub' /
            'portable washing machine' / 'tumble dryer' /
            'combined washer-dryer' null
```

Here, a respondent may choose any of the listed appliances or may give no reply at all.

- ☎ When the response list is displayed during a CATI interview, *null* does not appear in the list of ordinary responses. Instead, the interviewing program displays a message underneath the response list telling the interviewer whether or not this response is valid.

-
- ☞ For further details on entering null responses during a CATI interview, see section 33.14, No answer, don't know and refused.
-

Using *null* instead of a quoted text has two advantages:

- Uniformity of coding across all questions.
- Automatic cleaning of multipunched responses.

☎ Null responses are always coded with a – (code 11), regardless of how many responses there are in the list. In the example above, the named appliances will be coded with codes 1 to 6, and *null* (no answer) will be coded with a – code. If we had used the quoted text ‘No answer’ or ‘None of these’ instead, the interviewing program would assign this code 7 because it was the seventh response in the list. If there are more responses than will fit in one column, the interviewing program always codes *null* in the first column of the field leaving the remainder of the field blank.

- ☛ mrInterview treats *null* in a single-punched or multipunched response list as an ordinary categoric response and stores it in the database variable for that question. If *null* is present in a numeric or open-ended response list, mrInterview creates a categoric variable called *qname.Codes* in which to store the response.
- ☎ Some companies do not want *null* or no answer responses coded in the data. Instead, they choose to leave the columns for those questions blank. You can do this by setting the environment variable QCNONULL to any value, for example:

```
setenv QCNONULL 1
```

In a multipunched response list, using *null* in place of a quoted text simplifies the task of cleaning data prior to tabulation since the interviewing program prevents interviewers from entering *null* with any other response from the list.

There is no order of precedence between *null* and the other special responses *dk* and *ref* (see below). You may use any combination of these responses and may place them in any order.

4.2 Don't know

Quick Reference

To allow 'don't know' as an answer to a question, place the keyword **dk** at the end of the response list:

resp resp_type resp_list dk

The special keyword for Don't Know responses is **dk**. It behaves in the same way as *null*:

- It is not enclosed in single quotes.
- It comes at the end of the response list after the quoted texts or numbers.
- It is separated from predefined responses by a space.
- Interviewers cannot select it with any other response.



The interviewing program codes a *dk* response with an & (code 12). If the data covers more than one column, the & is coded in the first column and the remainder of the field is left blank.



mrInterview treats *dk* in a single-punched or multipunched response list as an ordinary categoric response and stores it in the database variable for that question. If *null* is present in a numeric or open-ended response list, mrInterview creates a categoric variable called *qname.Codes* in which to store the response.

There is no order of precedence between *dk* and the other special responses *null* and *ref*. You may use any combination of these responses and may place them in any order.

4.3 Refusals

Quick Reference

To allow interviewers to deal easily with refusals, place the keyword **ref** at the end of the response list:

resp resp_type resp_list ref

Sometimes a respondent refuses to answer a question. You can allow for this either by entering a text response in the list or by placing the special keyword **ref** at the end of the list. Like *null* and *dk*:

- It is not enclosed in single quotes
- It must come at the end of the response list after the quoted texts or numbers

- It is separated from predefined responses by a space, and
- Interviewers cannot select it with any other response.



ref is coded into the data as { (a multipunch of '0&'). If the response list uses a field of columns, the { will appear in the first column and the other columns will be blank.



mrInterview treats *null* in a single-punched or multipunched response list as an ordinary categoric response and stores it in the database variable for that question. If *null* is present in a numeric or open-ended response list, mrInterview creates a categoric variable called *qname.Codes* in which to store the response.

There is no order of precedence between *ref* and the other special responses *null* and *dk* (see above). You may use any combination of these responses and may place them in any order.

4.4 Other responses

'Other' is a general purpose response that appears in many single-punched and multipunched response lists to collect responses other than those specifically mentioned in the response list. Sometimes you will only want to know that the respondent used some other product or traveled by some other form of transport, whereas on other occasions you will want to know exactly what other product or mode of transport was used. There are two ways of doing this.

Other as a quoted response

The first way of dealing with other is to enter it as a quoted response in the list and follow it with an open-ended response asking what it refers to:

```
buse    ask 'And which brand do you use most often?'
          resp sp ('suds' / 'washo' / 'bubbles' / 'gleam') go ctn / 'other'
othuse  ask 'ENTER OTHER BRANDS HERE'
          resp coded(14)
ctn     continue
```

In this example, the respondent has a choice of five answers from which he/she may choose one. He/she may select one of the four listed brands or he/she may choose 'other', in which case he/she is asked for an open-ended reply. If you are not interested in what the other brand is, you simply omit the routing and the othuse question.

Specified other

Quick Reference

To generate an automatic ‘Other – please specify’ question, type:

```
resp resp_type resp_list other[(n)]
```

The default is to allocate 29 codes (three columns) for coding of open ends given with specified other, but you may change this by typing the number of codes you require in parentheses after *other*.

The second method of dealing with ‘Other’ is to use the keyword **other**. This comes at the end of the response list, after any precoded responses but before any of the special responses *null*, *dk* and *ref*. We call this **Specified Other** because it automatically generates a ‘please specify’ question similar to *othuse* in the example above:

```
use1      ask 'And which brand or brands do you use?'
           resp mp 'suds' / 'washo' / 'bubbles' / 'gleam' other
```

other is a keyword so it is not separated from the response list by a slash.

Compare this question with the one below:

```
use2      ask 'And which brand or brands do you use?'
           resp mp 'suds' / 'washo' / 'bubbles' / 'gleam' / 'other'
```

With the first question we will know what other brands the respondent uses whereas with the second one we will not.



Unless otherwise directed, the parser reserves three additional columns for coding responses given with specified other. When the parser reads a single-punched or multipunched response list, it allocates the first nine responses to the first column and then ten responses to subsequent columns. By reserving three columns for specified other, the parser allows you 29 codes for coding those responses.

If you would like to reserve more or fewer codes for a particular question, type the number of codes you require in parentheses immediately after the *other* keyword. For example:

```
use2      ask 'And which brand or brands do you use?'
           resp mp 'suds' / 'washo' / 'bubbles' / 'gleam' other(9)
```

This question will be allocated two columns altogether: one for the predefined responses and one for coding other responses.

- mrInterview creates two variables for specified other in addition to any other variables that it normally creates for the question. These variables are a text variable called *qname*.Other that holds the respondent's answer, and *qname*.Other.Codes that will hold the codes assigned to this response by the coding program.

Data allocation of specified other



Quick Reference

To request that specified other is always coded as a zero in the first column for a question, type:

CATI/CAPI/Web: **mapothzero**

CATI/CAPI/Web: **datamap othzero**

as the first line of the script.

Quoted other and specified other are normally coded according to their position in the response list. In the earlier examples, the quoted other at othuse would be coded as code 5, and specified other for the use1 question would be coded as 5.

You cannot change the coding for quoted other. With specified other, however, you can use **mapothzero** or **datamap othzero** to request that it is always coded as a zero in the first column for the question. If used, this keyword must appear *as the first line in the script*. This ensures that the statement will be executed for every interview which takes place, even if subsequent statements take different sets of respondents to different parts of the script. If you use these keywords anywhere else in your script, the parser will issue an error message.

-
- ☞ If your script runs subsurveys and you use *mapothzero* or *datamap othzero* in the main script, you should also include it as the first line in each subsurvey script if you want this facility to carry over into those scripts.
 - ☞ To find out more about subsurveys, section 25.2, Running one script from inside another.
-

4.5 Choosing your own words for other, null, dk and ref

Quancept CAPI, Quancept Web and mrInterview allow you to define your own texts to be displayed when the script contains *other*, *null*, *dk* or *ref*. In both cases, you define the new text either in the script or in an external file.

In Quancept CAPI



Quick Reference

To display your own choice of words for other specify, no answer, don't know and refused but still keep the default codes for those responses, type:

keyword = 'text to display'

where *keyword* is *other*, *null*, *dk* or *ref*.

You can define your own words to be displayed in place of the standard *other*, *null*, *dk* and *ref* texts. You do this by defining your own words within the script itself or separately in a qcw.lng file. If a special response text is set in both the script and qcw.lng, the setting in the script is used.

To define special response text in the script, type the standard keyword followed by an equals sign and then the text you wish to display enclosed in single quotes. Here is an example:

```
machine ask 'What types of laundry equipment do you have?
resp mp 'automatic washing machine' / 'twin-tub' /
        'portable washing machine' / 'tumble dryer' /
        'combined washer-dryer' null='none of these'
```

To define texts for special responses in a separate file, create a qcw.lng file in the Windows directory using any text editor. You define text and formatting for special responses as the first language in the [Translation] section.

[Translation]

Language=1

X1_1000=new dk text

X1_1001=new null text

X1_1002=new ref text

X1_1003=new Other text

For example:

```
[Translation]
Language=1
X1_1000=Do not know
X1_1001=No reply
X1_1002=Refused to answer
```

» You may also use any of the attributes that change the color, font or size of the text, as shown in section 22.1, Formatting instructions in CAPI scripts

In Quancept Web and mrInterview



Quick Reference



To define a replacement text for *null*, type:

```
set msgnull = 'text'
```

To define a replacement text for *dk*, type:

```
set msgdk = 'text'
```

To define a replacement text for *ref*, type:

```
set msgref = 'text'
```

To define a replacement text for *other*, type:

```
set msgoth = 'text'
```

To cancel a user-defined message and return to the default text, type:

```
unset variable_name
```

where *variable_name* is one of the message variable names shown above.

The standard answer texts are No answer, Don't know, Refused and Other. They are displayed in a bold font at the bottom of the response list. Unless you specify otherwise, the texts you define will be displayed in the same font that is used for ordinary responses in the list. For example:

```
set msgdk='<b>No idea whatsoever</b>'
```

displays the words 'No idea whatsoever' in bold whenever a response list contains the keyword *dk*.

You can define texts in the script or in the project's initialization file (Quancept Web) or the messages.qli file in the mrInterviewSource directory (mrInterview). If the same message variable is defined in both places, the setting in the script overrides the setting in the file. This is a useful feature because it means that you can switch between a global text defined in the file and a more specific text defined at particular points in the script.

-
- ☞ For information about the Quancept Web project initialization file see section 40.3, The project initialization file.
 - For information about the mrInterview messages.qli file see 'Customizing messages in mrInterview' in chapter 12, Assignment.
-

- The replacement text you define with *msgoth* applies only to the text displayed for the *other* keyword. It does not apply to the response-specific specified other responses that you can generate by appending *^o* to individual response texts. To use the same replacement text for all types of specified other, set the variable **msgothal** to a non-zero value at some point before the question where the replacements are to be made.

☞ See ‘Using *^o*’ below for further information about response-specific specified other.

4.6 Automatic open ends with sp and mp responses

Quancept offers two keywords for generating an automatic ‘please specify’ question when a certain response is chosen. These are *promptoth* and *^o*. Because *^o* is retained for backwards compatibility with older scripts, we strongly recommend that you use *promptoth* for new scripts. This is particularly important in the case of scripts that may be translated into other languages.

Using **promptoth**



Quick Reference



- To generate an automatic ‘please specify’ question when a certain response is chosen from a single-punched or multipunched response list, type one of the following:

CATI **resp resp_type 'resp1' promptoth / 'resp2' promptoth other**

CATI **resp resp_type 'resp1' promptoth / 'resp2' promptoth dummyother[(n)]**

mrInterview **resp resp_type 'resp1' promptoth / 'resp2' promptoth**

Open-ended response lists for product awareness studies often contain more than one ‘other’ response. For example, in a washing powder survey you might have responses such as Other Automatic, Other Biological, and so on. These are quoted responses, which means that you have no way of knowing how many responses were given in each category, or what they were.

The **promptoth** keyword solves this problem. When appended to a response text in a single-punched or multipunched response list, it causes the interviewing program to issue an automatic ‘Please specify’ question if that response is chosen. For this to work you must include specified other in the list as well. If this is missing, the *promptoth* flags will be ignored. Here is an example:

comment In mrInterview, there is no need to include 'other' in the comment response list unless you want an additional Other response know ask 'Which washing powders can you think of?@

INTERVIEWER: FOR EACH UNLISTED POWDER CHOSEN, CHECK WHETHER IT IS AUTOMATIC, BIOLOGICAL OR GENERAL PURPOSE@'

```
resp mp 'Suds' / 'Washo' / 'Other general purpose' promptoth /
'Suds Automatic' / 'Washo Automatic' /
'Other Automatic' promptoth / 'Suds Biological' /
'Washo Biological' / 'Other Biological' promptoth other
```

If the interviewer selects Other Automatic as one of the responses, the interviewing program waits until he/she presses Return to move to the next question and then prompts for the open end with the words 'For Other Automatic'. If more than one *promptoth* response is chosen, the interviewing program will prompt for each one in turn and will show the chosen response text in each prompt.

The prompt text is always generated by taking the response text and preceding it with the word 'for'. You may not change this. However, you may include a reminder to interviewers as to what is required in the body of the main text, as we have done in our example.

-  The parser allocates three columns for coding all open ends associated with this question, just as it does when the response list contains specified other. The texts of the open ends are written to the tex file in the format:

<response code:predefined text>:open-end

so that you can see which open end belongs to which predefined response. A complete record from the open-end text file might look like this (in the file the text is all one line):

```
000012000003KNOW      0011343601<6:Other Automatic>:Bubbles
<9:Other Biological>:Gleam
```

-
- ↖ If more than one response is chosen, the open ends and their related predefined responses are listed one after the other with no intervening punctuation and ***they all share the same serial number***. If the interviewer requests a serial number for manual recording of open ends, the interviewing program will issue the same number for all open ends belonging to the same question.
-

If the interviewer also chooses specified other, this will appear in the open-end text entry in the same way as the other open ends.

Using specified other with *promptoth* to switch on automatic open ends causes specified other to become a valid response to the question.

```
000012000003KNOW      0011343601<6:Other Automatic>:Bubbles<OTHER>:Fresh and
Clean
```

- ☎ Specified other may not be a suitable response to the question. If not, use *dummyoth* instead of *other* in the response text. This switches on automatic open-ends but does not display an Other response in the response list. For example:

```

know ask 'Which washing powders can you think of?@'
INTERVIEWER: FOR EACH UNLISTED POWDER CHOSEN, CHECK WHETHER IT
IS AUTOMATIC, BIOLOGICAL OR GENERAL PURPOSE@
resp mp 'Suds' / 'Washo' / 'Other general purpose' promptoth /
'Suds Automatic' / 'Washo Automatic' /
'Other Automatic' promptoth / 'Suds Biological' /
'Washo Biological' / 'Other Biological' promptoth dummyother

```

As with *other*, the parser allocates 29 codes for coding the question's open ends. If a number of responses have a prompted other, you can reserve more space for coding the responses by following *dummyother* with the number of codes to be reserved in parentheses.

You can also extract individual texts from a list of open ends entered for *promptoth* responses, whereas with *other* all texts are extracted as a single block.

☞ See section 14.9, Checking text responses, for details.

Using **^o**

Quick Reference

To generate an automatic 'please specify' question when a certain response is chosen from a single-punched or multipunched response list, type one of the following:

CATI/CAPI/Web **resp resp_type 'resp1 ^o' / 'resp2 ^o' / 'resp3' other**

CATI **resp resp_type 'resp1 ^o' / 'resp2 ^o' / 'resp3' dummyother**

mrInterview **resp resp_type 'resp1 ^o' / 'resp2 ^o' / 'resp3'**

The **^o** flag works similarly to the *promptoth* keyword. Note the slight difference in syntax: the **^o** must appear inside the quotes. Here is an example:

```

comment In mrInterview, there is no need to include 'other' in the
comment response list unless you want an additional Other response
know ask 'Which washing powders can you think of?@'
INTERVIEWER: FOR EACH UNLISTED POWDER CHOSEN, CHECK WHETHER IT
IS AUTOMATIC, BIOLOGICAL OR GENERAL PURPOSE@
resp mp 'Suds' / 'Washo' / 'Other general purpose ^o' /
'Suds Automatic' / 'Washo Automatic' /
'Other Automatic ^o' / 'Suds Biological' /
'Washo Biological' / 'Other Biological ^o' other

```

- ☎ Specified other may not be a suitable response to the question, in which case you would use `dummyother` instead of `other`, as shown in the previous section.

If specified other is not a suitable response in Quancept CAPI or Quancept Web scripts, you can write some additional code that displays an error if this response is chosen. The script below shows how to do this, but uses several keywords you've not yet met. The comments explain what each one does.

```

know    ask 'Which washing powders can you think of?@'
INTERVIEWER: FOR EACH UNLISTED POWDER CHOSEN, CHECK WHETHER
IT IS AUTOMATIC, BIOLOGICAL, FOR WOOLENS, OR GENERAL PURPOSE@
    resp mp 'Suds' / 'Washo' / 'Other general purpose ^o' /
          'Suds Automatic' / 'Washo Automatic' /
          'Other Automatic ^o' / 'Suds Biological' /
          'Washo Biological' / 'Other Biological ^o' other

comment Test whether specified other (item 10) was chosen
set othbit = bit(know/10)
if (othbit) {
    display 'Do not choose other. If you do not know the type
of powder, code it as Other General Purpose. Please re-enter codes@@'
    goto know
}

```

As with `promptoth`, you can extract individual responses from the list of responses gathered with `^o`.

- ▣ If you have used `msgoth` to define a replacement text for the `other` keyword, you can apply this replacement text to the prompts displayed for `^o` by setting the variable **msgothal** to a non-zero value at some point before the question where the replacements are to be made. For example:

```

set msgoth = 'Other, please type in'
know    ask 'Which washing powders can you think of?'
        resp mp 'Suds' / 'Washo' / 'Other general purpose ^o' /
              'Washo Biological' / 'Other Biological ^o' other
comment Make msgoth apply to ^o texts too
set msgothal = 1
uses    ask 'Which washing powders do you use?'
        resp mp 'Suds' / 'Washo' / 'Other general purpose ^o' /
              'Washo Biological' / 'Other Biological ^o' other

```

During the interview, the response list for 'know' will be displayed as defined in the script, except that the words 'Other, please type in' will be displayed as the prompt for specified other. When the 'uses' question is displayed, the responses Other general purpose and Other Biological will both be displayed as 'Other, please type in', the same as `other`.

4.7 Prompt text for numeric, open-ended and other responses

Quick Reference

To define a replacement text for the Answer prompt that is displayed with the input boxes for open-ended, *num* and *real* responses, type:

```
set msgansp = 'text'
```

To return to the default prompt, type:

```
unset msgansp
```

For example:

```
age ask 'How old are you?'
resp num 18 to 99
set msgansp = 'Please type your answer here.'
tv ask 'Which television channels do you watch?'
resp mp 'Channel 1' / 'Channel 2' / 'Channel 3' other
```

In this example, the age question will display a box with the label Answer. The tv question will display a box for an open-ended response that will be labeled 'Please type your answer here'. This text will be used for all numeric and open-end input boxes until a new text is defined or until an *unset* statement is read that returns the prompt to the default.

You can define the replacement text in the script or in the project's initialization file (Quancept Web) or the messages.qli file in the mrInterviewSource directory (mrInterview). If the same message variable is defined in both places, the setting in the script overrides the setting in the file. This is a useful feature because it means that you can switch between a global text defined in the file and a more specific text defined at particular points in the script.

-
- ☞ For information about the Quancept Web project initialization file see section 40.3, The project initialization file.
 - For information about the mrInterview messages.qli file see 'Customizing messages in mrInterview' in chapter 12, Assignment.
-

4.8 Sample script

```
comment Sample script for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 4, Special responses

comment CATI/CAPI/Web: Code specified other as zero in first column of field.
comment mrInterview: Ignored because it is not applicable.

mapothzero

comment Null and ref for no answer and refused
machine ask 'What types of laundry equipment do you have?'
    resp mp 'automatic washing machine' / 'twin-tub' /
           'portable washing machine' / 'tumble drier' /
           'combined washer-drier' null ref

comment Specified other for individual responses
comment The example shown is for CATI. For mrInterview, delete dummyother.
comment For CAPI/Web, replace promptoth with ^o inserted just before the
comment closing single quote of the response text, and replace dummyother
comment with other. You will also need to uncomment the next section that
comment tests whether 'other' was chosen
know ask 'Which washing powders can you think of?@

INTERVIEWER: FOR EACH UNLISTED POWDER CHOSEN, CHECK WHETHER
IT IS AUTOMATIC, BIOLOGICAL OR GENERAL PURPOSE@
    resp mp 'suds' / 'washo' / 'other general purpose' promptoth /
           'suds automatic' / 'washo automatic' /
           'other automatic' promptoth / 'suds biological' /
           'washo biological' / 'other biological' promptoth
           dummyother

comment CAPI/Web only: Test whether specified other (item 10)
comment was chosen
comment      set othbit = bit(know/10)
comment      if (othbit) {
comment          display 'Do not choose other. If you do not know the type
comment of powder, code it as Other General Purpose. Please re-enter codes@@'
comment          goto know
comment      }

comment Specified other with null and dk for No Answer and Don't Know
use      ask 'And which brand or brands do you use?'
    resp mp 'suds' / 'washo' / 'bubbles' / 'gleam'
           other null dk

comment CAPI only: to see user-defined texts for null/dk, comment
comment out the previous 'use' question and uncomment this version
comment use      ask 'And which brand or brands do you use?'
comment      resp mp 'suds' / 'washo' / 'bubbles' / 'gleam'
comment          null='No answer' dk='Don''t know'
comment
end
```

5 Keywords in response lists

This chapter deals with keywords that you might include in a response list in addition to the responses to the question. These are:

atoz	list responses in alphabetical order
autojump	automatically jump to the next question when a response is entered
echo	echo the selected responses on the screen
freeze	response list is common to several consecutive questions
go	routing with responses
mp <i>n</i>	maximum number of responses allowed in a multipunch
nodata	do not allocate columns to this question
ran	randomize the order of responses in the list
rot	rotate the order of responses in the list
rotnewsed	regenerate rotation pattern for every question
rotranfix	fix the position of a response in a randomized or rotated list
rotransub	group together a set of responses in a randomized or rotated list
slider	display a numeric response list as a slide bar
sp	single-punched answer in a multipunched response list
^s	single-punched answer in a multipunched response list
^v	scaling factor for single-punched responses

Also covered in this chapter is the function **convdata**, which applies a scaling factor to a response when that response is chosen from the list, and the topic of assigning unique IDs to responses.

5.1 Displaying responses in alphabetical order

Quick Reference

To display responses in a single-punched or multipunched response list in alphabetical order, type the keyword **atoz** after the response type:

resp *resp_type* atoz *resp_list*

If the response list is long or if it is often appended to between different runs of an ongoing project, you can help interviewers find responses quickly by displaying the list in alphabetical order. To do this, place the keyword **atoz** after the *sp/mp* response type, as shown below:

```
hard    ask 'Which types of hard cheese do you usually buy?'
resp mp atoz 'gorgonzola' / 'cheddar' / 'red leicester' /
          'sage derby' / 'feta' / 'red windsor' /
          'bavarian blue' / 'caerphilly' other
```

Using `atoz` does not affect the coding in the data file. The interviewing program codes responses according to their position in the original list, so gorgonzola is always code 1 and caerphilly is always code 8, even though these appear as the fifth and second responses in the sorted list.

- ☎ Quancept sorts according to the ASCII code sequence. If your scripts use languages such as Swedish whose characters are outside this sequence, you must specify the language using either the environment variable QCLANG or the *setlocale* callable function. The variable and function control not only the collation sequence for alphabetic sorting but also, amongst other things, the characters that are recognized as punctuation symbols and the positions at which line breaks occur in long texts.

☞ See 'General environment variables' in chapter 39, Managing CATI projects, for information about QCLANG, and 'Setting the language for alphabetic sorting, punctuation and line breaks' in chapter 27, Callfuncs, for information about *setlocale*.

- ☞ If the responses are set up as defined lists and some questions use subsets of responses from the master list, it can be useful at the interviewing stage to combine alphabetic displays with the Quancept CATI environment variable QCRETAIN. When set to 1, this variable causes responses in a list to be displayed with the same response number whenever the list is used. Normally, responses are numbered sequentially from 1, which can mean that a response will have different response numbers in different lists.
- ☞ See chapter 7, Defined response lists, for general information about using lists. For further information about QCRETAIN, see section 7.10, Using question names with lists, and chapter 42, Quancept CAPI design mode.
-

Additional features for alphabetic sorting in multilingual mrInterview scripts



Quick Reference

To override the default language for sorting, type:

set atozlang = *language_code*

where *language_code* is the three-character language code of the language you want to use.

To override the default sorting order, type:

set atozsort = *value*

where *value* is a numeric value as described later in this section.

To define how the sorting process should deal with differences in case and with non-alphabetic characters, type:

```
set atozflgs = value
```

where *value* is a numeric value as described later in this section.

To use the old, pre-version 2.0 method of sorting, type:

```
set atozmode = value
```

where *value* is any numeric value except zero.

In versions of mrInterview prior to 2.1, the *atoz* keyword did not always sort responses starting with accented characters correctly. For example, in a German script, responses whose first character had an umlaut would be sorted to the end of the list rather than appearing at the position of the corresponding unaccented character. mrInterview 2.1 and later uses a different sorting function which employs a locale code (a combination of a language code and a sort code), and a set of comparison flags to determine how responses should be sorted.

You should not normally need to make any changes to the locale code and comparison flags, but keywords are available for this should you wish to do so.

The default language code for the script is set according to the value of the *language* variable. You can override the language code used for sorting by specifying a new code with the **atozlang** variable. For example:

```
set atozlang = 'ESN'
```

to set the sorting language to Spanish.

-
- ☞ See section 12.9, Setting the language in multilingual mrInterview scripts, for information about the *language* variable.
Refer to http://msdn.microsoft.com/library/default.asp?url=/library/en-us/intl/nls_238z.asp for a list of language codes.
-

If you want to use a different sorting order, set the variable **atozsort** to one of the following values:

If <i>language</i> is	Set atozsort to	For
Japanese	0	Japanese XJIS order
	1	Japanese Unicode order
Chinese	0	Chinese BIG5 order
	0	PRC Chinese Phonetic order

If language is	Set atozsort to	For
Chinese	1	Chinese Unicode order
	2	PRC Chinese Stroke Count order
	3	Traditional Chinese Bopomofo order
Korean	0	Korean KSC order
	1	Korean Unicode order
German	1	German Phone Book order
Hungarian	0	Hungarian Default order
	1	Hungarian Technical order
Georgian	0	Georgian Traditional order
	0	Georgian Modern order

atoz normally ignores differences in case by converting all uppercase characters to lower case before sorting. You can change this aspect of the sorting process by setting the variable **atozflgs** to one of the following values:

Value	Sort type
0	Retain differences between upper and lower case
1	Ignore case differences (the default)
2	Ignore nonspacing characters
4	Ignore symbols
65536 (hex 10000)	Ignore Kanatype
131072 (hex 20000)	Ignore width
4096 (hex 1000)	Use the string sorting method

-
- ☞ For further information on these settings refer to
[http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/resources\(strings/stringreference/stringfunctions/comparestring.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/resources(strings/stringreference/stringfunctions/comparestring.asp).
-

If the new methods of sorting fail for any reason or the script sets the variable **atozmode** to a non-zero value, sorting will take place as in earlier versions of mrInterview. This means that responses will be converted to lower case and will then be sorted using the Unicode values of the characters.

5.2 Echoing responses on the screen



Quick Reference

Place the keyword **echo** at the end of the response list to echo the response entered for this question:

resp resp_type resp_list echo

A useful way of verifying important data is to echo back the response entered by the interviewer and to have him/her confirm that it is correct. You will not want to do this for every question, but it can be particularly useful with numeric responses where the script is designed to accept any number in a given range. For example:

```
cost      ask 'How much, on average, is your monthly grocery bill?'
          resp num 1 to 500  ref echo
```

If the average monthly bill is £230, this response is echoed back to the interviewer until he or she confirms that this is correct before continuing. If the answer is wrong, the interviewing program clears the response previously entered and waits for the interviewer to enter another answer.

5.3 Response lists referring to several questions

Quick Reference

If a single-punched or multipunched response list is common to a number of consecutive questions, define the response list with the first question and type **freeze** at the end of the response list.

resp sp or mp resp_list freeze

A frozen response list is displayed at the top of the screen until it is removed by another *resp* statement or an **unfreeze** statement.

To change the response type of a frozen response list from *sp* to *mp* or vice versa, type a response list containing just the new response type.

Sometimes you will have several questions which use the same response list. A typical example is with rating scales when you want to use the same set of ratings for a number of questions. You can, of course, type each question followed by the response list, but this is slow and unnecessary, especially when the Quancept language can do it for you.

The easiest way to tackle this requirement is to set the rating scale up as a **frozen response list** so that the interviewing program will display it permanently at the top of the screen. Questions which use that response list are then displayed below the response list, and only that section of the screen is cleared as each new question is displayed. The response list is removed only when a statement that signals the end of the frozen list is read. For example, the statements:

```
opinion ask 'Could you give me your overall opinion of Country
Fayre Cream Cheese? Was it ... '
      resp sp 'very good' / 'fairly good' / 'neither good nor bad' /
             'fairly poor' / 'very poor' freeze
```

will display the five responses in the list at the top of the screen until you remove them.

Once a response list has been frozen, all subsequent questions containing that response list may be entered without separate response lists. The interviewing program will know that the frozen response list refers to these questions too. Your script could therefore continue:

```
opinion ask 'Could you give me your overall opinion of Country
Fayre Cream cheese? Was it ... '
      resp sp 'very good' / 'fairly good' / 'neither good nor bad' /
             'fairly poor' / 'very poor' freeze
advert ask 'How did you rate the advertisement?'
package ask 'How would you describe the packaging?'
taste ask 'What did you think of the taste?'
```

You may switch the response type for a frozen response list from single punched to multipunched and vice versa by following the question with an abbreviated response list containing just the response type *sp* or *mp* as appropriate.

```
flav ask 'Which flavors did you try?'
      resp mp 'herb' / 'herb and garlic' / 'smoked salmon' freeze
prefer ask 'And which did you like best?'
      sp
```

A response list may be unfrozen either by another *resp* statement — that is, another response statement starting with the keyword *resp* — or by the keyword **unfreeze**. Here are two examples to illustrate this, both of which are equally correct:

```
comment Example 1
flav1 ask 'Which flavors did you try?'
      resp mp 'herb' / 'herb and garlic' / 'smoked salmon' freeze
prefer1 ask 'And which did you like best?'
      sp
buy1 ask 'Would you consider buying Country Fayre Cream Cheese?'
      resp sp 'yes' / 'no' null dk ref
comment Example 2
flav2 ask 'Which flavors did you try?'
      resp mp 'herb' / 'herb and garlic' / 'smoked salmon' freeze
prefer2 ask 'And which did you like best?'
      sp
      unfreeze
```

Let's look at the first example. The response list defined with *flav1* will be displayed at the top of the screen and will remain there until *buy1* is displayed with its new response list. The frozen response list will be understood as referring to *flav1* and *prefer1* even though the response type is changed from *mp* to *sp* at *prefer1*.

In the second example, the response list with flav2 is displayed on the screen until *unfreeze* is read after the response to prefer2 has been entered.

-
- ❖ Another way to deal with response lists shared by a number of questions is to use defined lists. Besides providing facilities for manipulating responses in lists, defined lists allow you to share a response list between nonconsecutive questions in the script.
 - ☞ To find out how to use defined lists for responses common to a number of questions, see chapter 7, Defined response lists.
-

5.4 Routing in single-punched and numeric response lists

Quick Reference

To insert routing instructions in single-punched and numeric response lists, use **go**:

```
resp sp 'resp1' go label1 / 'resp2' go label2 / 'resp3'  
resp num val1 go label1 / val2 go label2 / val3
```

A single-punched response list may contain routing when the next question to be asked depends on the answer to the current question. For example:

```
buy    ask 'Which type of cheese do you usually buy?'  
      resp sp 'hard cheese' go hard / 'soft cheese' go soft /  
             'cream cheese'
```

The respondent has a choice of three responses. If he/she answers ‘hard cheese’, the next question he/she answers will be ‘hard’; if he/she answers ‘soft cheese’, the next question is ‘soft’; and if he/she answers ‘cream cheese’, he/she continues with the next question in the script.

If the same routing applies to two or more consecutive answers in the list, you may enclose the group in parentheses and follow it with a single *go*:

```
buy    ask 'Which type of cheese do you usually buy?'  
      resp sp ('hard cheese' / 'soft cheese') go notcc /  
             'cream cheese'  
cream  ask 'Which flavors of cream cheese do you buy?'  
      resp mp 'herb' / 'herb and garlic' / 'smoked salmon'  
notcc  ask 'Have you ever bought cream cheese?'  
      resp sp 'yes' / 'no'
```

You can also use *go* with numeric response lists:

```
resp num 10- go exit / 11 to 20 go q15 / 21+
```

Any number less than or equal to 10 goes to exit; numbers in the range 11 to 20 go to q15, and responses of 21 or more continue with the statement immediately after this list.

-
- ☛ Do not use this sort of routing with multipunched response lists because it is possible that the responses given will result in conflicting routing. For example, if the previous example allowed more than one response and the respondent chose 'hard cheese' and 'cream cheese', you could not be sure whether the next question asked would be 'notcc' or 'cream', since the route taken depends on the order in which the interviewer enters the codes for these responses.
 - ☞ To set up routing for multipunched response lists, use the keywords described in chapter 14, Checking non-numeric responses.
-

Routing with null, dk and ref

If the same routing applies to *null*, *dk* and *ref* as to other responses in the list, you must enter it separately for each of those responses. Thus:

```
flav      ask 'Which is your favorite flavor?'
          resp sp ('herb' / 'herb and garlic') go herb /
                  'smoked salmon' null go more dk go more ref go exit
```

5.5 Number of responses chosen from a multipunched list

Quick Reference

To set a maximum for the number of responses that may be selected from a multipunched response list, type that number after the *mp* response type:

```
resp mp max resp_list
```

A maximum of zero allows any number of choices and is the default.

You can restrict the number of responses that a respondent may select from a multipunched response list. In a question which asks the respondent to choose two items from a list, you may enter this maximum as part of the response list so that the interviewing program will issue an error message if the interviewer enters more than two codes.

For example:

```
use ask 'Please select three ways in which you or your
household has used cheese in the last six weeks.'
resp mp 3 'sandwiches' / 'sauces' / 'with wine' /
'with biscuits' / 'in salads' / 'with burgers'
```

Here, the respondent may choose up to three answers. If the interviewer enters more than three codes, the interviewing program will display the error message ‘Too many responses (maximum 3) given *n*’ and will wait for the interviewer to re-enter three or fewer response codes.

-  Quancept CATI, Quancept CAPI and Quancept Web allocate columns in the data map based on the number of responses in the response list.
-  Because mrInterview stores data in a relational database, the issue of blank columns in the data does not arise. mrInterview creates a categoric variable that stores all the responses chosen.

5.6 Asking questions without storing responses

Quick Reference

To prevent the interviewing program from writing responses to the data files, place the keyword **nodata** at the end of the response list:

```
resp resp_type resp_list nodata
```

The Quancept language lets you define questions which will be asked during the interview but which will not be allocated any space in the data files. Answers given to these questions are available for use during the interview (for example, they may be displayed on the screen) and any routing associated with it is obeyed, but once the interview is over, they are lost. For example, the question:

```
natcurr ask 'What is your national currency?'
resp sp 'Pounds Sterling' / 'French Francs' /
'German Marks' / 'Dutch Guilders' / 'Italian Lira' /
'US Dollars' nodata
```

makes the respondent’s national currency available during the interview but does not allocate space for saving this information in the data file.

With open ends, *resp coded (0)* in a CATI, CAPI or Web script stores the text of the open end even though it does not allocate any columns for it to be coded into the data file. If you do not want the open-ended text stored at all, replace the (0) in the response list with the keyword *nodata*:

```
name ask 'What is your name?'
resp coded nodata
```

The respondent's name will be available during the interview, but once the interview is finished, the name is lost. (This works in mrInterview as well as in the Quancept variants.)

This is a useful facility for scripts which contain questions asking the respondent to verify that answers given earlier in the script or read in from the respondent's sample record are correct. If the existing data is correct, there is nothing to do; if it is incorrect, you can write statements in the script to correct it, so there is no need to keep the yes/no answers to the verification questions themselves. *nodata* is also useful with dummy questions used for manipulating responses during the interview but whose answers do not need to be stored as part of the data.

-
- ☞ For information on Quancept CATI's sample management system see the *Quancept Sample Management and Telephony Systems User's Guide*.
 - For information on mrInterview's sample management system see the online *mrInterview User's Guide*.
-

5.7 Rotation and randomization of responses

Responses in single-punched or multipunched response lists can be presented in rotated or random order. The response list is entered in exactly the same way as before, but the response type (either *sp* or *mp*) is followed by one of the keywords **rot** or **ran**.

Responses in rotated or randomized response lists are coded in the data according to their position in the original list rather than their position in the current rotation or randomization, so you can be sure that a code 1 in a certain column always refers to the first response in the list, and a code 2 always refers to the second response in the list.

Rotated response lists

Quick Reference

To present responses in a single-punched or multipunched response list in rotation, follow the response type with the keyword **rot**:

resp resp_type rot resp_list

When the interviewing program displays responses in rotation, the response displayed first on the screen depends on the number of responses in the list.

```
state ask 'Now I'm going to read you some statements people
have made about cheese. For each one I'd like you to tell me
whether you agree or disagree with the statement.@@
INTERVIEWER: Read each statement in the response list in turn and
code only those with which the respondent agrees.'
      resp mp rot 'Soft cheese is bad for some people' /
                  'Cheese is good value for money' /
                  'Cats like cheese as a special treat' /
                  'Cheese makes a healthy snack' /
                  'Cheese for supper gives you bad dreams' /
                  'Old hard cheese is best for catching mice'
```

The first time the question is asked, the second response in the list may be displayed first, followed by all other responses in the list in the order in which they appear in that list, with the first response at the end of the list:

Cheese is good value for money
 Cats like cheese as a special treat
 Cheese makes a healthy snack
 Cheese for supper gives you bad dreams
 Old hard cheese is best for catching mice
 Soft cheese is bad for some people

The second time the question is asked, responses will be presented in the same order except that the third response in the list will be displayed first, with the first and second responses at the end of the list:

Cats like cheese as a special treat
... other responses as shown above ...
 Soft cheese is bad for some people
 Cheese is good value for money

The third time, the response list will look like this:

Cheese makes a healthy snack
Cheese for supper gives you bad dreams
Old hard cheese is best for catching mice
Soft cheese is bad for some people
Cheese is good value for money
Cats like cheese as a special treat

When a questionnaire contains a number of rotated response lists with the same number of responses, the same rotation pattern is used for all those response lists. This means that if several questions use the same rotated response list, the responses to each of those questions will appear in the same order throughout an interview.

-
- 〝 In mrInterview 1.0 only, this was not the case because the rotation pattern was regenerated for each response list. To retain this behavior in later mrInterview scripts, set the variable **rotnwse** to 1.
-

Testing response patterns for rotated response lists

Quick Reference

When responses are presented in rotation, you may find the position of the first response in the original list with a statement of the form:

set *variable* = rotation

If you later want to analyze responses to see whether response patterns vary according to the position of a response in the list, you will need to save the rotation number as part of the data. Using the last rotation example shown above, the first response (Soft cheese is bad for some people) appears in the fourth position in the list, so the rotation value returned will be 4.

-
- 〝 For this statement to return the correct rotation value, you must place it *after* the question and response list to which it refers — in the example, after the response list for the question called ‘state’. If you place it before the question, the first rotation value will be zero, and all rotation values thereafter will be wrong.
-

To copy this value into the data, use a *fix* statement. For example:

```
state    ask 'Now I''m going to read you some statements people
have made about cheese. For each one I''d like you to tell me
whether you agree or disagree with the statement.@@
INTERVIEWER: Read each statement in the response list in turn and
code only those with which the respondent agrees.'
      resp mp rot 'Soft cheese is bad for some people' /
                  'Cheese is good value for money' /
                  'Cats like cheese as a special treat' /
                  'Cheese makes a healthy snack' /
                  'Cheese for supper gives you bad dreams' /
                  'Old hard cheese is best for catching mice'
      set pos = rotation
fixpos  fix (1) pos
```

 The *fix* statement is described in chapter 17.7, Inserting non-response data in the data file.

If the script contains more than one question with a rotated response list and those lists all contain the same number of responses, the rotation pattern will be the same in a given interview for all those lists. Therefore, you need save the rotation value only once for the whole batch of questions. Additionally, since the rotation value is set the first time rotation is requested, the *set* statement may be entered at any place in the script, as long as it is after the first question with a rotated response list.

When you analyze the data, you may need to calculate for yourself which response was presented first, based on the position of the first response in the original list. If the statement 'Soft cheese is bad for some people' is displayed in 4th place, you will need to work out for yourself that the statement 'Cheese makes a healthy snack' (number 5 in the original list) was the first item in the list as it was presented to the respondent.

Randomized responses lists

Quick Reference

To present responses in a single-punched or multipunched response list in random order, follow the response type with the keyword **ran**:

```
resp resp_type ran resp_list
```

The keyword *ran* causes the responses to be presented in a random order each time the list is used. Each response will always be displayed, but the internal ordering of the responses in the list will be ignored. For example:

```
state    ask 'Now I''m going to read you some statements people
have made about cheese. For each one I''d like you to tell me
whether you agree or disagree with the statement.@@
INTERVIEWER: Code only those with which the respondent agrees.'
resp mp ran 'Soft cheese is bad for some people' /
'Cheese is good value for money' /
'Cats like cheese as a special treat' /
'Cheese makes a healthy snack' /
'Cheese for supper gives you bad dreams' /
'Old hard cheese is best for catching mice'
```

will list the statements in a random order each time.

Rot and ran with loops

Rotation and randomization are commonly used with loops. Loops are simply a method of dealing efficiently with repetitive questions.

Many scripts contain rating sections where respondents are asked to say whether or not they agree with various statements. When the list of statements is long, respondents may sometimes give less accurate ratings to statements near the end of the list, so, to ensure a more accurate distribution of answers, you may decide to vary the order in which statements are presented.

Here is an example of randomized presentation. If you run several interviews using the sample script for this chapter, you will see how it works:

```
d4    protect 'I''m going to read you some comments which people have
made about this product. For each one will you say whether you
agree or disagree?@'
11    for com = 'It''s difficult to find it in the shops' /
      'Very tasty' / 'For health conscious people' /
      'Good value for money'    ran
qx    ask 'Do you agree or disagree with the statement: '+com+'?'
      resp sp 'Yes' / 'No' / 'No opinion'
next
```

-
- ☞ You will find a complete description of how to use loops in chapter 10, Repetitive questions, and chapter 11, Iteration and subscription.
-

Fixing a response's position in a list



Quick Reference



To fix the position of a response in a rotated, randomized or alphabetically sorted list so that it is always displayed in the same position in the list, type **rottranfix** after that response text:

```
resp resp_type atoz 'resp1' rottranfix / 'resp2' / 'resp3'
```

```
resp resp_type rot 'resp1' rottranfix / 'resp2' / 'resp3'
```

```
resp resp_type ran 'resp1' rottranfix / 'resp2' / 'resp3'
```

You may want to rotate, randomize or alphabetically sort most but not all responses in a list. A common example is when a multipunched response list contains special responses such as 'None of these' that you always want to appear at the end of the list. To prevent these responses being moved, type them in the position you want them to appear in the list and follow each one with the keyword **rottranfix** as shown here:

```
shop ask 'Which of the following supermarkets do you shop at?'
      resp mp ran 'Safeway' / 'Pricecheck' / 'Tesco' / 'Waitrose' /
           'Sainsbury''s' / 'None of these' sp rottranfix
```

In this example, the supermarkets will be displayed randomly, while the response 'None of these' will always appear last in the list. (The *sp* keyword between the response text and *rottranfix* indicates that this is a single-punched response in a multipunched list.)

-
- ❖ When *atoz* and *rottranfix* appear in the same response list, *rottranfix* overrides *atoz* so that responses tagged with *rottranfix* appear at the designated positions in the list, even if this means that they are not sorted alphabetically. In other words, the list as a whole is sorted alphabetically and then the *rottranfix* responses are switched back to their original positions in the list.
-

Grouping responses in a list



Quick Reference



To group a set of consecutive responses so that they remain together even if the response list is rotated, randomized or alphabetically sorted, place the keyword **rottransub** on the second and succeeding responses in each group:

```
resp resp_type atoz 'resp1' / 'resp1a' / 'resp2a' rottransub / 'resp3a' rottransub
```

```
resp resp_type rot 'resp1' / 'resp1a' / 'resp2a' rottransub / 'resp3a' rottransub
```

```
resp resp_type ran 'resp1' / 'resp1a' / 'resp2a' rottransub / 'resp3a' rottransub
```

All responses that are flagged with *rottransub* are attached to the most recent unflagged response. When the response list is rotated, randomized or sorted alphabetically, the reordering is based on the unflagged responses only. The responses flagged with *rottransub* remain in their original order and always follow the response to which they were attached.

The following example uses *rottransub* to group different kinds of responses about supermarkets. The first group consists of three comments about the staff, the second group consists of three comments about the store, and the third group consists of a single comment about prices (the response texts start with g1, g2 or g3 to help you identify the responses in each group). When the interviewing program rotates the response list, it rotates each group as a single entity rather than dealing with each response individually.

```
rate      ask 'Thinking about the last time you visited '+shop+'.  
Did you think that ...'  
      resp mp rot 'g1: The staff were helpful' /  
                  'g1: The staff were friendly' rottransub /  
                  'g1: The staff were smartly dressed' rottransub /  
                  'g2: The shop was clean and tidy' /  
                  'g2: There was a wide choice of products' rottransub /  
                  'g2: You could find things easily' rottransub /  
                  'g3: The prices were competitive'
```

If you use *rottransub* with *atoz*, the interviewing program sorts the responses alphabetically according to the first response in each group. If you were to sort the previous example alphabetically rather than rotating the responses, responses would appear on the interview screen in the following order:

g3: The prices were competitive
 g2: The shop was clean and tidy
 g2: There was a wide choice of products
 g2: You could find things easily
 g1: The staff were friendly
 g1: The staff were helpful
 g1: The staff were smartly dressed

Using rottranfix and rottransub in the same response list

If you use *rottranfix* and *rottransub* in the same response list, the response groups are re-ordered as usual, but any responses flagged with *rottranfix* are detached from their groups and retain their original positions in the list. The remaining responses in those groups remain grouped and rotate in the same way as other groups.

In the following example, there are three groups to be rotated, but the first text of the second group (The shop was clean and tidy) will always appear as the fourth item in the list, with other responses being placed before and after it.

```

rate      ask 'Thinking about the last time you visited '+shop+'.'
Did you think that ...
resp mp rot 'g1a: The staff were helpful' /
             'g1b: The staff were friendly' rottransub /
             'g1c: The staff were smartly dressed' rottransub /
             'g2a: The shop was clean and tidy' rottranfix /
             'g2b: There was a wide choice of products' rottransub /
             'g2c: You could find things easily' rottransub /
             'g3: The prices were competitive'
  
```

The first time the question is displayed, the responses may be shown in the following order:

g1a: The staff were helpful
 g1b: The staff were friendly
 g1c: The staff were smartly dressed
 g2a: The shop was clean and tidy
 g2b: There was a wide choice of products
 g2c: You could find things easily
 g3: The prices were competitive

The next time the question is displayed, response group g1 rotates to the end of the list, making group g2 the first group. However, because the first response in that group is flagged with *rottranfix*, it is detached and placed in position four in the list as it appears in the script:

g2b: There was a wide choice of products
g2c: You could find things easily

g3: The prices were competitive
g2a: The shop was clean and tidy

g1a: The staff were helpful
g1b: The staff were friendly
g1c: The staff were smartly dressed

The final repetition is as follows. Response group g2 has rotated to the bottom of the list but the *rotranfix* response has been detached and inserted in position four, in the middle of group g1:

g3: The prices were competitive

g1a: The staff were helpful
g1b: The staff were friendly

g2a: The shop was clean and tidy
g1c: The staff were smartly dressed

g2b: There was a wide choice of products
g2c: You could find things easily

5.8 Single-punched responses in a multipunched list

Quancept has two keywords for designating a single-punched response in a multipunched list. These are **sp** and **^s**. Because **^s** is retained for backwards compatibility with older scripts, we strongly recommend that where possible you use **sp** for new scripts. This is particularly important in the case of scripts that may be translated into other languages.

Using sp



Quick Reference



To flag an individual response in a multipunched response list as a single-punched response, follow the response text with **sp**:

```
resp mp 'resp1' / 'resp2' sp / 'resp3'
```

Often, you will want to include an answer such as 'None of the above' in a multipunched list and you will want some means of preventing that response from being selected with any of the other responses in the list. The Quancept language provides the keyword **sp** for this purpose.

Type the text of the response in the normal way, but add *sp* after the quote to show that this response should be treated as single punched. For example:

```
utensil ask 'Which of the following do you have?'
      resp mp 'cheese board' / 'cheese grater' / 'fondue set'
            'cheese knife' / 'none of these' sp
```

If the interviewer enters the code for ‘none of these’ with any other response code, the interviewing program will display the message ‘Response 5 must be single coded’ and will wait for the interviewer to re-enter an acceptable set of response codes.

-
- ❖ This restriction applies only to the interviewer. The interviewing program does not check for *set* statements in the script which insert invalid multipunches in the data.
-

Using ^s

Quick Reference

Append *^s* to a response in a multipunched response list to indicate that it is to be treated as single punched:

```
resp mp 'resp1' / 'resp2' / 'resp3' ^s
```

The flag *^s* works similarly to *sp*. Type the text of the response in the normal way, but before you enter the terminating single quote, type *^s* to show that this response should be treated as single punched. For example:

```
utensil ask 'Which of the following do you have?'
      resp mp 'cheese board' / 'cheese grater' / 'fondue set'
            'cheese knife' / 'none of these' ^s
```

-
- ❖ In versions of Quancept prior to version 7e.5, responses flagged with *^s* were always displayed at the end of the response list even if the list was presented in a rotated or random order. From version 7e.5 onwards, this is no longer the case: responses flagged with *^s* are rotated or randomized in the same way as other responses in the list. If it important that *^s* responses appear at the end of the list you should be able to achieve this using *sp* and *rotranfix* instead.
-

5.9 Scaling factors for numeric responses scale



Quick Reference

To apply scaling factors to responses in a single-punched list, follow responses with `^v` and a number:

```
resp sp 'resp1 ^vnumber' / 'resp2 ^v/number'
```

To scale a number up, type just the scaling factor; to scale a number down, type `/` before the scaling factor.

In some scripts, you may want to convert a number into a different form; for example, you may have a response in francs or marks that you wish to convert to pounds, or a distance in kilometers that you want to convert to miles. Conversions of this type are achieved by scaling the numeric response up or down by a given factor — that is, by multiplying or dividing the response by a predefined value.

This type of conversion in a script requires three statements:

1. A question with a single-punched response list that defines the scaling factors to be applied.
2. A question with a numeric response (either *num* or *real*) that is the value to be converted.
3. A statement that applies the appropriate scaling factor to the response.

We will look at each of these statements, and we will build a script that converts values in francs, marks, guilders, and so on into pounds sterling. You could use this in real life if you were running a project in several countries but wanted to merge the national data files into a single international file for analysis.

Setting the scaling factor

The first step is to find out which currency the respondent uses and to set the scaling factor to be the exchange rate between that currency and sterling. We do this by defining a single-punched response list in which each currency name is followed by `^v` and a conversion factor.

```
comment Exchange rates are tourist rates as at 29 Oct 1991
natcurr ask 'What is your national currency?'
resp sp 'Pounds Sterling ^v1' /
        'French Francs ^v/9.66' /
        'German Marks ^v/2.85' /
        'Dutch Guilders ^v/3.21' /
        'Italian Lira ^v/2140' /
        'US Dollars ^v/1.675'
```

Look at the values after each ^v. Every one except pounds sterling starts with a /. This signals a scaling down or division: the scaling factor of /3.21 for Dutch guilders indicates that there are 3.21 guilders to the pound, so to convert from guilders to sterling we must divide by 3.21. The factor for pounds sterling is a dummy value which we could have entered as /1 because there is no difference between dividing or multiplying by one. (The alternative would be to omit it and to skip the conversion statement if answers were already in sterling.)

Since it is unlikely that interviewers will be working in one country and calling another, you could have interviewers set the environment variable QCLOCALE to their international dialing code and then use this as a basis for setting the scaling factor automatically. For example:

```
comment Define currency as a dummy question ....
natcurr  dummyask 'What is your national currency?'
          resp sp 'Pounds Sterling ^v1' /
                  'French Francs ^v/9.66' /
                  'German Marks ^v/2.85' /
                  'Dutch Guilders ^v/3.21' /
                  'Italian Lira ^v/2140' /
                  'US Dollars ^v/1.675' nodata
comment Get value of interviewer's QCLOCALE variable ....
chkloc   set evar = 'QCLOCALE'
          callfunc('getenv',evar,country)
comment ... and set conversion factor accordingly
          if (country='44') {set natcurr=1}
          if (country='33') {set natcurr=2}
          if (country='49') {set natcurr=3}
          if (country='31') {set natcurr=4}
          if (country='39') {set natcurr=5}
          if (country='1')  {set natcurr=6}
```

Some of the statements in this example will be new to you, but the comments explain what is happening at each step. You will find full descriptions of these statements and other examples of their use later in this manual.

Find the data to be converted

The next step is to obtain the numeric data which needs to be converted. This is just a case of writing a standard question with an integer or real response. We will ask how much the household spends monthly on grocery shopping:

```
cost      ask 'How much, on average, is your monthly grocery bill?'
          resp real 1 to 500 ref echo
```

The data file will store the amount the respondent says shown to three decimal places. At this stage, no conversion has taken place.

Apply the scaling factor to the data

Quick Reference

To apply scaling factors, use the **convdata** callfunc:

```
callfunc('convdata', input, scale, output)
```

where *input* is the name of the variable containing the value to be converted, *scale* is the name of the single-punched question whose response defines the scaling factor, and *output* is the name of the variable to which the conversion will be written.

The final step is to apply the scaling factor. You do this with the **convdata** function.

If you decide to write the sterling conversion to a variable called *sterling*, you would type:

```
callfunc('convdata', cost, natcurr, sterling)
```

This tells the interviewing program to take the value in *cost*, multiply or divide by the value in *natcurr* and write the result out to *sterling*.

Suppose that the household spends 200 guilders a month on groceries. The value of *cost* will be 200 and the value of *natcurr* will be /3.21. Once the conversion has taken place, the value of *sterling* will be 62.305 pounds.

In this example, we have written our conversion out to a variable which did not exist prior to the conversion. Variables such as this which are not questions are called **temporary variables**. We will talk more about temporary variables later, but for the time being you should know that temporary variables have a response type just like questions. The response type of a question is determined by the *resp* statement which follows it, whereas the response type of a temporary variable is normally determined by the type of data you place in it.

With *convdata*, if you use a new temporary variable, the interviewing program will create it with the same type as the response you are converting. In this example, *cost* is a real variable so *sterling* is created as a real variable too.

If you use a temporary variable which you have used before in the script, Quancept will write the conversion to that variable, changing the response type of that variable if necessary. So, if you have already used a variable called *sterling* and have placed a whole number in it, its response type will be *num*. If the conversion is a real value, it will be written to the *sterling* variable as a real number and the response type of *sterling* will change from *num* to *real*.

☞ See chapter 12, Assignment, for more about temporary variables.

If the output variable is a question (perhaps a dummy question which you do not actually ask), you must ensure that it is of the same type as the variable you will be converting. If not, and you save a real conversion value in a question variable with a *num* type response, that value will be rounded to the nearest whole number. For example, 200 francs is 20.704 pounds sterling. When saved in a *num* type question variable, it will appear as 21. Similarly, 200 guilders is 62.305 pounds sterling, which would be saved as 62 in a *num* type question variable.

The advantage of writing the conversion to a question variable is that the value will be placed in the data file automatically unless you flag the response list to that question with the option *nodata*.

-
- ☞ For further information about *nodata*, refer to section 5.6, Asking questions without storing responses.
-

Now that we have looked at each component of a scaling question, let's put the whole thing together into a sample script which you can copy.

```

comment Define currency and sterling as dummy questions
natcurr  dummyask 'National currency'
        resp sp 'Pounds Sterling ^v1' /
                  'French Francs ^v/9.66' /
                  'German Marks ^v/2.85' /
                  'Dutch Guilders ^v/3.21' /
                  'Italian Lira ^v/2140' /
                  'US Dollars ^v/1.675' nodata
sterling  dummyask 'Sterling conversion'
        resp real 0 to 9999 echo

comment Get value of interviewer's QCLOCALE variable and set
comment conversion factor accordingly
chkloc    set evar = 'QCLOCALE'
          callfunc('getenv',evar,country)
          if (country='44') {set natcurr=1}
          if (country='33') {set natcurr=2}
          if (country='49') {set natcurr=3}
          if (country='31') {set natcurr=4}
          if (country='39') {set natcurr=5}
          if (country='1') {set natcurr=6}

comment Now ask the question
cost      ask 'How much, on average, is your monthly grocery bill?'
        resp real 1 to 500 ref echo

comment Convert cost to sterling. Because sterling is a question,
comment the value will be written to the data file automatically
        callfunc('convdata',cost,natcurr,sterling)
d1       display cost+' '+natcurr+' is '+sterling+' pounds sterling'
        end

```

5.10 Slide bars for numeric response lists



Quick Reference

To display a numeric response list as a slide bar, type:

```
resp num 'resp_list' slider('default')
```

A variation of this is:

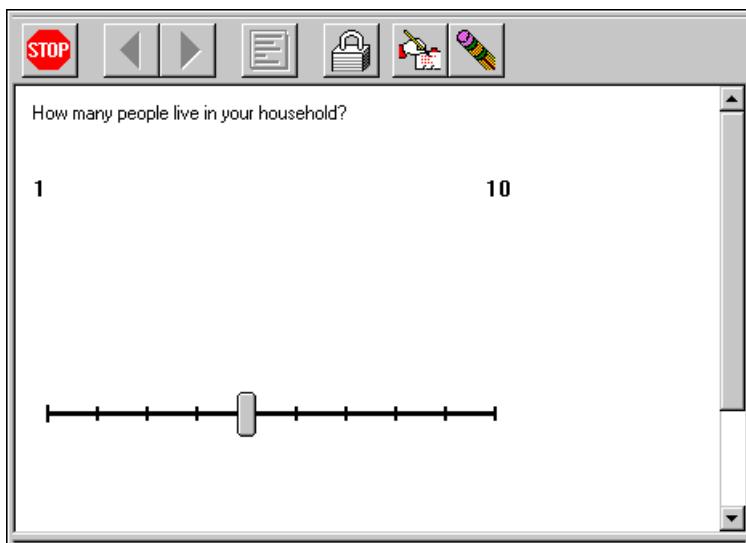
```
resp num 'resp_list' slider('paired') 'left_text' / 'right_text'
```

which requests a slide bar with text labels at either end.

Slide bars provide a graphic representation of a numeric response list. The range of acceptable responses is laid out along a line, somewhat like a ruler, and a movable button is provided for selecting a value on the ruler. For example, a question and response list defined as:

```
hsizer ask 'How many people live in your household?'
resp num 1 to 10 slider('default')
```

appears on the interviewing screen as follows:

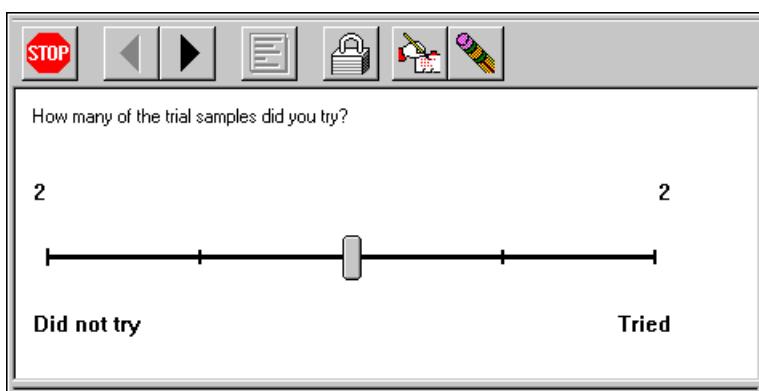


A numeric response can sometimes be derived from the response to an earlier related question. An example is a pair of questions that ask respondents how many trial samples they tried out of the total that they were given. One way of dealing with this is to ask two questions that use the same response list and to check afterwards that the sum of the two responses does not exceed the maximum allowed. A better way is to use a *paired* slide bar that collects the answers to both questions at once and ensures that the maximum is never exceeded.

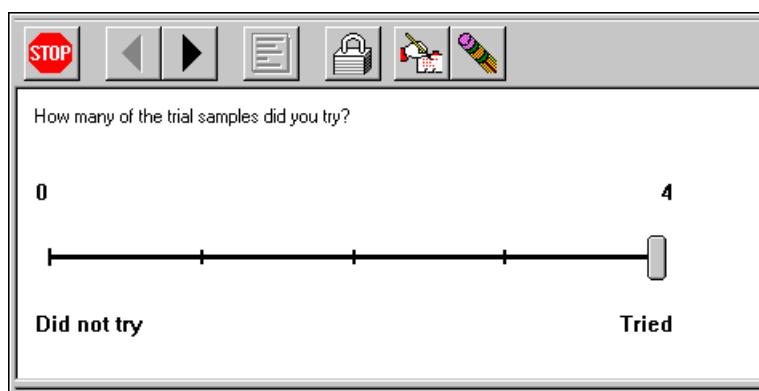
When you use a paired slide bar, you still need to define two questions to hold the responses, but you declare one of them as a dummy, because its responses can be obtained from the question with the slide bar. You would define the questions to do with trial samples as follows:

```
unused  dummyask 'Number of untried samples'
        resp num 0 to 4
used    ask 'How many of the trial samples did you try?'
        resp num 0 to 4 slider('paired') 'Did not try' / 'Tried'
        set unused = 4 - used
d1      display 'Tried '+used+ ' samples; Did not try '+unused+ ' samples'
        pause
```

The screen that the interviewer sees is:



The script asks the respondent how many trial samples were tried and calculates from the answer the number of trial samples not tried. The slide bar starts in the middle of the range. The text below the slide bar shows that the value on the right refers to the number of samples tried, so if the respondent tried all four samples the interviewer should move the slide bar as far to the right as possible. The values above the slide bar change accordingly:



When you define paired slider questions, you need to understand how Quancept converts the question into a slide bar and how it allocates the value set on the slide bar to the question variable. The points to remember are:

- The question text is displayed exactly as you define it. The text has no effect on the way the slide bar works or on the way the data is held.
- The numbers above the slide bar come directly from the response list. The starting value for the response list must be zero. If it is non-zero, the values displayed on the slide bar will be incorrect.
- The texts below the slide bar come directly from the *resp* statement. The first one is printed on the left of the slide bar and refers to the dummy question; the second is printed on the right of the slide bar and refers to the current question. The texts have no effect on the way the slide bar works and are simply there to remind the user which question he or she is answering.
- When the slide bar is displayed, Quancept tries to position the slider centrally within the range. This works best if the upper limit of the range is an even number.

5.11 Automatically jumping to the next question



Quick Reference

To jump automatically to the next question when a single-punched response is chosen using a mouse or light pen, type:

resp sp 'resp_list' autojump=number

where *number* is any value greater than zero.

To deactivate automatic jumping to the next question, type:

resp sp 'resp_list' autojump=0

For questions with single-punched response lists, you can instruct the interviewing program to jump automatically to the next question when a response is selected. This means that the respondent or interviewer does not have to select the forward arrow after entering the response.

For example:

```
buy    ask 'Which type of cheese do you usually buy?'
      resp sp 'hard cheese' go hard / 'soft cheese' go soft /
             'cream cheese' autojump=1
cream   ask 'Which type of cream cheese do you usually buy?'
      resp sp 'herbs' / 'herbs and garlic' / 'smoked salmon'
```

As soon as the interviewer selects the type of cheese that the respondent usually buys, the next question is displayed. If the respondent buys cream cheese, this is the ‘cream’ question. This question also has a single-punched response list, so as soon as the interviewer selects an answer the next question is displayed.

autojump is ignored for single-punched questions displayed using *multitask* and *freeze*. In this case, the interviewer must click the forward button to move to the next question.

autojump works for interviews that are conducted using a mouse or light pen because in these interviews any response can be selected using a single action — that is, by touching or clicking on a response button on the screen. This keyword has no effect with the CAPI keyboard interface because, for a question that has more than nine responses, two keystrokes may be needed to select a response.

-
- ☞ Rather than placing an *autojump* keyword on a response list in the script, you can set *autojump* for the whole questionnaire using QCompile or QCShell.
 - ☞ For details see ‘The Run tab’ in chapter 32, Parsing and compiling scripts.
-

5.12 Unique IDs for responses



Quick Reference



To apply a unique ID to a single-punched or multipunched response, type:

resp resp_type 'resp1' ('Id1') / 'resp2' ('Id2')

Unique IDs are special labels that you assign to responses in a single-punched or multipunched response list, and that you can use as an alternative to the response text or code when you want to refer to that response. Suppose, for instance, that you assign the unique ID ‘xgold’ to a product called Xeno’s Gold. When you want to refer to this product in logical tests, you will be able to use either its full name, or its position in the response list, or its unique ID.

One of the benefits of unique IDs is that they allow you to use the same script in a number of countries or markets, but to vary the product names or brand names between the countries or markets involved. If you assign unique IDs to the product or brand names that vary, and you use those IDs whenever you want to refer to those products, all you will have to change are the brand or product names at the point at which they are defined.

Unique IDs are recognized by all SPSS MR multiversion software. They cannot be translated. If you have a script that will be translated, you should give each response a unique ID so that you can use that ID instead of the response text in logical expressions. This allows Quancept to make comparisons using the text represented by the unique ID in the current interview language rather than reading an explicit text in a particular language from the questionnaire script. (In some instances, comparisons against the base (default) language work but you should not rely on this.)

You can use any combination of alphanumeric characters in an ID, up to a maximum of 512 characters. In Quancept, IDs must be unique within a response list but not necessarily across the script as a whole. If the same text appears in several response lists you may give it the same ID in all lists.

You can apply unique IDs to responses that appear in response lists attached to *ask*, *dummyask* or *define* statements, as well as to texts on *for* statements. Here is an example in a response list attached to an *ask* statement:

```
tpaste ask 'Which brand of toothpaste do you usually buy?'
resp sp 'Minty Fresh' ('minty') // 'Freshmint' ('fmint') /
      'Ice Blue' ('iceblue') / 'Xeno''s Pep' ('xenpep') /
      'Bright''s' ('brights') / 'Xeno''s gold' ('xgold') null
```

You can use unique IDs in the same way as response texts or codes in routing and *if* statements. When you do this, you need to type not only the identifier itself but also the single quotes and parentheses that surround it. For example:

```
if (tpaste='Ice Blue') {
    display 'Respondent uses Ice Blue'
}
if (tpaste = ('fmint')) {
    display 'Respondent uses Freshmint'
}
```

The first statement shows one way of checking whether a specific response was chosen from a single-punched list. The response text is enclosed in single quotes. Compare this with the second *if* statement that uses a unique ID instead of a response text. Here you have the unique ID enclosed in parentheses and single quotes, exactly as it appears in the response list.

-
- ☞ For information on these logical expressions see chapter 13, Logical expressions.
For information on flow control statements such as *if*, see chapter 9, Routing and flow control statements.
-

Unique IDs in mrInterview

❑ Unique IDs are more important in mrInterview than in Quancept and, although they are optional, you are strongly advised to use them because:

- They uniquely identify individual responses within the script.
Each response within a question's response list or on a *for* statement has its own unique ID. If the same response appears in more than one place in the script, you may use the same identifier for that response each time it is used. For example, if Brand A is present in two response lists and on a *for* statement, you may give it the same unique ID in all places.

-
- ☞ See section 10.2, Writing loops, for details on unique IDs for loops.
-

- They are used as variable names in the case data.

When the case data file is created for a script, a unique variable name is created to hold each response. When a question is part of a loop (that is, it is a repeated question), the variable name is a combination of the loop name, the question name and the loop control item's unique ID.

-
- ☞ See 'Labelling for statements' in chapter 10, Repetitive questions for details.
-

- They ensure that variables and responses are consistent across different versions of the script. Responses that have the same identifier in more than one version of the script are the same response. When activating a project multiple times, it is therefore important to ensure that responses that are the same have the same identifiers, and that responses that are different have different and unique identifiers. For example, if version 1 of the questionnaire has the response text 'Red' with the unique ID Red and you change the response text to 'Reddish colors', there is no need to change the unique ID in version 2 because this is still the same response. However, if you replace 'Red' with 'Blue' you must also change the unique ID to 'Blue' otherwise mrInterview will treat Blue as if it were Red.

The mrInterview parser generates unique IDs for any responses that do not have them. The procedure for doing this is shown below. However, this only a satisfactory solution in the early stages of a project when you are more concerned with the passage through the script than with the data gathered. When you start wanting to look at data, you must define your own unique IDs.

- If the response text ends with the `^o` (other specify) or `^s` (single response) marker, remove the marker.
- Take the first 45 characters of the response text, remove any nonalphanumeric characters except \$, _, # and @, and convert any spaces to underscores (two or more consecutive spaces are converted to a single underscore). If this makes the string shorter than 45 characters, and there are more characters available at the end of the response text, these characters are appended to the string to make it up to 45 characters again.
- If the resulting string starts with a number or \$, _, # or @, insert the letter 'e' at the start of the string.
- Compare the string against the user-defined and automatically generated unique IDs for all previous responses and, if the string is unique, make this the response's unique ID.
- If the string matches a previous ID, extend the string by appending an underscore and a number that is the response's position in the list (1 for the first response, 2 for the second, and so on). If this string is unique, it becomes the response's unique ID.
- If the string is still not unique, issue an error message. Normally, this procedure does not fail because it is unlikely that you will choose IDs that match the automatically generated IDs.

Using this procedure, the unique ID for '18-25 years' is e1825_years, the unique ID for '£20000–£30000' is e2000030000, and the unique ID for '\$20000–\$30000' is e\$20000\$30000.

-
- ❖ There is a limit of 999 characters per response if you do not define unique IDs. Responses of 1000 characters or more without unique IDs generate an ‘unspecified error’ in QCompile.
-

If you change a response text that does not have a user-defined unique ID or you change a user-defined unique ID, the parser creates a new response in the questionnaire definition file rather than altering the original ID for that response. The parser data merge process that updates the data model to match the metadata works with unique IDs rather than response texts and will not recognize that this response already exists in the data model. It therefore allocates a new position for this response in the case data. If you have already started interviewing on the project before changing the response text, you will need to combine the two responses from the case data in order to collect everyone who gave that response. It is for this reason that you are advised to create your own unique IDs.

mrInterview does not currently work with unique IDs that contain 8-bit characters. If your script contains 8-bit characters you should ensure that you define your own unique IDs for these responses.

If you define your own IDs, you are responsible for ensuring they are valid.

-
- ❖ Although unique IDs are very important in mrInterview scripts, they are not defined in all examples in this manual nor in all the associated sample scripts. Scripts that refer only to mrInterview have unique IDs defined; scripts that refer to Quancept and mrInterview do not.
-

5.13 Sample scripts

Script A

```
comment Sample script A for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 5, Keywords in response lists

shopper ask 'May I speak to the person who had the trial samples of
Country Fayre Cream Cheese?'
    resp sp 'yes' / 'no' go exit

comment Numeric question with slider
comment Sliders only work in CAPI. For CATI/Web/mrInterview, delete from
comment 'slider' to the end of the line in the 'used' response list
unused dummyask 'Number of untried samples'
    resp num 0 / 1 to 4
used ask 'How many of the trial samples did you try?'
    resp num 0 to 4 slider('paired') 'Did not try' / 'Tried'
    set unused = 4 - used
d1 display 'Tried '+used+ ' samples; Did not try '+unused+ ' samples'
    pause
    route(used=0) go exit

comment Frozen response list
opinion ask 'Could you give me your overall opinion of Country Fayre
Cream Cheese? Was it .... '
    resp sp 'very good' / 'fairly good' / 'neither good nor bad' /
        'fairly poor' / 'very poor' freeze
advert ask 'How did you rate the advertisement?'
package ask 'How would you describe the packaging?'
taste ask 'What did you think of the taste?'

comment Cancel frozen response list
othans ask 'Is there anything else you would like to say about
Country Fayre Cream Cheese?'
    resp coded

comment Routing in the response list
comment CAPI only: switch on automatic jumping to next question for
comment mouse or light pen input
comment CATI/Web/mrInterview: delete autojump=1 from response list
buy ask 'Which type of cheese do you usually buy?'
    resp sp 'hard cheese' go hard / 'soft cheese' go soft /
        'cream cheese' autojump=1
cream ask 'Which type of cream cheese do you usually buy?'
    resp sp 'herbs' / 'herbs and garlic' / 'smoked salmon'
    goto ctn
```

```
comment Response list displayed in alphabetical order
hard    ask 'Which type of hard cheese do you usually buy?'
        resp sp atoz 'gorgonzola' / 'cheddar' / 'red leicester' /
                  'sage derby' / 'feta' / 'red windsor' /
                  'bavarian blue' / 'caerphilly'
        goto ctn
soft    ask 'Which type of soft cheese do you usually buy?'
        resp sp atoz 'brie' / 'camembert' / 'port salut' /
                  'mozzarella' / 'limeswold' / 'danish blue'
ctn    continue

comment Restrict the number of selections from an mp list
use    ask 'Please select three ways in which you or your household
have used cheese in the last six weeks.'
        resp mp 3 'sandwiches' / 'sauces' / 'with wine' /
                  'with biscuits' / 'in salads' / 'with burgers'

comment Rotate order of responses in list seen by interviewer
state   ask 'Now I''m going to read you some statements people
have made about cheese. For each one I''d like you to tell me
whether you agree or disagree with the statement.@@
INTERVIEWER: Read each statement in the response list in turn and
code only those with which the respondent agrees.@
        resp mp rot 'Soft cheese is bad for some people' /
                  'Cheese is good value for money' /
                  'Cats like cheese as a special treat' /
                  'Cheese makes a healthy snack' /
                  'Eating cheese for supper gives you bad dreams' /
                  'Old hard cheese is best for catching mice'

comment Save position of the Soft cheese statement in list seen by
comment respondent
        set pos = rotation
fixpos  fix (1) pos

comment Single-punched answer in multipunched list
comment The example is for CAPI/Web/mrInterview. For CATI replace
comment the '^s' with sp after the closing single quote of the
comment response text
utensil ask 'Which of the following do you have?'
        resp mp 'cheese board' / 'cheese grater' / 'fondue set' /
                  'cheese knife' / 'none of these ^s'

comment Unsorted response in an alphabetically sorted list
shop    ask 'At which shop do you do your main grocery shopping?'
        resp sp atoz 'Safeways' / 'Asda' / 'Sainsbury' / 'Netto' /
                  'Co-op' / 'Morrisons' / 'Aldi' / 'Tesco' /
                  'None of these' rotranfix go exit
```

```
comment Response groups in a rotated list
rate      ask 'Thinking about your most recent visit to '+shop+',
did you think that ....@@
INTERVIEWER: Code all that the respondent agrees with'
resp mp rot 'The shop was clean and tidy' /
'There was a wide choice of products' rotransub /
'You could find things easily' rotransub /
'The prices were competitive' rotransub /
'The staff were helpful' /
'The staff were friendly' rotransub /
'The staff were smartly dressed' rotransub

exit      display 'Thank you. Goodbye.'
end
```

Script B

 For this script to work you must have the environment variable QCLOCALE set to one of the values shown in the script.

```
comment Sample script B for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 5, Keywords in response lists (CATI only)

shop      ask 'At which shop do you do your main grocery shopping?'
resp sp atoz 'Safeways' / 'Asda' / 'Sainsbury' / 'Netto' /
            'Co-op' / 'Morrisons' / 'Aldi' / 'Tesco' /
            'None of these' rotranfix go cost

cost      ask 'How much, on average, is your monthly grocery bill?'
resp real 1 to 5000 ref go exit echo
natcurr dummyask 'National currency'
resp sp 'Pounds Sterling ^v1' / 'French Francs ^v/9.66' /
        'German Marks ^v/2.85' / 'Dutch Guilders ^v/3.21' /
        'Italian Lira ^v/2140' / 'US Dollars ^v/1.675' nodata
sterling dummyask 'Sterling conversion'
resp real 0 to 9999
chkloc    set evar = 'QCLOCALE'
callfunc('getenv',evar,country)
if (country='44') {set natcurr=1}
if (country='33') {set natcurr=2}
if (country='49') {set natcurr=3}
if (country='31') {set natcurr=4}
if (country='39') {set natcurr=5}
if (country='1') {set natcurr=6}

comment Convert cost to sterling.
callfunc('convdata',cost,natcurr,sterling)
d1       display cost+' '+natcurr+' is '+sterling+' pounds sterling'
pause
```

Quancept and mrInterview Scriptwriter's Manual

```
tpaste ask 'Which brand of toothpaste do you usually buy?'
resp sp 'Minty Fresh' ('minty') // 'Freshmint' ('fmint') /
       'Ice Blue' ('iceblue') / 'Xeno''s Pep' ('xenpep') /
       'Bright''s' ('brights') / 'Xeno''s gold' ('xgold') null

if (tpaste =('fmint')) {
qfm           ask 'What is it about '+tpaste+' that you
particularly like?'
resp coded
}

ishop      ask 'Have you ever used the Internet for shopping?'
resp sp 'Yes' / 'No' go exit

comment Suppress standard specified other prompt when using
comment response-specific prompts
where      ask 'What prompted you to do this?'
resp mp 'Newspaper/magazine advertisement' promptoth /
        'Newspaper/magazine article' promptoth /
        'Mailshot' promptoth / 'TV/radio advertisement' /
        'Friend/relative' dummyother

exit      end
```

6 Writing multimedia scripts

If you are scriptwriting for Quancept CAPI or Quancept Web, you can incorporate multimedia elements in your script. By multimedia we mean sound or video clips and pictures which are played or shown to the respondent. In Quancept CAPI you can also record respondents' open-ended responses as a picture that has been 'written' on screen or record their voices.

Quancept CATI with the QTS provides facilities for recording spoken open ends as audio files.

This chapter describes the following keywords for use in multimedia scripts:

audio	record an open-ended response as an audio file
#audio=playback	play a sound file
#audio=record	record an open-ended response as an audio file
	display a picture
maxrecord	set a maximum recording time for an open-ended response
maxvideo	play a video clip that takes up the entire screen
mmaudio	display a button that plays an audio file
mmautoplay	play an audio file automatically
mmcaption	determine where response texts are printed in relation to pictures
mmpicsize	specify the size of the picture to be displayed
mmpicture	display a picture
mmpicunit	define the units to be used in sizing a picture
mmvideo	play a video clip in a frame
scribble	allow respondents to write responses using an electronic pen
set samprate=	set sampling rate for audio recordings
set sampbits=	set the number of sampling bits for audio recordings

6.1 Displaying pictures in Quancept CAPI



Quick Reference

To display picture files as part of the question text, type:

```
label  ask 'question_text' mmpicture='filename.bmp'  
resp resp_type resp_list
```

To display picture files as buttons in the response list, type:

```
label  ask 'question_text'  
resp resp_type 'resp_text' mmpicture='filename.bmp' / ...
```

Pictures are a useful means of increasing the visual appeal of an interview as well as being a practical way of displaying images for brand awareness or advertising recognition.

Quancept CAPI can display bitmap images held in files with a .bmp extension. Almost all graphics packages can save graphics in this format, so if the pictures you want to use aren't already saved as bitmaps, you should be able to convert them simply by loading them into a graphics package and saving them as bitmaps.

You can store a script's bitmap files in the same directory as the script or in another directory. If you store the files in the same directory as the script, you need only give the filename with *mmpicture*. If the files are in a different directory, you must enter the file's full pathname, with a drive letter if appropriate. For example:

```
qlogo  ask 'This is the logo for Active Holidays. Do you think it  
shows what the company does?' mmpicture='c:\pictures\active.bmp'  
      resp sp 'Very well' / 'Quite well' / 'Neither well nor badly' /  
           'Not very well' / 'Not at all well'
```

With questions and display texts, you can display a number of pictures by entering a list of *mmpicture* options separated by commas. The pictures will be displayed one below the other, under the question or display text. For example:

```
q8    ask 'Which of these logos is the one for Active Holidays?'  
      mmpicture='active.bmp', mmpicture='fake1.bmp', mmpicture='fake2.bmp'  
      resp sp 'The first one' / 'The second one' / 'The third one' dk
```

In response lists, pictures are associated with individual responses and will be displayed as buttons during the interview. Each response must therefore have its own *mmpicture* option, but it is acceptable to mix responses with pictures and text-only responses in the same list.

For example:

```
activ  ask 'Which water sports have you done during the last five years?'
resp mp 'Snorkeling' mmpicture='snorkel.bmp' /
       'Scuba diving' mmpicture='scuba.bmp' /
       'Surfing' mmpicture='surf.bmp'/
       'Wind surfing' mmpicture='windsurf.bmp'/
       'None of these ^s'
```

If you define routing in the response list using *go varname*, the *mmpicture* keyword must come after the routing instruction as shown here:

```
activ  ask 'And which is your favorite?'
resp mp 'Snorkeling' go snorkel mmpicture='snorkel.bmp' /
       'Scuba diving' go scuba mmpicture='scuba.bmp' /
       'Surfing' go surf mmpicture='surf.bmp'/
       'Wind surfing' go windsurf mmpicture='windsurf.bmp' /
       'None of these ^s'
```

If you will be using the same set of response texts and pictures several times during the script, you can create a defined response list whose responses are associated with pictures. You might find this particularly useful in an advertising awareness study where you want respondents to say which brand or product a particular picture represents.

☞ See chapter 7, Defined response lists, for details.

Another way of increasing the flexibility of the script is to place the name of the picture file in a variable and then give the name of the variable with *mmpicture*. This allows you to vary the pictures displayed according to responses gathered earlier in the interview; for example, you could display a picture of the respondent's favorite brand, or a picture associated with a film that the respondent has seen recently. The way to do this is as shown below:

```
set varname='filename.bmp'
qname ask 'question_text' mmpicture='[+varname+]'
resp resp_type resp_list
```

☞ See section 8.8, Text substitution in response lists, for further details on using variables in this way.

Positioning response text relative to a picture



Quick Reference

To define the position of response text relative to a picture button, type:

resp resp_type 'resp_text' mmpicture='fname.bmp', mmcaption=number

where *number* indicates the position of the response text.

When a response list contains picture buttons, the interviewing program displays each picture with its response text on the right. If you want a response text to appear in a different position, or not at all, you can use **mmcaption** with one of the following values:

Use	To specify that response text
0	Is not displayed
1	Appears above the picture
2	Appears to the right of the picture (the default)
3	Appears below the picture
4	Appears to the left of the picture

The following example displays response pictures but no response texts:

```
activ ask 'Which of the following water sports have you tried  
in the last five years?'  
        resp mp 'Snorkeling' mmpicture='snorkel.bmp', mmcaption=0 /  
                'Scuba diving' mmpicture='scuba.bmp', mmcaption=0 /  
                'Surfing' mmpicture='surf.bmp', mmcaption=0 /  
                'Wind surfing' mmpicture='windsurf.bmp', mmcaption=0 /  
                'None of these ^s'
```

☞ *mmcaption* must follow *mmpicture*. If *mmcaption* precedes *mmpicture*, the interviewing program may not display the picture and response text in their correct positions on screen.

Defining a picture's size



Quick Reference

To define the size of a picture, type:

```
label  ask 'question_text'
      resp resp_type 'resp_text'
      mmpicture='fname.bmp', mmpicsize=number, [mmpicunit=unit]
```

where *number* is a measurement that is interpreted as either pixels or millimeters depending on the value of *unit*. *unit* is either 0, indicating that measurements are in pixels (the default), or 1, indicating that measurements are in millimeters.

The interviewing program displays picture buttons at the pictures' original sizes. This may be the effect that you want, but at times you may need more control over the display size. The keywords **mmpicsize** and **mmpicunit** give you this control by enabling you to set a size for a picture and the unit in which you have defined that size.

Use *mmpicsize* to define the display size for a picture. The interviewing program interprets this measurement in pixels. If you are defining a picture's size in millimeters instead, add *mmpicunit=1* to the definition. For example:

```
activ  ask 'Which of the following water sports have you tried
in the last five years?'
      resp mp 'Snorkeling' mmpicture='snorkel.bmp', mmpicsize=50,
            mmpicunit=1 /
      'Scuba diving' mmpicture='scuba.bmp', mmpicsize=50,
            mmpicunit=1 /
      'Surfing' mmpicture='surf.bmp', mmpicsize=50,
            mmpicunit=1 /
      'Wind surfing' mmpicture='windsurf.bmp', mmpicsize=50,
            mmpicunit=1 / 'None of these ^s'
```

6.2 Playing video clips



Quick Reference

To play a video clip in a section of the interviewing screen, type:

```
label ask 'question_text' mmvideo='filename.avi'  
resp resp_type resp_list
```

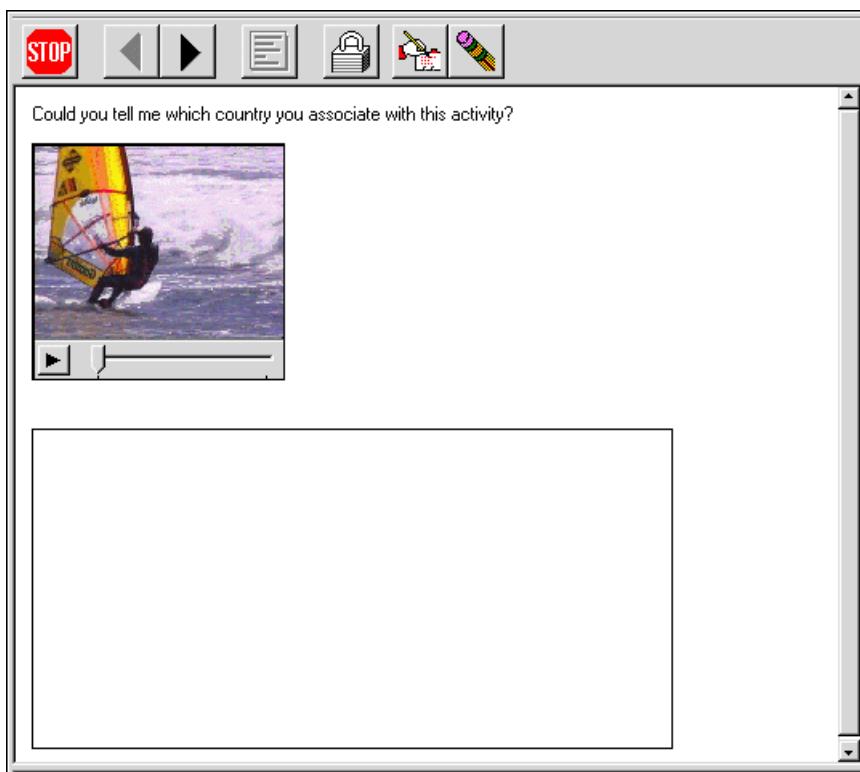
To play a video recording on a full screen, type:

```
label ask 'question_text' maxvideo='filename.avi'  
resp resp_type resp_list
```

When the interviewing program reaches a question that uses the *mmvideo* keyword, it displays the first frame of the video in its own window. The interviewer can play and stop the video using the controls that appear below it. For example, the question:

```
surfing ask 'Could you tell me which country you associate with  
this activity?' mmvideo='windsurf.avi'  
resp coded
```

will be displayed by the interviewing program like this:



If, instead of *mmvideo*, you use *maxvideo* to play the video on the full screen, a button labeled 'video' appears. When you press the button, the video fills the screen and the play/stop controls appear at the bottom of the screen.

The video recordings you use in a script must be stored as .avi or .mpg files. To use mpg files users need to have MPEG hardware and software installed on their PCs. Any hardware and software that is used must have MPEG-standard MCI drivers. If you use MPEG hardware that has nonstandard MCI drivers, QCompile will be unable to run the mpg files.

You can store video files in the same directory as the script or in a different directory. If a file is in the same directory as the script, you need only give the filename. If the file is in a different directory, you must enter the full pathname, with a drive letter if appropriate:

```
surfing ask 'Which country you associate with this activity?'
mmvideo='n:\video\holz\windsurf.avi'
resp coded
```

☞ You cannot use *mmvideo* on *display* statements.

For increased flexibility, the name of the video recording file may be stored in a variable:

```
set varname='filename.avi'
label ask 'question_text' mmvideo='[+varname+]'
resp resp_type resp_list
```

☞ See section 8.8, Text substitution in response lists, for further details on using variables in this way.

6.3 Playing sound recordings



Quick Reference

To play a sound recording, type:

```
label ask 'question_text' mmaudio='filename.wav'
resp resp_type resp_list
```

Any sound recording you want to include in an interview must be stored in .wav format. You can store this file in the same directory as the script or in a different directory. If the file is in the same directory, you need only give the filename with *mmaudio*. If the file is in a different directory, you must enter the file's full pathname, giving the drive letter if necessary. If you need to type a pathname for the file, separate the pathname's components with two backslashes rather than one.

For example:

```
radioad ask 'I''m going to play you a radio advertisement about  
wind surfing in Australia. Please tell me what you think about it.'  
mmaudio='n:\sounds\radio_ads\windsurf.wav'  
resp coded
```

When the interviewing program reaches a question that uses *mmaudio*, it displays the question text with a button below it for playing the named sound file.

The name of the sound recording file may be entered explicitly or stored in a variable:

```
set varname='filename.wav'  
label ask 'question_text' mmaudio='[+varname+]'  
resp resp_type resp_list
```

» You may not use *mmaudio* on *display* statements.

Playing audio files automatically



Quick Reference

To play a sound file automatically when the question is displayed, type:

```
label ask 'question_text' mmautoplay='filename.wav'  
resp resp_type resp_list
```

Sound files named with *mmaudio* do not start playing until you press a button. If you want the sound to begin as soon as a question is reached, replace *mmaudio* with *mmautoplay*. This plays the audio file automatically so no audio button appears on the interview screen.

6.4 Recording open ends in audio format



Quick Reference

To save an open-ended response as a sound recording, type:

```
label ask 'question_text'  
resp coded audio
```

If the interviewing PC has a sound card, you can record open-ended responses as audio files. This could be useful for telephone interviews or for personal interviews where the interviewer visits respondents in their own homes.

When the interviewing program reaches an open-ended question that uses the *audio* keyword, it allocates a serial number to the response and opens a new file in which the respondent's words will be recorded. The interviewing program names this file qcserial.wav, where *serial* is the response serial number. For example, if the response has been allocated the serial number 1492, the file will be called qc1492.wav. The directory in which this file is created depends on the AudioResultDirName setting in the interviewing computer's quantime.ini file.

If you want to play the recording back to the respondent later in the interview, use *mmaudio* as follows:

```
advert ask 'If you could make your own advertisement for
wind surfing, what would it be like?'
      resp coded audio
qcheck ask 'So what you''re saying is ...' mmaudio='[+advert+]'
      resp sp 'Yes, that''s correct' / 'No, not at all' go advert
```

Setting a maximum recording time



Quick Reference

To set the maximum recording time for an open-ended response, type:

```
label ask 'question_text' maxrecord=number_of_seconds
      resp coded audio
```

The default recording time is 60 seconds.

Sound recording files can grow quite large, so if you're expecting or requiring only a short response, it's worth limiting the recording time. This reduces the size of the recording files. In exceptional cases, you can also use **maxrecord** to increase the recording time by entering a value greater than 60 seconds. For example:

```
advert ask 'If you could make your own advertisement for
wind surfing, what would it be like?' maxrecord=75
      resp coded audio
```

increases the sound recording time to one minute and 15 seconds.

Setting the sample rate for recordings



Quick Reference

To set the sample rate, type:

set sampRate=value

where value is one of the sample rates supported by the PC's sound card. The default is 11025.

To set the sample bits, type:

set sampBits=value

where value is either 8 or 16 depending on the PC's sound card. The default is 8 bits.

The interviewing program records open ends at 11025Hz in 8-bit format. You can change the sample rate or the sample bits to any value supported by the sound card in the interviewing PC. Common sound card defaults are:

Sound Quality	Sample frequency (Hz)	Sample bits
Compact Disk	44100	16
Radio	22050	8
Telephone	11025	8

When setting the sample bits, values less than 8 are converted to 8, and values greater than 8 are converted to 16

-
- ❖ Increasing the sample rate significantly increases the size of the WAV files created during interviewing. This affects the amount of disk space occupied by the files on the interviewing computer as well as the amount of time required to upload the files to the Project Control Center.
-

6.5 Saving an open end as a picture



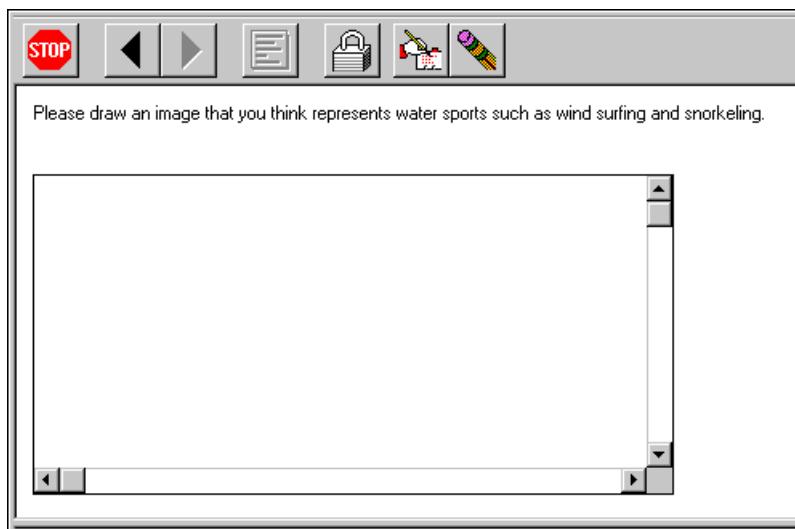
Quick Reference

To save an open-ended response as a bitmap picture, type:

```
label  ask 'question_text'
      resp coded scribble
```

If the interviewing PC has a mouse or light pen attachment, you can have the respondent write his/her response on the screen for storage as a bitmap file. You may find this method useful if your project is running in a shopping mall or similar location where outside noises may distort sound recordings.

When the interviewing program reaches a question that uses the *scribble* keyword, it displays a box in which respondents can ‘write’ their response on screen using an electronic light pen:



Like sound recordings, the respondent’s written response is held in a file named according to the serial number that the interviewing program allocates to each open-ended response. Since the files hold bitmap images, they are called qcserial.bmp. For example, if the response has been allocated the serial number 891, the file will be called qc891.bmp. The directory in which this file is created depends on the BitmapResultDirName setting in the interviewing computer’s quantime.ini file.

If you need to clear the contents of the picture box while you’re testing your script, click the right mouse button. (If you’re using a left-handed mouse, click the left mouse button.)

If you want to display a *scribbled* response later in the interview, use *mmpicture* as shown in the following example:

```
image ask 'Please draw an image that you think represents water
sports such as wind surfing and snorkeling.'
resp coded scribble
display 'Let me show you what you drew.' mmpicture='[+image+']'
```

6.6 Displaying images in Quancept Web



Quick Reference

To display an image as part of a question or response text, type:

```

```

inside the question or response text at the point you want the image to appear. By default, *pathname* is treated as relative to the location of qcweb.dll.

In the following example, the name of the image file is entered as a simple filename, so Quancept will look for the file in the directory containing qcweb.dll (usually \InetPubs\scripts).

```
activ ask 'Which country do you associate with this activity?
'
resp sp 'Australia' / 'USA' / 'South Africa' other dk
```

It is not a good idea to store image files in the scripts directory because respondents do not have read permission for this directory. Also, do not place image files in the projects directory, or in any subdirectory below it, because this will allow respondents to download the survey results by typing <http://server/qcweb/projname/results/projname.drs>. The best solution is to create a virtual root with read permissions to contain images for all projects. For example, if the virtual root is defined as

```
/intimages=c:\qcweb\qcserver\images
```

and you place the file surf.gif in this directory, you would write the previous example as:

```
activ ask 'Which country do you associate with this activity?
'
resp sp 'Australia' / 'USA' / 'South Africa' other dk
```

↖ Images for must be in GIF or JPEG (JPG) formats.

6.7 Displaying images in mrInterview



Quick Reference

To display an image as part of a question or response text, type:

```
<img src=/server/SPSSMR/ImageCache/ImageCache.aspx?project=projname&file=filename>
```

inside the question or response text at the point you want the image to appear. *server* is the name of the Web Service machine, *projname* is the name of the current project, and *filename* is the filename of the image file to be displayed.

mrInterview uses a mechanism known as the Image Cache for managing images and making them available to applications that need them. When you activate a project, the activation process copies any image files named in the script from the project's source directory into the master project directory, FMRoot\Master\project. The Image Cache contains a copy of all the image files used by all projects and makes them available to applications. If an image is updated in a project directory, the copy in the Image Cache is updated automatically.

To display an image during an interview you need to name the image cache application, the project, and the file to be displayed. For example:

```
activ ask 'Which country do you associate with this activity?  
&lt;img src=/zeus/SPSSMR/ImageCache/ImageCache.aspx?  
project=sports&file=surf.gif&gt;'  
resp sp 'Australia' / 'USA' / 'South Africa' other dk
```

If you have a number of images to display, you can place the image cache reference and the project name in a variable and then substitute that variable in the tag, as shown below.

```
set images = '/zeus/SPSSMR/ImageCache/ImageCache.aspx?project=sports'  
activ ask 'Which country do you associate with this activity?  
&lt;img src=[+images+]&file=surf.gif&gt;'  
resp sp 'Australia' / 'USA' / 'South Africa' other dk
```

Images for must be in GIF or JPEG (JPG) formats.

6.8 QTS Record and Play



QTS Record and Play is the name of the module in the Quancept Telephony System that allows you to play sound files to respondents during an interview and to record open-ended responses. The notes in the following sections describe the statements needed in a script to provide this functionality. For further details see your SMS and QTS documentation.

Playing sound files



Quick Reference

To play a sound file as part of a question, type:

```
qname ask '#audio=playback[/start=value1][/stop=value2], sound_filename; qtext'  
resp resp_type resp_list
```

The default is /start=any/stop=any which prompts the interviewer to press any key to play or stop the sound file. This option must be entered in lower case and may not contain spaces.

- ❖ The # character must be the first character in the question text otherwise the sound file will not be played.

When the audio instruction is reached during an interview, the interviewer sees the question displayed in the usual way.

The optional /start= parameter lets you determine when the sound file will be played. Your company may have found, for instance, that it is better to wait until the interviewer has asked the question before playing the recording, in which case you will want to have the interviewer press a key to start the playback. The optional /stop= parameter lets you specify a key that the interviewer can press to stop the playback before the end of the sound file. Possible values for /start and /stop are as follows:

any	Play/stop the sound file when the interviewer presses any key. This is the default.
return	Play/stop the sound file when the interviewer presses Return.
none	Play/stop the sound file as soon as the question is displayed.
<i>function key</i>	Play/stop the sound file when the interviewer presses the designated function key.
<i>cursor key</i>	Play/stop the sound file when the interviewer presses the designated cursor key. Valid key names are up, down, left, right, home, end, ins, del and esc.
" <i>character</i> "	Play/stop the sound file when the interviewer presses <i>character</i> .

- ❖ Not all function and cursor keys are available on all terminal types.

When the question is displayed, qtip also displays a message above the Response: prompt telling the interviewer which key to press to start recording. For example, if the script specifies /start=any, the message will say 'Hit any key to start recording'.

Both the interviewer and the respondent hear the sound file, and the interviewer also sees the message ‘>>AUDIO PLAYBACK<<’ displayed just above the Response: prompt on his/her screen, followed by a note of what key to press to stop the playback. The interviewer can talk to the respondent while the sound file is playing.

In the following example, the interviewer is reminded to read out the question before pressing Return to play the sound file. The interviewer presses Return to start playback and may press function key F1 if he/she wants to stop the playback before the end of the sound file.

```
Qm1    ask '#audio=playback/start=return/stop=f1, [+fmtune+];
        Can you tell me which film this music came from?
        @[Press Return AFTER reading question]
        @[interviewer: FILM NAME IS '+fmname+']@
            resp sp 'Correct answer'/ 'Incorrect' dk
```

This example uses two variables, fmtune to replace the name of the sound file, and fmname to store the film name. The variables were declared and set earlier in the script. If you were to replace the variables with specific names the script might look like this:

```
Qm1    ask '#audio=playback/start=return/stop=f1,starwars;
        Can you tell me which film this music came from?
        @[Press Return AFTER reading question]
        @[interviewer: FILM NAME IS STAR WARS]@
            resp sp 'Correct answer'/ 'Incorrect' dk
```

Audio files are played back at a fixed sound level. If interviewers or respondents find that the playback is too loud or too quiet your system administrator can change it.

Recording open ends



Quick Reference

To record a response and, optionally, the question during an interview, type:

```
qname    ask '#audio=record[/start=value1][/stop=value2] [,userID]; question_text'
        resp coded
```

The default is /start=any/stop=any which prompts the interviewer to press any key to start or stop the recording. This option must be entered in lower case and may not contain spaces.

-
- ❖ The # character must be the first character in the question text otherwise nothing will be recorded.
-

When the audio instruction is reached during an interview, the interviewer sees the question displayed in the usual way.

The optional /start= and /stop= parameters let you specify how the interviewer will start and stop recording. Valid keys are the same as for starting and stopping playback. The default is to prompt the interviewer to press any key to start the recording. The interviewer sees the words '>>AUDIO RECORDING<<' just above the Response: prompt, followed by a note of which key to press to stop recording.

If /stop is not specified, recording continues until the interviewer presses any key. This stops the recording but does not move to the next question. Instead, the question remains open and the interviewer may enter additional response text manually, pressing Return twice to terminate the text in the usual way. The response written to the .tex file consists of the name of the wav file followed by the text that the interviewer typed.

qtip does not normally allow interviewers to leave an open-end without either typing in a response text or requesting a text serial number. Interviewers working on scripts that record open-ends should therefore have the environment variable QCNULL set to 1 so that qtip will accept a null response when open-ends are recorded.

In the following example, both the question and the answer will be recorded. The interviewer is reminded to press Return before reading out the question — otherwise only the respondent's answer would be recorded. Recording stops when the interviewer presses function key F1.

```
Qm3    ask '#audio=record/start=return/stop=f1;  
      [Press Return BEFORE reading question]  
      @Can you tell me why you didn't go to see that film?  
      @[interviewer: PROBE FULLY]@'  
          resp coded
```

-
- ❖ Tests suggest that by recording the question, you reduce the chance of interviewer error and ensure that you capture the respondent's complete answer.
-

When recording responses, qtip creates the following files:

/hostname/project/qctextid.wav
for open-ended and other specify responses, where *textid* is the response serial number

/hostname/project/projname_respserno_varname_iter.wav
for other questions, where *respserno* is the respondent serial number, *varname* is the question name and *iter* is the iteration number

In both cases, *hostname* is the name of the telephony computer.

Recording subsurveys

Quick Reference

To record an entire subsurvey, type:

```
qname ask '#SUB=scriptname #audio=record/start=value/stop=none'
      resp num 1
```

Vaild options for starting recording are */start=any* (the default), */start=return* and */start=none*.

☞ *#SUB* must come before *#audio*.

The recording is stored in a file called *projname_respnun_qname_iteration.wav*, but because the question is not open-ended you cannot listen to the recording with Verbastat. You can, however, listen to it using standard Windows multimedia tools.

Subsurvey scripts can contain open-ended questions with their own */stop* and */start* instructions. Note, however, that this requires two DSPs, one for the subsurvey recording and one for the question, so you need to use this facility with care.

☞ See section 25.2, Running one script from inside another, for information about subsurveys.

6.9 Sample script



In order to run this script successfully, you will need to create files of the appropriate type with the following names: *snorkel.bmp*, *scuba.bmp*, *surf.bmp*, *windsurf.bmp*, *windsurf.avi*, and *windsurf.wav*. The pictures or sounds in the files are not important as long as the filenames are correct.

```
comment Sample script for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 6, Writing multimedia scripts (CAPI only)

comment Display sized pictures next to response choices
activ   ask 'Which of these water sports have you tried in the
last five years?'
      resp mp 'Snorkeling' mmpicture='snorkel.bmp' /
             'Scuba diving' mmpicture='scuba.bmp' /
             'Surfing' mmpicture='surf.bmp' /
             'Wind surfing' mmpicture='windsurf.bmp' /
             'None of these ^s' go goodbye

comment Display video file
windsurf ask 'Could you tell me which country you associate with this
activity?' mmvideo='windsurf.avi'
      resp coded

comment Play audio file
radioad ask 'I''m going to play you a radio advertisement about
wind surfing in Australia. Please tell me what you think about it.'
mmaudio='windsurf.wav'
      resp coded

comment Collect response in the form of an audio recording
comment Limit time to 75 seconds
advert   ask 'If you could make your own advertisement for
wind surfing, what would it be like?' maxrecord=75
      resp coded audio

comment Collect response in the form of a drawing
image   ask 'Please draw an image that you think represents water
sports such as wind surfing and snorkeling.'
      resp coded scribble
      display 'Let me show you what you drew.' mmpicture='[+image+]'
      pause

goodbye display 'Thank you for time today. Goodbye.'
end
```

7 Defined response lists

The usual way of writing response lists is to type them in full after the question to which they refer. If a response list refers to a number of questions which appear consecutively in the script, you may enter the response list with the first question and then freeze it for the other questions which use it.

Sometimes the questions which use the same response list are scattered throughout the script. One approach is simply to type the response list in full for each question (or to use commands within your system editor to copy the response list from one place to another). A better method is to use a defined response list. This allows you to define the response list once, usually at the start of the script, and to give it a name. Then, whenever you want to use the responses in the list, you just type the name of the list on the *resp* statement for the current question. This has the added advantage of reducing the amount of space required to store your script both as a text file and as a compiled interviewing program.

Defined lists are also useful when you have a master list of responses which you want to break down into sublists or manipulate in some other way during the script. A good example here is questions for aided and unaided awareness, where you want the list for the aided awareness question to contain only those responses not mentioned unaided. The interviewing program usually codes responses in a list sequentially from code 1, starting with the first response in the list. By using defined lists, you can have the responses in the sublist (for example, the responses for the aided awareness question) coded according to their position in the complete list of responses.

A third advantage of defined lists is when you have several scripts and you want the data from all scripts to be tabulated together. By using lists and assigning codes according to a master list, you can ensure that the data from one script is compatible with that from all other scripts in the project.

Keywords associated with defined lists are:

define	assign a name to a list of responses
list	response type used with defined lists

7.1 Defining response lists

Quick Reference

To set up a defined response list of single-punched or multipunched texts, type:

```
list_name define 'resp1' / 'resp2' / 'resp3'
```

Defined response lists contain responses which, when used as the response list to a question, may be single punched or multipunched. The response texts are therefore enclosed in single quotes and separated by slashes.

The list name, like a label, may be up to eight characters long and must start with a letter. So that your script is easy to understand, try to make the list name relevant to the contents of the list.

Here is a defined list containing various flavors of yogurt:

```
yogmain define 'apricot and nectarine' / 'banana' / 'black cherry' /  
    'strawberry' / 'raspberry' / 'melon and passion fruit' /  
    'forest fruits' / 'pineapple' / 'apricot' / 'orange'
```

We have typed several responses on each line to save space, but you may prefer to type one response per line when you write your scripts.

The name of this list tells us that this is the main list of flavors. We call this the **master list**. A script may have any number of master lists.

You can define **sublists** associated with a master list by taking certain responses from the master list and defining them as lists with their own names. If different flavors of yogurt are available in different brands, you can define one sublist for each brand:

```
alpine define 'banana' / 'black cherry' / 'forest fruits' /  
    'melon and passion fruit' / 'pineapple' / 'apricot'  
kentish define 'apricot' / 'orange' / 'raspberry' / 'pineapple' /  
    'apricot and nectarine'  
dfresh define 'black cherry' / 'strawberry' / 'raspberry' / 'apricot'
```

The parser and the interviewing program do not distinguish between upper and lower case in response texts, so it does not matter if, for example, you type the responses in the main list all in upper case and those in the sublists all in lower case. When a defined response list is displayed as the response list to a question, the interviewing program displays the responses in the case they are in the master list and codes them according to their position in that list.

If texts contain spaces, the response in the master list must have the same number of spaces in the same places as the corresponding text in the sublist if the responses are to be matched. If the spacing is different, the response in the sublist will be omitted when the sublist is displayed.

- ⌚ When the order of responses in a sublist differs from those in the master list, as in the kentish list, Quancept Web displays responses in the order in which they appear in the master list.
- ☎ In Quancept CATI, Quancept CAPI and mrInterview, you need not type the responses in a sublist in the same order as they appear in the master list. In the kentish list, the order of responses is not the same as in the yogmain list. However, the order of the individual responses in the display is taken from the sublist. This provides you with great flexibility in the way you deal with lists. In long-running or panel studies, you may want to alter the order of responses in sublists so that new or key items always appear at the top of the list. Another way of ordering lists might be by price, so that the most expensive items appear at the top of the list. As long as you do not alter the order of the master list, your new data will always be compatible with what has already been collected.

You may define lists anywhere in your script, but we recommend defining them at the start of the script so that they are easy to find if you need to refer to them while writing or testing the script.

7.2 Column allocation with defined lists

When the parser finds a defined list, it does not allocate any columns in the Quancept CATI, Quancept CAPI or Quancept Web data map or in the mrInterview database at this point. This happens when the parser finds the list named on a *resp* statement. Instead, it stores the response texts and the name of the list for future reference.

-  When you use the list shown below as the response list to a question, the parser allocates two columns to that question because the list contains ten codes. Responses will be coded according to their position in this list, so, for example, banana yogurt will be a code 2 in the first column and orange yogurt will be a code 0 in the second column.

```
yogmain define 'apricot and nectarine' / 'banana' /
    'black cherry' / 'strawberry' / 'raspberry' /
    'melon and passion fruit' / 'forest fruits' /
    'pineapple' / 'apricot' / 'orange'
```

-  The mrInterview parser creates a single categoric variable to store however many responses are selected. Each response is represented by a unique code that is assigned to it when the questionnaire definition file is created.

7.3 Using a master list as a response list

Quick Reference

To use a master list as the response list to a question, type a *resp sp* or *resp mp* statement with the name of the list as the response list:

```
resp sp master_list
resp mp master_list
```

To use a master list as a response list to a question, type the question using an *ask* statement in the normal way and follow it with a *resp* statement defining whether the question requires a single-punched or multipunched response and then the name of the defined list to use. For example:

```
yogbrd define 'Alpine' / 'Helen Bradley''s' / 'Kentish Farm' /
    'Finborough' / 'Alsace' / 'Dairy Fresh'
unaided ask 'When I mention the word yogurt, which brand names do you
think of?'
    resp mp yogbrd
usual ask 'And which brand do you usually buy?'
    resp sp yogbrd
```

Both questions use the same response list, but one allows multipunching while the other requires a single punch. Responses will be displayed exactly as they are written in the defined list, and will be listed in the order in which they appear in the list definition.



Since the defined list contains six responses, the parser will allocate one column to each question, and responses will be coded according to their position in that list. So, Alpine yogurt is code 1 and Dairy Fresh yogurt is code 6.

-
- ❖ You may use different master lists with different questions. If you do this, remember that, as far as Quancept and mrInterview are concerned, there is no relationship between the responses in one master list and those in another, even if some responses are common to two or more of those lists.

This is especially important if you are using assignment to merge the answers to several questions into one variable; for example, to build up a list of brands or products mentioned. If you are considering doing this and the questions whose answers you want to merge use defined lists in the form shown in this section, we strongly advise you to read the list of points at the end of 'Question labels' in section 12.4, Assigning responses to questions, before writing your script.

7.4 Null, dk and ref with defined lists

Quick Reference

To allow *null*, *dk* and *ref* as answers to a question which uses a defined list, type those keywords at the end of the *resp* statement for that question:

```
resp sp list_name null dk ref  
resp mp list_name null dk ref
```

Defined lists may not contain the keywords *null*, *dk* or *ref*. If you want to allow these responses, you must append the keywords to the end of the *resp* statement in the normal way. For example:

```
yogbrd    define 'Alpine' / 'Helen Bradley''s' / 'Kentish Farm' /  
                  'Finborough' / 'Alsace' / 'Dairy Fresh'  
unaided   ask 'When I mention the word yogurt, which brand names do you  
think of?'  
          resp mp yogbrd dk  
usual     ask 'And which brand do you usually buy?'  
          resp sp yogbrd null ref
```

7.5 Specified other with defined lists

Quick Reference

To include specified other in a response list that comes from a defined list, include *other* on the *resp* statement:

```
resp sp list_name other
resp mp list_name other
```

Defined response lists may contain specified other. With Quancept CATI, Quancept CAPI and Quancept Web, the total number of columns allocated for coding the whole response list depends on where exactly you put specified other. There are several possibilities:

- On the defined list and the *resp* statement (CATI and Web only)
- On the defined list only (CATI and Web only)
- On the *resp* statement only (CATI, CAPI, Web and mrInterview)

With mrInterview, the position of specified other does not affect the way in which data is collected.

Specified other on the defined list and *resp* statement



If you have a response list containing specified other and also have specified other on the defined list, then the parser will allocate one code for each quoted response, one code for specified other and three extra columns for coding the answers to specified other. For example:

```
yogbrd define 'Alpine' / 'Helen Bradley''s' / 'Kentish Farm' /
    'Finborough' / 'Alsace' / 'Dairy Fresh' other
unaided ask 'When I mention the word yogurt, which brand names do you
think of?'
    resp mp yogbrd other
```

generates one column of seven codes — six for the listed brand names and one for specified other — plus three extra columns. This is the same as writing:

```
unaided ask 'When I mention the word yogurt, which brand names do you
think of?'
    resp mp 'Alpine' / 'Helen Bradley''s' / 'Kentish Farm' /
        'Finborough' / 'Alsace' / 'Dairy Fresh' other
```

If the script contains a *mapothzero* or *datamap othzero* statement, the interviewing program will use code 0 for specified other rather than code 7.

-
- ☞ For more details on *mapothzero* and *datamap othzero*, see ‘Data allocation of specified other’ in chapter 4, ‘Special responses’.
-

Specified other on the defined list only



If the defined list includes specified other but the *resp* statement does not, then the interviewing program will display only the quoted responses in that list. For example:

```
yogbrd define 'Alpine' / 'Helen Bradley''s' / 'Kentish Farm' /
               'Finborough' / 'Alsace' / 'Dairy Fresh' other
unaided ask 'When I mention the word yogurt, which brand names do you
think of?'
resp mp yogbrd
```

allocates one column and six codes only. It is the same as:

```
unaided ask 'When I mention the word yogurt, which brand names do you
think of?'
resp mp 'Alpine' / 'Helen Bradley''s' / 'Kentish Farm' /
               'Finborough' / 'Alsace' / 'Dairy Fresh'
```

If the presence of specified other requires that a new column be started in the data file (that is, it is the 10th, 20th, and so on, code in the list and the script does not contain a *mapothzero* or *datamap othzero* statement), then the map file will allocate an extra column to the response, even though it will always be blank. If the script contains *mapothzero* or *datamap othzero*, code 0 in the first column would normally be used for specified other and no extra blank column will be generated.

Specified other on the *resp* statement only

This is the same as having specified other on both the defined list and the *resp* statement, as described earlier. Thus,

```
yogbrd define 'Alpine' / 'Helen Bradley''s' / 'Kentish Farm' /
               'Finborough' / 'Alsace' / 'Dairy Fresh'
unaided ask 'When I mention the word yogurt, which brand names do you
think of?'
resp mp yogbrd other
```

has the same effect as writing:

```
unaided ask 'When I mention the word yogurt, which brand names do you
think of?'
resp mp 'Alpine' / 'Helen Bradley''s' / 'Kentish Farm' /
               'Finborough' / 'Alsace' / 'Dairy Fresh' other
```

7.6 Other keywords with defined lists

If a defined list that potentially allows multipunching contains a response that can only be selected by itself, you may flag this response with the *sp* (or *^s*) flag, as you would in an ordinary response list. For example:

```
yogmain define 'apricot and nectarine' / 'banana' / 'black cherry' /
    'strawberry' / 'raspberry' / 'melon and passion fruit' /
    'forest fruits' / 'pineapple' / 'apricot' / 'orange' /
    'none of these' sp
```

You also may use any of the following keywords in a *resp* statement which uses a defined list:

atoz	mmpicture	promptoth	rottranfix
echo	mp <i>n</i>	ran	rotransub
freeze	nodata	rot	

Examples are:

```
yogmain define 'apricot and nectarine' / 'banana' / 'black cherry' /
    'strawberry' / 'raspberry' / 'melon and passion fruit' /
    'forest fruits' / 'pineapple' / 'apricot' / 'orange'/
    'none of these ^s'
tried ask 'Which flavors of yogurt have you tried?'
resp mp atoz yogmain null
best ask 'Which three flavors do you like best?'
resp mp 3 yogmain
```

Responses in defined lists can have unique IDs assigned to them in the same way that you assign identifiers to responses in ordinary lists.

When a response list uses sublists, any differences in the use of *promptoth*, *rotransub*, *rottranfix* or *sp* (^s) between a response in the sublist and the same response in the master list are resolved as follows:

- If the response list is specified as *sublist in master*, responses behave according to their flags in *sublist*.
- If the response list is specified as *not sublist in master*, responses behave according to their flags in *master*.

☞ You will find full documentation of all keywords except *mmpicture* in chapter 5, ‘Keywords in response lists’.
See section 6.1, Displaying pictures in Quancept CAPI, for information on *mmpicture*.
For information about sublists see section 7.8, Using sublists as response lists, and section 7.9, Using responses not in a sublist.

7.7 Reserving space for extending lists



Quick Reference

To reserve space for adding extra responses to a defined list, follow *define* with the maximum number of responses the list will contain, enclosed in parentheses:

list_name define (n) response texts

If a defined list includes specified other, you may notice that one or two responses crop up frequently in the answers given with specified other. You can save time at the coding stage of a project by adding these responses to the defined list so that they can be selected as precodes in future interviews. To do this without disrupting the data map, you need to have sufficient columns available to take these additional answers at the point at which the question occurs. For example, if a defined list contains 15 responses, a question which uses that list will be allocated two columns. Two columns can store 19 codes, so you have room for another four responses in the list without affecting the data map.

Relying on response lists not filling the columns allocated to them is risky and may mean that you do not always have sufficient columns available for expansion. The way to do this is to enter the maximum number of responses you expect to have at the same time as you define the list. This causes the parser to allocate sufficient columns for that number of responses even if the list contains fewer than that at the time you parse the script.

For example:

```
ices    define (19) 'Ice cream' / 'Frozen chocolate bars' / 'Mousse' /  
           'Frozen cakes/puddings' / 'Ice lollies' other
```

requests space for 19 codes (two columns) even though there are only six responses in the list.

If you define more responses in a list than there are columns to store them (that is, more than 19 responses in the response list shown above), the parser will issue an error message.

☞ If you define your lists in this way, take care if the lists contain specified other. The interviewing program always codes this using the code number immediately after that of the last quoted response in the list. If you insert extra responses in the list, the code assigned to specified other will change.

If you think you may be altering the number of responses in any defined lists, you are advised to use *mapothzero* or *datamap othzero* at the start of the script to have the interviewing program code specified other as code 0 in the first column of a question rather than according to its position in the response list. If you use this facility, adding responses to a defined list once interviewing has begun will have no effect on the integrity of your data.

☞ For further details about *mapothzero* and *datamap othzero*, see ‘Data allocation of specified other’ in chapter 4, ‘Special responses’.

7.8 Using sublists as response lists

Quick Reference

To use a sublist as a response list, type:

```
resp sp sublist in master_list
resp mp sublist in master_list
```

You can use sublists as response lists, too. The general format of a *resp* statement is shown above. This tells the parser and the interviewing program to code responses chosen from the sublist according to their positions in the master list and, in Quancept CATI, Quancept CAPI and mrInterview, to display responses according to their position in the sublist.

Do not enclose the list names in single quotes as you would do with normal responses, otherwise, the interviewing program will take them as responses rather than as the names of defined response lists.

As an example, the statements:

```
yogmain define 'apricot and nectarine' / 'banana' /
    'black cherry' / 'strawberry' / 'raspberry' /
    'melon and passion fruit' / 'forest fruits' /
    'pineapple' / 'apricot' / 'orange' /
    'none of these' sp
dfresh define 'black cherry' / 'strawberry' / 'raspberry' / 'apricot'
freqdf ask 'Which flavor do you buy most frequently from the Dairy
Fresh range of yogurts?'
    resp sp dfresh in yogmain
```

will offer the flavors in the sublist dfresh as possible responses to the question freqdf. If the response texts are written in different cases in the sublist from the way they appear in the master list, the interviewing program will display them in the case in which they appear in the master list.

- ☎ If the responses appear in a different order in the sublist and the master list, Quancept CATI and Quancept CAPI display the responses on the interviewer's screen in the order in which they are defined in the sublist. If the respondent buys raspberry yogurt most frequently, this will appear as a code 5 in the data file because raspberry is the fifth response in the master list. It does not matter that it is the third response in the sublist that is displayed.
- 🌐 In Quancept Web, responses in a sublist are always displayed and coded in the order they appear in the master list.



Quancept CATI, Quancept CAPI and Quancept Web use ‘pattern matching’ to compare the response text in the sublist against the text in the master list. Items in the sublist must be spelled in exactly the same way as in the master list, although case can be different, as mentioned earlier. For example, orange in the sublist will match Orange in the master list (and will be displayed as Orange) but will not match oranges.

If you test your script and find that the interviewing program is still not displaying the correct responses in a sublist, check that you have not added a space between the single quotes and the response text — although you know that ‘orange’ means the same as ‘orange’, the interviewing program does not and will therefore treat them as different responses.

❖ Responses in the sublist should match only one item in the master list.

All responses within the sublist itself should be unique — for example, you should not use ‘orange’ and ‘Orange’ as two separate items in the same sublist. When you parse a script containing multiple occurrences of the same item in a sublist, the parser will warn you that you may lose data. This is because the interviewing program will match all duplicates within the sublist to the first occurrence of the item in the master list and code them accordingly. This is particularly difficult to detect if you use blank responses. For this reason, we strongly recommend that you never use multiple blank responses in a sublist.



The parser compares the number of responses in the sublist with the number of responses in the master list and issues the following warning if there are more responses in the sublist:

WARNING: The sublist x has more entries than the master list y

This often happens if you have added the *other* keyword to the *resp* line, so you should choose to continue the compilation when prompted to do so.



mrInterview removes any spaces, tabs, carriage return or line feed characters from the start and end of responses. It then displays responses in the order they appear in the sublist and codes them according to their position in the master list. If you want responses displayed and coded according their positions in the master list, use regedit to assign a non-zero value to the registry key HKEY_LOCAL_MACHINE\Software\SPSS\mrInterview\1.0\SifLimitOrderMaster. Note that this must be done with care, not only on your own machine while you are testing the script, but also on the machine on which the live project will run.

7.9 Using responses not in a sublist

Quick Reference

To display responses present in the master list but not in a named sublist, precede the sublist name on the *resp* statement with the word **not**:

```
resp sp not sublist in master_list
resp mp not sublist in master_list
```

For example, the statement:

```
othflav ask 'And what flavors do you buy in other brands?'
            resp mp not dfresh in yogmain
```

selects all responses present in the master list but not in the list called dfresh.

Using the lists defined at the start of this chapter, the responses displayed by this statement would be apricot and nectarine, banana, melon and passion fruit, forest fruits, pineapple, and orange.

7.10 Using question names with lists

Quick Reference

To use answers to a question as the response list to a second question, type:

```
resp sp qname in master_list
resp mp qname in master_list
```

To use answers not chosen, type:

```
resp sp not qname in master_list
resp mp not qname in master_list
```

In both statements, *qname* is the name of a question whose response list is a defined response list. That list must be either the master list or a sublist of it.

Once a question has been answered, the answers given are stored in a variable with the name of that question (for example, brand might store the brand of yogurt that the respondent usually buys). When the question name stores responses chosen from a defined list, you may use that question name as part of a response list anywhere you would normally use a sublist name. This is especially useful with aided and unaided awareness questions where you want to ask a series of questions, each time reducing the response list so that it excludes the items already mentioned.

For example:

```
yogbrd  define 'Alpine' / 'Helen Bradley''s' / 'Kentish Farm' /  
          'Finborough' / 'Alsace' / 'Dairy Fresh'  
unaided  ask 'When I mention the word yogurt, which brand names do you  
          think of?'  
          resp mp atoz yogbrd other  
buynow   ask 'And which of those brands do you currently buy?'  
          resp mp atoz unaided in yogbrd freeze  
everbuy  ask 'And are there any which you have bought previously but do  
          not buy any more?'  
aided    ask 'Have you heard of any of the following brand names?@  
          INTERVIEWER: Read each name in turn. Code only those for which the  
          answer is yes.'  
          resp mp atoz not unaided in yogbrd null
```

If the respondent names Helen Bradley's, Kentish Farm and Alsace brands at the unaided question, the response lists for the buynow and everbuy questions will contain just those responses. Notice that the response list for these questions is frozen on buynow.

The aided question has a different list so it unfreezes the previous response list. The list for this question will contain only those responses which were not mentioned at the unaided question — namely, Alpine, Finborough and Dairy Fresh.

Notice how we have used *atoz* to have the responses displayed in alphabetical order.

- ☎ If the interviewer also has the environment variable QCRETAIN set to 1, the responses in the lists will always be displayed with the same response codes rather than being renumbered from 1 each time. Responses are numbered according to their position in the main list named on the *resp* statement — in this case, 'yogbrd'. The first screen the interviewer sees will be:

```
1/UNAIDED  
When I mention the word yogurt, which brand names do you think of?  
  1 Alpine  
  5 Alsace  
  6 Dairy Fresh  
  4 Finborough  
  2 Helen Bradley's  
  3 Kentish Farm  
  7 OTHER  
  
Reply may be NULL  
.. Reply may be a combination of the above  
  
Response:
```

- ☎ The screen for the aided question several screens later will be:

```
1/AIDED
Have you heard of any of the following brand names?
INTERVIEWER: READ EACH NAME IN TURN. CODE ONLY THOSE FOR WHICH
THE ANSWER IS YES.
1 Alpine
4 Finborough
3 Kentish Farm

Reply may be NULL
.. Reply may be a combination of the above

Response:
```

If you compare this with the previous screen, you will see that the responses are still shown in alphabetical order and that the responses which were unnamed at the unaided awareness question still have the same codes at the aided awareness question as they did before. This is a function of the interviewer setting the environment variable QCRETAIN to 1. If QCRETAIN is unset or is set to any value other than 1, the response lists will be numbered sequentially from 1 each time.

Using *atoz* and QCRETAIN in this way are purely cosmetic features for the benefit of the interviewer. They have no bearing at all on the way the data is coded into the data file. The responses in our example will be coded in the data according to their position in the yogbrd list as it was originally defined.

Specified other with not question in list'

When you have a response list of the form 'not question in list' and the response list to the original question contains specified other, whether or not specified other is part of the current response list depends on whether *other* also appears as part of that list. For example:

```
yogbrd define 'Alpine' / 'Helen Bradley''s' / 'Kentish Farm' /
                 'Finborough' / 'Alsace' / 'Dairy Fresh' other
usbuy   ask 'Which brand do you usually buy?'
        resp mp yogbrd other
othbuy  ask 'Are there any other brands that you also buy?'
        resp mp not usbuy in yogbrd other
```

At both questions, the respondent may choose any of the named brands or he/she may choose specified other to name a different brand. If *other* was not included in the second response list, the respondent would be forced to choose one of the listed responses.

7.11 Finding the relevant sublist

Quick Reference

If the sublist to be used may vary between respondents, use **resp list** to select the sublist for the current interview:

```
resp list list_name1 ['print_name1'] / list_name2 ['print_name2']
```

If the list names are not suitable for use as texts in a response list, you may follow each list name with a text to be displayed in the response list.

In the examples so far, we have known which sublist to use. This will not always be the case; for example, you may not know which sublist of flavors to display until you have asked which brand the respondent buys. The way to deal with this problem is to write a question that has as its responses the names of the sublists that may be used. This is done using a *resp* statement of the form:

```
resp list sublist1 / sublist2 / sublist3 / ...
```

where *sublist1* to *sublist3* are the names of the sublists. Our first step is to find out which brand the respondent usually buys, so we write:

```
comment Assume that a list of flavors is defined for each brand
yogbrd define 'Alpine' / 'Helen Bradley''s' / 'Kentish Farm' /
            'Finborough' / 'Alsace' / 'Dairy Fresh'
yogmain define 'apricot and nectarine' / 'banana' /
            'black cherry' / 'strawberry' / 'raspberry' /
            'melon and passion fruit' / 'forest fruits' /
            'pineapple' / 'apricot' / 'orange' /
            'none of these' sp
alpine define 'banana' / 'black cherry' / 'forest fruits' /
            'melon and passion fruit' / 'pineapple' / 'apricot'
usual ask 'Which brand of yogurt do you usually buy?'
      resp list alpine / bradley / kentish / finb / alsace / dfresh
```

The interviewing program displays each list name as a possible response and waits for the interviewer to code the appropriate brand. Since the answer to this question will be used as part of a 'list in list' response list, the interviewing program will allow only one brand to be selected.

In this example, the list names are sometimes abbreviations of the brand names. This is acceptable when writing a script, but it is not very helpful to interviewers who need to know the full brand names. Therefore, the parser allows you to follow each list name with a text which will appear in place of the list name when the response list is displayed during an interview. Each text must be enclosed in single quotes, as for any other text response:

```
usual ask 'Which brand of yogurt do you usually buy?'
resp list alpine / bradley 'HELEN BRADLEY''S' /
kentish 'KENTISH FARM' / finb 'FINBOROUGH' /
alsace / dfresh 'DAIRY FRESH'
```

- ☎ So that the list looks consistent when displayed for the interviewer, we have entered the display texts in upper case to match the way the interviewing program displays list names in response lists. This is a cosmetic change, since some interviewers may find it confusing to see some responses in upper case and others in lower case.

Now that the name of the brand bought is stored in 'usual', you may use this question name as the basis of a response list which displays the flavors available in that brand. For example:

```
prefer ask 'Which flavor do you like best?'
resp sp usual in yogmain
```

When the interviewing program reads this response list, it will substitute for 'usual' the response that it represents. There are six permutations for the response list:

resp sp alpine in yogmain	resp sp bradley in yogmain
resp sp kentish in yogmain	resp sp finb in yogmain
resp sp alsace in yogmain	resp sp dfresh in yogmain

If the respondent usually buys Kentish Farm yogurt, the response list to this question will contain all the flavors defined as being part of the kentish sublist. Having found the appropriate sublist, we can write the rest of the script in the normal manner, referring to 'usual' each time we need to refer to the brand bought.

The special responses *null*, *dk* and *ref* are not allowed with the keyword *list* because they would cause the rest of the script to be invalid if they were chosen (for example, you might have a statement which becomes 'resp sp null in master'). Instead, include the texts 'Don't know', 'No answer' and 'Refused' as ordinary responses in the defined list, flagged with *sp* or *^s* to ensure that they are always single punched.

other, *dummyother* and *promptoth* are not valid with *resp list* because of the likelihood that the resultant list name would be invalid.

- ☎ If you want to vary the order that list names appear on the screen, you may use *rot* or *ran* to display them in rotated or randomized order. You cannot use *atoz* to sort the names in alphabetical order, nor can you control the order of the list names using *rotranfix* or *rotransub*.

Displaying responses not chosen from a variable sublist

We explained earlier that a response list of the form:

```
resp mp not usbuy in yogmain
```

displays all the responses which were not mentioned at the question usbuy. To do this, the interviewing program compares the responses stored in usbuy with those in the master list, yogmain, and displays any that are present in the master list only.

When the response list to the first question (here, usbuy) is defined with resp list or resp mp you cannot write the response lists as simply as this. The reason is that the value of the first question is the name of a list rather than a quoted response text. The example below illustrates what you should do in this situation.

```
usual    ask 'Which brand of yogurt do you usually buy?'
        resp list alpine / bradley 'HELEN BRADLEY''S' /
                      kentish 'KENTISH FARM' / finb 'FINBOROUGH' /
                      alsace / dfresh 'DAIRY FRESH'
buy      ask 'Which flavors do you usually buy in this brand?'
        resp mp usual in yogmain freeze
comment  Build list of responses not chosen from sublist
nomore   dummyask 'Flavors no longer bought'
        resp mp yogmain
11       for flav = usual in yogmain
        set flavnum = iteration
        set given = bit(buy/flavnum)
        if (.not. given) {
            set nomore = flavnum
        }
next
tried    ask 'Are there any flavors in this brand that you''ve bought
        previously but which you no longer buy?'
        resp mp nomore in yogmain null dk
```

The key question in this example is ‘tried’. What we would like to write as its response list is:

```
resp mp not buy in yogmain null dk
```

because we want all responses which are not chosen at the buy question. However, that question has a variable sublist as its response list; that is, the sublist used depends on which brand the respondent usually buys. So, the statement shown here displays all responses not mentioned at ‘buy’ regardless of whether they were present in the sublist for the respondent’s usual brand.

To get the responses we want, we define a dummy question, nomore, and assign the chosen flavors to the dummy question manually via statements in the loop. The loop starts at the *for* statement and ends at the *next* statement. The *for* statement tells it to read the sublist chosen at the ‘usual’ question and compare it with the master list (this is basically the same as writing a response list).

For each response that is present in both lists, the interviewing program checks whether it was chosen at 'buy'. If not, we assign it as a response to the dummy question and return to the start of the loop to deal with the next response.

Once the interviewing program has tested the last response in the chosen sublist, the loop ends and the interviewing program uses the responses we have assigned to the dummy question as the response list for the 'tried' question.

An additional point to notice about this example is the way we have set the value of the dummy question to null before we enter the loop. This is in case the interviewer snaps back and changes the answer to 'buy' so that some responses which had been assigned to the dummy question are no longer valid.

☞ For further information on loops, see chapter 10, 'Repetitive questions', and chapter 11, 'Iteration and subscription'.

For further details on assigning values to variables and questions, see chapter 12, 'Assignment'.

7.12 Blank responses in defined lists

Quick Reference

If you have a defined list that you may need to expand but that contains certain responses that must appear at the end of the list, you can pad the list with blank responses:

```
list_name define 'resp1' / ' ' / ' ' / 'respn'
```

We explained earlier how you can reserve space in the Quancept CATI, Quancept CAPI or Quancept Web data map for adding extra responses to defined lists as the project progresses. We said that if you have a list of nine responses and you want to reserve space for 15 responses, you would enter the *define* keyword as `define(15)`.

Another way of allocating space for extra responses is to include some blank responses in the list which you can replace with response texts once interviewing has begun. Suppose, for example, you have a list containing a response 'None of these' which must always be the last response in the list. You suspect that the list may be incomplete, so you need to be able to add responses to the list before the None response. To do this, enter as many blank dummy responses as you think you will need at the appropriate points in the list. By a blank response, we mean a space enclosed in single quotes (' ').

☞ Multiple blank responses are not appropriate for sublists. See section 7.8, Using sublists as response lists, for more information.

In the example that follows, the listed brands are entered as codes 1 to 6 and 'None of these' as code 9; the blank responses reserve codes 6 to 8 for future use.

```
yogbrd define 'Alpine' / 'Helen Bradley''s' / 'Kentish Farm' /
    'Finborough' / 'Alsace' / 'Dairy Fresh' /
    ' ' / ' ' / 'None of these ^s'
```

During the interview, all nine codes will be displayed, but those that have been reserved for future use will have no response text after them. Thus, if the question is:

```
yogbrd define 'Alpine' / 'Helen Bradley''s' / 'Kentish Farm' /
    'Finborough' / 'Alsace' / 'Dairy Fresh' /
    ' ' / ' ' / 'None of these ^s'
unaided ask 'When I mention the word yogurt, which brand names do you
think of?'
resp mp yogbrd
```

the response list on the interviewer's screen will be:

```
1 Alpine
2 Helen Bradley's
3 Kentish Farm
4 Finborough
5 Alsace
9 Kentish Farm
7
8
9 None of these
```

-
- ¶ The space between the pair of quotes in blank responses is very important. If you forget it, any responses in the list after that blank response will be ignored. If in the previous example we typed one of the blank responses as two consecutive single quotes, the interviewing program would not display 'None of these' as part of the response list seen by the interviewer.
-

7.13 Sample script

```
comment Sample script for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 7, Defined response lists

comment CATI/CAPI/Web: Code specified other as 0 in first column of questions.
comment Ignored by mrInterview because it is not applicable.
    mapothzero

comment Define the lists. Note that the restriction on the size of the ices
comment list is not applicable to mrInterview and is therefore ignored.
yogbrd  define 'Alpine' / 'Helen Bradley''s' / 'Kentish Farm' /
          'Finborough' / 'Alsace' / 'Dairy Fresh'
yogmain define 'apricot and nectarine' / 'banana' / 'black cherry' /
          'strawberry' / 'raspberry' / 'melon and passion fruit' /
          'forest fruits' / 'pineapple' / 'apricot' / 'orange'
alpine   define 'banana' / 'black cherry' / 'forest fruits' /
          'melon and passion fruit' / 'pineapple' / 'apricot'
bradley  define 'apricot and nectarine' / 'melon and passion fruit' /
          'strawberry' / 'banana' / 'forest fruits'
kentish  define 'apricot' / 'orange' / 'raspberry' /
          'pineapple' / 'apricot and nectarine'
finb     define 'strawberry' / 'raspberry' / 'apricot' / 'pineapple'
alsace   define 'banana' / 'black cherry' / 'forest fruits' / 'apricot'
dfresh   define 'black cherry' / 'strawberry' / 'raspberry' / 'apricot'
ices     define (19) 'Ice cream' / 'Frozen chocolate bars' / 'Mousse' /
          'Frozen cakes/puddings' / 'Ice lollies'

comment Use master list only with specified other to force prompting
comment for other brand names, and null
unaided  ask 'When I mention the word yogurt, which brand names do you
think of?'
    resp mp atoz yogbrd other null

comment Only list responses not mentioned at unaided
aided    ask 'Have you heard of any of the following brand names?
@INTERVIEWER: READ EACH NAME IN TURN. CODE ONLY THOSE FOR WHICH THE
ANSWER IS YES.'
    resp mp atoz not unaided in yogbrd null

comment Now choose a sublist to work with. CATI displays list names in upper
comment case, so replacement names have been entered in upper case to match.
comment mrInterview uses lower case throughout.
usual    ask 'Which brand of yogurt do you usually buy?'
    resp list alpine / bradley 'HELEN BRADLEY''S' /
          kentish 'KENTISH FARM' / finb 'FINBOROUGH' /
          alsace / dfresh 'DAIRY FRESH'

comment Response lists contain flavors in the brand chosen at 'usual'
buy      ask 'Which flavors do you usually buy in this brand?'
    resp mp usual in yogmain
```

```
prefer    ask 'Which flavor do you like best?'
          resp sp buy in yogmain
why      ask 'Why is this?'
          resp coded
comment   Build list of responses not chosen from sublist
nomore   dummyask 'Flavors no longer bought'
          resp mp yogmain
          unset nomore
11       for flav = usual in yogmain
          set flavnnum = iteration
          set given = bit(buy/flavnnum)
          if (.not. given)
          {
              set nomore = flavnnum
          }
next
tried    ask 'Are there any flavors in this brand that you''ve bought
          previously but which you no longer buy?'
          resp mp nomore in yogmain null dk

comment Use an expandable list. Do some interviews and then add
comment some extra responses to the defined list to see what happens
cold     ask 'Which cold or frozen desserts do you buy most frequently?'
          resp sp ices other null ref
othcold  ask 'And are there any others that you buy on a regular basis?'
          resp mp not cold in ices other null ref

thank    display 'Thank you for your co-operation'
end
```

8 Displaying information on the screen

The Quancept language includes several words for displaying specific text on the screen during an interview. This text may be information for the interviewer or answers given by the respondent earlier in the interview. Keywords that display information or control how information is displayed are:

control='dropdown'	display responses in a drop-down list
control='listbox'	display responses in a list (combo) box
display	display text temporarily on the screen
hsetcols	set the number of columns for displaying response lists horizontally
maxtime	maximum time to display text
messagebox	display text in a message box
mintime	minimum time to display text
pause	wait until the interviewer presses a key to continue
protect	display text permanently on the screen
readfile	extract text from a file
setcols	set the number of columns for displaying response lists horizontally
vsetcols	set the number of columns for displaying response lists vertically
xdisplay	read text from a file and display it temporarily on the screen
xprotect	read text from a file and display it permanently on the screen

In addition to these keywords, this chapter also discusses ways of displaying data as part of a text, and of highlighting selected words within text.

8.1 Displaying temporary text

Quick Reference

To display text temporarily on the screen, type:

[label] **display** 'text'

In mrInterview, the label is mandatory. If the script will be translated using mrTranslate, all *display* statements must have labels otherwise they cannot be translated.

In Quancept CATI, if the displayed text is overwritten too quickly by another text, follow it with:

pause

This displays a message asking the interviewer to press any key to continue.

The simplest way to display something on the screen is to use the **display** command. This word is followed by a space and the text to be printed which, like a question, must be enclosed in single quotes. For example, at the end of an interview you may want to display a message reminding the interviewer to thank the respondent for his/her help. You could add this to your script like this:

```
exit    display 'Thank you for your help. Goodbye'
```

A *display* statement can display up to 1024 characters; that is, roughly 12 lines of 80 characters. If you are displaying answers to questions as part of the text, it is possible that although the number of characters in the text (including question names) is less than 1024, the limit will be exceeded when the responses are substituted during the interview. You can solve this problem by spreading the text over several consecutive *display* statements. The interviewing program lets you display up to 21 lines of text at a time if there is no question text on the screen, or 10 lines if a question text is displayed at the same time.

Text displayed in this manner remains on the screen until the interviewing program receives input from the interviewer. This input may be the answer to a question, where the display text is on the interviewer's screen above the question text, or the pressing of a key in response to the keyword **pause**. The effect of *pause* is to freeze the screen temporarily and display a message asking the interviewer to press any key to continue.

- ☎ When the CATI and CAPI interviewing programs display text on screen, they show the text as it appeared at the point the *display* statement was executed. If, as part of its text, a *display* statement shows the value of a variable, the interviewing program show the variable's value as it was when the *display* statement was executed. If later statements change the value of the variable, the interviewing program does not update the variable's value in the display statement.
- 💻 The Quancept Web and mrInterview interviewing programs always update variables in *display* statements so that the statement reflects the current value of the variable.

☞ For further information about displaying the values of variables in text see section 8.8, Text substitution in response lists.

- 💻 When an mrInterview interview finishes or is stopped either by a *stop* or *signal* statement in the script or by the respondent clicking the Stop button, mrInterview displays a page confirming that the interview has terminated. The default page contains the words 'End of interview. Thank you for your participation.'. There are three ways of replacing this text with a text of your choice:
 - Place a *display* statement at the end of the script or at the point that the interview terminates, containing the texts you wish to display in these situations.
 - Define page templates containing the text you want to display in these situations and use *set curpgstyle* statements in the script to use these templates when necessary.

- • When you activate the project, cancel the option to use the default interview pages and enter the URLs or template names of the pages you want to display at the end of a completed interview and when an interview is stopped.
-
- ☞ See chapter 23, 'Using page layout templates in mrInterview', and the section entitled 'Interview Templates tab' in chapter 32, 'Parsing and compiling scripts' for further details.

Displaying pictures



You can display pictures with *display* in the same way that you display pictures with questions. You use the same keywords too, which are as follows:

	display a picture (Quancept Web and mrInterview)
mmpicsize	picture size (Quancept CAPI)
mmpicture	display a picture (Quancept CAPI)
mmpicunit	unit in which picture size is defined (Quancept CAPI)

☞ For further details on these keywords see chapter 6, 'Writing multimedia scripts'.

Changing the appearance of the text

Quancept CAPI, Quancept Web and mrInterview all have extensive text formatting capabilities which include changing the color or size of the text, or the font in which it is displayed. Changing the appearance of a word or phrase is a good way of drawing attention to that text: you could use it for displaying special instructions or warnings, or for highlighting key texts such as brand names.

The keywords you use for changing a text's appearance vary between Quancept CAPI and Quancept Web and mrInterview. The examples shown here are simply provided to give you an idea of how to make these changes.



Here is an example for a CAPI script:

```

display 'Here is the first picture.'
mmpicture='c:\\pix\\cheese\\ad1.gif'
qad1   ask 'Which brand of cheese do you think it is advertising?'
      resp sp 'limeswold' / 'countryside cheddar' go ctn /
            'mouse''s choice' / 'farm fresh' other dk
wrong  display 'No. It was <color:red>Countryside Cheddar<color:black>'
ctn    continue

```



The same example written for Quancept Web and mrInterview is:

```
comment For mrInterview replace \\ in image pathname with /
display 'Here is the first picture. '
qad1 ask 'Which brand of cheese do you think it is advertising?'
      resp sp 'limeswold' / 'countryside cheddar' go ctn /
           'mouse''s choice' / 'farm fresh' other dk
wrong display 'No. It was
<font color="red">Countryside Cheddar<font color="black">'
ctn   continue
```

-
- ☞ For a complete description of text formatting keywords see chapter 22, ‘Screen management with graphical user interfaces’.
-

A note about display statements and design mode



If you intend to use QCompile’s design mode to manipulate the layout of your script on the interview screen, you must give every *display* statement in the script a label. Statements that do not have a label and that are repositioned in QCompile may be displayed incorrectly by the interviewing program. By applying a label to a *display* statement, you ensure that any screen position for that statement is retained after repositioning.

-
- ☞ For more information about design mode, see chapter 42, ‘Quancept CAPI design mode’.
-

8.2 Displaying text in a message box



Quick Reference

To display text in a message box, type:

messagebox 'text'

Message boxes are a useful means of emphasizing messages or highlighting inconsistencies in a respondent’s answers. For instance, you might define a message for the interviewer to read out if the sum of three values given by the respondent does not add up to 100%. You can set this message up so that it is displayed in a message box that the interviewer or respondent can cancel by clicking on the OK button.

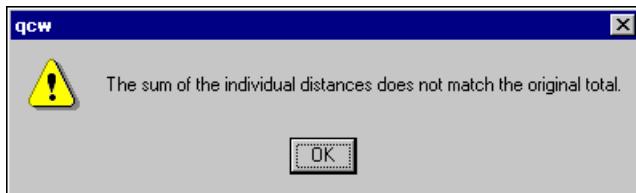
```
journey ask 'How many miles do you travel to work each day?'
       resp num 1 to 50
       set total=0
11      for howgo = 'train' / 'underground train' / 'bus' /
           'car/motorcycle' / 'bicycle' / 'foot'
```

```

howfar      ask 'How far do you travel by '+howgo+'?'
            resp num 0 to 50
            total = total + howfar(*)
next
if (total <> journey )
{
    messagebox 'The sum of the individual distances does not
match the original total.'
    goto journey
}

```

The message box that this script displays is:



8.3 Controlling the display time for text



Quick Reference

To define the how long a text will be displayed, type:

display 'text' [mintime=number] [maxtime=number]

where *number* is a number of seconds.

When you define the minimum length of time for which display text will appear on screen, the interviewer is unable to continue with the interview until at least the given number of seconds has elapsed. When you define a maximum display time, the interviewing program automatically displays the next screen when the given time has passed.

In the following example, the text is displayed for five seconds, after which the interviewing program continues automatically with the next statement:

```

if (total <> journey ) {
d1      display 'The sum of the individual distances does not
match the original total.' mintime=5 maxtime=5
        goto journey
}

```

-
- ❖ QCompile does not check the minimum and maximum display time range when the *mintime* and *maxtime* keywords appear in the same statement. When you define display times in this way, make sure that the minimum display time does not exceed the maximum display time.
-

mintime and *maxtime* are also valid on *ask* and *multitask* statements. When used with *multitask*, they apply to all questions named on that statement and must be entered before the list of question names, as follows:

label multitask 'text' [mintime=n1] [maxtime=n2] dumq1 / dumq2 / ...

-
- ❖ Use *mintime* and *maxtime* with care on *ask* statements; be especially careful with *maxtime*. If you define a question with a maximum display time, the interviewing program displays the question for that length of time and then moves on to the next screen. If the question has not been answered during this time, it is treated as unanswered and has a NULL value.
-

8.4 Protected texts

Quick Reference

To display a text until it is explicitly removed, type:

label protect 'text'

To remove a protected text, type:

unprotect label

Another method of displaying information is to use a **protect** statement that displays a text on the screen until it is explicitly removed by an **unprotect** statement. This is the ideal way to keep a permanent reminder of the respondent's demographic details or the title of the survey on the screen. The same rules for text lengths apply to protected text as to displayed text.

Here is an example that uses protected text to display part of a question text which is common to a number of questions:

```
chprot  protect '@I''d like to ask you about different types of
cheeses. For the moment I''m interested in all varieties of
cheese - that is, hard cheeses and soft cheeses, cheeses made
from cows'' milk and cheeses made from goats'' milk@@'
qallch  ask 'First, let me ask which cheeses can you think of?'
        resp mp atoz cheese other
usech   ask 'And which of those are eaten by people in your household?'
        resp mp atoz qallch in cheese null
unprotect chprot
```

When several blocks of information are to be displayed, they are listed on the screen in the order in which they appear in your script file. If one block of text is erased from the middle of a set of protected texts the lower texts are moved up to avoid lots of blank spaces on the screen.

 When the CATI interviewing program displays protected text on screen, it shows the text as it appeared at the point the *protect* statement was executed. If, as part of its text, a *protect* statement shows the value of a variable, the interviewing program shows the variable's value as it was when the *protect* statement was executed. If later statements change the value of the variable, the interviewing program does not update the variable's value in the protected text.

 The Quancept CAPI, Quancept Web and mrInterview interviewing programs update variables in *protect* statements so that the statements reflect the current value of the variable.

When using a *protect* statement to display text, note that the statement is only removed by an *unprotect* statement. If the script includes routing, you must ensure that the *unprotect* statement is reached by all respondents. To illustrate this, we'll look at a simple example.

```
p1      protect 'Questions about hard cheese'
qhard  ask 'Which hard cheeses can you think of?'
        resp mp hard in cheeses other null dk
        route (qhard=null .or. qhard=dk) go p2
eat    ask 'And which of those cheeses do you eat yourself?'
        resp mp qhard in cheeses
        unprotect p1
p2      protect 'Questions about soft cheese'
qsoft  ask 'Do you ever eat soft cheese?'
        resp sp 'Yes' / 'No'
```

In this example, the p1 *protect* statement marks the beginning of a section about hard cheeses and is accompanied by an accompanying *unprotect* statement at the end of the section. However, any respondents who give a *null* or *dk* response to the question 'qhard' will be routed over the *unprotect* statement. The hard cheese *protect* statement will therefore remain on the interview screen.

The following script avoids this problem by preceding the *unprotect* command with a *continue* statement. This provides a label to which respondents giving a *null* or *dk* response to 'qhard' can be routed before the *unprotect* statement is executed.

```
p1      protect 'Questions about hard cheese'
qhard  ask 'Which hard cheeses can you think of?'
        resp mp hard in cheeses other null dk
        route (qhard=null .or. qhard=dk) go ctn
eat    ask 'And which of those cheeses do you eat yourself?'
        resp mp qhard in cheeses
ctn   continue
        unprotect p1
p2      protect 'Questions about soft cheese'
qsoft  ask 'Do you ever eat soft cheese?'
        resp sp 'Yes' / 'No'
```

-
- ❖ We recommend that you do not put protected texts inside loops as this can cause problems for interviewers if they need to skip back to a question within a loop.

When the interviewing program displays text on screen, it shows the text as it appeared at the point the *protect* statement was executed. If, as part of its text, a *protect* statement shows the value of a variable, the interviewing program continues to show the variable at the value it had when the *protect* statement was executed.

Displaying the current question name in CAPI and mrInterview interviews



Quick Reference



- To display the name of the current question during an interview, type:

label protect ['Text'+]qname

near the beginning of the script.

To display the full name of the current question, including the iteration number, if any, type:

label protect ['Text'+]qnameful

In Quancept CAPI and mrInterview it is not always easy for the person completing the questionnaire to know the name of the current question. However, they may need to know it if they have problems at a specific question and need to report exactly where the problem occurred. To solve this problem, you can put a *protect* statement near the beginning of the script that displays the name of each question on the screen as the interview progresses. The statement uses the special variable called **qname** which the interviewing program interprets to mean ‘the name of the current question.’ For example:

```
p1    protect 'The name of the current question is:' +qname
```

As the interview progresses, *qname* is updated to reflect the question being asked.

- ❖ In CATI interviews conducted using mrInterview, you may want interviewers to write down open-ended responses rather than typing them as the respondent speaks. In order to be able to enter these handwritten responses against the correct questions later on, interviewers will need to know the full name of the question, including any iteration number, so that they can write this down with each response. The variable *qnameful* provides the full name, and can be used with the *pgtitle* variable to display the full question name as the page title for each question.

☞ For more information on special variables, see chapter 12, ‘Assignment’.

For more information about *pgtitle*, see section 23.9, Choosing a page template for a question.

8.5 Reading texts from files



Quick Reference

To display text from a file, type one of the following:

xdisplay 'pathname'

label **xprotect 'pathname'**

If the file is in the project directory you may enter just the name of the file rather than the full pathname.

The keywords **xdisplay** and **xprotect** are variations of *display* and *protect* which read their texts from a file rather than from the script. This can be useful when you have a long text to display or when you want to build up a library of standard texts for use in any script.

Text files may contain up to 21 lines of text. Characters such as @, + or single quotes that have a special meaning in the script have no special meaning in the text file: they are simply characters to be displayed in the text. To start a new line in the text, press the Return key when you are typing; to display an apostrophe, type just one single quote. Since single quotes and + have no special meaning in the file, this means that you cannot display the values of variables in the text.

When using *xdisplay* and *xprotect*, follow them with the name of the file to be read, enclosed in single quotes. For example:

`xdisplay 'intro'`

If the file is in a different directory from the script, remember to enter its full pathname. For example:

`xdisplay '/u/PROJECTS/intro'`

-
- » It is not necessary for the files to exist when the script is compiled, but if they do not exist when interviewing starts an error will occur.
-

One of the main advantages of using *xdisplay* and *xprotect* is that since the text is contained in external files, the size of the compiled script (the qoc file) is drastically reduced, thereby allowing space for more complex script structures.

8.6 Reading single lines from look-up files



Quick Reference

To retrieve a single line from a look-up file, type:

```
callfunc('readfile', fname_var, line_key, text_var)
```

If *line_key* is a number, the interviewing program retrieves the text from that line in the file; if it is a text, the interviewing program retrieves the first line that starts with that text (but not the text of *line_key* itself). The retrieved text is copied to *text_var*.

Sometimes you may want to display a single line from a file rather than the whole file. One example might be where the response list contains an abbreviated form of a long response and you want to display the full text so that interviewers can verify that they have chosen the correct item from the list.

Another reason could be that you have a file of standard texts that are common to many of the scripts you write, and you want to be able to call up the one that suits your current requirements. For example, you may have a standard text for interviews with housewives and other standard texts for female students, women who work full time, female pensioners, and so on.

The **callfunc** **readfile** allows retrieval of text from a look-up file during the course of an interview. The text is located either by its line number, or by searching for a line starting with a given string. Only one line may be read from the file at a time.

For example, you would write:

```
set filename = 'chserate'  
callfunc('readfile',filename,4,thisone)
```

to copy the contents of the fourth line from a file called 'chserate' into the variable 'thisone'. You could then display the text by naming the variable on a *display* or *protect* statement:

```
display thisone
```

In the next example, the line key is a variable containing the text to be located in the file rather than a line number:

```
set filename = 'chserate'  
set findme = 'mched'  
callfunc('readfile',filename,findme,thisone)
```

This example searches for the first line in 'chserate' that starts with the characters 'mched'. The remaining characters on that line will be copied into a variable called 'thisone'. For example, if 'chserate' contains the lines:

```
fchedFarmhouse Cheddar
mchedMature Cheddar Cheese
```

the value of ‘thisone’ will be ‘Mature Cheddar Cheese’; that is, it does not include the string ‘mched’.

Here is another example that selects six statements at random from a look-up file. The example uses some keywords that you have not yet met. These are *for* and *next* which mark the start and end of a group of statements to be executed repetitively. The values on the *for* statement determine the number of repetitions. In this example, this is ten because there are ten statements in the look-up file. However, statements inside the loop count the number of statements selected and route out of the loop after six have been chosen.

```
comment Name the look-up file
      set filename = 'chserate'
comment Make choices at random from ten items
      set numasked = 0
      for stmtnum = 1 to 10 ran
comment Count number of times the loop has been started (i.e.,choices
comment made) and go to exit if more than 6
      set numasked = numasked + 1
      route (numasked > 6) go exit
comment Read statement from file
      callfunc('readfile',filename,stmtnum,thisone)
comment Display statement as question text
state      ask thisone
      resp sp 'agree very much' / 'agree a little' /
              'neither agree nor disagree' / 'disagree a little' /
              'disagree very much' freeze
      next
exit      display 'Thank you for your help. Goodbye'
```

When you read information from a look-up file into a script, you must take care to test the file both as a script designer and with the interviewer’s login configuration. The interviewing program expects to be able to access the look-up files mentioned in the script. You must therefore ascertain that there are no permissions problems that would prevent any Quancept program, particularly the interviewing program, having access to those files.

Look-up files do not have to be present when you parse the script, however. This enables you to design a script without the need to have all look-up files present and allows work on those external files to be completed in an extended time frame.

- ☞ If an external file contains defined lists or questions that are used by the main survey, the parser will report warnings or errors if this information is referenced without being present.
- ☞ The next section explains more about displaying answers and the values of other variables in texts. Also, see chapter 10, ‘Repetitive questions’, for full details of the *for* and *next* keywords.

8.7 Displaying data in texts

Quick Reference

To display the value of a variable or a mixture of text and variables, type:

[label] **display** [’text’+]variable_name [+’text’]

These rules also apply to *ask*, *protect* and *messagebox* statements which display text and/or variables on the screen.

When displaying information such as demographic details you will want to have a standard statement format that can be used for any respondent. This is best illustrated by examples as follows:

demog display 'Respondent '+name+' is '+age+' years old.'

could give:

Respondent Alan is 30 years old. or Respondent Carol is 19 years old.

Name and age are the labels of the questions whose responses you want displayed or protected at the appropriate points in the given text. The response to each question is always uniquely related to the current respondent. The text that is common to each respondent is enclosed in single quotes, together with any extra spacing required (notice the space after the word Respondent, for example). The labels of the questions whose responses are to be printed are enclosed in plus signs (+).

If the response appears at the end of the text it is preceded by a plus sign rather than enclosed in plus signs:

dname display 'The respondent''s name is '+name

might return ‘The respondent’s name is Carol’. If you add an extra plus sign by mistake the parser will issue an error message when checking your script.

If the response is at the start of the display, the question label is followed by a single plus sign. So,

dtype display name+ likes '+type+' cheese'

might give us ‘Benjamin likes limeswold cheese’.

If you want to display the answer to a question or the value of a temporary variable without any supporting text, just type the question or variable name and nothing else. For example, you would type:

dname display name

to display the respondent’s name by itself.

The length of a text with variable substitution varies according to the length of the values substituted. Numbers are represented in the minimum space required to represent them. The number 468 occupies three spaces only, whereas 19731 occupies five spaces. Single-punched responses are displayed as the word they represent, and multipunched responses are printed as a list of choices, with choices separated by a semicolon (red; green; blue, for instance). Open ends are displayed in as much space as is required, even if this means extending the protected or displayed text over several lines.

- ☎ When the CATI interviewing program displays text on screen, it shows the text as it appeared at the point the *display* or *protect* statement was executed. If the statement shows the value of a variable, the interviewing program show the variable's value as it was when the statement was executed. If later statements change the value of the variable, the interviewing program does not update the variable's value in the displayed statement.
- 🌐 The Quancept Web and mrInterview interviewing programs always update variables in *display* and *protect* statements so that the statement reflects the current value of the variable.
- 💻 The CAPI interviewing program updates variables' values in *protect* statements but not in *display* statements.

The examples so far have all used *display* statements. Substitution is valid in any text, including question texts and response lists. In some cases you can even vary a complete question text to suit the respondent just by using substitution.

For example:

```

film    ask 'Which film did you watch?'
        resp sp 'Batman' / 'Superman' / 'Superman II'
q2      ask 'How many children were in your party?'
        resp num 0 / 1 go one / 2 to 9 go many
        set qtxt = 'What do you think children would particularly
like about '
        goto ctn
one     set qtxt = 'What did the child particularly like about '
        goto ctn
many    set qtxt = 'What did the children particularly like about '
ctn     continue
like    ask qtxt+film
        resp coded

```

The main point about this example is the way we build up the text of the question based on the responses to earlier questions. When we come to defining the question, we enter the name of the variable containing the question text after the *ask* where we would normally type the question text.

-
- ☞ There is a further example of building question texts using variables in section 12.3, Text assignments. See also section 8.8, Text substitution in response lists.
-

In most circumstances there is no limit on the number of variables you may display in a text, apart from the fact that the whole text including variable values should fit on one screen. However, if you are using callfuncs there is a limit of 300 variables per displayed text.

8.8 Text substitution in response lists

Quick Reference

To display the value of a question or temporary variable as a response in a response list, type the name of that variable enclosed in [+ ... +] in place of the response text:

```
resp sp 'respI' / '[+varname+]' / ...
resp mp 'respI' / '[+varname+]' / ...
```

The [] characters are required as part of the statement. They do not denote an optional parameter.

Sometimes you will want to display the answer to a previous question as a possible answer to a second question. The way you write this in the script is similar to that described above for displaying data in texts, except that the label of the question containing the data is enclosed in square brackets and plus signs, rather than just plus signs.

One example of using this feature is when you want to display an answer given with specified other as part of a later response list.

-
- ☞ There are several ways of extracting answers given using specified other; the method used here is one that is easy to follow without having a lot of background information.
 - ☞ See chapter 4, ‘Special responses’, for a complete discussion of using specified other responses in the script.
-

This example uses an *othertext* statement to extract specified other responses into a temporary variable (that is, a variable that is not a question). The variable is then used in the response list to tell the interviewing program to display the value currently held in that variable.

```
use    ask 'How is cheese used in your household?'
       resp mp 'in sandwiches' / 'with crackers' / 'with wine' /
              'for cooking' other
       set qoth = othertext(use,5)
pop    ask 'And, of those uses, which is the most popular use?'
       resp sp 'in sandwiches' / 'with crackers' / 'with wine' /
              'for cooking' / '[+qoth+']'
```

The parser places the names of temporary variables in a file called *script.qtv*. If you create and parse your script in one directory and then move the compiled qoc file to a different directory for interviewing, you must also copy the qtv file. If the interviewing program is unable to find the qtv file, it will display the name of the temporary variable in the response list rather than the text of the other specify that it contains.

You can also use this notation to insert responses as texts in loop value lists, and with defined lists. Here is our previous example written using defined lists:

```

        set qoth = 'other'
usage  define 'in sandwiches' / ' with crackers' /
           'with wine' / 'for cooking' / '[+qoth+]'
use    ask 'How is cheese used in your household?'
       resp mp usage other
comment Check whether cheese is used in some other way
       set oth = bit(use/5)
       if (oth) {
qoth1   ask 'How else is cheese used in your household?'
       resp coded(9)
       set qoth = qoth1
}
pop    ask 'And, of those uses, what is the most popular use?'
       resp sp use in usage

```

If the respondent's household uses cheese in sandwiches, with wine and in mousetraps, the response list for 'pop' would contain these three responses and nothing else.

The main point to notice here is the way the possibility of another way of using cheese is catered for when the list is first defined.

In the final example, below, you see how to use a loop to display up to three 'other' responses in a subsequent question. If you are new to script writing you may prefer to skip this example.

```

chs     define 'Cheddar' / 'Edam' / 'Red Leicester' / 'Other cheese (1)'/
           'Other cheese (2)'/ 'Other cheese (3)'
qlike  ask 'Which cheeses do you like ?@'
       resp mp chs

comment Set chsname to 4, 5 and 6 (the punches in chs/qlike
comment corresponding to the "Other cheese" options
11     for chsname = 4 to 6
       comment Check whether punch 'chsname' (4,5 or 6) was chosen at
comment qlike. If yes, ask interviewer to enter the name of the
comment Other cheese in question
       set yes=bit(qlike/chsname)
       if (yes) {
othchs    ask 'INTERVIEWER: Enter ONE (and ONLY ONE) of the
other cheeses mentioned@'
       resp coded
}
next

```

```
comment First name entered at othchs is displayed after Red Leicester;
comment the second name after that and the third name last
chs2      define 'Cheddar' / 'Edam' / 'Red Leicester'/
           '[+othcsh(1)+]'/ '[+othchs(2)+]'/ '[+othchs(3)+]'

comment Responses selected at qlike are set into 'likes' so we have
comment a question in which, for example, "Gouda" is displayed
comment instead of "Other cheese (1)".
likes      dummyask ''
            resp mp chs2
            unset likes
            set likes=qlike

comment Response list for cfav shows all cheeses selected at qlike
comment with their correct names
cfav      ask 'And which of these is your favorite?'
            resp sp likes in chs2

comment Loop through all cheeses selected at qlike with their correct
comment names
l1       for choice = likes in chs2
qtry      ask 'Did you try '+choice+' because ...@'
            resp mp 'you saw an ad on TV' / 'it cost less'/
                   'you prefer the taste' / 'special promotion'/
                   'None of these' sp Other dk
            next
```

-
- ☞ Defined lists are covered in chapter 7, 'Defined response lists'
See also section 14.2, Checking multipunched responses.
-

8.9 Highlighting text during an interview



Quick Reference

To highlight a word or phrase, enclose it in the characters [+*so+] and [+*se+]:

[+*so+]text[+*se+]

The square brackets [] are part of the syntax. They do not denote an optional parameter.

Highlighting is an effective method of emphasizing certain key words or phrases on the interviewer's screen. To highlight a word or phrase, enclose it in the characters [+*so+] and [+*se+], which stand for **S**tand**O**ut and **S**tandard **E**nd. For example, the text:

```
chprot    protect 'Now I''d like to ask you about different
types of cheeses. For the moment I''m interested in
[+*so+]all varieties[+*se+] of cheese - that is, hard
cheeses and soft cheeses, cheeses made from cows'' milk
and cheeses made from goats'' milk@@'
```

will be displayed during the interview with the words 'all varieties' shown in reverse video (for example, black writing on a white background rather than white writing on a black background).

Highlighting can also be useful for differentiating between text to be read to the respondent and instructions to the interviewer.

-
- ❖ Highlighting works only if your terminal is able to switch into and out of reverse video, and if the character sequences for those changes are set in the terminal definition. If you try this facility and it does nothing, check with your system administrator to find out whether your terminal has support for reverse video.
-

8.10 Scrolled response lists in interviews



Quick Reference

To increase or decrease the number of columns displayed on screen, use the statement:

```
callfunc('setcols',num_cols)
```

When the interviewing program encounters a long response list during an interview, it displays the response texts in three columns in an attempt to fit the whole list on the screen. If the response list is very long, or if it is preceded by a large amount of question text, there may still be insufficient room to display the whole response list. The interviewing program then displays as many responses as will fit on the first screen and prompts the interviewer to press + to see the next screenful.

Sometimes you can force the response list all onto one screen by increasing the number of columns allowed per screen, for example, by allowing four columns instead of three. Texts that are too long for the narrower column width are truncated, and you will need to decide whether this is acceptable.

If you have a response list in which the texts are more than one line long, you may set the number of columns to one. The interviewing program will then display responses in a single column down the screen and will wrap very long texts round onto the next line.

Once read, a *setcols* statement refers to all subsequent response lists until the end of the script or until another *setcols* statement is read.

8.11 Specifying the number of columns for response lists

Quick Reference



To display a response list in a given number of columns, type:

resp resp_type resp_list [other] [h]setcols=number [null] [dk] [ref]

or

resp resp_type resp_list [other] vsetcols=number [null] [dk] [ref]

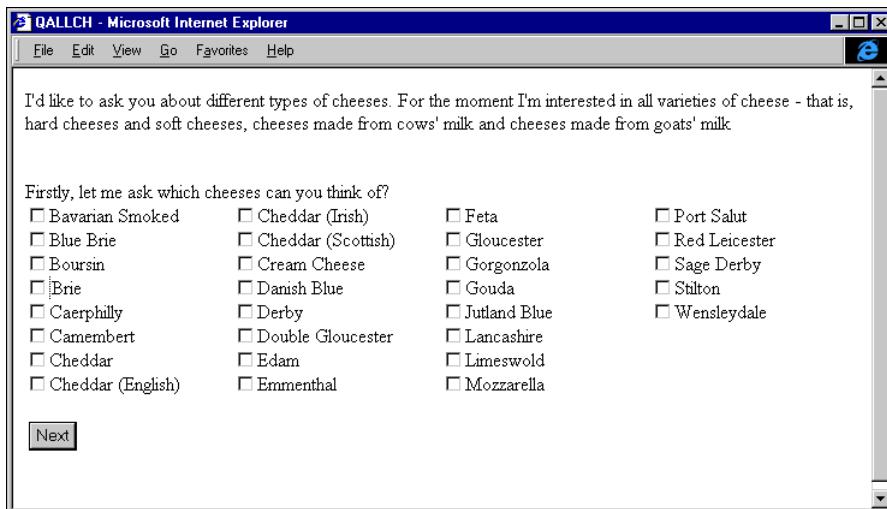
where *number* is the number of columns in which responses are to be displayed. Use *hsetcols* to display responses across the screen or *vsetcols* to display them down the screen. In CAPI only, *setcols* is a variant of *hsetcols*.

The CAPI interviewing program normally calculates the number of columns it needs to display a response list, whereas the Web and mrInterview interviewing programs display the response list as a single column down the screen. **setcols**, **hsetcols** and **vsetcols** give you more control over the way responses are presented during an interview by enabling you to specify the number of columns to be used and how responses are to be allocated to those columns.

All three keywords create a given number of columns on the page; the difference is the way in which responses are allocated to the columns and, in CAPI, the position of the response texts relative to the buttons. With *vsetcols*, responses are allocated to columns vertically so that the first and second responses appear in rows one and two of the first column. For example:

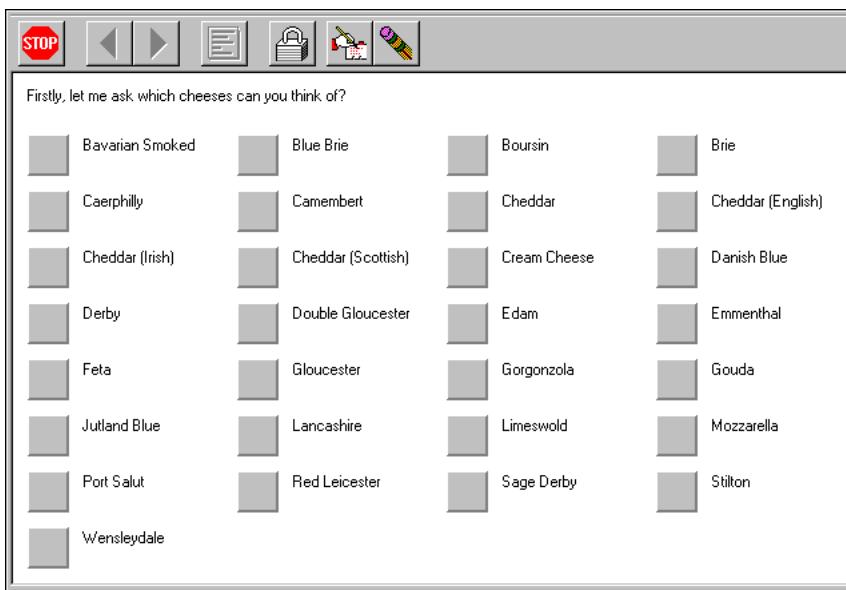
```
cheese    define  'Gouda' / 'Gorgonzola' / 'Cheddar' /  
                  'Cream Cheese' / 'Edam' / 'Caerphilly' /  
                  'Bavarian Smoked' / 'Brie' / 'Cheddar (Irish)' /  
                  'Lancashire' / 'Sage Derby' / 'Port Salut' /  
                  'Cheddar (Scottish)' / 'Danish Blue' / 'Feta' /  
                  'Jutland Blue' / 'Wensleydale' / 'Boursin' /  
                  'Red Leicester' / 'Double Gloucester' /  
                  'Gloucester' / 'Derby' / 'Emmenthal' / 'Stilton' /  
                  'Mozzarella' / 'Camembert' / 'Blue Brie' /  
                  'Cheddar (English)' / 'Limeswold'  
qallch  ask 'Firstly, let me ask which cheeses can you think of?'  
resp mp atoz cheese  vsetcols=4
```

is displayed in Quancept Web and mrInterview as:



The display for Quancept CAPI is similar except that the buttons are larger.

With *hsetcols* in Quancept Web and mrInterview and *setcols* in Quancept CAPI, responses are allocated to columns horizontally so that the first response appears at the top of the first column and the second response appears at the top of the second column. Here is the cheese example displayed using *setcols* in Quancept CAPI (the Quancept Web *hsetcols* display is similar but has smaller selection boxes):



In Quancept CAPI, *hsetcols* is a variant of *setcols* which displays the response texts left-aligned above the buttons rather than on their right. This type of layout takes up more space on the screen and is best suited to short response lists such as the one shown below:



When you use *setcols*, *hsetcols* or *vsetcols*, any responses flagged with *sp* or *^s* (a single-punched response in a multipunched list) are displayed at the end of the list. Special responses such as *null* and *dk* are omitted from the columns and are displayed on the left of the interviewing screen underneath the standard response texts. The three keywords must follow *other* if it is on the list but may come before *null*, *dk* and *ref*.

Setting the width of response columns

Quick Reference



Quancept Web and mrInterview attempt to make the columns of equal widths by dividing the width of the browser's screen by the number of columns required.

To define column widths for Quancept CAPI, type:

resp resp_type resp_list [h|v]setcols(width)=number

where *number* is the number of columns to use and *width* is the percentage of the screen width that each column may occupy.

When you display a response list in a fixed number of columns, the interviewing program distributes the columns evenly over the total width or length of the screen.

- In Quancept CAPI, you may specify the column width as a percentage of the total screen width. For example:

```
usech ask 'How is cheese used in your household?'
resp mp 'in sandwiches' / 'with crackers' / 'with wine' /
       'for cooking' / 'other' hsetcols(15)=5
```

forces the response columns to take up a smaller percentage of the screen than the default, with each of the five columns taking up just 15% of the screen.

8.12 Drop-down lists and combo boxes



Quick Reference

- To present responses as a drop-down list, type:

```
label ask 'question_text'
resp resp_type resp_list control='dropdown'
```

- To present responses as a combo (list) box, type:

```
label ask 'question_text'
resp resp_type resp_list control='listbox'
```

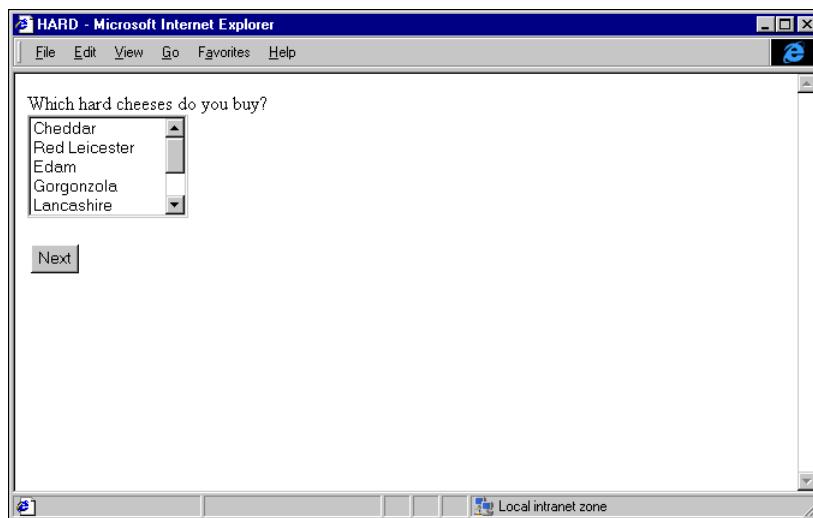
The *control* keyword must follow the last quoted response text.

Do not use specified other in response lists that you display in this way. Instead, include Other as an ordinary response and then, after the question, test for this response and prompt for what Other represents, if necessary.

A list box displays five responses at a time, starting with the first five answers in the response list. The box has a built-in scroll bar for viewing the rest of the list. The respondent can select one or more responses from the list by clicking on them, with multiple selections being made using SHIFT+CLICK and CTRL+CLICK. The question:

```
hard ask 'Which hard cheeses do you buy?'
resp mp 'Cheddar' / 'Red Leicester' / 'Edam' / 'Gorgonzola' /
'Lancashire' / 'Bavarian Smoked' / 'Gouda' /
'Stilton' control='listbox'
```

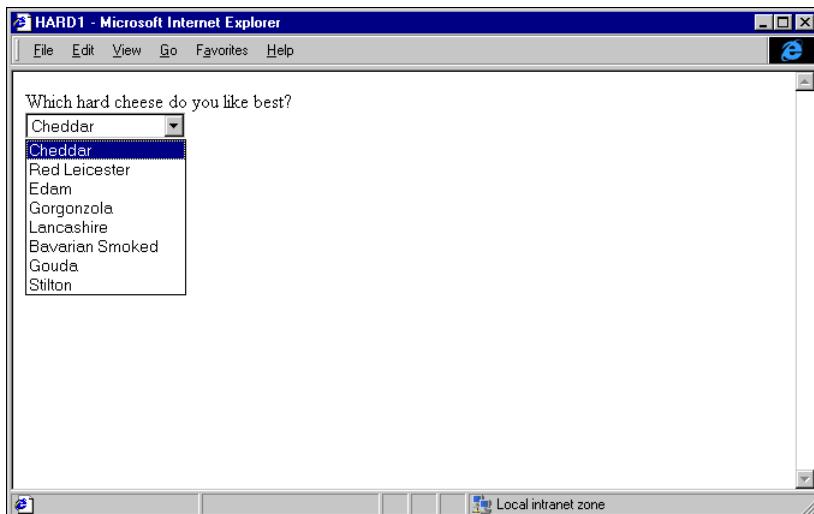
is displayed as:



A drop-down list displays just the first response in a box with an arrow button, and the respondent can open the whole list in a separate box by clicking on that button. If the full list is longer than the space available to display it, the interviewing program makes the box as long as possible and adds a scroll bar for viewing the rest of the list.

Drop-down lists are designed for use with single-punched response lists, and should only be used with questions of that type. (If you use *dropdown* with a multipunched list, you will always see a list box because this is the only way that HTML can allow selection of multiple responses.) When the respondent selects a response, the listing box closes and the response chosen is displayed in the first box.

The previous question (changed to be a single-punched response list) appears as shown below once the respondent has clicked on the arrow button. When the respondent selects a response, the drop-down list closes and the respondent's selection is displayed in the single box:



As the illustration shows, the interviewing program always highlights the first response in the list. If you are concerned that this may encourage respondents to select this response without looking at any others, you can make this response a dummy text and then follow the question with a logical test that repeats the question if the respondent chooses this answer. The example below shows one way of doing this:

```
best    ask 'Which hard cheese do you like best?'
resp mp 'Please select one' /
      'Cheddar' / 'Red Leicester' / 'Edam' / 'Gorgonzola' /
      'Lancashire' / 'Bavarian Smoked' / 'Gouda' /
      'Stilton' control='dropdown'
if (best = 'Please select one') {
d1      display 'You did not choose a valid response from the list.
Please try again.'
      goto best
}
```

-
- ❖ Some browsers have search facilities whereby typing a letter takes you straight to the next response that starts with that letter. In other browsers, this facility is either not available at all, or it behaves differently between drop-down and list boxes.
-

8.13 Sample script

This section contains the sample script for this chapter, plus a listing of the look-up files called by the script.

```
comment Sample script for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 8, Displaying information on the screen

cheese    define  'Gouda' / 'Gorgonzola' / 'Cheddar' /
            'Cream Cheese' / 'Edam' / 'Caerphilly' /
            'Bavarian Smoked' / 'Brie' / 'Cheddar (Irish)' /
            'Lancashire' / 'Sage Derby' / 'Port Salut' /
            'Cheddar (Scottish)' / 'Danish Blue' / 'Feta' /
            'Jutland Blue' / 'Wensleydale' / 'Boursin' /
            'Red Leicester' / 'Double Gloucester' /
            'Gloucester' / 'Derby' / 'Emmenthal' / 'Stilton' /
            'Mozzarella' / 'Camembert' / 'Blue Brie' /
            'Cheddar (English)' / 'Limeswold'

comment CATI only: Read introduction from file
comment For CAPI/Web/mrInterview, replace this line with a standard display
comment statement that contains the introductory text itself.
        xdisplay 'intro'

comment Permanent display text
comment Highlighting. For CAPI, replace [+*so+] with <b+> and
comment [+*se*+] with <b->. For Web/mrInterview, replace [+*so+] with <b>
comment and [+*se*+] with </b>
chprot    protect '@I''d like to ask you about different types of
            cheeses. For the moment I''m interested in [+*so+]all varieties[+*se+]
            of cheese - that is, hard cheeses and soft cheeses, cheeses made
            from cows'' milk and cheeses made from goats'' milk@@'

comment CATI only: Use four columns for the response list
        callfunc('setcols',4)
qallch    ask 'Firstly, let me ask which cheeses can you think of?'
        resp mp atoz cheese
usech     ask 'And which of those are eaten by people in your household?'
        resp mp atoz qallch in cheese null

comment CAPI/Web/mrInterview only: Use four columns for the response list
comment qallch    ask 'Firstly, let me ask which cheeses can you think of?'
comment         resp mp atoz cheese vsetcols=4
comment usech     ask 'And which of those are eaten by people in your
comment         household?'
comment         resp mp atoz qallch in cheese null vsetcols=4

comment Remove permanent display text
        unprotect chprot
```

```
comment CATI only: Revert to two columns
    callfunc('setcols',2)

comment CATI/Web/mrInterview only
usehh    ask 'How is cheese used in your household?'
        resp mp 'in sandwiches' / 'with crackers' / 'with wine' /
              'for cooking' other

comment CAPI only: Display response text above each response button
comment instead of to the right.
comment usehh ask 'How is cheese used in your household?'
comment      resp mp 'in sandwiches' / 'with crackers' / 'with wine' /
comment          'for cooking' other hsetcols=5

comment Get value of other
    set qoth = othertext(usehh,5)

pop      ask 'And what is the most popular use?'
        resp sp 'in sandwiches' / 'with crackers' / 'with wine' /
              'for cooking' / '[+qoth+]'

comment Temporary displayed text
d1      display 'Now I''m going to read you some statements people
        have made about cheese. For each one I''d like you to tell me whether
        you agree very much, agree a little, neither agree nor disagree,
        disagree a little , or disagree very much.@@'
        pause

comment CATI/CAPI only: Ask for ratings on six statements chosen
comment at random from a file of 10 statements.
comment Web/mrInterview: Delete or comment out all statements between
comment here and the line containing 'next'. To achieve the same
comment functionality in Quancept Web and mrInterview you would use an
comment ODBC database.
        set filename = 'chserate'
        set numasked = 0
11      for stmtnum = 1 to 10 ran
        set numasked = numasked + 1
        route (numasked > 6) go exit

comment Read statement from file
        callfunc('readfile',filename,stmtnum,thisone)
state    ask thisone
        resp sp 'agree very much' / 'agree a little' /
              'neither agree nor disagree' /
              'disagree a little' / 'disagree very much'
        freeze

next
```

```
comment CAPI only: Display closing text in a message box
comment For CATI/Web/mrInterview, replace 'messagebox' with 'display'
exit      messagebox 'Thank you for your help. Goodbye'
end
```

Look-up files

Here is the look-up file used with *readfile* from which this script selects six statements:

```
Soft cheese is bad for some people
Cheese is good value for money
Cats like cheese as a special treat
Cheese makes a healthy snack
Goat''s milk cheese is expensive
Blue cheeses are too strong for me
Eating cheese for supper gives you bad dreams
Children should eat cheese as part of their basic diet
The smell of cheese makes me feel ill
Old hard cheese is best for catching mice
```

The file of introductory text, intro, contains the following:

```
Hello. I''m calling from SPSS MR, an independent market
research company.
```

9 Routing and flow control statements

The interviewing program normally executes statements in a script in the order in which they occur. Some statements are provided to alter this normal order of execution by allowing you either to skip over certain questions or to ask a question more than once. In this chapter, we will discuss statements for skipping to different parts of the script. We call this **routing**. We will also cover some statements which allow you to alter the order in which a group of questions is presented:

else	questions for respondents failing the <i>if</i> condition
go	route to a named statement
goto	route to a named statement
gotosms	return to the SMS menu
if	start of conditional questions
ifsnap	test for snapbacks
newran	randomize blocks of statements
newrot	rotate blocks of statements
onresp	routing based on keyboard input
ranend	end of questions to be presented in random order
ranstart	start of questions to be presented in random order
rotend	end of questions to be presented in rotation
rotstart	start of questions to be presented in rotation
route	conditional routing
ststest	actions for test interviews only
testingrun	actions for test interviews only

-
- Take care when you use routing in a script. It is easy to make a mistake and skip over questions by accident when you move from one part of a script to another.

For example, if you collect and store information in variables that your script will depend on later in the script, be sure that any questions that define or initialize the variables cannot be missed accidentally due to routing statements or snapbacks. If this happens in your script, those variables will be set to unexpected values.

9.1 Routing in single-punched and numeric response lists

Quick Reference

To attach routing instructions to single-punched and numeric responses, type:

```
resp sp 'resp1' go label / 'resp2'  
resp num val1 go label / val2  
resp real val1 go label / val2
```

If the same routing applies to a set of consecutive single-punched responses, you may enclose the list of texts in parentheses and follow the closing parenthesis with the name of the statement to skip to, as shown below:

```
resp sp ('resp1' / 'resp2') go label / 'resp3'
```

Routing allows you to skip certain questions and is often part of a response list. It is used when the next question to be asked is dependent upon the response to the current question. For example, you may wish to assess people's opinions on public transport, with the questions varying according to the type of public transport used.

One of the keywords associated with routing is **go**. You use it in single-punched and numeric response lists to define routing for an individual response or group of responses in the list. *go* must always be followed by the label of the next statement to execute. We call this the **routing destination**. All statements between *go* and the routing destination are then ignored when that response is chosen. Here is an example of a screening question which uses *go*:

```
qb ask 'Do you travel regularly by train / may I speak with someone  
who regularly travels by train?'  
resp sp 'respondent travels regularly by train' /  
      'no one travels regularly by train' go term1 /  
      'regular train traveler not available' go appt
```

In this example, the interviewing program routes to the label 'term1' if no one in the household travels regularly by train. If the person who does travel regularly by train is not available, the interviewing program routes to the label 'appt' so that an appointment can be arranged. If the current respondent travels regularly by train, there is no routing required, so the interviewing program will continue with the next question in the script.

If the response type is numeric (integer or real), you may specify a response as a range. For example:

```
resp num 10 to 20 go small / 21 to 30 go large
```

If the response is any whole number between 10 and 20, the script routes to a label ‘small’. If the response is between 21 and 30 the script routes to ‘large’. The use of ranges saves you from entering the same routing for each possible response. Here is another example of routing where the response list combines ordinary ranges with open-ended ranges:

```
age      ask 'How old are you?'
        resp num 12- go child / 13 to 19 go teen / 20+ go adult
```

With single-punctured lists, responses which share the same routing may be grouped with parentheses:

```
intro  ask 'Please may I speak to the person responsible for
        grocery shopping?'
        resp sp ('Correct person speaking' /'Will transfer you') go ctn /
        'Not available at present' go appt / 'Refusal'
```

9.2 Routing outside response lists

Quick Reference

To define routing outside a response list, type:

```
goto label
```

Where it is necessary to use routing on its own, use the word **goto** instead of *go*. In the example below, the question asked after q2c depends on the answer given to q2c. Respondents who think the service is bad or appalling answer question q3c, while those who think it is excellent, very good or satisfactory answer q3cx. After this, all respondents continue with the next statement in the script. The script below shows one way of dealing with this type of requirement.

```
q2c      ask 'What is your general opinion of the service offered
        for commuters?'
        resp sp ('Excellent' / 'Very Good' / 'Satisfactory') go q3cx /
        'Bad' / 'Appalling'
q3c      ask 'Why do you say that?'
        resp coded
        goto ctn

comment Respondents who think the service is excellent, very good
comment or satisfactory
q3cx     ask 'Even though you think the service offered for commuters is
        '+q2c+' is there anything about it which you think could be improved?'
        resp coded

ctn      continue
```

The two questions to be answered by the two groups of respondents are both listed after q2c. The routing in the response list to q2c takes respondents who think the service is excellent, very good or satisfactory straight to q3cx, omitting q3c, which does not apply to them. Having answered q3cx, they then continue with the next statement in the script.

Respondents who think the service is bad or appalling have no routing in the response list, so they continue with q3c. The question after this is q3cx, which does not apply to them, so we precede this with a *goto* statement that routes them around this question and on to the next one, which does apply.

-
- ☞ You will find another way of writing this script in section 9.5, Conditional questions (using if and else).
-

9.3 Conditional routing

Quick Reference

When routing is dependent on the outcome of a test, type:

route (*logical expression*) **go** *label*

It is possible that you may wish to skip questions in certain cases only. The Quancept language provides for this with the word **route**. When this is used, routing depends upon the result of a **logical expression**. This is a statement which compares the value of a variable against a fixed value. If the variable has the given value, the expression is true and the routing will be followed; if the variable does not have the given value, the expression is false and the routing will be ignored.

-
- ☞ See also section 13.2, Symbols used with logical expressions.
-

Here are two similar examples of routing. The first example tests the value of a single-punched response, while the second shows one way of testing the value of a multipunched response. In both cases, we are checking whether the respondent chose Victoria Station as the destination:

```
comment Single punched example
spdest ask 'Which station do you usually arrive at on your
inward journey?
@INTERVIEWER: If station varies, code the one used most often.'
    resp sp 'Kings Cross' / 'Euston' / 'Victoria' /
        'Waterloo' / 'Charing Cross' / 'Cannon Street' /
        'Marylebone' / 'Paddington' / 'Moorgate' /
        'St Pancras' other
    route (spdest = 'Victoria') go vict
```

```

comment Multipunched example
mpdest ask 'Are there any other stations that you travel to?'
resp mp 'Kings Cross' / 'Euston' / 'Victoria' /
      'Waterloo' / 'Charing Cross' / 'Cannon Street' /
      'Marylebone' / 'Paddington' / 'Moorgate' /
      'St Pancras' other
route (mpdest = '<Victoria>') go vict

```

The main difference between the two examples is the format of the logical expression used with *route*. The first is checking a single value, so we can enter that value in the expression exactly as it appears in the response list. The second example is checking a response which may consist of more than one answer. The angle brackets around Victoria in the second logical expression tell the interviewing program to return a true value if Victoria was chosen with any other response which appears to the left (<) or right (>) of it in the response list.

In both examples, the next question in the script is executed if Victoria station is not mentioned.

-
- ☞ You will find further examples of using *route* with multipunched responses in ‘Symbols with multipunched responses’, in chapter 13, ‘Logical expressions’.
-

9.4 Routing dependent on keyboard input



Quick Reference

To define routing for responses which are not part of response lists, type:

onresp 'text' go label

The routing we have talked about so far has all referred to responses defined in the response list for particular questions. The **onresp** statement lets you define routing for responses which are not part of a response list. For example, interviewers may be told to type ‘term’ if the respondent refuses to continue with the interview. Once defined, responses of this type may be entered at any point during the interview, regardless of whether they appear in the response list to a question.

```

onresp 'term' go exit
qa      ask 'Do you work on the railways? Is anyone in your immediate
family associated with this field?'
resp sp 'yes' go term1 / 'no' nodata
qb      ask 'Do you travel regularly by train / may I speak with someone
who regularly travels by train?'
resp sp 'respondent travels regularly by train' /
      'no one travels regularly by train' go term1 /
      'regular train traveler not available' go term2
      nodata
other statements
exit    display 'Thank you for your help'

```

Your script may contain as many *onresp* statements as you like. There are, however, some words of warning:

- Although there are no rules governing where in the script such statements may or may not occur, you are advised to place them at the beginning of the script so that they are immediately apparent to anyone else working with the script.
- *onresp* statements are evaluated and stored before interviewing begins and cannot be altered during the course of the interview. If you have two or more statements with the same response text but with different destinations, all except the first will be ignored.
- *onresp* as a method of routing should be used with care, since it may allow the interviewer to skip unexpectedly to different sections of a script before key variables have been set or to skip to parts of the script which are irrelevant to the current respondent.

9.5 Conditional questions (using if and else)

Quick Reference

To define blocks of conditional statements, type:

```
if (logical expression) {
    statements.1
}
[else {
    statements.2
} ]
```

The keywords **if** and **else** provide an alternative to *route* when you have questions which are designed for specific respondents only. They are particularly useful when you have several groups of questions each for respondents with different characteristics — for example, people who commute daily by train, people who use trains only occasionally, people who commute daily by car, people who use their cars only for nonbusiness purposes.

☞ There is a limit of 300 tests per logical expression on an *if* statement.

☞ See chapter 13, ‘Logical expressions’, for more information.

If and *else* are keywords and may not start in column 1, but any questions inside the braces (curly brackets) behave exactly as any other question — that is, their labels must start in column 1. In the syntax statement above, we have placed the { on the same line as the *if/else* keyword and have lined the } up with the *if/else* to which it belongs. The statements inside the braces are indented. Quancept does not enforce this layout, but we recommend that you use it because it makes it easy to see which statements belong in each group and to see where each group begins and ends. Where space allows, we will use this layout in our examples.

-
- ❖ If the braces contain only one statement, you may usually write the whole *if* statement all on one line; for example:

```
if (country <> 'UK') {goto ctn}
```

The exception is when the statement to be executed is an *include* statement. In this case, the *include* statement must be entered on a new line.

An *if* need not always be followed by an *else*, but an *else* must always be preceded by an *if*. In many cases, there will be actions to be carried out only if the logical expression is true. Since the *else* section would be blank, the parser and the interviewing program will accept an *if* clause by itself. In this situation, anyone not satisfying the logical expression simply continues with the next statement in the script after the } which closes the *if* section. Here is an example:

```
comm    ask 'Are you a daily commuter?'
        resp sp 'Yes' / 'No' go home
trans   ask 'Which mode of transport do you use most frequently?'
        resp sp 'Train' / 'Car'
        if (trans='Train') {
ticket  ask 'What type of ticket do you buy?'
        resp sp 'Single' / 'Return' / 'Daily Travelcard' /
                'Weekly Season' / 'Monthly Season' /
                'Annual Season' other
}
home   ask 'Are you able to do some of your work at home?'
        resp sp 'Yes' / 'No'
```

In this example, people who commute daily by train are asked which type of ticket they buy, whereas everyone else (people who do not commute daily by train) continues with the home question which follows the closing brace of the *if* clause. The people who commute by train answer the questions ticket and home.

If you want to ask the people who fail the *if* test a different question before they reach the home question, you can follow the *if* with an *else* clause:

```
comm    ask 'Are you a daily commuter?'
        resp sp 'Yes' / 'No' go home
trans   ask 'Which mode of transport do you use most frequently?'
        resp sp 'Train' / 'Car'
        if (trans='Train') {
ticket  ask 'What type of ticket do you buy?'
        resp sp 'Single' / 'Return' / 'Daily Travelcard' /
                'Weekly Season' / 'Monthly Season' /
                'Annual Season' other
}
        else {
qlnc    ask 'Do you ever use the train to travel to work?'
        resp sp 'Yes' / 'No'
}
```

```
home      ask 'Are you able to do some of your work at home?'
resp sp 'Yes' / 'No'
```

When you use *if* and *else* together in this way, the parser will not accept any statement, not even a comment which is normally ignored, between the closing brace of the *if* and the *else*. If you want to explain the purpose of the *else* clause, you must place the comment inside the braces. The example below illustrates the use of comments with *else*.

```
comment Daily commuters by train ....
    if (trans='train') {
ticket      ask 'What type of ticket do you buy?'
resp sp 'Single' / 'Return' /
        'Daily Travelcard' / 'Weekly Season' /
        'Monthly Season' / 'Annual Season' other
q2c      ask 'What is your general opinion of the service
offered for commuters?'
resp sp 'Excellent' / 'Very Good' / 'Satisfactory' /
        'Bad' / 'Appalling'
if (q2c='Bad' .or. q2c='Appalling') {
comment ... who think the service is bad or appalling
q3c      ask 'Why do you say that?'
resp coded
}
else {
comment ... who think the service is excellent/very good/satisfactory
q3cx      ask 'Even though you think the service offered
for commuters is '+q2c+' is there anything about it which you think
could be improved?'
resp coded
}
comment End of daily commuters by train
}
else {
comment People who are not daily commuters by train
q1nc      ask 'Do you ever use the train to travel to work?'
resp sp 'Yes' / 'No'
}
```

Suppose we have three people:

- | | |
|----------|--|
| Person A | commutes daily by train and thinks the service is bad. Person A is asked questions ticket, q2c and q3c. |
| Person B | commutes daily by train and thinks the service is very good. Person B is asked questions ticket, q2c and q3cx. |
| Person C | commutes but does not use the train. Person C is asked question q1nc. |

As the previous example shows, *if* and *else* may be nested. This means that you can accumulate characteristics on an incremental basis, each time reducing the number of respondents eligible to answer the next question. If you do this, you will need to be very careful where you put the braces; otherwise, your script may not do exactly what you expect.

If and *else* can be nested up to ten levels deep; that is, you can have ten *if(exp)* groups before the innermost group is terminated by a } line. To break out of any *if... else* group to a question outside the group, use *goto*. However, your script will often be easier to follow and to debug if you avoid nesting and break conditional statements into short groups instead.

Note that *else* always refers to the *if* at the corresponding level, as shown in our example. Where a nested *if* does not have an *else*, you may return to the higher level by entering just a } in place of the *else { ... }* clause:

```

outer    if (exp) {
d1          display 'this is the outer if-group'
inner      if (exp) {
            display 'this is the inner if-group'
        }
comment there is no else statement for the inner group, so we
comment terminate the inner group with a } sign. Without this,
comment the else would be attached to the inner if clause.
        }
        else {
d2          display 'this is the outer else-group'
    }

```

9.6 Rotating and randomizing questions



Quick Reference

To present questions in random order in a Quancept CATI interview, type:

```

ranstart
    questions
ranend

```

To present questions in rotation in a Quancept CATI interview, type:

```

rotstart
    questions
rotend

```

Sets of questions may be presented in rotated or random order during an interview. Here is an example of how to present questions in random order.

```
ranstart
floor  ask 'Cleanliness of train floors and windows'
       resp sp rating
station ask 'Cleanliness of stations'
       resp sp rating
rely   ask 'Reliability of service'
       resp sp rating
staff  ask 'Politeness and helpfulness of staff'
       resp sp rating
price  ask 'Price of tickets'
       resp sp rating
ranend
```

All the questions between *ranstart* and *ranend* will be presented in a different random order for each interview. This does not mean that the columns containing the responses will differ. The interviewing program assigns columns to questions in the order in which they appear in the script file.

A similar thing happens with *rotstart* and *rotend*; the questions enclosed by these words are presented in rotation, although the actual columns assigned to those questions remain unchanged.

The only other statement that is valid in a rotated or randomized group is *comment*.

-
- ❖ Do not include any type of routing or logical test in the group, not even as part of a response list, as this will only work if the question defining the routing is displayed before the question to which the routing refers.
-

The order of rotation with *rotstart* and *rotend* is not the same as that used by the *rot* keyword. In the first interview in a session, *rot* displays responses in the order they appear in the script and then moves the first response to the end of the list for the second interview. *rotstart/end* display the last question in the block first in the first interview, followed by all other questions in order. In the second interview, the last response moves back to the end of the list and the first question in the block is displayed first.

9.7 Rotating and randomizing blocks of questions



Quick Reference

To present single questions or groups of statements in rotation or in a random order in Quancept CAPI, Quancept Web and mrInterview interviews, type:

rotstart		ranstart
<i>statements</i>		<i>statements</i>
newrot	or	newran
<i>statements</i>		<i>statements</i>
newrot		newran
<i>statements</i>		<i>statements</i>
rotend		ranend

Rotation and randomization facilities in Quancept CAPI, Quancept Web and mrInterview are more flexible than in Quancept CATI. Rather than allowing only questions in the block, they allow other statements as well, such as assignments, logical tests and routing. When you write these types of blocks, you need to think carefully about how the statements in the block relate to one another and how rotation and randomization will affect the logic of the block. For example, suppose you have the following questions:

```

q19  ask 'Do you usually travel by car or by public transport?'
      resp sp 'Car' go q21 / 'Public transport'
q20  ask 'Question about public transport'
      resp .....
      goto ctn
q21  ask 'Question about cars'
      resp .....
      ctn  continue
    
```

The logic of the script requires that these statements remain in the order they appear here. If you put them inside a rotated or randomized block, the order will change and the results will not be what you intended.

In order that Quancept knows how to re-order the statements within a rotated or randomized block, you must precede each question or question group with a *newrot* or *newran* statement as appropriate. If there is more than one statement between each *newrot* or *newran* line, Quancept will treat those statements as a single statement and will reposition the group within the block, keeping the statements in the group in the order in which they are defined.

For example:

```
comment start of randomized block and first group
    ranstart
floor      ask 'Cleanliness of train floors and windows'
            resp sp rating
            if (floor='Poor' .or. floor='Very poor') {
p1          ask 'Why do you say this?'
            resp coded
        }
station    ask 'Cleanliness of stations'
            resp sp rating
            if (station='Poor' .or. station='Very poor') {
p2          ask 'Why do you say this?'
            resp coded
        }
comment start second group
    newran
rely       ask 'Reliability of service'
            resp sp rating
            if (rely='Poor' .or. rely='Very poor') {
p3          ask 'Why do you say this?'
            resp coded
        }
staff      ask 'Politeness and helpfulness of staff'
            resp sp rating
            if (staff='Poor' .or. staff='Very poor') {
p4          ask 'Why do you say this?'
            resp coded
        }
price      ask 'Price of tickets'
            resp sp rating
            if (price='Poor' .or. price='Very poor') {
p5          ask 'Why do you say this?'
            resp coded
        }
comment end of randomized block
    ranend
```

In this example, all the statements between *ranstart* and *newran* form a single group and must be presented in the order they appear here. The statements between *newran* and *ranend* form another group which must also be presented in the order they appear. However, the two groups may be presented in any order during an interview, so some respondents will be asked about cleanliness first whereas other will be asked about reliability, staff and price first.

If you want to use an identical series of logical tests several times within the same block, you can create a reusable script section in a library (include) file and include it from the main script. Statements in the included file must not have labels as these will not be unique if the file is used more than once. If you need to pass information to the included file, place that information in variables in the main script and these will be come available to the file whenever it is included. This option is not applicable to the example script because the logical test goes on to ask a question, and all questions must have labels.

 For more information on reusing statements, see section 25.1, Loading library files.

9.8 Testing for snapbacks



Quick Reference

To test for a snapback, place the following statement before the first question in your script:

set variable = ifsnap

Snapback is the term used in interviewing to describe going back from the current question to a previously answered question to check or change its answer. After a snapback, interviewers may step through each question in turn until they reach the point at which the snapback occurred, or they may enter a special response to return immediately to the original point in the script.

In certain circumstances, you may want to test whether an interviewer has snapped back to a question. This allows you, for example, to define one set of actions for the first pass through a script and another set for when questions are re-asked after a snapback.

The test for a snapback is achieved by adding an *ifsnap* assignment statement to your script before any questions are entered. The named variable will be true if the interviewer has snapped back and false if not. You may then test the value of this variable and define different sets of actions based on its value. For example:

```
comment 'snapping' becomes true if the interviewer snaps back
      set snapping = ifsnap

comment Save the start time of the interview
comment If the interviewer has snapped back, istart will retain
comment its original value from the first pass through the interview
      if (.not. snapping) {
          set istart = timeofday
      }
startt  fix (6) istart
```

```
comment Question cotype is always asked, regardless of snapbacks
comment Questions q1 and q2 are only asked on the first pass
comment through the interview; they are skipped on snapbacks
cotype ask 'What is your company''s main business?'
    resp sp 'Banking/Finance' / 'Insurance' / 'Computer software' /
        'Marketing/Public Relations' / 'Market Research'

    if(.not. snapping) {
11        for i = 'legal' / 'financial' / 'personnel'
            set iters = iteration
q1        ask 'Do you have a '+i+' department?@'
            resp sp 'yes' go q2 / 'no'
            set depts(iters) = 0
            goto nxt1

q2        ask 'Could you give me the name of the person in
charge of your '+i+' department?@'
            resp coded dk
            set depts(iters) = 1
nxt1        next
    }

comment Variable 'elapsed' holds the length of the interview in minutes,
comment including snapbacks
        set finish = timeofday
        set elapsed = finish - istart
finish    fix (6) elapsed
end
```

This simple example (provided as sample script B for this chapter) illustrates two uses of *ifsnap*:

- to protect data collected on the first pass through the interview
- to route

The question cotype will always be asked, regardless of whether or not the interviewer has snapped back. However, the questions about departments and heads of departments will be asked only if this is the first pass through the script.

The other point of interest in this script is the use of the temporary array called depts. An array is a single variable broken down into a number of cells. Each cell of the array stores a value of 0 if a company does not have a certain department or 1 if it does. In this example, dept(1) refers to the legal department, dept(2) refers to the financial department, and dept(3) refers to the personnel department. If the script contains questions about, say, the legal department, you could test the value of depts(1) and route respondents around this section of the script if their company doesn't have a legal department.

Questions involving loops and snapbacks can become very complex, particularly when interviews are stopped and restarted. The simplified example above, using *timeofday*, may not work properly under all circumstances. You should test your scripts thoroughly before using them on a live project. The next section shows a more complex example that ensures that the time of day written into the data files is always the time at which the *timeofday* statement was originally executed, not the time at which it was re-executed after a stop or snapback:

-
- ☞ To find out more about loops, see chapter 10, ‘Repetitive questions’, and chapter 11, ‘Iteration and subscription’.
 - For more about arrays, see section 11.5, Subscription of temporary variables.
 - The commands the interviewer types for snapbacks are described in section 33.27, Snapping backwards and forwards in an interview.
 - See also section 39.2, Environment variables, for information on QCNOCLEAN. It is used to keep all data, including responses to any questions that go off-path.
-

In summary, *ifsnap* returns a value of 1 in the following cases:

- After rewinding to the top of the script and retracing the script in order to locate the question specified on the interviewer’s snap command. This applies to both backward and forwards snaps, and happens whether or not the snap is successful. For example, if the interviewer types <q6 and routing prevents q6 being reached, *ifsnap* is still set to 1 because a snap was attempted.
- If the question currently displayed already has an answer because the interviewer snapped back to an earlier question, or because a *goto* statement in the script routed to an earlier question, and the interviewer is now stepping forwards through the script. In this case, *ifsnap* remains set to 1 until the interviewer reaches a question after the one from which the original snap was made. The presence of *pause*, *goto*, subsurveys and interactive callable functions before the first unanswered question has no effect on the value of *ifsnap*.

-
- ☞ See ‘Routing around ifsnap assignments’ for details.
-

- When a stopped interview is restarted and qtip is walking through the stopped part of the interview to find the first unanswered question.
- When interviews are being reviewed and the reviewer skips to the end of the interview.

Timeofday and snapbacks

 *timeofday* returns the time at which the statement was executed and is normally used as a means of fixing into the data the time at which the interview took place. If the interviewer snaps back during the interview, the interviewing program silently walks through the script from the beginning up to the question to which the interviewer returned. The responses saved for questions are re-assigned to the question variables and any *set* statements are re-executed. This means that the time of day will now be the time at which the statement was re-executed rather than the original time.

The same thing happens when an interview is stopped and restarted.

The script shown below illustrates a way of storing the time of day in the data without having it altered by snapbacks or stopping and restarting the interview.

```
comment Response list is 00:00am to 23:59:59pm
tim1 dummyask 'store time of day'
    resp num 0 to 235959
comment If time of day has already been saved, copy it into x1
    if (tim1 <> null) {
        set x1 = tim1
    }
comment Check for snapbacks. If this is the first time through the
comment script, save the time of day in x1
    set sb = ifsnap
    if (.not. sb) {
        set x1 = timeofday
    }
comment At this point, x1 contains the original time of day, either
comment because it has just been set for the first time, or because it
comment has been copied out of the dummy question in which it was held.
comment Now copy x1 into tim1.
    set tim1 = x1
```

Routing around ifsnap assignments

If a script contains an *ifsnap* assignment followed at some point by a *goto* statement that returns the interview to a point before the *ifsnap* assignment, the value of *ifsnap* when it is reached the second time will be 1 if the last question displayed before the assignment already had a response when it was displayed. As an example, consider the following script:

```
q1    ask 'Question 1'
      resp sp 'red' / 'green' go close
q2    ask 'Question 2'
      resp sp 'red' / 'green' go close / 'yellow'
      set x = ifsnap
      - other questions -
      goto exit
close display 'Thank and close'
      pause
      goto q1
exit  end
```

In the first interview, the respondent chooses green at q1 and jumps directly to the close statement, bypassing the *ifsnap* assignment. When the respondent is returned to q1, he changes his response to red. This changes the interview path and q2 is displayed for the first time. The respondent chooses yellow at q2. This time the *ifsnap* assignment takes place and x is set to 0, after which the interview ends. This is expected behavior because *ifsnap* was not on the interview path at the time the *goto* was executed.

In the second interview, the respondent chooses red at q1 and then green at q2. The interview jumps directly to the close statement, bypassing the *ifsnap* assignment. When the respondent is returned to q1, he leaves the original answer unchanged and moves on to q2. The current answer to q2 is green but the respondent changes this to yellow. This changes the path through the interview and the *ifsnap* assignment is executed. *The unexpected behavior is that x is set to 1 because as far as qtip is concerned, a snapback has taken place.* The reason that qtip thinks this is because the last question displayed before the *ifsnap* assignment already had an answer when it was displayed for the second time.

If you do not want *ifsnap* to be set with this type of routing, you should unset all the questions between the routing destination and the *ifsnap* assignment (q1 and q2 in this example).

9.9 Returning to the SMS menu



Quick Reference

To return the interviewer to the SMS menu, type:

gotosms

When the SMS menu is displayed at the start of an interview, it contains an option for proceeding with the interview. When you return to the menu during an interview, this option is no longer relevant so it is suppressed. However, other options may be available that were not available at the start of the interview.

Being able to return to the SMS menu is a useful facility in scripts which have long or complex screeners. It lets you:

- ensure that the data file contains data only for interviews in which the respondent answered more than just the screening questions,
- dispose of phone numbers belonging to unsuitable respondents by directing the interviewer to select from the SMS menu a code which places those numbers in, for example, the file of dead (unwanted) numbers,
- arrange a callback for when the required respondent will be available without having to save any data for the screening questions asked so far.

-
- ☞ You may not use *gotosms* as a means of stopping the interview to be restarted at the same place later on. Although this may appear to work in some scripts, in others it will not work at all, since the facility was not designed with this use in mind. If you want the interview to be saved and restarted from the point at which it was stopped, you must use a *stop* statement and have the interviewer arrange the callback without returning to the SMS menu.
 - ☞ Stopping interviews is discussed in chapter 16, ‘Ending the interview’.
-

Consider the following script:

```
intro    protect 'Hello, I''m calling from SPSS MR . . . . '
          screener
qa       ask 'Do you work on the railways? Is anyone in your immediate
          family associated with this field?'
          resp sp 'Yes' go term1 / 'No' nodata
qb       ask 'Do you travel regularly by train / may I speak with
          someone who regularly travels by train?'
          resp sp 'respondent travels regularly by train' /
          'no one travels regularly by train' go term1 /
          'regular train traveler not available' go term2
          nodata
qc       ask 'INTERVIEWER: If new respondent, repeat introduction,
          then ask:@@
          Have you traveled by train during the last two weeks?'
          resp sp 'Yes' go ctrl1 / 'No' nodata
term1   display 'INTERVIEWER: Terminate with No suitable respondent'
          pause
          gotosms
term2   display 'INTERVIEWER: Arrange a callback'
          pause
          gotosms
ctrl1   main
```

This script contains three screening questions whose purpose is to select respondents who travel regularly by train, and who have traveled by train during the last two weeks, and who do not belong to households containing someone who works for the railways. Interviews with respondents who fail any of these requirements are terminated. For example, if no one in the household travels regularly by train, the interviewer sees a message telling him/her which result code to select, and the SMS menu is displayed. Once the interviewer selects the result code for 'No suitable respondent', the number is disposed of as defined in SMS and the next number to call is selected. No data is added to the data file for that interview.

We use a similar method to deal with callbacks for suitable respondents who are not currently available. The script returns to the SMS menu and the interviewer selects the option to arrange a callback for a later time.

Without *gotosms*, you would need to use routing statements to reach the *end* statement. This would cause a record containing just a respondent serial number and card type to be added to the data file.

9.10 Actions for test interviews only (CATI)



Quick Reference

To check whether the script is running in automatic test mode, type:

set variable = testingrun

When you are testing a script in automatic mode, the interviewing program decides which answers to give to each question, which in turn determines the route taken through the script. There are two situations when this may be unsatisfactory:

- When the script starts with screening questions and the interview is terminated if respondents fail these questions. If the eligibility conditions are tight, you may find that you have many interviews terminated in the screener when the main purpose of the run is to test the main section of the script.
- When the script contains open ends whose formats are checked to ensure they match a specified pattern (for example, postcodes or zip codes). The interviewing program has no way of knowing what is required in a particular open end, so it enters a dummy text with a serial number and accepts this as a valid response format. If the script later contains statements whose execution depends on the text having a particular value, these statements will not be executed because the text will never satisfy the test conditions.

In these cases, you will want some way of overriding the interviewing program's own selections with responses of your own, without affecting the way the script works when not in test mode. The keyword associated with this facility is **testingrun**, and you use it as follows:

```
set variable = testingrun
if (variable) {
    statements
}
```

If you are running the interviewing program in automatic test mode, the variable will be true and the statements inside the braces will be executed. When interviewers are running the interviewing program in the standard way, the variable will be false and the statements will be ignored.

Here is the screening section of a script showing how you can bypass the statements which terminate the interview:

```
qb      ask 'Do you travel regularly by train / may I speak with
           someone who regularly travels by train?'
           resp sp 'respondent travels regularly by train' /
                  'no one travels regularly by train' go term1 /
                  'regular train traveler not available' go term2
           nodata
term1   display 'INTERVIEWER: Terminate with No suitable respondent'
           pause
```

```
comment Check whether this is a testing run. If so, don't terminate
comment the interview
    set istest = testingrun
    if (istest) {
        goto ctrl
    }
    gotosms
term2  display 'INTERVIEWER: Arrange a callback'
pause
if (istest) {
    goto ctrl
}
gotosms
ctrl   main
```

For open ends whose text formats are checked, you could assign a standard value or one of a standard set of values which will always pass the test. For example:

```
pcode  ask 'What is your post code?'
      resp coded
      if (istest) {
          set pcode = 'NW6 2EG'
      }
```

☞ See section 34.1, Automatic testing, for information on how to test scripts in automatic mode.

9.11 Actions for test interviews only (mrInterview)



Quick Reference

To define actions that apply to test interviews but not live interviews, place the following statements in the script:

```
if (ststest) {
    statements
}
```

When you activate a project, you specify whether the project's status is Test, Active or Inactive. The activation process stores this information in the project's DataCollection.Status variable. The *ststest* variable in the script makes the value of this variable available during the interview and allows you to define actions that apply only to test interviews.

A typical example is to display a Stop button for test interviews so that you can stop the interview in a controlled manner if you find the script does not produce the results you expected, rather than leaving the interview to time out or forcibly terminating the interview. When the project is reactivated as an active project, *ststest* will always return a False value so the Stop button will not be displayed.

Here is the code that displays a Stop button for test interviews:

```
if (ststest) {  
    set hidestop = 0  
}
```

9.12 Sample scripts

This section provides two sample scripts to illustrate routing and flow control. Sample script A contains examples of all the statements and techniques covered in this chapter. Sample script B is for CATI only and is designed to illustrate the use of *ifsnap* with *timeofday* to time how long an interview or portion of an interview took to complete.

Script A

```
comment Sample script A for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 9, Routing and Flow Control Statements

comment CATI/CAPI only: Define getout as special interviewer response
comment recognized anywhere. Comment out for Web/mrInterview.
    onresp 'getout' go exit

rating define 'Excellent' / 'Very good' / 'Good' / 'Acceptable' /
    'Bad' / 'Very bad' / 'Totally unacceptable'

intro protect 'Hello, I''m calling from SPSS MR. I wonder if you
can spare some time to answer a few questions about train travel.'

comment Routing as part of the response list
qa      ask 'Do you work on the railways? Is anyone in your immediate
family associated with this field?'
        resp sp 'Yes' go term1 / 'No' nodata
qb      ask 'Do you travel regularly by train / may I speak with someone
who regularly travels by train?'
        resp sp 'respondent travels regularly by train' /
            'no one travels regularly by train' go term1 /
            'regular train traveler not available' go term2
            nodata
qc      ask 'INTERVIEWER: If new respondent, repeat introduction, then
ask:@@ Have you traveled by train during the last two weeks?'
        resp sp 'Yes' go ctrl1 / 'No' nodata

comment CATI with SMS only: Return to SMS menu
comment CATI without SMS/CAPI/Web/mrInterview: comment out this section and
comment remove the comments from the two 'goto exit' statements
term1  unprotect intro
nsr    display 'INTERVIEWER: Terminate with No suitable respondent'
        pause
        gotosms
term2  unprotect intro
cbk   display 'INTERVIEWER: Arrange a callback'
        pause
        gotosms
```

```
comment CATI without SMS/CAPI/Web/mrInterview: Terminate interview instead
comment term1    unprotect intro
comment         goto exit
comment term2    unprotect intro
comment         goto exit

ctn1    unprotect intro

comment CATI/mrInterview only: Test for snapbacks
comment CAPI/Web: comment out the 'set' statement, the 'if' line
comment and the line containing '}'
      set snapback = ifsnap
      if (snapback) {
region   ask 'Which rail region do you live in?'
      resp sp 'South London' / 'North West Kent' /
              'Western' / 'Chilterns' / 'North London' /
              'Anglian' other
}
london  ask 'Do you ever travel to London by train?'
      resp sp 'Yes' / 'No'

comment Conditional routing outside response lists
      route (london = 'No') go comm

indest  ask 'Which station do you arrive at on your inward
journey?@INTERVIEWER: If station varies, code the one used most often.'
      resp sp 'Kings Cross' / 'Euston' / 'Victoria' /
              'Waterloo' / 'Charing Cross' / 'Cannon Street' /
              'Marylebone' / 'Paddington' / 'Moorgate' /
              'St Pancras' other
length   ask 'How long does this train journey take?'
      resp sp 'Less than 15 minutes' / '15 to 30 minutes' /
              '30 to 45 minutes' / '45 minutes to 1 hour' /
              'Over 1 hour'
comm    ask 'Are you a daily commuter?'
      resp sp 'Yes' go trans / 'No'

comment Mandatory routing outside a response list
      goto ctn2

trans   ask 'Which mode of transport do you use most frequently?'
      resp sp 'Train' / 'Car'

comment Flow control with if/else
      if (comm='yes' .and. trans='train') {
ticket   ask 'What type of ticket do you buy?'
      resp sp 'Single' / 'Return' / 'Daily Travelcard' /
              'Weekly Season' / 'Monthly Season' /
              'Annual Season' other
```

```

q2c      ask 'What is your general opinion of the service
offered for commutes?'
        resp sp 'Excellent' / 'Very Good' / 'Satisfactory' /
        'Bad' / 'Appalling'
        if (q2c='Bad' .or. q2c='Appalling') {
q3c      ask 'Why do you say that?'
        resp coded
    }
    else {
q3cx      ask 'Even though you think the service offered for
commutes is '+q2c+' is there anything about it which you think
could be improved?'
        resp coded
    }
else {
q1nc      ask 'Do you ever use the train to travel to work?'
        resp sp 'Yes' / 'No'
    }

ctn2      continue

p1      protect 'Now I''m going to ask your opinion on various
aspects of the train service. For each one I''d like you to tell me
whether it is excellent, very good, good, acceptable, bad, very bad or
totally unacceptable.@@'

comment Start random presentation of questions
comment CAPI/Web/mrInterview: uncomment the newran statement.
        ranstart
comment Start of first group
floor      ask 'Cleanliness of train floors and windows'
        resp sp rating
seats      ask 'Cleanliness of seats'
        resp sp rating
station    ask 'Cleanliness of stations'
        resp sp rating
comment Start of second group
comment     newran
rely       ask 'Reliability of service'
        resp sp rating
staff      ask 'Politeness and helpfulness of staff'
        resp sp rating
food       ask 'Availability of refreshments'
        resp sp rating
price      ask 'Price of tickets'
        resp sp rating
comment End random presentation of questions
        ranend
        unprotect p1

```

```
thank      display 'That''s the last of my questions. Thank you for your  
help and goodbye.'  
exit      end
```

Script B

```
 comment Sample script B for Quancept & mrInterview Scriptwriter's Manual  
comment Chapter 9, Routing and Flow Control Statements  
comment CATI and mrInterview only: Demonstrates ifsnap  
  
comment 'snapping' becomes true if the interviewer snaps back  
      set snapping = ifsnap  
  
comment CATI only: Save the start time of the interview.  
comment For mrInterview, comment out the whole if clause.  
comment If the interviewer has snapped back, istart will retain  
comment its original value from the first pass through the interview  
      if (.not. snapping) {  
          set istart = timeofday  
      }  
startt  fix (6) istart  
  
comment Question cotype is always asked, regardless of snapbacks  
comment Questions q1 and q2 are only asked on the first pass  
comment through the interview; they are skipped on snapbacks  
cotype  ask 'What is your company''s main business?'  
      resp sp 'Banking/Finance' / 'Insurance' / 'Computer software' /  
            'Marketing/Public Relations' / 'Market Research'  
  
      if(.not. snapping) {  
l1       for i = 'legal' / 'financial' / 'personnel'  
          set iters = iteration  
q1       ask 'Do you have a '+i+' department?@'  
          resp sp 'yes' go q2 / 'no'  
          set depts(iters) = 0  
          goto nxt1  
  
q2       ask 'Could you give me the name of the person in  
charge of your '+i+' department?@'  
          resp coded dk  
          set depts(iters) = 1  
nxt1     next  
      }  
  
comment CATI only: Variable 'elapsed' holds the length of the interview  
comment in minutes, including snapbacks For mrInterview, comment out the  
comment next three lines.  
      set finish = timeofday  
      set elapsed = finish - istart  
finish   fix (6) elapsed  
end
```


10 Repetitive questions

Repetitive questions are questions that are asked more than once in an interview. You will often find them in brand studies, for example, where respondents are asked for their opinions on a number of brands. Sometimes the question text will be the same each time the question is asked; at other times it will vary slightly, perhaps by showing the name of the current brand.

The Quancept language offers a loop facility which allows you to define repetitive questions once only and to include additional statements which indicate how many times and in what order the questions should be displayed. Keywords for loops that are described in this chapter are:

colgrid/rowgrid	display responses in a grid format
for	start a loop
grouped	control how loop data is saved
next	end a loop
scrtmout	time-out delay for infinite loops
scrtmpas	time elapsed since last page was displayed

The chapter ends with some suggestions for how to improve the efficiency of your scripts and how to use loops to reduce the size of the data maps.

10.1 How loops work

Before we discuss the keywords to use for loops, it is important to understand the concepts on which loops are based. A loop starts with a statement which determines the number of times the loop will be repeated. This is done by specifying a numeric range or a list of texts and using a special variable, which we call a *pointer*, to point to each value or text in turn. The first time the loop is started, the pointer will point to the first value or text, the second time it is started, it will point to the second value or text, and so on. When the pointer is pointing to the last value or text, and the last statement in the loop is reached, the interviewing program knows that the loop is finished and will continue with the statement immediately after the loop.

Loops are terminated by a special keyword that sends the interviewing program back to the statement that marks the start of the loop. The pointer is moved to the next value or text in the list, as described above.

The interviewing program keeps track of the number of times a loop has been started and the number of times the statements inside the loop have been executed. The number of times the loop has been started is called the *iteration*; the number of times a question has been asked is called the *subscript*. The subscript and iteration are not always the same..

☞ We explain the terms *iteration* and *subscript* in more detail and show how you can use them in scripts in chapter 11, ‘Iteration and subscription’.

10.2 Writing loops

Quick Reference

To define a set of repetitive questions or actions, enclose those statements in the words **for** and **next**:

```
[label] for variable = value_list  
        statements  
next
```

Labelling *for* statements is optional in Quancept and mandatory in mrInterview.

Each of the statements enclosed by the *for* and *next* lines is repeated once for each item in the value list. If there are three items in the list, the loop is repeated three times.

The value list may take several forms. With numeric ranges, you would type, for example:

```
loop1 for persnum = 1 to 9
```

to repeat the loop nine times.

Lists of nonconsecutive numbers have the numbers separated by slashes:

```
loop1 for months = 6 / 12
```

The statement repeats the loop twice, once for 6 months and once for 12 months.

When the value list consists of texts, it resembles a *sp/mp* response list — each text is enclosed in single quotes and separated by a slash, and may be followed by a unique ID (highly recommended for mrInterview). The statement:

```
loop1 for sections = 'home news' ('home') / 'foreign news' ('foreign') /  
                    'editorial' ('editorial')
```

repeats the loop three times, the first time for home news, the second time for foreign news and the third time for editorial.

There are three options if the loop refers to a defined list, all of which mirror the syntax for using defined responses lists with questions. Examples are:

```
loop1 for flavor = master  
loop2 for flavor2 = q1 in master  
loop3 for flavor3 = not q2 in master
```

The formats of *for* statements for loops with defined lists look similar to those of response lists which use defined lists, and they work in the same way.

-
- ☞ For more information on using loops with defined lists see the section entitled, ‘Specified other and defined lists with loops’ later in this chapter.
-

The pointer indicates the current position in the value list. Let’s look at an example:

```
sect  for sections = 'home news' / 'foreign news' / 'editorial'
freq   ask 'How often do you read the '+sections+ ' section of this
newspaper?'
      resp sp 'All the time' / 'More often than not' /
             'Sometimes' / 'Never'
      next
```

We have already explained how the pointer, called ‘sections’, points to each section of the newspaper in turn. We can substitute the value of the pointer in the question text using the standard text substitution rules.

-
- ☞ The standard text substitution rules are discussed in section 8.7, Displaying data in texts.
-

The first time the question is asked, it will be displayed as ‘How often do you read the home news section of this newspaper?’ When the respondent has replied, the pointer moves to foreign news and the question is displayed as ‘How often do you read the foreign news section of this newspaper?’

The loop is repeated once more for the editorial section. The interviewing program knows when it has reached the end of the loop for the last time because it will have reached the end of the value list. The interviewing program then leaves the loop and asks the question which comes immediately after the line containing the word *next*.

The value list may contain 1000 items.

- ☎ During an interview, the interviewing program retains the first 256 characters of the texts which control the loop (for example, home news, foreign news, editorial). If the number of characters in the texts in the value list is greater than 256, you will need to ask the interviewing supervisor to set the environment variable QCLOOPLONG to an appropriately higher value for each interviewer working on this project. This may be particularly important if the loop is controlled by respondent-specific texts, as described below.

-
- ☞ For more information about QCLOOPLONG, see section 39.2, Environment variables.
-

Labelling for statements



Labelling *for* statements in scripts for mrInterview is equally as important as defining unique IDs for responses. When a question appears in a loop, the name of its variable in the case data is created from the label of the *for* statement, the question's name and the unique ID of the current item in the loop's value list. For example, if the loop is defined as:

```
sect  for sections = 'home news' ('home') / 'foreign news' ('foreign') /  
      'editorial' ('editorial')  
freq   ask 'How often do you read the '+sections+ ' section of this  
newspaper?'  
      resp sp 'All the time' ('all') / 'More often than not' ('often')/  
      'Sometimes' ('sometimes') / 'Never' ('never')  
      next
```

the labels for freq will be sect[{home}].freq for the first iteration, sect[{foreign}].freq for the second, and sect[{editorial}].freq for the third.

If the loop control list is numeric, the unique IDs of the control values are replaced by the numeric values. So, for example:

```
loop2  for week = 1 to 2  
q9       ask 'Which product did you try during week '+week+'?  
      resp sp 'Product A' ('proda') / 'Product B' ('prodB')  
      next
```

generates variables with the names loop2[1].q9 and loop2[2].q9.

Loops with ranges

Quick Reference

To execute a loop for certain values within a numeric range, define the increment with **step**:

[label] for variable = lowest to highest step increment

When using ranges in a value list, you might find the **step** facility useful. It allows a range to be specified, but only uses certain numbers within that range, as shown in the example below:

```
lpos  for pos = 10/20/30 to 40 step 2/70
```

pos takes the values 10, 20, 30, 32, 34, 36, 38, 40 and 70 successively. This is because we specified '30 to 40 step 2'. If we had specified '3 to 18 step 3', pos would take the values 3, 6, 9, 12, 15 and 18.

If the highest value in the range is not exactly divisible by the step value, the loop is repeated for values up to the highest value in the range which is divisible by that number. For example:

```
for pos2 = 1 to 7 step 4
```

repeats the loop twice, the first time for pos2=1 and then again for pos2=5.

Specified other and defined lists with loops

A *for* statement of the form:

```
loop1 for paper = read in master
```

is an extremely efficient way to ask a series of repetitive questions about a set of items chosen from a defined list. When the parser allocates columns to questions in the loop, it assumes that the loop will be repeated as many times as there are items in the master list. However, when the interviewing program conducts the interview, it will execute the loop only as many times as there are answers given to the question called 'read'.

If the defined list contains specified other, the interviewing program will substitute any answers recorded via that keyword in place of specified other when the loop is executed. If the interviewer writes down the response instead of typing it, the interviewing program will display the words 'answer at response *n*', where *n* is the serial number of the open end instead, and the interviewer will have to read the answer from his/her handwritten sheet. If the respondent chooses several other flavors, they will all be substituted at once because they are treated as a single response.

Consider the following script:

```
papers define 'The Daily Bugle' / 'The Morning Herald' /  
    'The Informer' / 'Today''s Gossip' / 'News and Views' other  
read ask 'Which daily newspapers do you read regularly?@  
INTERVIEWER: Record only first other paper mentioned'  
        resp mp papers other  
loop1 for pub = read in papers  
ratep ask 'On a scale of 1 to 5, how would you rate  
    '+pub+'. Please use 1 for very good and 5 for very bad'  
        resp num 1 to 5  
next
```

If the respondent regularly reads the Informer and the Worker's Daily, the loop will be repeated twice. The first time, the interviewing program will substitute the text 'The Informer' in place of the pub variable in the rating question. The second time, it will substitute the words:

```
<OTHER>:The Worker's Daily
```

If the response to specified other continues over several lines, the interviewing program will reformat it to fit in with the rest of the question text and will mark the original line breaks with | signs.

The interviewing program treats a specified other response as a single answer even though it may contain a number of answers (for example, the respondent names more than one newspaper that does not appear in the list). Because of this, it is not possible to separate the individual responses to specified other.

-
- ☞ If you do not want answers given with specified other to be preceded with <OTHER>, you may replace specified other with other as an ordinary quoted response and add an extra question to prompt for the unlisted answer. If you write the papers out as a set of quoted texts in the loop value list, you can then substitute the value of the extra question as a text in the list using the notation [+qname+].
 - ☞ There is an example of this in section 10.8, Respondent-specific texts in value lists.
-

Multiask in loops



multiask displays two or more questions on the screen at the same time. You can use *multiask* in a loop, but remember the following points:

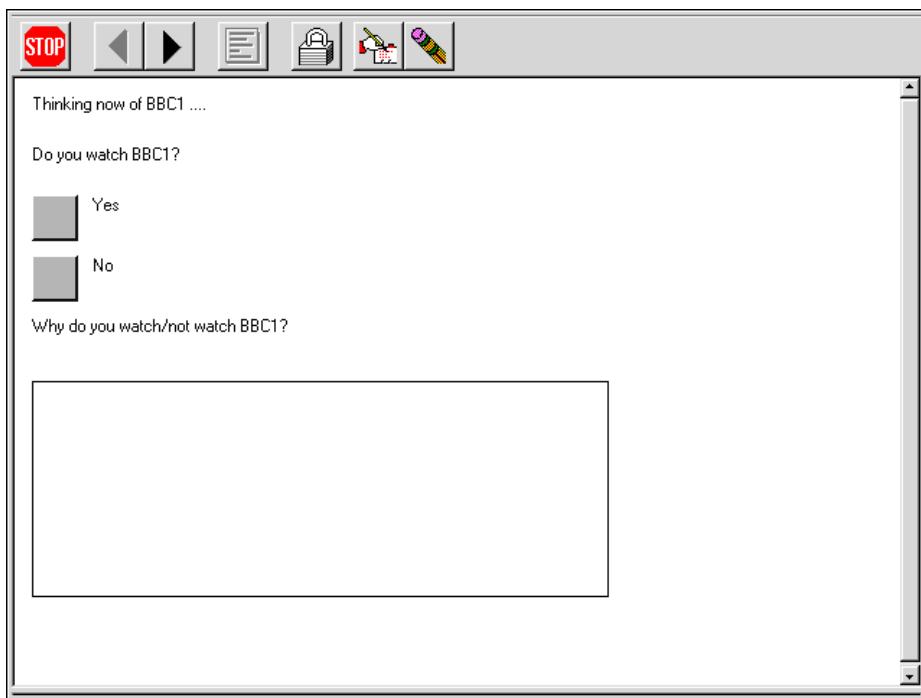


- The dummy questions must be defined in the same loop.
- The references to the dummy questions on the *multiask* statement must be subscripted.

The following example uses *multiask* in a loop:

```
chan   for channel = 'BBC1' / 'BBC2' / 'ITV' / 'Channel 4'  
watch  dummyask 'Do you watch '+channel+'?'  
       resp sp 'Yes' / 'No'  
hours  dummyask 'Why do you watch/not watch '+channel+'?'  
       resp coded  
both   multiask 'Thinking now of '+channel+' ....'  
       watch(*) / hours(*)  
next
```

This is displayed in Quancept CAPI as:



Nested loops

Any *for ... next* loop can itself contain other loops. This is called ***nesting***. These loops may be nested up to ten levels deep (that is, ten *fors* before the first *next*), but you should always be sure to check that each *for* has a *next* otherwise, the script will not work. Here is an example of a nested loop:

```

pubs      for pub = 'The Daily Bugle' / 'The Morning Herald' / 'The Informer'
ratep      ask 'On a scale of 1 to 5, how would you rate
           '+pub+'. Please use 1 for very good and 5 for very bad'
           resp num 1 to 5
comment Ask about each section of the current newspaper in turn
sect       for sections = 'home news' / 'foreign news' / 'editorial' /
           'arts' / 'leisure activities' / 'features' /
           'health/medical' / 'education'
freq       ask 'How often do you read the '+sections+ ' section of
           this newspaper?'
           resp sp 'All the time' / 'More often than not' /
           'Sometimes' / 'Never'
rate       ask 'On a scale of 1 to 5, how would you rate that
           section. Please use 1 for very good and 5 for very bad'
           resp num 1 to 5
           next
next
  
```

The outer loop is concerned with the different newspapers listed on the *for* statement. The pointer, *pub*, points to each newspaper in turn. There is just one rating question in the outer loop before the start of the inner loop. The inner loop asks the respondent to rate the various sections of the current newspaper on a scale of 1 to 5. When the interviewing program finishes the inner loop for the first publication, it leaves that loop and returns to the start of the outer loop ready to get the name of the next newspaper in the value list.

-
- ☞ To find out about other keywords for nested loops, see section 10.6, Column allocation with loops.
-

10.3 Routing with loops

Routing within loops is almost the same as routing used anywhere else in your script. However, there are some important points which you must remember:

- Routing into a loop is forbidden by the parser. If you wish to ask a question in a loop, you must make sure that you go to the *for* statement and follow it through to the end. If you do not want to execute the loop for all items in its value list, you will need to include a routing statement immediately after the *for* statement to check the current item in the value list and route to the end of the loop if it is not a value that you want.
- Do not use *go* in a response list to route out of a loop. The only safe way to break out of a loop before its completion is to use *goto* and a statement name, or *route* followed by a logical expression. The pointer variable will then retain its current value. This means that if this is the second pass through the loop, the pointer variable will retain the value of the second item in the value list.
- Do not skip backward out of a loop to a statement before the start of that loop because the results of re-entering that loop are unpredictable.

There will be times when you want a certain response to a question to cause the program to skip to *next*, even though that question may not be the last question in the loop. You cannot tell the interviewing program to skip to *next* because it is a keyword, not a label. Instead, you give *next* a label and tell the program to route to the given label:

```
sect      for sections = 'home news' / 'foreign news' / 'editorial' /  
                  'leisure activities' / 'features'  
freq      ask 'How often do you read the '+sections+ ' section of  
your newspaper?'  
          resp sp 'All the time' / 'More often than not' /  
                  'Sometimes' / 'Never'  
comment Check the answer to a question in the loop  
        route (freq(*) = 'Never') go nxtsec  
rate      ask 'On a scale of 1 to 5, how would you rate that  
section. Please use 1 for Very good and 5 for Very bad'  
          resp num 1 to 5  
nxtsec    next
```

In this example, the respondent is asked to rate various sections of a daily newspaper on a scale of 1 to 5. If the respondent never reads a particular section, we assume that he/she has no interest in it and therefore skip the rating question. The statement that does this is the *route* statement. This checks the answer given to the freq question and routes to the statement labeled nxtsec. This is *next*, which returns to the start of the loop and moves the pointer to the next section name.

The (*) after the question name in the routing statement enables us to differentiate between the answers given to the rating question each time it was asked. Why do we need this? As you know, you can refer to a response by giving its question name — in this example, to find out whether the respondent reads the home news section, you look at the value of freq. However, you can also look at this variable to find out whether the respondent reads the foreign news, sports or editorial sections, among others.

When the interviewing program finds a question in a loop, it looks at the *for* statement to see how many different answers or sets of answers that question may have. In this example, there are five section names, so each question may have up to nine answers. The interviewing program differentiates between each of these answers by giving it a number according to its position in the list — home news is 1, foreign news is 2, and so on. To find out whether the respondent reads the home news section, you need to give the question name and the position of home news in the loop list — that is, freq(1).

Because we are writing the loop in general terms rather than specific to one section, we use the * to represent a section's position in the list. When we are asking about home news, which is the first section in the list, the interviewing program substitutes 1 in place of the asterisk; when we are asking about foreign news, which is the second section, the interviewing program substitutes 2 in place of the asterisk, and so on. Thus, on the first pass through the loop, the *route* statement will be read as if it had been written:

```
route (freq(1) = 'Never') go nxtsec
```

and will refer to home news; on the third pass it will be read as:

```
route (freq(3) = 'Never') go nxtsec
```

and will refer to the editorial section.

We call this method of referring to questions in loops *subscription*.

☞ You will find further information about subscription and more examples in chapter 11, 'Iteration and subscription'.

Terminating interviews with infinite loops



Quick Reference

To define the time-out delay for infinite loops, type:

```
set scrtmout = value
```

where *value* is the number of milliseconds to wait before terminating the interview.

To refer to the amount of time that has elapsed since the previous page was displayed, use the **scrtmpas** variable.

Infinite loops usually occur when a loop contains a *go* or *goto* statement that routes to a location within the loop; for example:

```
lp1    for i = 1 to 10
      xx      continue
              goto xx
      next
```

The check for an infinite loop is made whenever a go, goto or next statement is executed, and involves checking the amount of time that has passed since the previous page was shown. If this amount of time exceeds the time-out value specified in *scrtmout* then the interview is terminated. The default time-out value for infinite loops is 55000 milliseconds (55 seconds), and this value is independent of the main interview time-out setting that is stored in the registry. If you set *scrtmout* to zero then no time-out checking for loops takes place.

If you need to refer to the amount of time that has elapsed since the previous page was displayed, use the variable *scrtmpas*. The interviewing program sets this variable every time the time-out check is done, but never reads it.

When a time-out occurs, a ‘script timeout error’ message is written to the log file showing the values of *scrtmout* and *scrtmpas*, the keyword that caused the error, and the line at which the offending keyword occurs in the script.

10.4 Rotation, randomization and alphabetical sorting

rot and **ran** are the keywords associated with rotational and random presentation of responses in a response list, and **atoz** sorts responses in alphabetical order. They may equally well be used in loops to cause the values for a loop to be implemented in rotation or at random. For example:

```
pubs  for pub = 'The Daily Bugle' / 'The Morning Herald' /
          'The Informer'  rot
ratep    ask 'On a scale of 1 to 5, how would you rate
'+pub+'. Please use 1 for Very good and 5 for Very bad'
      resp num 1 to 5
next
```

The first time this loop is used (that is, when the first respondent is interviewed), the respondent may be asked about the Informer, then about the Daily Bugle and finally about the Morning Herald. The next respondent may be asked about the Morning Herald, The Informer and the Daily Bugle, in that order. When the third respondent is asked these questions, he/she will first be asked about the Daily Bugle, then about the Morning Herald and finally about the Informer.

Using the keyword *ran* causes questions to be asked about the different newspapers in a random order. For example, the first respondent may be asked about the Daily Bugle, the Informer and the Morning Herald, whereas the next respondent may be asked about the Morning Herald, the Daily Bugle and the Informer.

Data is always saved according to the order of items in the loop list. Thus, answers about the Daily Bugle will be followed by those for the Morning Herald and then the Informer.

 If the values in the loop list are texts, you may also attach the keywords **rottranfix** and **rottransub** to individual values in the loop. Use *rottranfix* to force a value to retain its original position in the value list, or *rottransub* to define a group of values that is to be kept together when the other values in the list are re-ordered.

↳ See section 5.1, Displaying responses in alphabetical order, for more information about *atoz*.

See section 5.7, Rotation and randomization of responses, for more information about *rot*, *ran*, *rottranfix* and *rottransub*.

See section 10.6, Column allocation with loops, for further information on how the parser allocates columns to questions in loops.

10.5 Using freeze with loops

Where all questions in the loop share the same response list, you may use the keyword **freeze** on the first response list. The list will be displayed at the top of the screen and will remain in operation for all iterations of the loop as long as it is not unfrozen by the keyword *unfreeze* or another *resp* statement.

-
- ☞ See section 5.3, Response lists referring to several questions, for further information about *freeze*.
-

10.6 Column allocation with loops



Quick Reference

If you want responses to appear in the data in the order in which they are given rather than the order in which the questions appear, append the word **grouped** to the loop's value list:

```
for variable = value_list grouped
```

The parser allocates columns to questions in the order in which those questions appear in the loop, so responses are grouped by question rather than the order in which they were presented. For example, suppose we have three questions in a loop which is to be repeated twice. The interviewing program stores all responses to the first question before the responses to the second question, as follows:

one(1) one(2) two(1) two(2) three(1) three(2)

If you use *grouped* to alter this so that responses appear in the data in the order in which they are given, responses will then be stored as follows:

one(1) two(1) three(1) one(2) two(2) three(2)

In both cases, one, two and three are the question names, and the numbers in parentheses refer to the number of times the loop has been repeated. If there is only one question in the loop, there is no point in using *grouped*, since responses are always stored in the order in which they are given.

There are three rules for grouped nested loops:

- If the outer loop is grouped, then all inner loops will be grouped automatically. There is no need to place *grouped* on the *for* line of an inner loop if it is already on the *for* line of the outermost loop.
- The parser does not allow a grouped inner loop inside an ungrouped outer loop, so if grouping is required, you must group all loops in the current set.

- The start and end of an inner loop must be separated from the start and end of its outer and/or further inner loops by at least one question.

Here are two examples. The first shows what is correct and the second shows what is incorrect:

```
comment Example 1 - Correct
o1    for outer = 1 to 5 grouped
        Questions here
m1    for middle = 1 to 3
        Questions here
i1    for inner = 1 to 4
        Questions here
inner    next
        Questions here
middle    next
        Questions here
outer next

comment Example 2 - Incorrect
o2    for outer = 1 to 5
i2        for inner = 1 to 4 grouped
        Questions here
inner    next
outer next
```

Example 2 is incorrect because there are no questions between the start of the outer loop and the start of the inner loop or between the end of the inner loop and the end of the outer loop.

10.7 Displaying responses in grids



Quick Reference

To display questions as a grid rather than as a series of repeated questions, type:

```
for grid_type " variable = value_list
label      ask 'question_text'
            resp resp_type resp_list
next
```

grid_type is **rowgrid** to display *value_list* as the rows of the grid and *resp_list* as the columns, or **colgrid** to display *value_list* as the columns of the grid and *resp_list* as the rows.

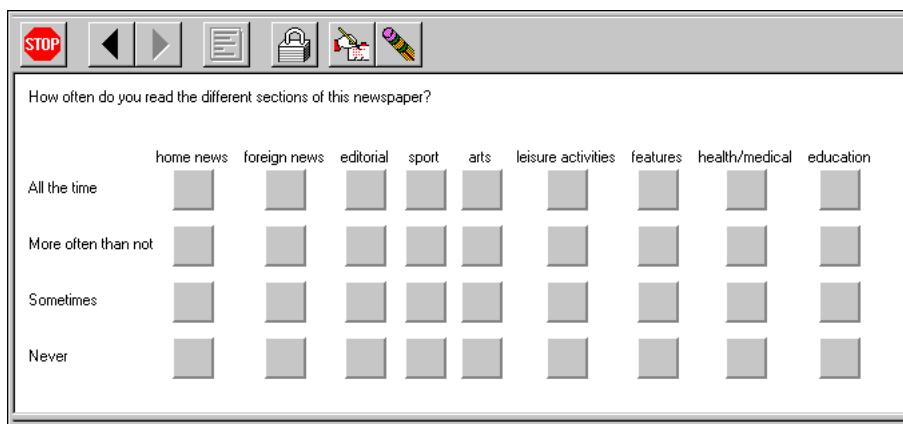
Quancept's and mrInterview's graphical displays let you display repeated questions in response grids similar to those you may have seen in printed questionnaires.

You can display the grid in two ways, depending on where you want to print the items in the value list and where you want to print the responses to the question. The examples below use the same question to illustrate the difference between the two options.

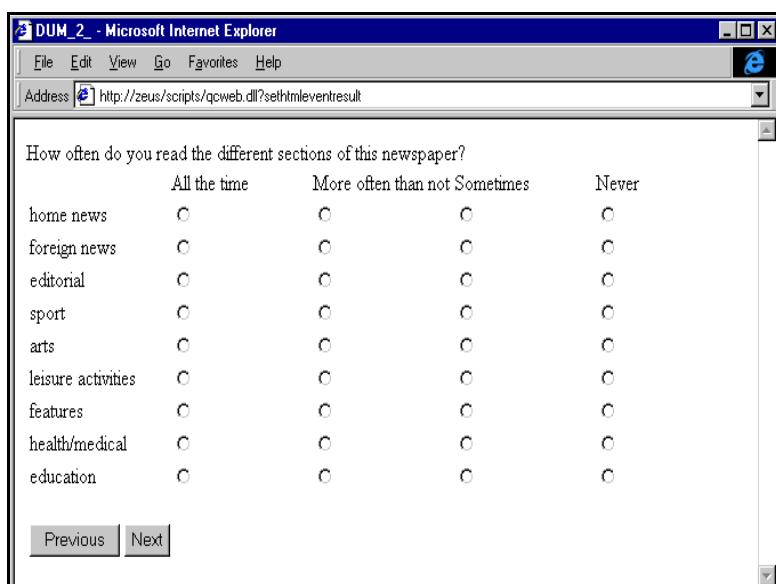
In the first example, the question is defined as:

```
sect  for colgrid '' section ='home news' / 'foreign news' /
          'editorial' / 'sports' / 'arts' /
          'leisure activities' / 'features' /
          'health / medical' / 'education'
often      ask 'How often do you read the different sections of your newspaper?'
resp sp 'All the time' / 'More often than not' /
          'Sometimes' / 'Never'
next
```

which displays the value list as the columns of the grid and the response list as the rows. In Quancept CAPI it looks like this:



With *rowgrid*, Quancept Web and mrInterview display the grid as:



-
- ❖ Grids that use questions with single-punched or multipunched responses can contain one question only.
-

Specified other in grids



You may use specified other with questions that appear in grids, but the interviewing program treats it as ***an ordinary response and does not display a text box*** for entry of an open-ended text.

You should bear this in mind when considering whether to define a single repeated question as a grid or whether to use a conventional loop. If it is important to know what other responses the respondent wished to give, then you should use a loop rather than a grid to gather the data.

Setting column widths in response grids



Quick Reference

To display questions as a grid rather than as a series of repeated questions, type:

```
for grid_type (row% [, col%] ) " variable = value_list
label           ask 'question_text'
                resp resp_type resp_list
next
```

grid_type is **rowgrid** to display *value_list* as the rows of the grid and *resp_list* as the columns, or **colgrid** to display *value_list* as the columns of the grid and *resp_list* as the rows.

row% specifies the percentage of the screen to be occupied by the grid's row text column, and *col%* specifies the percentage of the screen to be used for each of the grid's body columns.

When Quancept CAPI displays a question as a grid, it makes each column wide enough to display the longest text that appears in it. If a column contains a long text it may be so wide that the interviewer needs to scroll through the grid in order to see all of its columns. If the texts are of widely varying lengths, the resulting display will be uneven and messy.

Quancept Web and mrInterview attempt to make the grid columns of equal widths by dividing the width of the browser's screen by the number of columns required (including the row texts). In most cases it is successful. The underlying commands that achieve this are new with HTML version 3.2 and sometimes they are not honored, in which case some columns may be of unequal widths.

One way round this is to control the column width by including new line instructions that force the text to be wrapped at that point. Another is to specify the percentage of the screen that the grid's side text column and body columns will occupy. For example:

```
sect    for rowgrid (15,15) '' section ='home news' / 'foreign news' /  
           'editorial' / 'sports' / 'arts' /  
           'leisure activities' / 'features' /  
           'health / medical' / 'education'  
often    ask 'How often do you read the different sections of your newspaper?'  
           resp sp 'All the time' / 'More often than not' /  
           'Sometimes' / 'Never'  
next
```

Here, the *rowgrid* statement specifies that the side column text and the body columns will each take up 15% of the screen.

- If you are using mrInterview you can control the column widths by defining a grid template that specifies how the grid should appear on the page.

☞ See chapter 24, Templates for grids, for further details.

Numeric or real questions in grids



A grid that uses questions with single-punched or multipunched responses can contain one question only. A grid that uses questions with numeric or real responses can contain more than one question as long as all question have the same response type (that is, you cannot have a grid with both numeric and real responses). For example:

```
d1    display 'Now, please tell me how often you did the following  
           activities in the last week/month'  
l1    for colgrid '' i = 'Last week' / 'Last month'  
walk   ask 'Brisk walk of more than 15 minutes'  
           resp num 0 to 100  
jog    ask 'Jogging more than 1 kilometer'  
           resp num 0 to 100  
aerobics ask 'Aerobics or other similar exercise class'  
           resp num 0 to 100  
swim   ask 'Swimming'  
           resp num 0 to 100  
next
```

The script produces the following effect:

Now, please tell me how often you did the following activities in the last week/month

	Last week	Last month
Brisk walk of more than 15 minutes	<input type="text"/>	<input type="text"/>
Jogging more than 1 kilometer	<input type="text"/>	<input type="text"/>
Aerobics or other similar exercise class	<input type="text"/>	<input type="text"/>
Swimming	<input type="text"/>	<input type="text"/>

Next

Notice how the text that is common to all questions has been defined on a *display* statement above the grid. This reduces the amount of text displayed in the grid itself. If you still find that your question texts are long, use the @ character to mark the points at which you want the text to break onto a new line.

Numeric grids are often used to obtain information under headings that are unlikely to apply to all respondents. In the previous example, there will probably be respondents who did not do aerobics or swim at all. Quancept Web and mrInterview do not allow respondents to proceed until all questions have a response. Normally, this means they have to fill in every box in a grid, which is time consuming. It is possible, however, to preset values into the grid so that respondents only have to fill in the answers that apply and proceed immediately, which makes the survey easier to complete while still allowing for detailed questioning.

The following script shows how to preset all the questions to zero:

```

11      for j = 1 to 2
          set walk(j) = 0
          set jog(j) = 0
          set aerobics(j) = 0
          set swim(j) = 0
      next
d1      display 'Now, please tell me how often you did the following
activities in the last week/month'
      for colgrid '' i = 'Last week' / 'Last month'
walk      ask 'Brisk walk of more than 15 minutes'
          resp num 0 to 100
jog       ask 'Jogging more than 1 kilometer'
          resp num 0 to 100
aerobics  ask 'Aerobics or other similar exercise class'
          resp num 0 to 100
swim      ask 'Swimming'
          resp num 0 to 100
      next

```

The screenshot shows a Microsoft Internet Explorer window titled "DUM_3 - Microsoft Internet Explorer". The page content is a survey question asking about the frequency of various activities. The question is: "Now, please tell me how often you did the following activities in the last week/month". Below the question, there are two columns of labels: "Last week" and "Last month". Under each column, there are four activities listed with corresponding input fields. The activities and their labels are:

Activity	Last week	Last month
Brisk walk of more than 15 minutes	<input type="text" value="0"/>	<input type="text" value="0"/>
Jogging more than 1 kilometer	<input type="text" value="0"/>	<input type="text" value="0"/>
Aerobics or other similar exercise class	<input type="text" value="0"/>	<input type="text" value="0"/>
Swimming	<input type="text" value="0"/>	<input type="text" value="0"/>

At the bottom left of the form is a "Next" button. At the bottom right of the browser window is a status bar with the text "Local intranet zone".

-
- ❖ You cannot currently define default values for questions that appear by themselves in single-question grids. You can get round this problem by displaying those questions using *multitask* instead.
-

The interviewing program sees the grid as a single item, so routing statements, assignment statements and callable functions are not permitted inside the grid. This means that you cannot use the results of one question in the grid to influence the responses that are accepted to subsequent questions in the grid. For example, if the respondent has given a figure for the total weekly food bill, you cannot keep a running total inside the grid and display an error message as soon as the running total exceeds the original total. Instead, you have to add up the responses from the grid questions and compare that total with the original total after the grid, and then route the respondent back to the start of the grid if the figures do not tally.

There are limits on the size of a numeric grid. These are:

- no more than 25 rows
- no more than 100 columns
- no more than 500 cells altogether

mrInterview does not normally allow blank responses in numeric grids. There are two ways of changing this:

- Set the variable `emptyok` to a non-zero value before the grid. This allows blank cells in the grid and treats the question as an unanswered question.
- Add `null` to the response list and set `silentnl` to a non-zero value before the grid. This accepts blank cells in the grid and codes the question as unanswered. You may prefer this method if you want to differentiate between answered and unanswered questions.

-
- ☞ See ‘Automatic null coding when no response is chosen’ in chapter 10, Repetitive questions, for information about *silentnl*, and ‘Allowing blanks in numeric grids’ in chapter 10, Repetitive questions, for information about *emptyok*.
-

10.8 Respondent-specific texts in value lists

Quick Reference

To use the value of a variable as an item in a value list, type:

[label] **for** variable = 'text1' / '[+var_name+]'

The value list need not consist solely of quoted texts; it may also contain variables which represent respondent-specific texts. This follows the same principles as described in section 8.6, Reading single lines from look-up files, for substituting respondent-specific information in response lists. For example:

```

knowp    ask 'Which daily newspapers can you name?'
        resp mp 'The Daily Bugle' / 'The Morning Herald' /
                'The Informer' / 'Today''s Gossip' /
                'News and Views' / 'Other'
        if (read = '<Other'') {
oth      ask 'INTERVIEWER: enter name of first other paper
mentioned'
        resp coded
    }
pubs    for pub = 'The Daily Bugle' / 'The Morning Herald' /
                'The Informer' / 'Today''s Gossip' /
                'News and Views' / '[+oth+'
        route (pub = null) go nxtp
people   ask 'What type of people do you think read '+pub+'?'
        resp mp 'People who want to read real news' /
                'Left-wing people' / 'Financial people' /
                'People who want local news' /
                'People who like scandal'
nxtp    next

```

If the respondent mentions a newspaper called the Worker’s Paper that does not appear in the original list, the publication loop will be repeated six times with the text ‘The Worker’s Paper’ being substituted in the value list in place of [+oth+].

When you substitute responses as items in a loop list, you need to consider whether the response will ever be null and, if so, skip the questions in the loop. In this example, [+oth+] will be null if the respondent names listed only newspapers, so we route those respondents to the end of the loop. Since [+oth+] is the last item in the list, the interviewing program will then execute the next statement in the script.

10.9 Sample scripts

This chapter provides two sample scripts. The first script shows how to use the commands covered in this chapter that apply to any version of Quancept and to mrInterview. The second script is aimed at Quancept CAPI, Quancept Web and mrInterview users only and shows how to use the *colgrid/rowgrid* keywords for displaying responses in a grid.

Script A

```

comment Sample script A for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 10, Repetitive questions (loops)

papers      define 'The Daily Bugle' / 'The Morning Herald' /
              'The Informer' / 'Today''s Gossip' / 'News and Views'
read        ask 'Which daily newspapers do you read on a regular basis?@'
INTERVIEWER: RECORD ONLY FIRST OTHER PAPER MENTIONED.'
             resp mp papers other

comment CATI/CAPI/Web: Test the script with and without grouped loops.
comment mrInterview: grouped is ignored.
comment For each newspaper mentioned, in random order, ....
pubs        for pub = read in papers ran grouped
ratep       ask 'On a scale of 1 to 5, how would you rate '+pub+'.
             Please use 1 for Very good and 5 for Very bad'
             resp num 1 to 5
comment ... for each of the listed sections ....
sect        for sections = 'home news' / 'foreign news' /
              'editorial' / 'sport' / 'arts' /
              'leisure activities' / 'features' /
              'health/medical' / 'education'
freq        ask 'How often do you read the '+sections+
             section of this newspaper?'
             resp sp 'All the time' / 'More often than not' /
             'Sometimes' / 'Never'

comment Check the answer to a question in the loop
route (freq(*) = 'Never') go nxtsec
rate        ask 'On a scale of 1 to 5, how would you
             rate that section. Please use 1 for Very good and 5 for Very bad'
             resp num 1 to 5
nxtsec      next
comment Separate inner and outer loop ends with a question
            goto nxtpub
qdum1       ask ''
             resp coded nodata
nxtpub     next

```

```
comment Question separating grouped loops
persons ask 'How many adults, that is people aged 18 years and over, are
there in your household?'
      resp num 1 to 9
d1      display 'Now I''m going to ask some questions about
each person in the household. I''ll refer to them by number. Please
count yourself as person number 1.'
      pause
pers   for persnum = 1 to 9 grouped
      route (persnum > persons) go nxt
      if (persnum = 1) {
          set relat(1) = 'respondent'
          goto employ
      }
relat   ask 'What is the relationship of person number '
+persnum+ ' to yourself?'
      resp sp 'respondent' / 'husband' / 'wife' /
           'son/daughter' / 'sister' / 'brother' / 'father' /
           'mother' / 'other'
employ   ask 'Which of the following most closely matches
your/that person''s work status?'
      resp sp 'Employed full-time' / 'Employed part-time' /
           'On a training course' / 'At school' /
           'At university' / 'Unemployed' / 'Retired' /
           'Looks after the household' null
nxt     next
thank   display 'Thank you for taking the time to answer our questions.'
      pause
      end
```

Script B



```
comment Sample script B for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 10, Repetitive questions (grids)
comment (CAPI/Web/mrInterview only)

papers   define 'The Daily Bugle' / 'The Morning Herald' /
           'The Informer' / 'Today''s Gossip' / 'News and Views'
read      ask 'Which daily newspapers do you read on a regular basis?@
           SELECT ONLY FIRST OTHER PAPER MENTIONED.'
           resp mp papers other

comment CAPI/Web: Test the script with and without grouped loops.
comment mrInterview: grouped is ignored.
comment For each newspaper mentioned, in random order, ....
11       for pub = read in papers ran grouped
ratep     ask 'On a scale of 1 to 5, how would you rate '+pub+'.
           Please use 1 for Very good and 5 for Very bad'
           resp num 1 to 5
```

```
comment ... for each of the listed sections ....
sect      for colgrid '' sections = 'home news' / 'foreign news' /
           'editorial' / 'sport' / 'arts' /
           'leisure activities' / 'features' /
           'health/medical' / 'education'
freq      ask 'How often do you read the different
           sections of this newspaper?'
           resp sp 'All the time' / 'More often than not' /
           'Sometimes' / 'Never'
           next
comment Separate inner and outer loop ends with a question
           goto nxtpub
qdum1    ask ''
           resp coded nodata
nxtpub   next

comment Question separating grouped loops
persons   ask 'How many adults, that is people aged 18 years
           and over, are there in your household?'
           resp num 1 to 9
d1        display 'Now I''m going to ask some questions about
           each person in the household. I''ll refer to them by number. Please
           count yourself as person number 1.'
           pause
pers      for persnum = 1 to 9 grouped
           route (persnum > persons) go nxt
           if (persnum = 1) {
               set relat(1) = 'respondent'
               goto employ
           }
relat     ask 'What is the relationship of person number '
           +persnum+ ' to yourself?'
           resp sp 'respondent' / 'husband' / 'wife' /
           'son/daughter' / 'sister' / 'brother' /
           'father' / 'mother' / 'other'
employ   ask 'Which of the following most closely matches
           your/that person''s work status?'
           resp sp 'Employed full-time' / 'Employed part-time' /
           'On a training course' / 'At school' /
           'At university' / 'Unemployed' / 'Retired' /
           'Looks after the household' null
nxt      next
thank    display 'Thank you for taking the time to answer our questions.'
           pause
           end
```


11 Iteration and subscription

This is the second chapter about repetitive questions. In the previous chapter, we looked at how to write various types of loops, and how to control the allocation of columns to the questions inside loops. In this chapter, we will talk about testing responses given to questions in loops and will explain the concepts of iteration and subscription, both of which are central to the correct and efficient use of loops.

New keywords introduced in this chapter are:

iteration	the number of times a loop has been executed
subscript	the number of times a question in a loop has been asked

The chapter ends with some suggestions for how to improve the efficiency of your scripts and reduce the size of the data maps using loops.

11.1 Counting loop iterations

Quick Reference

To find the number of times a loop has been executed, type:

set variable = iteration

It can be useful to know how many times a loop has been started — for example, if you have a large value list and you do not want to repeat the loop for every item in the list. **iteration** is a keyword that is incremented each time the *for* statement is read and keeps track of the number of times a loop has been started. In order to use this information you must store it in a variable. You can think of a variable as a storage box, rather like a question name, in which you can place values that you want to use later in the script. In this case, the variable will store a number which tells you how many times the loop has been started.

Iterations with rotated and randomized value lists

If you rotate or randomize the items in the *for* statement, the value which *iteration* returns corresponds to the position of the current item in the original value list rather than the number of times the loop has been started. For example:

```
11      for prog = 'Family Films' / 'Children''s Films' / 'Comedies' ran
          set iter = iteration
d1      display 'iteration is '+iter
swatch   ask 'Which channel do you usually watch for '+prog+'?'
          resp sp 'Sky One' / 'The Movie Channel' / 'Filmnet' /
          'RTL Plus' other null
next
```

If this loop were not randomized, the iteration would always be 1 for family films, 2 for children's films, and 3 for comedies, in that order. *swatch(1)* would always hold the name of the channel watched for family films, *swatch(2)* would always store the channel watched for children's films, and *swatch(3)* would always store the channel watched for comedies.

If the randomization results in the loop being repeated for children's films, family films and comedies, the iteration values in *iter* will be 2, 1 and 3, reflecting the order in which items were selected from the value list. The references for the data, however, will still be the same as if there were no randomization; *swatch(2)*, for example, will still store the answer about children's films, just as it would if there were no randomization.

To pick up the absolute iteration value (that is, the number of times the loop has been started), you should use a temporary variable and increment it by 1 each time the loop is started, as shown below:

```
set count = 0
11      for prog = 'Family Films' / 'Children''s Films' /
          'Comedies' ran
          set count = count + 1
          set iter = iteration
d1      display 'iteration is '+iter+' and count is '+count
swatch   ask 'Which channel do you usually watch for '+prog+'?'
          resp sp 'Sky One' / 'The Movie Channel' / 'Filmnet' /
          'RTL Plus' other null
next
```

Now, even if the iteration values are 2 (children's films), 1 (family films) and 3 (comedies), the number of times the loop has been restarted will be correctly reflected as 1, 2 and 3 in the variable called *count*.

Notice that the temporary *count* variable is set to 0 before it is used. This is in case the interviewer snaps back and repeats the loop. If the start value was not reset, the interviewing program would keep adding 1 to the current value of *count*. You may use an *unset* statement instead of the assignment statement if you prefer.

☞ See section 12.13, The *unset* statement, for more information.

Using the iteration value in the script

One way of using the iteration value is when you have a loop with a rotated or randomized list and want to repeat the loop fewer times than there are items in the list — for example, three times when there are six items in the list:

```

set lpct = 0
11    for i = 'a' / 'b' / 'c' / 'd' / 'e' / 'f' rot
          set lpct = lpct + 1
          route (lpct > 3) go nxt
q7      ask ....
        resp ....
nxt     next
qout    ask ...
        resp ...

```

In this example, the items in the value list are to be rotated and the respondent is to be asked about only three of them. There is no way of knowing which three items will be presented to a given respondent, so we count the number of times the loop has been started and route according to this. Once the *for* statement has been read for the fourth time, the logical expression with *route* will be true and the interviewing program will skip to the end of the loop.

You can also use the iteration number for checking the individual answers to questions in loops. We'll talk about this when we look at subscription later in this chapter.

Iteration in nested loops

To understand how the iteration is calculated for nested loops, consider the following example:

```

o1 for outer = 1 to 2
      set outiter = iteration
qa      ask ...
      .....
i1      for inner = 1 to 3
      set initer = iteration
qb      ask ...
      .....
      next
qc      ask ...
      .....
      next

```

The first time the outer loop is used, its iteration value (outiter) is 1. This value does not change until the outer *for* statement is executed the second time. During this time the inner loop is executed three times with iteration values (initer) of 1, 2 and 3 for each pass through the inner loop.

When we reach the end of the inner loop the third time through, we return to the start of the outer loop. The iteration value for the outer loop becomes 2. This value again remains constant while the inner loop is executed. Since we are restarting the inner loop from the beginning, the iteration value for this loop is reset to 1 regardless of the fact that this is now the fourth time we are asking qb. By the time we reach this question on the third pass through the inner loop, 'initter' will be 3 and 'outiter' will still be 2.

From this, you can see that the iteration value of the inner loop is not dependent upon that of the outer loop. If you want to check the total number of times the inner loop has been executed, you will need to refer to the subscript value of the questions in the loop (for example, to check whether this is the fifth time the question has been asked) or use a temporary variable which is not reset each time the outer loop is restarted.

11.2 Referring to answers given in loops

Quick Reference

To find out how many times a question in a loop has been asked, type:

set *variable* = *subscript*

Responses to questions are represented by their question names: if the question name is age, the respondent's age, whatever it may be, is represented by the name age. Any references to the value of age are references to the age of the current respondent.

In loops, the question name will refer to different responses at different times. For example:

```
11      for prog = 'Family Films' / 'Children''s Films' / 'Comedies'
swatch    ask 'Which channel do you usually watch for '+prog+'?'
          resp sp 'Sky One' / 'The Movie Channel' /
                  'Filmnet' / 'RTL Plus' null
          go nxtprog
why       ask 'Why do you mainly watch that channel for '+prog+'?'
          resp sp 'It''s in a language I can understand' /
                  'Best selection' / 'It''s free' / 'Other'
          next
```

In this example, the variable why will refer to three types of answers, as defined in the loop's value list. If you want to refer to one of the answers to that question later in the script — for example, to check whether the respondent watches comedies on a certain channel because it's free — you will need some way of differentiating between the three answers that this question will have. If the respondent said he/she watches comedies and family films on Sky One because it is free, how can you be sure of picking up the 'It's free' response that goes with comedies rather than the one that refers to family films? The answer is to follow the question name with a number indicating which answer to swatch you wish to check. In this case the number is 1, 2 or 3. This number is called the *subscript*.

In our example:

- why(1) means the answer to why the 1st time it was asked (that is, for family films)
- why(2) means the answer to why the 2nd time it was asked (that is, for children's films)
- why(3) means the answer to why the 3rd time it was asked (that is, for comedies)

Therefore, to check whether the respondent watches comedies on Sky One because it's free, you would write a statement which includes the logical expression `why(3)='It''s free'`. This form of cross-reference works well as long as you are able to enter the absolute value of the subscript. In many cases, however, you will want to use subscription inside a loop to check the answer to a previous question in the loop. If we expand the example, you may want to check whether the respondent would still watch that channel if it were a pay channel, or to skip to the end of the loop ready to ask about the next type of program if the respondent watches that channel for some reason other than cost.

This means that you will need some general method of entering the subscript for why, since it will vary according to which type of program you are currently asking about. The script therefore becomes:

```

11      for prog = 'Family Films' / 'Children''s Films' / 'Comedies'
swatch    ask 'Which channel do you usually watch for '+prog+'?'
          resp sp 'Sky One' / 'The Movie Channel' /
                  'Filmnet' / 'RTL Plus' null go nxtprog
why       ask 'Why do you mainly watch that channel for '+prog+'?'
          resp sp 'It''s in a language I can understand' /
                  'Best selection' / 'It''s free' / 'Other'
          route (why(subs) <> 'It''s free') go nxtprog
nxtprog   next

```

This script as it stands is incomplete because we have not determined how the value of `subs` is to be calculated.

There are four methods of defining subscripts, three of which are valid with this particular example.

- If the values on the `for` statement are numeric and are sequential from 1, you may use these values as the subscripts.
- If the loop is unnested you may use the iteration value.
- You may use the special notation *.
- You may use the **subscript** keyword.

The first method is unsuitable for this example because it requires a sequential numeric `for` statement (ours has text values), but all other options are acceptable, so we will write three scripts and check the differences.

We will use the loop iteration value first:

```
11      for prog = 'Family Films' / 'Children''s Films' / 'Comedies'
swatch    ask 'Which channel do you usually watch for '+prog+'?'
          resp sp 'Sky One' / 'The Movie Channel' /
                  'Filmnet' / 'RTL Plus' null go nxtprog
why       ask 'Why do you mainly watch that channel for '+prog+'?'
          resp sp 'It''s in a language I can understand' /
                  'Best selection' / 'It''s free' / 'Other'
          set iter = iteration
          route (why(iter) <> 'It''s free') go nxtprog
nxtprog   next
```

The first time the loop is executed, the iteration value returned by the *set* statement is 1. This is then incremented by 1 for each subsequent pass through the loop. The first time we reach the *route* statement, it is read as:

```
route (why(1) <> 'It''s free') go nxtprog
```

which checks whether the answer to ‘why’ the first time it was asked was ‘It’s free’. On the third pass, this statement is read as:

```
route (why(3) <> 'It''s free') go nxtprog
```

which checks whether the answer to ‘why’ the third time it was asked was ‘It’s free’. If we had not subscripted ‘why’ in this way, the parser would have issued an error message when we checked the script.

Now let’s look at the same script with the special * subscript:

```
11      for prog = 'Family Films' / 'Children''s Films' / 'Comedies'
swatch    ask 'Which channel do you usually watch for '+prog+'?'
          resp sp 'Sky One' / 'The Movie Channel' /
                  'Filmnet' / 'RTL Plus' null go nxtprog
why       ask 'Why do you mainly watch that channel for '+prog+'?'
          resp sp 'It''s in a language I can understand' /
                  'Best selection' / 'It''s free' / 'Other'
          route (why(*) <> 'It''s free') go nxtprog
nxtprog   next
```

In this example, the interviewing program keeps track of how many times ‘why’ has been asked and always substitutes the current subscript value in place of the asterisk. This method of writing subscripts is ideal for loops of all kinds (whether or not the loop is nested), since everything is automatic and there is less chance of your script failing due to errors on your part.

The final option is to use the **subscript** keyword:

```

11      for prog = 'Family Films' / 'Children''s Films' / 'Comedies'
swatch    ask 'Which channel do you usually watch for '+prog+'?'
          resp sp 'Sky One' / 'The Movie Channel' /
                  'Filmnet' / 'RTL Plus' null go nxtprog
why       ask 'Why do you mainly watch that channel for '+prog+'?'
          resp sp 'It''s in a language I can understand' /
                  'Best selection' / 'It''s free' / 'Other'
          set subs = subscript
          route (why(subs) <> 'It''s free') go nxtprog
nxtprog  next

```

The only difference between this script and the iteration example is that the keyword *subscript* has been used in place of the keyword *iteration*. The loop works in exactly the same way as both previous ones, returning a subscript of 1 the first time the question is asked, 2 the second time, and so on. Thus, it is apparent that when a loop is unnested, the subscript count of questions in that loop is the same as the loop iteration count.

The advantage of using *subscript* instead of * comes with complex loops where you want to refer to answers given in previous iterations of the loop. In this case, you'll need to keep track not only of the current subscript but also of any previous subscripts related to questions asked in earlier iterations of the loop.

Specified other in loops (negative subscripts)



↖ This information is included for backwards compatibility only. Negative subscription is still a valid method of retrieving specified other responses, but it is easier to use *othertext* instead.

When you want to refer to an answer given in response to the automatic prompt issued by specified other, you use the special subscript -1. In loops, the * subscript takes care of this, either substituting the current iteration value if specified other was not chosen or a corresponding negative value (-1, -2, for instance) if it was. Let's add specified other to the list of channels in our previous example and change the wording of 'swatch':

```

11      for prog = 'Family Films' / 'Children''s Films' / 'Comedies'
swatch    ask 'Which channel do you usually watch for '+prog+'?'
          resp sp 'Sky One' / 'The Movie Channel' /
                  'Filmnet' / 'RTL Plus' other
          null go nxtprog
why       ask 'Why do you mainly watch '+swatch(*)+' for '+prog+'?'
          resp sp 'It''s in a language I can understand'
                  'Best selection' / 'It''s free' / 'Other'
nxtprog  next

```

If the respondent chooses a channel from the list, the interviewing program substitutes it in place of the question name swatch in the second question. If the respondent uses specified other to name a channel that is not in the list, the interviewing program substitutes the name given at the 'Please specify' question and also the open-end serial number that it has given to this response. For example:

```
Why do you mainly watch SER      123/Sky Movies for family films?
```

This looks untidy on the interview screen, and interviewers may find it confusing, so a better approach is to pick up the current subscript value with *subscript* and then take care of the change of subscript from a positive to a negative value yourself.

The subscript for specified other is always the same as the main question subscript except that it is negative instead of positive. Thus, if the question subscript is 2, the subscript for specified other for that question will be -2. The script below shows you how to set a negative subscript:

```
11      for prog = 'Family Films' / 'Children''s Films' / 'Comedies' ran
comment Get the subscript for use with specified other
      set subs = subscript
swatch    ask 'Which channel do you usually watch for '+prog+'?
      resp sp 'Sky One' / 'The Movie Channel' /
              'Filmnet' / 'RTL Plus'
      other go setneg null go nxtprog
      set whatchan = swatch(subs)
      goto why
comment Negative subscription to pick up text of specified other
setneg     set negsubs = 0 - subs
            set whatchan = swatch(negsubs)
why       ask 'What are the reasons that you mainly watch '
          +whatchan+ for '+prog+'?
            resp mp 'It''s in a language I can understand' /
                  'Best selection' / 'It''s free' / 'Other'
nxtprog   next
```

This method displays just the open-ended text typed by the interviewer and nothing else.

Subscription when assigning values to repeated questions

When a question is defined in a loop its responses are held in an array in which the cells or elements are named *qname(1)* to *qname(max_repeats)*.

When you define a loop, the size of any questions or variables named in that loop is controlled by the number of values on the *for* statement. All three of the following examples generate arrays with five cells:

```
11  for x = 1 to 5
12  for y = 1 / 3 / 5/ 7 / 9
13  for z = 'red' / 'blue' / 'green' / 'yellow' / 'orange'
```

The exception is when the loop refers to a question that has already been defined outside the loop. In this case, the number of cells in the question array is taken from the original question definition. If the size of the question array as it was originally defined is less than the size implied by the loop in which its values are set, then qtip fails with an error stop message. This is particularly important if the question was originally defined in a loop that has a different number of control values to the current one.

When referring to questions or other variables in loops, you must specify the question name and the subscript so that the correct cells of the array can be updated. The * notation calculates the subscript based on the number of times the loop has been started. When a numeric loop has values that start at 1 and increment by 1, as in the 11 example, you can generally use the loop control variable instead of * and the results will be the same. In all other cases it is safest to refer to the subscript as *.

The following examples illustrate the difference between using * and using a numeric loop control variable as the subscript. The first example uses the * notation to refer to the cells of q5 and will fail with an errorstop message. This is because the q5 question array was originally defined as having 16 cells whereas the loop that sets its values implies that it has only three cells.

```
comment This script sets q5(1)=3, q5(2)=9 and q5(2)=16.
comment q5(4) to q5(16) are blank
      for i = 1 to 16
q5  dummyask ''
      resp num 1 to 100
11  for x = 3 / 9 / 16
      set q5(*) = x
      next
```

Since * is based on the number of times the loop has been started, only the first three cells of q5 are updated.

The second example uses the loop control variable as the subscript, and allows you to set the values of specific cells in the array according to the current value of the loop control variable:

```
comment This script set q5(3)=3, q5(9)=9 and q5(16)=16
      for i = 1 to 16
q5  dummyask ''
      resp num 1 to 100
11  for x = 3 / 9 / 16
      set q5(x) = x
      next
```

11.3 Subscription with sequential numeric for statements

It is sometimes possible to use the values in the *for* statement as the question subscripts. When the values in the *for* statement are numeric and are sequential from 1, you may use the name of the pointer variable on the *for* statement as the subscript. The example below illustrates this feature:

```
comment Set counters to zero. total is total hours watched TV
comment films is the number of hours spent watching films
    set total = 0
    set films = 0
comment Test each day of the week
11      for day = 1 to 7
tv       ask 'How long did you spend watching TV on day '+day+'?'
        resp real 0 to 168
comment Count total time watched
        set total = total + tv(day)
comment Check how long was spent watching films
fwatch   ask 'How much of that time were you watching films?'
        resp num 0 to 168
comment Count total time for films
        set films = films + fwatch(day)
next
```

Since the variables *films* and *total* are not questions, their values will not be stored in the data file automatically. You may store them by using *fix* statements.

☞ The *fix* statement is explained in section 17.7, Inserting non-response data in the data file.

11.4 Subscripts with nested loops

When loops are nested, the subscript (number of times the question was asked) for questions in the inner loop is no longer the same as the loop iteration value (number of times loop was started) so you must pick up the subscript value using * or *subscript*.

When we were first discussing loops and iterations, we said that with nested loops the iteration count of the inner loop is reset to 1 each time the outer loop is started. If we have an outer loop with three values and an inner loop with five values, the iteration values for the inner loop will be 1 to 3 for the first value in the outer loop, 1 to 3 again for the second value in the outer loop, and so on.

Any questions in the inner loop will be asked a maximum of 15 times in all and will have subscripts of 1 to 15; subscripts 1 to 3 for the first pass through the outer loop and 4 to 6 for the second, and so on. Here is an example to clarify this.

```
11      for item = 'TV' / 'Satellite TV' / 'Videos'
numhr   ask 'How long, to the nearest quarter hour, did you spend
        watching '+item+'?'
        resp real 0 go nxtitem / 0.25 to 168
```

```

comment Create temporary variable to sum up hours for this item
    set timchk = 0.0
12     for ptype = 'films' / 'news programs' / 'documentaries' /
            'situation comedies' / 'other programs'
ptime      ask 'How much of that time was spent watching ' +ptype+ '?'
            resp real 0 to 168
comment Add hours for current program into the total
    set timchk = timchk + ptime(*)
next
route (timchk = numhr(*)) go nxtitem
d1      display 'The sum of the three times you have just given me is
            '+timchk+' but the total time you gave earlier is ' +numhr+'. Could I
            just recap on those figures?'
            pause
            goto numhr
nxtitem  next

```

As the comments show, the purpose of these loops is to find out the total time spent watching TV, satellite TV and videos, and to verify this by checking how much time was spent watching different types of programs. If the accumulated total for an individual item does not match the overall total, the questions for that item are asked again.

The outer loop has 3 iterations and the inner one has 5, meaning that the question ptime will be subscripted in the range 1 to 15. The question numhr is part of the outer loop, so its subscript will be between 1 and 3, depending on whether we are asking about TV, satellite TV or videos. We set the variable timchk to 0 to ensure that it accumulates only one set of figures. If we did not do this, it would accumulate the total number of hours watched for all three items.

For each item, we then enter the inner loop. We ask the question ptime for each of the five program types, and the variable timchk is incremented by the number of hours given. Once the inner loop has been finished for one item, the accumulated total is compared with the original total for that item (numhr) and, if they match, the interviewing program goes to nxtitem and then back to the next iteration of the outer loop. This sets the iteration and subscript for numhr to 2 and resets the iteration for the inner loop to 1. It does not alter the subscript for ptime. This remains set to 5 and will be incremented by 1 when the question is asked the sixth time.

- ☞ In this example, we have checked that the sum of the times for the individual programs is the same as the total number of hours spent watching TV, satellite TV or videos by adding the times into a temporary variable and then comparing the value of that variable with the original total. An alternative would be to use *limitresp* to alter the responses allowed by the ptime question so that the sum of the individual times could never exceed the total.
- ☞ There is an example of this in section 3.5, Variable values in numeric ranges.

11.5 Subscription of temporary variables



Quick Reference

If you will be subscripting temporary variables, define the maximum number of cells per subscripted variable with the **arrays** statement:

arrays *max_size*

Any variable that is not a question may be subscripted as if it were a question. Thus, it is possible to say:

```
set a(1) = b + c
set a(2) = bit(q5/6)
set a(3) = 'taxi'
```

The variable 'a' has been divided into 3 cells, each one storing a different item of information. We call variable 'a' an *array*.

Arrays are useful if you need to store several related responses or values and want some visible means of telling that they are related. For example, say we have a loop asking which channels the respondent watches for different types of films. If there are some questions later in the script for people who watch the Movie Channel, we might want to set a flag each time the respondent mentions that channel. If we call the array mc, then mc(1) might be set to 1 if the respondent watches family films on the Movie Channel, mc(2) might be set if he/she watches children's films on that channel, and mc(3) might be set if he/she watches comedies on that channel.

```
unset mc(1)
unset mc(2)
unset mc(3)
11   for prog = 'Family Films' / 'Children''s Films' / 'Comedies' ran
        set subs = subscript
swatch   ask 'Which channel do you usually watch for '+prog+'?'
        resp sp 'The Movie Channel' / ('Sky One' / 'Filmnet' /
        'RTL Plus') go nxtprog null go nxtprog
comment Set appropriate cell of array if key response was given in
comment this iteration
        set mc(subs) = 1
nxtprog next
```

If you use this facility, it is advisable to define the maximum number of subscripts or cells that any temporary variable may have so that the parser can calculate how much space will be required for these variables. The word for this is *arrays* followed by a number showing the maximum number of subscripts that any temporary variable will have:

```
arrays 10      or    arrays 31
```

The number you use is arbitrary. However, if you try to use a variable with more than the given number of cells (for example, var(10) when *arrays* is 9) an error of the form ‘maxsubscript 9 retrval 10’ will occur when the script is used for interviewing.

Note that one *arrays* statement covers all temporary variables in a script. If you use a(1) to a(6) and b(1) to b(4), you write the *arrays* statement as, say, *arrays 6*.

11.6 Reducing the size of data maps with loops



- Any script that is required to select a small number of items from a large list is automatically allocated sufficient column space to accommodate the complete list, even if most of them will be left blank by the interviewing program. Careful use of loops can significantly reduce the size of the data map. The sections below look at some common requirements and offer suggestions on how best to write a script for each situation.
- mrInterview stores data in a relational database for outputting in any format of your choice, rather than in card-column format, so there is no need to worry about saving space in the data file. Nevertheless, you may find it useful to look at the scripts in the following sections as they illustrates ways of dealing with particular scripting requirements.

A few responses at random from a large list

The purpose of this script segment is to have the interviewing program choose a number of items at random from a large list and to retain the order of choice without having columns allocated to all items in the list. The respondent does not do the choosing.

The first method is to place the choice question in a loop which is repeated as many times as you want to make choices. However, with long response lists, it is likely that you will be reserving columns unnecessarily. For example, if the response list contains 99 items from which five are to be chosen, each iteration will reserve ten columns that will only ever contain one code. The example below suggests an alternative method which reduces the number of columns required to a minimum.

We will look at the script first (it has some comments that may help you) and then go through it statement by statement.

```

comment Dummy questions save order of selection (respnnum) and
comment choices made (country).
loop1      for i = 1 to 5
respnnum    dummyask 'Country positions in order of selection'
            resp num 0 to 99
            next
country    dummyask 'Five choices will be made from this list'
            resp mp 'England' / 'Ireland' / 'Scotland' / 'Wales' /
                    'France' / 'Belgium' / 'Holland' / 'Germany' /
                    'Italy' / 'Spain' / 'Portugal' / 'Austria' /
                    'Switzerland' / 'Monaco' / 'Norway' / 'Sweden' /
                    'Denmark' / 'Finland' / 'Iceland' nodata

```

```
comment Make five choices
      unset chnum
loop2    for choice = 1 to 5
      set chnum = 1
comment Choose country number at random and save as current choice
loop3    for which = 1 to 19 ran
      set respnum(choice) = which

comment Test whether this country has been chosen already. chnum
comment is only equal to choice when we've tested all choices made so
comment far. If this is the first country chosen, then choice=chnum
comment and go to get the next choice. If not, cycle through the
comment choices made so far until we've checked each one. Only then
comment will this routing take effect
testme      route (choice = chnum) go break
            route (which = respnum(chnum)) go back
            set chnum = chnum + 1
            goto testme

back       next
comment Current choice compared against all previous choices and
comment found to be unique
break      set country = respnum(choice)
            set ansnum = respnum(choice)
            set txt = elm(country/ansnum)
ctry       display 'Country selected is '+txt
next
```

The script starts by defining two dummy questions. `respnum` will store as a numeric value the position of each chosen country in the country response list. `country` will store the names of the selected countries. Note that the script makes the selections, not the respondent, the idea being that the respondent is then questioned on the five randomly selected countries.

Because we want to select five countries, we repeat loop1 five times so that we have space in the data file for each answer separately. The position number of the first country selected will be stored in `respnum(1)` and will be copied into the data file in the columns allocated to that iteration of the question. Similarly, the position number of the last country chosen will be stored in `respnum(5)` and will be copied into the data file in the columns allocated to that iteration. Thus you'll have not only the response numbers chosen but also the order in which they were selected.

The reason we keep track of the order of selection in this way rather than with *mention* is because the script makes each selection separately, rather than selecting five responses in one pass.

`country` is an ordinary question with a multipunched response list which defines the list of countries from which selections will be made. As selections are made and checked for uniqueness (that is, not already chosen), they are assigned as responses to the question so that they can be used later in the script. The questions are defined as *nodata*, since we don't need to store the answers in the data file as we are already doing this with the `respnum` array.

The rest of the script is a large loop (loop2) concerned with making selections and copying response numbers and texts into respnum and country. Within this, we have another loop (loop3), which compares the current selection against all previous selections to ensure that the selection is unique before it is saved. If the selection is not unique, it is ignored and another one is made.

loop2 is repeated five times, once for each choice. The statement `set chnum=1` initializes a temporary variable which is used in the checking section for cycling through the choices saved so far. It is always set to 1 at the start of each new selection to ensure that all selections made so far are checked. When chnum and choice have the same value, we know that we have checked all choices.

For example, if the third selection has just been made, choice will be 3 and chnum will be 1. The current selection is compared against the first selection and, assuming they are different, chnum is set to 2, so that the current selection is compared against the second selection. This process continues until a match is found or all choices have been checked.

loop3 makes the selection and checks for uniqueness. It can be repeated up to 19 times per choice (there are 19 countries to choose from), and countries are selected at random. The country's response number is saved in the appropriate cell of respnum — respnum(1), for example, if this is the first choice made. Notice that the country name has not yet been saved.

If choice and chnum have the same value we know this is the first selection so there is no checking to be done and we skip out of loop3 to the break statement in loop2. This assigns the country name corresponding to the chosen response number to the country question and displays it on the screen. For example, if respnum(1) is 4, the script will assign the text 'Wales' to country because it is the fourth response in the list.

If choice and chnum have different values, we know that some choices have been saved and we need to compare the current selection against them. This is where we start incrementing chnum. However, to start with, chnum is 1 and is pointing to the first choice saved.

If the current response number matches the response number in respnum(1), we have a duplicate selection and we return to the start of loop3 to make another random selection. If the current response number differs from the response number in respnum(1), we increment chnum so that we can look at the second saved choice in respnum(2).

Once the current selection has been compared against all existing selections and no matches have been found, we skip out of loop3 and assign the appropriate country name to country as described above. Once this is done, execution returns to the start of loop2 so that chnum is reset and ready to check the next selection.

Restricted questions in nested loops

The previous example illustrated an efficient method of using loops for automatic selection of a few responses from a long list. This section shows another method of saving space in the data file when you use nested loops. If the questions in the inner loop are to be asked only for certain items in the value list for the outer loop, you can save considerable space in the data file by allocating columns in the data only for the number of times the questions in the inner loop are actually asked.

In the example below, the additional question is to be asked for Washo and Gleam only, so instead of reserving five columns for this question (there are five items in the value list for the outer loop), we want to reserve only two. We will look at the script now and then explain in more detail what it is doing afterwards.

```
reason      define 'Used before' / 'Heard about from a friend' /
            'Never heard of'
allbrd      define 'Suds' / 'Washo' / 'Bubbles' / 'Sparkle' / 'Gleam'
keybrd     define 'Washo' / 'Gleam'
comment Define question for inner loop as a dummy to which we can
comment assign values later
dloop      for item = 'Washo' / 'Gleam'
q2          dummyask 'How heard of product'
            resp sp reason nodata
            next

comment A dummy question with the key brands as the response list
comment for use later as an index into the q2 loop
q2idx      dummyask 'Index number for key brands'
            resp sp keybrd
comment Start of script proper
loop1      for brand = allbrd
            set iter = iteration
q1          ask 'How favorable are you towards '+brand+'?'
            resp sp 'Very favorable' / 'Favorable' /
            'Neither favorable nor unfavorable' /
            'Unfavorable' / 'Very unfavorable'

comment If this is a key brand, work out the index value for q2
comment and ask the question. Then set the answer into the data
comment area already allocated to q2
loop2      for test = keybrd in allbrd
            if (brand = test) {
                unset q2idx
                set q2idx = brand
                set index = nbit(q2idx)
q2get      ask 'Why did you say you were '+q1(iter)+'
            about '+brand+'?
            resp sp reason nodata
            set q2(index) = q2get(iter)
}
            next
        next
```

The main point to notice is that the extra question (q2) is defined, but not asked, before the main loop. At this point, we are reserving the columns we require for the extra question and are setting up an indexing system to ensure that later, when we copy data into those columns, we copy it into the right column.

The script as it appears to the interviewer starts at the label loop1. The start of the loop is straightforward and asks how favorably the respondent rates the current brand. After this, we're ready to ask the extra question for the key brands. This is the start of loop2.

We check whether the current brand for loop2 is a key brand and, if so, set the value of the index question (q2idx) to the name of that brand. We then use *nbit* to find the position of that brand in the response list to the index question. Since the response list to the index question is identical to the value list for the loop containing the extra question, the brand's position in the response list will be the same as the subscript value for the extra question related to that brand. For example, Washo is the first response in the list, and q2(1) will store the answer related to that brand.

Next, we ask the extra question but do not store the data at this point. Instead, we assign it to the appropriate cell of the q2 array we defined earlier.

If we had used a conventional nested loop for this example, we would have three columns in the data (for Suds, Bubbles and Sparkle) which would always be blank. With long value lists where the additional questions are required for a few items only, these blank columns can soon mount up. By using two separate loops, we are forcing the parser to allocate the minimum number of columns required.

Loops with all respondent-specific texts

This example shows a more complex script using respondent-specific texts in the value list. It illustrates how you can build a loop whose value list is completely dependent on responses chosen by the respondent. The script is also a useful reminder of how loops in general work, having five loops in all, one of which is nested.

```

papers      define 'The Independent' / 'The Guardian' / 'The Sun' /
              'The Daily Telegraph' / 'The Express' / 'The Daily Mirror' /
              'The Sunday Mirror' / 'The Independent on Sunday'
bought      dummyask 'Dummy for storing all papers bought'
resp mp papers
dloop       for keep = 1 to 3
just3        dummyask 'Select three papers at random'
resp coded(0)
next
ctl1        continue
comment Find which newspapers were bought each day and net them
comment into the bought variable for a weekly list
loop1       for days = 'Monday' / 'Tuesday' / 'Wednesday' /
              'Thursday' / 'Friday' / 'Saturday' / 'Sunday'
q1          ask 'Which newspaper(s) did you buy on '+days+'?'
resp mp papers in papers
set bought = q1
next

```

```
comment Loop to select 3 newspapers at random from those mentioned
      unset count
loop2    for choice = bought in papers ran
      set count = count + 1
      route (count > 3) go nxt
      set just3(count) = choice
nxt      next
comment Loop for the 3 chosen newspapers only
loop3    for pubs = '[+just3(1)+]' / '[+just3(2)+]' / '[+just3(3)+]'
loop4    for opin = 'Load of rubbish' / 'Good editorial' /
      'Good business section' / 'No real news'
q2          ask 'Thinking of '+pubs+, how much
do you agree or disagree with the statement:@'+opin+'@ Please use
a scale of 1 to 5, where 1 is agree very much and 5 is do not
agree at all.'
      resp num 1 to 5
next
next
```

The script starts with two dummy questions. The bought question is designed to store all newspapers bought throughout the week, even though the respondent is asked about each day's purchases separately. The just3 question is part of a loop, so that we have an array of three cells for storing the names of the three newspapers we want to talk about in more detail.

The loops for daily purchases and choosing three publications (loop1 and loop2) are straightforward and are amply explained by the comments in the script.

The script ends with a nested loop. The outer one (loop3) has as its value list the three newspapers chosen at random by the script. These are entirely dependent on the responses earlier in the interview. Notice the use of an array here rather than three separate variables. This helps to make the script easy to follow, since we know that all cells of the array store related information.

The inner loop (loop4) simply displays a newspaper name and asks for a rating on a given statement.

As with the previous scripts, this one illustrates an efficient method of dealing with this type of requirement. Because the rating questions are dealt with as a separate loop, we keep the number of columns used in the data file to a minimum.

11.7 Sample script

```
comment Sample script for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 11, Iteration and subscription

loop1      for item = 'TV' / 'Satellite TV' / 'Videos'
numhr      ask 'How many hours, to the nearest quarter hour, did you
spend watching '+item+'?'
            resp real 0 go nxtitem / 0.25 to 168
comment Create temporary variable to sum up hours for this item
            set timchk = 0.0
loop2      for ptype = 'films' / 'news programs' /
            'documentaries' / 'situation comedies' /
            'other programs'
ptime      ask 'How much of that time was spent
watching ' +ptype+'?'
            resp real 0 to 168

comment Add hours for current program into the total
            set timchk = timchk + ptime(*)
next
if (timchk <> numhr(*)) {
d1      display 'The sum of the times you have just
given me is '+timchk+' but the total time you gave earlier is '
+numhr(*)+'. Could I just recap on those figures?'
            pause

comment CAPI alternative to the 'display/pause' statements
comment      messagebox 'The sum of the times you have just
comment given me is '+timchk+' but the total time you gave earlier
comment is '+numhr(*)+'. Could I just recap on those figures?'

            goto numhr
}
nxtitem  next

comment Respondents who didn't watch satellite tv skip the next section
route (numhr(2) = 0) go exit

comment CATI only: Set up empty array for use in next loop
arrays 3
unset mc(1)
unset mc(2)
unset mc(3)
loop3      for prog = 'Family Films' / 'Children''s Films' /
            'Comedies' ran
comment Get the subscript for use with specified other
comment CAPI only: delete 'go setneg' from the response list
            set subs = subscript
```

```
swatch      ask 'Which channel do you usually watch for '+prog+'?'
             resp sp 'Sky One' / 'The Movie Channel' / 'Filmnet' /
                   'RTL Plus' other go getoth null go nxtprog
             set whatchan = swatch(subs)
             goto why

getoth      set whatchan = othertext(swatch(subs),5)

why         ask 'What are the reasons that you mainly watch '
            +whatchan+ for '+prog+'?
             resp mp 'It''s in a language I can understand' /
                   'Best selection' / 'It''s free' / 'Other'

comment Test current answer to the question
         route (why(*) <> '<It''s free>') go setit
pay         ask 'Would you still watch '+whatchan+ if it became a
            subscription channel?
             resp sp 'yes' / 'maybe' / 'no' dk
setit       route (whatchan <> 'The Movie Channel') go nxtprog

comment Set appropriate cell of array if key response was given in
comment this iteration
         set mc(subs) = 1

nxtprog    next
             if (mc(1)=1 .or. mc(2)=1 .or. mc(3) = 1) {
pay2        ask 'You said that you watch The Movie Channel which is
            a pay channel. Do you consider that the films shown on that channel
            are ... '
             resp sp 'Very good value for money' /
                   'Good value for money' /
                   'Not good value for money' /
                   'A waste of money'
             }
exit       display 'Thanks for your help'
             pause
             end
```

12 Assignment

Assignment is the process of giving a value to a variable. This chapter explains how assignment works in Quancept and introduces the following statements:

notemp	make temporary variables illegal
remove	remove a response from those selected from a multipunched list
set	assign a value to a variable
substitute	assign replacement text to a variable
unset	assign a null value to a variable
warntemp	warn when temporary variables are used

This chapter also describes the **longtext** callable function that stores responses to several questions under one variable name, and introduces a number of special variables that you can use with *set* to provide specific types of information during an interview.

12.1 Types of variables

A *variable* is an area of storage space which the interviewing program uses during the course of an interview. There are two types of variables: question variables and temporary variables. Question variables are the labels of questions and they store the responses given to those questions. Question variables may be real questions which are asked of the respondent or dummy questions whose answers are assigned by statements in the script. The values of question variables are always written to the data file unless the response list contains the keyword *nodata*.

Temporary variables are variables which you use to store information on a temporary basis during the interview. They are not normally written to the data file, although you may request that the interviewing program does this by naming the variable on a *fix* statement.

-
- ☞ See section 5.6, Asking questions without storing responses, for information about *nodata*.
 - See section 17.7, Inserting non-response data in the data file, for information about *fix*.
-

You have already seen temporary variables in many of the examples. The pointer variable in a loop is a temporary variable whose value changes each time the loop is restarted, as are many of the variables you have seen used with *set* statements.

Temporary variables are particularly useful for setting flags, or internal markers, indicating the respondent's position in the script, or for signaling that a certain combination of answers has been given. You are strongly advised to use a different temporary variable for each action or flag in the script and to use meaningful names for each variable. This makes the script easy to read and reduces the likelihood of a variable having an unexpected value after a routing statement or a snapback (that is, where the interviewer returns to a question which has already been asked).

-
- ☞ See section 33.27, Snapping backwards and forwards in an interview, to find out about snapbacks.
-

For example, if you use a temporary variable called *j* for a number of different purposes throughout the script, it will not be immediately clear what *j* refers to at a given point in the script, nor will it be so easy to determine what its correct value should be at that point. On the other hand, if you use a variable called, say, *numvac*s to count the number of vacation days taken in the past year and another variable called *europe* to indicate that at least one vacation was taken in Europe, your script immediately becomes easier to understand and to debug if it does not behave as you hoped.

12.2 The set statement

Quick Reference

To assign a value to a variable, type:

```
set variable = value
```

Question variables are normally assigned values by the interviewer typing in the code, number or text which represents the respondent's answer to the question. Temporary variables have their values assigned with a *set* statement. If the variable does not exist, the interviewing program creates it. If the variable already exists, the interviewing program overwrites its current value with the new value defined on the *set* statement, as long as the data type of the value is compatible with that of the variable.

Data types

The type of data that a question variable may contain is defined by the response type on the *resp* statement. The data type of a temporary variable is determined by the type of data you place in it the first time you use it. For example, suppose you write:

```
set name = 'Benjamin'
```

Using computer terms, Benjamin is a 'text string' so the parser creates the variable as a text variable. (This is the equivalent of a question with an open-ended response). If you reuse the variable 'name' later in the script, the parser will expect you to place a text value in it. Similarly, if you write:

```
set total = 0
```

the parser creates total as an integer variable which may store whole numbers only. To create total as a real variable, you must define its starting value as a real variable:

```
set total = 0.0
```

If you reuse a numeric variable, the parser expects you to use it with a numeric value. You may assign a real value to an integer variable or an integer value to a real variable, but the interviewing program will modify the value to suit the data type of the variable.

-
- ☞ See section 15.2, Variable types with arithmetic assignments, for more information about numeric assignments.
-

12.3 Text assignments

If the value you wish to assign to a variable is a text, you must enclose it in single quotes. You can use this form of assignment to build up questions which have variable texts. In the following example, we are assigning specific values to ‘pers’ and ‘verb’, depending upon the number of children in a family.

```
q1    ask 'Do you have any children?'
      resp sp 'Yes' / 'No' go exit
q2    ask 'How many?'
      resp num 1 / 2+ go t2
t1    set pers = 'your child'
      set verb = 'Has'
      goto q3
t2    set pers = 'any of your children'
      set verb = 'Have'
q3    ask verb+' '+pers+' been swimming during the last six weeks?'
      resp sp 'Yes' / 'No'
```

If there is only one child in the family, the script sets ‘pers’ to the value ‘your child’ and ‘verb’ to ‘has’. If there are several children, the values of these variables are altered accordingly. If the respondent has one child only, question q3 will be displayed as ‘Has your child been swimming during the last six weeks?’ . If the respondent has more than one child, the question will be displayed as ‘Have any of your children been swimming during the last six weeks?’ .

If necessary, you can build up a complete question text using variables. This would be useful where you have a standard question text in which you want to insert different variables. For example:

```
compcar    ask 'Does your company provide company cars for its
            employees?'
            resp sp 'Yes' / 'No' go ctn  dk go ctn
numcars1   ask 'How many employees have company cars?'
            resp num 1 to 999 dk go numcars2
            set manytxt2 = numcars1
            goto stxt1
numcars2   ask 'How many do you know of?'
            resp num 0 go ctn / 1 to 999
            set manytxt2 = numcars2
stxt1     set manytxt1 = 'How many of those '
```

```
set manytxt3 = ' cars were purchased within the last 6 months?@@'
many    ask manytxt1+manytxt2+manytxt3
        resp num 0 to 999
ctn     continue
```

The question to look at in this example is called many. Its text is a concatenation of the values of manytxt1, manytxt2 and manytxt3. manytxt1 and manytxt3 are the same for all respondents, but manytxt2 varies according to whether the respondent answered numcars1 or numcars2.

12.4 Assigning responses to questions

You may use *set* to assign responses to dummy questions in the script. There are three ways of doing this:

- Using the position of the response in the list. This is suitable for single-punched and multipunched response lists.
- Using the response text or value. This is suitable for all response types, although you must take care that you enter a number rather than a text for numeric and database response lists.
- Using the question name. This is suitable when you want to assign all the responses given to one question to the dummy question.

We will look at each of these possibilities below.

☞ You can also assign values to ordinary questions in the script. This is dangerous, since your changes may alter the path through the script or otherwise invalidate the data. If you need to alter the value of a question, copy it into a dummy question and alter the value of the dummy question instead.

☎ In versions of Quancept CATI up to and including version 7.7, if a script assigned an out-of-range response to a categoric question, Quancept CATI terminated the interview and wrote an error message to the tiperrs file. In version 7.8 and later, an assignment of an out-of-range response code (response position) is silently ignored, whereas for an assignment of a non-existent response text, Quancept CATI writes an error message to the tiperrs file and also displays that message at the foot of the interview screen. A blank response being assigned to the variable whose value was being set.

With out-of-range numeric assignments, Quancept CATI accepts the assignment and writes the out-of-range value to the drs file.

☒ mrInterview rejects out-of-range assignments for categoric questions but does not terminate the interview. Instead, it leaves the question's original response intact and writes the error message ‘–Sif: sp/mp question error, value is out of range; *qname*; *value*’ to the log file. mrInterview also accepts out-of-range numeric assignments, but writes the following warning message to the log file: ‘–Sif: num question warning, value is out of range; *qname*; *value*’.

- ➊ Quancept Web silently ignores out-of-range assignments and does not write data for that question to the data file. It is therefore important to check and test your script thoroughly to ensure that there are no errors of this kind.

Response texts

If the dummy question has a single-punched or multipunched response list, you may assign a response to it by typing that response text, enclosed in single quotes, on a *set* statement. If the question is single punched, the assignment will overwrite any value that the variable currently has, whereas if the variable is multipunched, the value will be added to any that the variable already has.

In the example below, we want to know which magazines the respondent knows and, if he/she mentions any of the 'key' magazines, to record them separately in the order in which they were mentioned. We have one question with a multipunched list which stores all the magazines named and a set of three dummy questions to store the key brands in the order in which they were mentioned:

```

house      define 'House and Garden' / 'Homes and Gardens' /'Ideal Home' /
            'Beautiful Home' / 'Living' / 'Good Housekeeping' /
            'Family Circle' / 'Traditional Homes'

comment Key magazines are Ideal Home, Beautiful Home and
comment Traditional Homes
dloop      for i = 1 to 3
hfirst      dummyask 'Key house magazines in order of mention'
            resp sp house
            next
hknow      ask 'Which home furnishing/management magazines can you
think of?'
            resp mp house other

comment Clear dummy question in case the interviewer snaps back
dloop2     for i = 1 to 3
            unset hfirst(i)
            next

comment Find order in which magazines were mentioned. If this
comment is a key brand, assign that response text to the
comment appropriate cell of hfirst
hloop1    for hpos = 1 to 8
            set tmpmag = mention(hknow,hpos)
            if (tmpmag = 3) {
                set hkeymag = hkeymag + 1
                set hfirst(hkeymag) = 'Ideal Home'
            }
            if (tmpmag = 4) {
                set hkeymag = hkeymag + 1
                set hfirst(hkeymag) = 'Beautiful Home'
            }
}

```

```
        if (tmpmag = 8) {
            set hkeymag = hkeymag + 1
            set hfirst(hkeymag) = 'Traditional Homes'
        }
next

comment mrInterview only: Treat valueless variables as null variables
set noaneqnl = 1
route (hkeymag = null) go ctn
hmag      display 'The key magazines known were mentioned in the
following order:@
hloop2    for key = 1 to 3
            route(key>hkeymag) go nxt1
            display hfirst(key)+', '
nxt1      next
pause
ctn       continue
```

-
- ☞ The section that deals with order of mention does not work in Quancept Web. However, it is still worthwhile studying the script as it shows the general principles of testing for specific responses and updating dummy questions based on the results of the tests.
 - ☞ For further information on *mention* in Quancept Web, see 'Order of mention for multipunched responses' in chapter 14, 'Checking non-numeric responses'.
-

The script makes extensive use of subscripted variables to ensure that it is as short and efficient as possible. The loop is repeated up to eight times, once per publication. We find the order in which magazines were mentioned and then check whether the current mention is a key publication. If so, we increment the count of key publications found and then assign the magazine name to the appropriate cell of hfirst.

If the respondent mentions Ideal Home, Living, Traditional Home and Homes and Gardens, in that order, the value of hfirst(1) will be Ideal Home and the value of hfirst(2) will be Traditional Home. hfirst(3) will be null because the respondent was not aware of the third key brand.

Notice the loop that unsets the dummy question hfirst. This ensures that the question will be reset to null if the interviewer snaps back and changes the hknow question so that different key brands are mentioned. Without this, Quancept could report that certain key brands were mentioned when they were not.

The *set* statements in this example show the texts in the same combinations of upper and lower case as are used in the defined list. This is not essential, because the interviewing program does not differentiate between case in assignment statements. As long as the texts contain the same letters, punctuation and number of spaces in the same places, the interviewing program is able to match the assignment text with its partner in the response list. If the assignment text does not match a text in the response list, the interview terminates with an error message.

-
- ☞ This notation is not valid for assigning respondent-specific responses defined as [+response+]. Use the response number instead, as described below.
-

- ☒ If you write scripts of this kind for mrInterview, you will need to set the variable *noaneqnl* to a non-zero value as shown in this example. Unlike Quancept, mrInterview does not treat valueless variables the same as variables that have been assigned a null value, so the logical expression (*hfirst=null*) will always be false. By setting *noaneqnl* to a non-zero value, you force mrInterview to treat valueless variables the same as null variables. An alternative is to write a separate expression to deal with empty (unanswered) questions.

☞ See chapter 14.11, Testing for null-response and unanswered questions, for details.

Abbreviating long response texts

Here is another example of using response texts in assignments. It suggests a method of shortening very long response texts into a form suitable for displaying in a response list, while retaining the original text for use in the rest of the interview.

```

qdum      dummyask 'Very long response texts'
          resp mp 'Volkswagen Scirocco GT' / 'Volkswagen Scirocco GTI'
          'Volkswagen Scirocco GTX' / 'Vauxhall Astra GTE' /
          'Porsche 944 Turbo Cabriolet' nodata

comment Real question uses abbreviated response texts
own       ask 'Which cars do you own?'
          resp mp 'Scirocco GT' / 'Scirocco GTI' / 'Scirocco GTX' /
          'Astra GTE' / 'Porsche 944'
comment Assign response codes to dummy question
          set qdum = own
comment Pick up original response text. The bit keyword checks
comment whether the response in position i was given to qdum. If
comment so, we use the elm keyword to get to response text.
11       for i = 1 to 5
          set given = bit(qdum/i)
          route (.not. given) go back
          set car = elm(qdum/i)
why       ask 'Why did you buy a '+car+'?'
          resp coded
back      next

```

The full names of the cars are too long to display in the response list, so we define them as a response list to a dummy question. The response list displayed during the interview contains abbreviated response texts in the same order as those they represent.

Once we know which cars the respondent owns, we assign those response codes to qdum as if it had been asked as a proper question. The loop checks which cars the respondent owns and obtains the long response text to display as part of the next question.

Response numbers

You may also make single-punched and multipunched assignments based on a response's position in the response list. Here is a variation of the magazine example (using a different list and key brands) to illustrate this:

```
womans    define  'Cosmopolitan' / 'Woman' / 'Woman''s Own' /
              'My Weekly' / 'Woman''s Journal' / 'Options' /
              'Woman''s Realm' / 'New Woman'
comment Key woman's magazines are Options, New Woman, Cosmopolitan
dloop      for i = 1 to 3
wfirst     dummyask 'Key woman''s magazines in order of mention'
            resp sp womans
            next
wknow      ask 'Which woman''s magazines can you think of?'
            resp mp womans other

comment Clear dummy question in case the interviewer snaps back
dloop2     for i = 1 to 3
            unset wfirst(i)
            next

comment Find order in which magazines were mentioned. If this
comment is a key brand, assign that response number to the
comment appropriate cell of wfirst.
wloop1    for wpos = 1 to 8
            set tmpmag = mention(wknow,wpos)
            if (tmpmag = 1) {
                set wkeymag = wkeymag+1
                set wfirst(wkeymag) = 1
            }
            if (tmpmag = 6) {
                set wkeymag = wkeymag+1
                set wfirst(wkeymag) = 6
            }
            if (tmpmag = 8) {
                set wkeymag = wkeymag+1
                set wfirst(wkeymag) = 8
            }
            next
comment mrInterview only: Treat valueless variables as null variables
comment    set noaneqnl = 1
            route (wkeymag = null) go ctn
dmag       display 'The key magazines known were mentioned in the
                  following order:@'
```

```
wloop2    for key = 1 to 3
          route(key>wkeymag) go nxt1
dmag2      display hfirst(key)+', '
nxt1      next
pause
ctn       continue
```

In this example, the key brands are in positions 1, 6 and 8 in the list. To assign those responses to the dummy questions, simply type the response numbers.

If you enter a number larger than the number of responses in the list, the columns in the data file will be blank for that question.

Question labels

The third method of assigning responses to questions is to copy the values belonging to one question variable into another one. This often happens when you have a dummy question to which you want to assign responses as they are given at various points in the script. A common example is a brand awareness study in which you ask respondents to list the brands or products they are aware of. Then, for all brands or products not spontaneously mentioned, you ask specifically whether or not a respondent knows a particular brand or product.

Later parts of the script may need to refer to all brands or products known as a single group rather than as the two groups in which they were collected. By merging, or **netting**, the two groups into a summary question, you can refer to the known brands as a single entity.

The example below shows the last part of the magazine script, in which we merge the two sets of key brands mentioned into a single dummy question variable called allkey. We have included the *unset* statement required to make this part of the script function correctly with snapbacks.

```
allmags   define 'Cosmopolitan' / 'Woman' / 'Woman''s Own' /
           'My Weekly' / 'Woman''s Journal' / 'Options' /
           'Woman''s Realm' / 'New Woman' /
           'House and Garden' / 'Homes and Gardens' /
           'Ideal Home' / 'Beautiful Home' / 'Living' /
           'Good Housekeeping' / 'Family Circle' /
           'Traditional Homes' other
allkey    dummyask 'All key brands mentioned'
resp mp allmags

comment Sections for women's and house magazines as shown above
comment Copy values of hfirst and wfist into allkey. Unset
comment allkey first to ensure it is empty in case of snapbacks
unset allkey
aloop    for subs = 1 to 3
          set allkey = wfist(subs)
          set allkey = hfirst(subs)
next
exit     display 'All key brands mentioned were: @'+allkey
pause
```

The advantage of collecting responses in this way is that you still have the two sets of key publications in separate variables should you wish to refer to them individually, but at the same time you have a netted response referring to all key publications collectively.

-
- ☞ The response list for the dummy question must match the response lists of the questions whose responses are to be merged.
-

If the dummy response list contains different texts, the responses merged will be those in the dummy list which are in corresponding positions to those selected from the proper response lists:

```
all      dummyask ''  
        resp mp 'red' / 'yellow' / 'blue' / 'green'  
flav    ask 'Which flavors do you like?'  
        resp mp 'strawberry' / 'peach' / 'plum' / 'rhubarb'  
        set all = flav
```

If the respondent chooses peach and plum, the dummy question will be assigned the responses yellow and blue.

If you use dummy questions and assignment in this way, other points you should bear in mind are:

- The questions whose responses are to be merged must either:
 - have the same response list (that is, the same number of quoted responses in the same order),
 - or use defined response lists of the form ‘sublist in masterlist’, which share the same master list (see chapter 7, ‘Defined response lists’).

If you forget to define the dummy question, a temporary variable with the given name will be created, but it will have no value because there will be no responses associated with that name.

- If you do not want the dummy question to become part of the data, append the option *nodata* to the response list. If you do put it in the data file, the columns associated with that question will contain the codes of the responses which were assigned to that question by the *set* statement.
- If you assign a question with a multipunched response list to a question with a single-punched response list, only the first multipunched response selected by the interviewer will be assigned.
- If you are using questions that have a specified other response, and you want to copy the other responses from one question into the other you must write a separate set statement to copy those responses. You can use negative subscription or *othertext* to pick up the responses. If you use negative subscription, you will need a statement of the form:

```
set q2(-1) = q1(-1)
```

-
- ☞ See section 14.7, Displaying specified other answers, for details
-

Assigning mp variables to sp variables in Quancept CATI v7.8



The behavior described above is the way all versions of Quancept CATI before version 7.8 work. From version 7.8 onwards you can choose how assignments of multipunched responses to single-punched variable works by setting the environment variable QCSET to 0, 1 or 2. When QCSET is 0, assignment works in the pre-version 7.8 way. When QCSET is 1, assignment is always based on response codes or positions and out-of-range assignments are ignored; this is the default. When QCSET is 2, assignment is based on response texts and out-of-range assignments generate error messages at the foot of the interview screen and in the tiperrs file, but do not cause the interview to fail. For example:

```
q1 ask 'Choose a color.'
      resp sp 'red' / 'blue' / 'green' / 'yellow'
q2 ask 'Now choose some more colors.'
      resp mp 'orange' / 'red' / 'white' / 'purple' / 'blue' / 'brown'
      set q1 = q2
```

Assume that the original answer to q1 is always green. The table below shows how the value of QCSET affects the final value of q1:

QCSET	q2	new value for q1
0	blue, red	green (unchanged) because although blue is present in the q1 response list, it is position 5 in the q2 response list. There is no position 5 in the q1 response list.
0	purple, brown, white	yellow (match by response position).
0	brown, orange, blue	green (unchanged) because brown does not appear in the q1 response list, and there is not a response 6 in the q1 list.
1	red, orange	blue.
1	blue, red	green (unchanged) because there is not a response 5 in the q1 list.
1	purple, brown, white	yellow.
2	blue, red	blue.
2	purple, brown, white	green (unchanged) because purple does not appear in the q1 response list. An error message is generated.

12.5 Assignments with null, dk and ref

To assign a null, dk or ref value to a temporary variable, enter the appropriate keyword on the right side of the assignment statement:

```
set tmpvar1 = null  
set tmpvar2 = dk  
set tmpvar2 = ref
```

You may use this type of assignment only with questions where null, dk and/or ref are valid responses to those questions. If the response list does not allow dk as a response, an assignment of the for *set q1=dk* will be rejected by the parser with the message ‘Variable q1 assigned or compared to impossible value’.

- ▣ Quancept CATI, Quancept CAPI and Quancept Web treat unanswered questions and valueless variables the same as questions and variables that have been assigned null values. mrInterview does not. If your script contains logical expressions that test whether a question or variable has a null value, the script will behave differently depending on whether interviews are conducted with Quancept or mrInterview. If you want mrInterview to treat unanswered questions and valueless variables as if they were null, you will need to set the variable *noaneqnl* to a non-zero value at or near the top of the script.

☞ See chapter 14.11, Testing for null-response and unanswered questions, for details.

12.6 Assigning values to subscripted variables

You can usually assign values to subscripted variables in the same way that you assign values to unsubscripted variables. The exception is when using the notation *set qname(*)=value* to assign values to questions in a loop.

- ⌚ In Quancept Web, the notation *set qname(*)=value* works only when used with questions that appear in the same loop as the assignment statement. For instance:

```
11    for i = 1 to 5  
        set qnum(*)=3  
        qnum      ask 'Choose a number.'  
        resp num 1 to 10  
    next
```

-
- ↖ If you use this notation to refer to a question in a previous loop, Quancept Web always uses a value of 1 for the subscript. This type of subscription works properly in Quancept CATI, so if you are using a script that was originally written for CATI interviewing you must change these statements before using the script for Web interviewing.
-

- ➊ The notation `set qname(*)=value` does not work when `qname` is asked in a future loop. The question is assigned a value of zero or a very large number.

The notation `set qname(number)=value` does not work at all.

The notation `set qname(variable)=value` works correctly.

Therefore, if you need to assign default values to a repeated question outside the grid in which the question appears, use a loop of the form shown below instead:

```
11  for i = 1 to 4
    set num(i)=0
next
```

You are most likely to want to do this if you have grids containing questions with numeric or real responses and you want to define default values for those questions before they are asked. For instance:

```
11  for i = 1 to 2
    set meat(i)=0
    set fish(i)=0
next
d1  display 'What percentage of your grocery bill went on ...
12  for colgrid '' i = 'This week' / 'Last week'
meat   ask 'Meat'
       resp real 0 to 100.0
fish   ask 'Fish'
       resp real 0 to 100.0
next
```

12.7 Special keywords with set

Questions and other variables defined in the script can be displayed to show values specific to the current interview. In addition to answers gathered as part of the script, other interview-specific information exists which you may also find useful. This includes the time of day at which the interview takes place and the name of the interviewer conducting the interview.

You can obtain this and other similar information by including the statement:

```
set name = keyword
```

in the script, where `keyword` defines the information required. Keywords you may use are:

Keyword	Returns
<code>date</code>	Date when the interview was started. Quancept CATI returns the date in the format yyymmdd, whereas Quancept CAPI and Quancept Web return it in the format mmddyy. The date is returned as a text not a number.

Keyword	Returns
dayofweekn	Day of week as a number (Sunday=1).
dayofweeks	Day of week as a three-character string (for example, Mon). In CATI, if the interviewing program's messages have been translated into your local language and your local language is defined with the LANGUAGE environment variable, the days of the week will be displayed in that language.
heaptop	The 'heapspace' is the amount of memory allocated to each interviewing program process. Every interviewer has a default amount of 50000 bytes. Setting a variable to <i>heaptop</i> and then displaying the variable shows how much memory the interviewer's program has used up. You can use it to test whether scripts are too big for the interviewing program using the default heapspace. <hr/> ☞ You can increase the default heapspace with QCHEAPMAX. See section 39.2, Environment variables.
ifsnap	Sets the variable to true if the interviewer has reached the <i>set</i> statement by snapping back or false if not (that is, this is the first time the statement has been executed). <hr/> ☞ You will find more information about this facility in section 9.8, Testing for snapbacks.
intname	Interviewer's name. In Quancept CATI, this is the name entered at the interviewing program prompt or as defined with the environment variable QCWHOAMI. If the interview is stopped and restarted, the name is that of the interviewer completing the interview, rather than the name of the person starting it. In Quancept CAPI, this is the name defined in QCShell or on QCompile's Run tab. In Quancept Web and mrInterview, this is the unique identification code that is assigned to each interview. See section 36.4, Displaying the respondent's interview ID, for further details.
login	System login name of interviewer as in the Unix password file. If the interview is stopped and restarted, the name is that of the interviewer completing the interview, rather than the name of the person starting it.

Keyword	Returns
partofday	Part of day local to the interviewer (morning=midnight to noon; afternoon=noon to 6 p.m; evening=6 p.m to midnight). In CATI, if the interviewing program's messages have been translated into your local language and your local language is defined with the LANGUAGE environment variable, the parts of day will be displayed in that language.
qcproject	The project name (that is, the name of the script).
respondent	Respondent ID of the current respondent.
testingrun	Sets the variable to true if the interviewing program is running in automatic test mode or false if it is not.
<hr/>	
	☞ To find out more, see section 9.10, Actions for test interviews only (CATI).
<hr/>	
time	Time at which this statement was executed. The time is returned as a text in the format hh:mm:ss.
timeofday	Time of day local to the interviewer in the format hhmmss. This is a text not a number.
userinfo	System user number of interviewer as in the file /ip/users. If the interviewing program messages have been translated into your local language and your local language is defined with the LANGUAGE environment variable, the interviewing program will display the parts of day and days of the week in your language rather than in English

A simple introduction may be written as follows:

```
set pod = partofday
set myself = intname
intro    display 'Good '+pod+'. I ''m '+myself+' calling on behalf of
          SPSS MR Ltd.@Would you be able to spare some time now to answer
          a few questions on your weekly shopping habits?'
```

Information returned by these keywords may be written into the data file with *fix* statements.

☞ See section 17.7, Inserting non-response data in the data file, to find out about writing to the data file with *fix*.

Automatic null coding when no response is chosen



Quick Reference

To allow respondents to click Next without selecting an answer, include *null* in the response list and add the following statement to the script:

```
set silentnl = 1
```

To return to the default of requiring respondents to select a response, type:

```
set silentnl = 0
```

When a respondent clicks Next without choosing a response, mrInterview normally displays an error message and does not let the respondent continue until a response has been chosen. You can override this restriction by requesting that mrInterview should select null whenever no other response is selected.

For example:

```
comment Default behavior - respondents must select a response at every question
tbag    ask 'Do you use tea bags, loose tea or both?'
        resp sp 'Tea bags' ('bags') / 'Loose tea' ('loose')/
                  'Both' ('both') null
comment Switch on silent null - null will be selected automatically if
comment respondents click Next without choosing an answer. Null does not
comment appear in the response list that the respondent sees.
        set silentnl = 1
        if (tbag='Tea bags') {
shape      ask 'Which shape tea bags have you tried?'
        resp mp 'Round' ('round') / 'Square' ('square') /
                  'Pyramid' ('pyramid') null
        set count = numb(shape)
        if (count>1) {
diff       ask 'In your experience, does the shape of the
            bag affect the quality, taste or flavor of the tea?'
            resp sp 'Yes' ('yes') / 'No' ('no') null
        }
}
comment Revert to default behavior
        set silentnl = 0
drink     ask 'How many cups of tea do you usually drink per day?'
        resp num 1 to 30
```

You may use *silentnl* with categorical and numeric grids and multiasks, but it must apply to all iterations of the grid or all questions in the multiask. You cannot switch the feature on for some questions or iterations but not others.

Allowing blanks in numeric grids



Quick Reference

To allow respondents to leave blank cells in numeric grids, add the following statement to the script:

```
set emptyok = 1
```

To return to the default of requiring all cells to contain a valid response, type:

```
set emptyok = 0
```

If you want to be able to differentiate between answered and unanswered questions, you may prefer to use *silentnl* rather than *emptyok* because this will code the blank cells as null responses.

☞ See ‘Automatic null coding when no response is chosen’ for details.

Customizable error messages in Quancept Web



Quick Reference

To customize error messages that Quancept Web displays during interviews, type:

```
set message_variable = 'message text'
```

Quancept Web has a number of special variables that you may use to customize some of the error messages that respondents may see during an interview. The names and functions of these variables are:

Variable	Represents	Default if variable not set
msgansp	The prompt for an answer	Answer:
msgbrbut	Text to display if the respondent uses the browser's navigation buttons	You have use the browser buttons, please use Next/Previous buttons below
msginvan	Text to display if the respondent gives an invalid answer	Incorrect answer(s), please answer again
msgmisan	Text to display if an answer is missing	Missing answer(s), please try again

You can change a text as many times as you wish within an interview.

You can define these message texts using variables in a project's ini file. If the same message is defined in the ini file and the script, the script version overrides the text in the ini file. This is a useful feature because it means that you can switch between a default text defined in the ini file and a more specific text at certain points in the script. To cancel a message defined in the script and revert to the one in the ini file, simply unset the message variable.

Error messages are displayed in red at the top of the page, or as specified in the template file. In Quancept Web, the standard messages are displayed in red at the top of the text frame and are separated from the question text by a line. In contrast, user-defined error messages are displayed in the ordinary text font below the question text, with the result that respondents may miss them. You cannot display user-defined error messages at the top of the frame but you can specify a different color and have a line printed below them. Here's an example:

```
set msginvan = '<font color="red">That answer is not acceptable.  
Please try again.</font><br><hr width="100%"><br>'
```

This statement displays the error message in red. It then starts a new line, displays a horizontal rule across the full width of the frame, and then starts another new line.

12.8 Customizing messages in mrInterview



Quick Reference

To customize error and other messages that mrInterview displays during interviews, type:

```
set message_variable = 'message text'
```

Most of the error messages to do with invalid responses are held in variables and can be customized to suit your requirements. Each variable name consists of an identifier that represents the error, and a marker that represents the type of question at which the error occurred. For example, the variable *msgnuman* stores the message to be displayed when a non-numeric answer is entered in a numeric response box. *msgnum* is the identifier, and *an* is the marker that indicates that this is the answer to an ordinary question.

Similarly, the variable *msgnumar* stores the same message as *msgnuman*, except that it is used for rowgrids and includes a reference to the row that is in error. There are five markers:

- an** Ordinary answers
- ar** Rowgrids
- ac** Colgrids
- ma** Multiasks where errors are reported at the top of the multiask using
<mrData0 /ErrorField>
- ag** Numeric grids

Each group of error variables has the same default message text; for example, the variables `msgmisan`, `msgmisar`, `msgmisac`, `msgmisam` and `msgmisag` all report ‘Missing answer(s), please try again’. However, the messages for rowgrids, column grids and multiask are preceded by a reference that indicates the row, column or sub-question that is in error.

Message variables are as follows:

Variable	Function and Default Text
<code>msgmisan</code>	Displayed when a question has no response selected. The default is ‘Missing answer(s), please try again.’
<code>msgmisar</code>	
<code>msgmisac</code>	
<code>msgmisam</code>	
<code>msgmisag</code>	
<code>msgsgan</code>	Displayed when more than one answer is chosen from a single response list. The default is ‘Only one answer is allowed, please try again.’
<code>msgsgar</code>	
<code>msggangac</code>	
<code>msgsgam</code>	
<code>msgsgag</code>	
<code>msgnuman</code>	Displayed when a non-numeric value is entered for a numeric question. The default is ‘Answer <i>value</i> is not numeric, please try again.’
<code>msgnumar</code>	
<code>msgnumac</code>	
<code>msgnumam</code>	
<code>msgnumag</code>	
<code>msgrngan</code>	Displayed when a numeric value is entered that is outside the acceptable range for the question. The default is ‘Answer <i>value</i> is not in range <i>range</i> , please try again.’
<code>msgrngar</code>	
<code>msgrngac</code>	
<code>msgrngam</code>	
<code>msgrngag</code>	
<code>msgcntan</code>	Displayed when the number of responses chosen from a multiple response list exceeds the number of responses that the script will accept for this question. The default is ‘There are too many answers, only <i>n</i> are allowed.’
<code>msgcntar</code>	
<code>msgcntac</code>	
<code>msgcntam</code>	
<code>msgcntag</code>	
<code>msgsolan</code>	Displayed when a multiple response list contains single choice responses, and a single choice response is selected with other responses. The default is ‘Answer <i>text</i> cannot be combined with other answers.’
<code>msgsolar</code>	
<code>msgsolac</code>	
<code>msgsolam</code>	
<code>msgsolag</code>	
<code>msgotman</code>	Displayed when the respondent selects Other but leaves the related text box empty. The default is ‘Answer <i>text</i> does not have any text.’
<code>msgotmam</code>	

Variable	Function and Default Text
msgotian	Displayed when the respondent enters an open-ended text but does not also select the Other check box or radio button. The default is ‘Answer <i>text (index)</i> has some text but is not selected.’
msgotiam	Displayed when an other specify response is too long. The default is ‘Answer <i>name (number)</i> has too much text, current length <i>curlen</i> , maximum <i> maxlen</i> .’
msgtxlam	Displayed when an open-ended response is too long. The default is ‘The answer has too much text, current length <i>curlen</i> , maximum <i> maxlen</i> .’
msgtxlan	

You can also customize non-error messages and other texts that appear on the interview screen by defining replacement texts using the following variables:

Variable	Description	Default if variable not set
msgansp	The prompt for an answer	Answer:
msgoth	The prompt for specified other responses.(Does not affect ^o texts unless msgothal is set to 1.)	Other.
msgbrbut	Text to display if the respondent uses the browser’s navigation buttons	You have use the browser buttons, please use Next/Previous buttons below.
msgintrn	Displayed when there is an error in mrInterview itself. You should not normally expect to see this message.	Internal error <i>description</i> .
msgend	The message that is displayed at the end of the interview. If you specify a template or URL to be displayed at the end of the interview, any text on this page override the value set in this variable.	End of interview. Thank you for your participation.
msgalrcmp	The message that is displayed when a respondent tries to restart an interview that is already complete.	The interview is already completed.
msgrgto	The character to use for separating the minimum and maximum values in a range displayed by any of the <i>msgrng</i> variables.	A hyphen (for example, 1–10).
msgrror	The character to use for separating ranges in a list of ranges displayed by any of the <i>msgrng</i> variables.	A comma (for example, 1–10, 11–20).

Making message texts translatable — version 2.2 and later method

mrInterview version 2.2 comes with a file of translated message texts. The file is usually installed as c:\Program Files\SPSS MR\mrInterview\Server\ErrorMessage.mdd (although the pathname may be different if mrInterview was installed somewhere other than c:\Program Files) and it contains message texts in the following languages: Afrikaans, Chinese (Hong Kong dialect), English, Dutch, French, German, Italian, Japanese, Korean, Spanish and Turkish.

The interviewing system (specifically, the mrEngine service) loads this file when it starts and then passes a reference to the file to each interview as it starts. When the interviewing program needs to display a message, it looks for the message in the appropriate language in the following places:

- In the message variables defined in the script, as in earlier versions of mrInterview (see below).
- In the project's mdd file or in ErrorMessage.mdd. Each document is searched using the current labeltype, context and language; if that fails, the search is repeated using the current language and the default labeltype and context; if that fails, the search is repeated using the first language named in the mdd file and the default labeltype and context.

If the message cannot be found in any of these locations, the interviewing program uses the built-in English default message text.

-
- ☞ Storing message texts in the project's mdd file without using the message variables is only likely to happen if you want different messages for a particular project. The messages could be added to the mdd file using MDM Explorer or a program of your own making.
-

There are a number of variables that you can use to change the way in which mrInterview searches for message texts.

Variable name	Description
msglang	mrInterview takes the current language for message texts from the <i>language</i> variable. Use <i>msglang</i> to locate messages in a different language.
msglangd	When the message cannot be found in the current language, mrInterview searches for the language in the first language defined in the mdd file. Use <i>msglangd</i> to define a different first language. (Note that the current language in ErrorMessage.mdd is never used.)
msgmdord	The default is to search the project's mdd file before ErrorMessage.mdd. To reverse the order, set <i>msgmdord</i> to a non-zero value.
msgmdidx	Normally, mrInterview uses the label text for the first element with index 0, but you can set the index to a different value using <i>msgmdidx</i> . The first element has index 0, the second element has index 1, and so on.

Variable name	Description
msgprefix	mrInterview searches for messages using the names of the variables as they were known in earlier versions (for example, <i>msgsngan</i> , <i>msgmisan</i>), but mrInterview 2.2 and later allows you to insert a prefix in front of the standard names. If you do this, you must define the prefix you have used using the <i>msgprefix</i> variable. mrInterview will then search for message texts whose variable names start with the given prefix. For example, if you write <i>set msgprefix='Messages.'</i> , mrInterview will search for messages defined as <i>Messages.msgsngan</i> , <i>Messages.misan</i> , and so on.

Making message texts translatable — pre-version 2.2 method

-
- ❖ The notes that follow describe the only way in which message texts could be translated. This method is retained for backwards compatibility so that pre-version 2.2 scripts continue to work. However, you may prefer to use the ‘version 2.2 and later’ method for new scripts.
-

Neither the default texts for these variables nor any replacement texts that you define in the script are translatable. If you want these messages to be translated, you may define them as the responses to dummy questions at the start of the script. The names of the questions must be the names of the variables whose text appears in the response list. For example:

```
msgmisan    dummyask ''
             resp sp 'Missing answer' nodata
msgbrbut    dummyask ''
             resp sp 'Browser buttons not valid. Use Next/Previous instead' nodata
```

To save you typing, the full list of error messages is available in the file *messages.qli* in the main project source directory (usually *c:\mrInterviewSource*), together with the default texts for *null*, *dk*, *ref* and *other*. If you include this file in your script, the texts will be available for translating with *mrTranslate*. Translations into various European and Far Eastern languages are provided with *mrInterview* and are installed as *mdd* files in the project source directory.

To set up a project to use translated error messages, do the following:

1. Insert the statement *include messages.qli* at or very near the top of the script.
2. Compile the script.
3. Open the *mdd* file using *mrTranslate* and make the translations.

You can define more than one text for a variable if you wish. This allows you to display different messages for a particular error at different points in the script. To do this, define the texts you want to use for the variable using a dummy question, and then use *set* statements to select the message text to be used at different points in the script. For example:

```

msgmisan dummyask ''
    resp sp 'You have forgotten to choose an answer' /
        'Please choose an answer from the list' nodata
    set msgmisan=1
q1    ask ....
    resp ....
q2    ask ....
    resp ....
    set msgmisan=2
q3    ask ....
    resp ....
q4    ask ....
    resp ....

```

If the respondent clicks Next without giving an answer to q1 or q2 the message ‘You have forgotten to choose an answer’ is displayed. If the respondent does not choose an answer for q3 or q4, the message ‘Please choose an answer from the list’ is displayed.

If you define multiple texts in this way but do not indicate which one is to be used, mrInterview will use the first text defined for the variable.

12.9 Setting the language in multilingual mrInterview scripts



Quick Reference

To set the language for the script, type:

set language = (value)

where *value* either a question or temporary variable holding the number or the three-character code of the language as it is known to the questionnaire definition file, or an integer that is the number of the language in the questionnaire definition file. Languages are numbered from 0 not from 1.

mrInterview provides a quick and easy way of delivering questionnaires in a number of languages. You write the script in the usual way and then it is translated using mrTranslate. The translated texts are held in the project’s questionnaire definition file and can be picked up from there during interviews.

The language in which the interview is to be conducted is determined by the script, either through a direct question or through a variable whose value is set by the script. For example, if information about a respondent’s native or preferred language is held in the sample record, this information can be extracted and used by the script to set the language for the interview. If no language is defined, then the project’s base language is used. This is the language in which the script is written or the language that was chosen as the base language the last time the project was activated.

If you need to ask the respondent what language to use in the interview, you might type:

```
langq ask 'Choose a language for the interview.'  
resp sp 'English' ('ENU') / 'Spanish' ('ESN') /  
      'French' ('FRA') / 'Danish' ('DAN')  
set language = (langq)
```

The three-character codes after the language names are the unique identifiers used for the languages in the questionnaire definition. If the respondent wants to be interviewed in Danish, the *set* statement assigns the code DAN to the language variable and mrInterview then extracts the texts in that language from the questionnaire definition.

-
- ❖ A temporary variable that has been assigned the value of categorical variable retains the value of that variable in the language that was current at the time the assignment was made, regardless of any language changes that may occur later in the interview.
-

If the language information is available as part of the sample, you can extract it from the sample record and assign it to the *language* variable as follows:

```
comment Extract language from lang and save in ilang  
callfunc('getsmvar',lang,ilang)  
set language = (ilang)
```

This example assumes that the sample record stores the language as a three-character code in the lang variable.

The questionnaire definition (mdd) file can hold several versions of a script. If you translate a script and then make changes to it, the reparsing and recompilation process adds a new version of the script to the questionnaire definition file and carries forward translations from the original version into the new version. It is then only necessary to translate any new or changed texts.

-
- ☞ For information on setting the interview language in Quancept CATI scripts, see section 27.2, Scripts for multilingual CATI interviews.
-

12.10 Saving copies of SQL statements that write data



Quick Reference

To save copies of all SQL statements that write data, type:

```
set logsql = 1
```

mrInterview writes interview data to an SQL database in the project directory. It also maintains a log file called ISEnn.tmp in which it records the actions it has taken and any problems that it encountered. When a log file reaches approximately 4Mb in size, mrInterview closes that file and opens a new one. This process continues until there are ten log files. Once the tenth file reaches its maximum size, mrInterview starts overwriting the first log file.

If mrInterview is unable to write out data at the end of the interview, it writes the failed SQL statements to these log files and also to a unicode file called SifSql.log in the LogSql subdirectory of the Project directory. You can usually retrieve the data that should have been written to the database from this file.

If you want SifSql.log to contain all SQL statements that write data, not just those that failed, place the statement:

set logsql = 1

in your script, or define a dummy question called logsql and set it to 1. (Any non-zero value is acceptable for both the variable and the dummy question.) You can place this statement anywhere in the script, so you might decide to save the SQL statements only if the respondent has reached a certain point in the interview.

12.11 Different texts for Web and WebCATI interviewing

- In order to reach the widest cross section of potential respondents, companies are increasingly running multimode projects, where data is gathered using a number of different methods. Version 2.2 and later of mrInterview provides facilities for standard self-administered (Web) interviews, as well as the more traditional interviews conducted by an interviewer contacting potential respondents by telephone (WebCATI), all using a single questionnaire file.

When a questionnaire is used for both Web and WebCATI interviewing, you will sometimes want to vary the question text between the two types of interviews. A common example is a question about gender. In a Web interview, you will usually want to use a minimal question text that asks respondents to choose gender from a list—for example, ‘Are you ...’. In a WebCATI interview, you may want to prompt the interviewer to enter the respondent’s gender without asking the question—for example, ‘INTERVIEWER: Enter respondent’s gender’.

The Dimensions Data Model that writes the questionnaire texts (the metadata) to the mdd file is able to store a number of different texts for each question or displayable text. By default, mrInterview reads question and displayable texts from the Question context within the Label label type.

To use different texts for different types of interviews, you can define a new context that contains the texts you want to use, and then place statements in the script that specify the context from which the texts for the current interview should be read.

-
- ☞ Refer to Dimensions Development Library for more information about contexts and label types. This is available as a free download from <http://www.spss.com/spssmr/DDL/>.)
-

The following table shows the texts to be used in a short questionnaire that can be completed by respondents themselves or by telephone interviewers.

Question	Text for Web (self completion)	Text for WebCATI
gender	Are you...?	INTERVIEWER: Enter respondent's gender
Age	How old are you?	How old are you?
Likes	What did you like about the product? Please be as specific as possible.	What did you like about the product? INTERVIEWER: PROBE FULLY.
Dislike	Was there anything about the product that you disliked? Please be as specific as possible.	Was there anything about the product that you disliked? INTERVIEWER: PROBE FULLY.
Buy	How likely is it that you would buy this product?	How likely is it that you would buy this product?

To implement these changes you would do the following:

1. Write the questionnaire using the Web interviewing texts.
2. Insert a section at the start of the script that tests whether this is a self completion or interviewer driven interview, and sets the context to Question for a self completion interview or to CATI (or another name of your choice) for interviewer driven interviews. See the next section for details.
3. Parse and compile the questionnaire.
4. Use the DimensionNet Files activity to download the mdd file onto your computer.
5. Run MDMLLabelManager and add a context (for example, CATI) that matches the context you used in the questionnaire.
6. Set the context alternative to Question and change the questionnaire base language so that it is not read-only.
7. Run mrTranslate to insert the WebCATI texts in the CATI context (you need only insert the texts that differ).
8. Use Files to upload the mdd file into the project's folder within your User area.
9. Activate the questionnaire for CATI use.
10. Parse and compile the questionnaire to regenerate the sif file using the revised mdd file.
11. Test the questionnaire to ensure that the different texts are selected at the appropriate times.

Defining the context in the script



Quick Reference

To define the context for the current respondent, place the following statement at or near the beginning of the questionnaire script:

```
set context = 'value'
```

where *value* is the name of the context from which the texts for the current interview are to be read.

A text's context in the mdd file determines the environment in which it will be used. The default context is Question. When setting up a script for both Web and WebCATI interviewing, you can use the Question context for one set of interview texts and then define a separate context to store the texts that differ for the other interviewing environment.

For example:

```
set svar = 'wc'
callfunc('getsmvar',svar,ivtype)
if (ivtype='cati') {
    set context='CATI'
}
else {
    set context='Question'
}
```

In this example, the interview type is being read from the wc field in the sample record. If wc contains CATI then the interview is a WebCATI interview being conducted by an interviewer, so the texts for the interview will be read from the CATI context. If wc contains anything else the texts from the default Question context are used.

Defining the label type in the script



Quick Reference

To change the label type used during an interview, type:

```
set labeltyp = 'value'
```

where *value* is the name of the label whose contexts you want to use.

The Label Type defines a text's type. The usual label type for mrInterview texts is Label.

12.12 Other variables that can be set in mrInterview

- ■ The following variables can be set to control the behavior of various features in mrInterview:

Variable	Function
dsplymax	Increases the maximum number of <i>display</i> and <i>protect</i> statements that may appear on a single page. When the limit is reached, the message ‘too many <i>display</i> or <i>protect</i> statements’ is written to the ISE log file and the 101st text is displayed. (The message also includes the text of the 101st <i>display/protect</i> statement and the current maximum.) Further <i>display</i> and <i>protect</i> statements are silently ignored.
nolimoth	In versions of mrInterview prior to version 2.1, a response list of the form <i>resp sp not q1 in list1</i> , where the response list to q1 contained <i>other</i> , would always include <i>other</i> in the response list to the current question even if it was chosen at q1. In version 2.1 and later, <i>other</i> is treated like an ordinary quoted response and is included in the current response list only if it was not chosen at q1. To revert to the earlier behavior, set <i>nolimoth</i> to a non-zero value.
ofpathem	When a question changes from being on the path through the interview to being off-path (that is, it is no longer applicable), mrInterview does not delete the data held in that variable. This can occasionally lead to situations where a questionnaire behaves incorrectly because the interviewing program is basing its decisions on the data in an off-path question. For example:

```

q1 ask 'Which brand do you prefer?'
    resp sp 'Brand A' ('brda') / 'Brand B' ('brdb')
    if (q1='Brand B') {
        q2 ask '...?'
            resp sp 'Yes' ('yes') / 'No' ('no')
        }
        q3 ask '...'
            resp coded
            if (q2='yes') (
d1     display '...'
            )

```

Suppose the respondent chooses Brand B at q1 and Yes at q2, and then snaps back from q3 to q1 and chooses Brand A instead. q2 now becomes off-path and the respondent goes straight from q1 to q3. However, because the answer to q2 has not been cleared, the d1 display statement will still be executed even though it is not appropriate for the current path through the interview.

Setting *ofpathem* to a non-zero value causes mrInterview to treat all off-path variables as if they were blank.

If an off-path question reverts to being on the interview path, the original response to that question becomes available when the page for that question is displayed. The question appears with the original response preselected, so the respondent can just click Next to continue.

Variable	Function
ofpathmd	Although mrInterview holds off-path data in memory until the interview ends, it does not write that data to the database. From version 2.2 onwards, you can specify whether off-path data is to be saved by setting the variable <i>ofpathmd</i> to one of the following values: <ul style="list-style-type: none"> 0 Always clear off-path data when the interview terminates, regardless of how or why this happens. This is the default. 1 Clear off-path data only when the script terminates normally. Do not clear off-path data when the interview ends in the middle of the script due to a time-out or the respondent clicking Stop, or because the interviewing server was shut down. If the script does not terminate normally and data is written to the database, any off-path data will be included with the valid interview data. 2 Never clear off-path data. Off-path data will always be written to the database with the valid interview data.

12.13 The unset statement

Quick Reference

To set a variable to its original valueless state, type:

unset *variable_name*

With questions, *unset* clears the response that was given or assigned to that question. This has the effect of blanking out the columns in the data file associated with those questions so that it appears that the questions have never been asked.

Additionally, once a question has been unset, the interviewing program sees it as unanswered (for example, as if it has been skipped using routing) and the interviewer will not be able to snap back to it during the rest of the interview. If the interviewer snaps back to a question before the unset one, the unset question will be treated as a completely new question and the interviewer will not be able to snap forward past it or type " to retain its original value.

unset is especially useful with temporary variables which accumulate totals or with dummy questions which are assigned values during the course of the script, since it ensures that the temporary variable or dummy question is always reset before values are added. We talked about this earlier when we looked at assigning answers to dummy questions, but here is a simple numeric example to explain the process in detail:

```
qa    ask .....
resp .....
unset total
```

```
11    for value = 1 to 5
        set total = total + value
    next
qb    ask 'question text'
resp .....
```

The loop is repeated five times and simply adds the current loop value into the total variable. At the end of the loop, total will be 15. If the interviewer snaps back to qa and then returns to qb, the interviewing program will reset total to its original valueless state before repeating the loop a further five times. Thus, total will always be 15 at the end of the fifth iteration.

If we had omitted the *unset* statement, total would still be 15 after the first set of iterations, but after a snapback to qa and then a further five repetitions of the loop, the value of total would be 30.

12.14 Removing a single response from a multipunched selection

Quick Reference

To remove a single response from those selected from a multipunched list, type:



remove *qname response*

where *qname* is the question from which *response* should be removed. *response* is either the position of the response in the response list, or the response text in single quotes, or the name of a temporary variable containing a suitable integer or text value.

Sometimes during interviewing you may wish to remove responses that have been written to the data, or to limit the responses available to a respondent on the basis of how he or she answered earlier questions. For example, you might have included a fake or rival product in a response list in order to test the respondent's market awareness, but you only want to store data for your own products. The *remove* statement allows you to modify the data collected by removing a response from the results of a multipunched question.

If you want to remove a fixed response from a list of responses chosen, you can enter the response text or response number as part of the command:

remove *qname 'resp_text'*

For example:

```
brands ask 'Which brands can you name?'
      resp mp 'Suds' / 'Washo' / 'Gleam' / 'Sparkle' / 'Fresh and Clean'
comment Remove competitors' brands (Suds and Sparkle) from the list
      remove brands 'Suds'
      remove brands 4
```

To remove several responses from a multipunched question's response data, use the remove keyword in a loop. When using remove inside loops you must identify the response to be removed by using its iteration number. If you try to identify the response using its text the parser will issue an error message.

For example, to remove the respondent's favorite brands from the list of those that the respondent knows, you would set up a defined list so that you can refer to the responses by number:

```

brands    ask 'Which brands of washing powder can you name?'
          resp mp master
fav      ask 'And which of these are your favorites?'
          resp mp brands in master
11       for prefer = fav in master
          set iter = iteration
          remove all iter
          next
notfav   ask 'If your favorite powders were not available, which
          other powders might you consider using?'
          resp mp all in master

```

12.15 Checking for temporary variables

Quick Reference

To have the parser issue a warning when a script uses temporary variables, type:

warntemp

To prevent an interviewing program being created if the script contains temporary variable, type:

notemp

If used, these keywords should appear near the beginning of the script. (If the script contains the keyword *mapothzero*, this must remain as the first line in the file).

Take the following script as an example:

```

warntemp
q1    ask 'Do you have any children?'
      resp sp 'Yes' / 'No' go exit
q2    ask 'How many?'
      resp num 1 / 2+ go t2
t1    set pers = 'your child'
      set verb = 'Has'
      goto q3
t2    set pers = 'any of your children'
      set verb = 'Have'

```

```
q3      ask verb+ ' +pers+ ' been swimming during the last six weeks?'
        resp sp 'Yes' / 'No'
exit    end
```

When you parse this script, qparse will create a qoc file but will warn you of the existence of temporary variables (that is, those used with the *set* statements) as follows:

```
script:13 WARNING Variable pers is a temporary variable never defined
script:13 WARNING Variable verb is a temporary variable never defined
Quancept Parse-0 errors, 2 warnings
```

If the script contained the keyword *notemp* instead, the warning messages would be the same, but the last line would report 2 errors and 0 warnings instead.

12.16 Combining responses and other values with longtext



Quick Reference

To store responses to several questions under one variable name, type:

```
label  dummyask '[text]'
      resp coded
      callfunc('longtext', label, var1, var2, ... var30)
```

where *label* is the name of the variable to hold the responses and *var1* to *var30* are the names of question variables or temporary variables whose values are to be stored.

You can use the callable function *longtext* to store responses to several questions and the contents of temporary variables in one variable. The responses stored in this way can be of any type, including open-ended and numeric, and different types of responses can be saved in the same *longtext* variable. This will be useful if you want to keep several very long open-ended responses in one variable.

-
- ❖ If you use *longtext*, you should instruct interviewers to enter all verbatim responses during the interview, rather than waiting for the prompt ‘Do you want to append/check verbatims now (y/n) ?’. This is because the values stored in the *longtext* variable are the values in the individual question variables at the moment the *longtext* callfunc is executed. Therefore, *longtext* can contain only values that are entered during the actual interview. Any verbatims changed after the prompt will not be recorded in the *longtext* variable.
-

The following example will store the responses to both whyprod and whybrand in the dummy question probrd:

```

whyprod  ask 'Why did you buy that product?'
          resp coded
whybrand ask 'Why did you buy that particular brand?'
          resp coded
probrd   dummyask ''
          resp coded
          callfunc('longtext',probrd,whyprod,whybrand)

```

When *longtext* is executed for the first time within a script, the interviewing program creates a directory called *longtext* within the directory *script.dir* and, within this, a file whose name is the text serial number of the dummy question used to store the responses. Each subsequent call of *longtext* creates another file in the *longtext* directory.

Files in the *longtext* directory have the following format:

```

Serial: respondent number, Text: text serial number, Target Var: varname(iter)
Variable qlabel_1=
response to first question
Variable qlabel_2=
response to second question
Temporary variable number number=
value for temporary variable

```

As you will see, question names (*qlabels*) are preceded by the word ‘Variable’ and are followed by the responses to the question, as entered by the interviewer. Temporary variables are preceded by the words ‘Temporary variable number’ and are followed by the value the variable contained when *longtext* was executed. (If your *longtext* statement used question names only, the file for that statement will contain Variable entries only.)

The example below contains three questions. The text serial number of the *longtext* variable will depend on how many texts have been entered before the point in the script at which *longtext* is executed. Assuming the respondent answers ‘other’ at question reason1 and the interviewer enters a verbatim response the first time the interviewing program is run, the variable bigopen will be the second open-ended response entered in the interview (it is entered by the *longtext* callfunc). The interviewing program will therefore create a *longtext* directory containing a file called 2. The contents of this file are shown below after the Quancept code.

The second time the code is executed, if the respondent gives an *other* answer and the interviewer enters a verbatim response at question reason1, the interviewing program will create a file called 4 in the *longtext* directory and so on.

```
reason1    ask 'Please indicate which of the following factors influenced
           your choice of holiday'
           resp mp 'price' / 'destination' / 'advice from friends' /
                 'advice of travel agent' other
pricel     ask 'Approximately how much did you pay for your holiday,
           per person?'
           resp num 0 to 200 / 201 to 500 / 501+
           if (pricel < 1) {
               set temp1 = 'free'
               goto bigopen
           }
           if (pricel < 201) {
               set temp1 = 'cheap'
               goto bigopen
           }
           if (pricel < 501) {
               set temp1 = 'average'
               goto bigopen
           }
           set temp1 = 'expensive'
bigopen    dummyask ''
           resp coded
           callfunc('longtext', bigopen, reason1, pricel, temp1)
end
```

If the responses given to the questions in the above example are:

Question	Responses
reason1	2 (destination), 3 (advice from friends), 5 (other) I saw a fantastic television program which made the place look like paradise.
pricel	650

the contents of the file called 2 will be:

```
Serial:1, Text:2, Target Var:BIGOPEN(1)
Variable REASON1=
destination; advice from friends; ;SER 1/<5:OTHER>;I saw a
fantastic television program which made the place look like paradise.
Variable PRICE1=
650
Temporary variable number 3=
expensive
```

12.17 Assigning a replacement value to a variable



Quick Reference

To assign a replacement value to a variable, type:

```
substitute variable_name value
```

where *variable_name* is the name of the variable and *value* is the value with which you want to replace *variable_name* each time it appears in the script.

As you are writing a script, you may find yourself reusing the same value a number of times throughout the script; for example, you may have a number of loops that are all repeated for the same range of values, or a number of questions that all ask for a maximum of three responses. If the script will be used once only, then it is not a problem to type the exact values for each loop or response list each time. However, if you are writing a script that will be re-used in a modified form, and you need to change the loop ranges or the number of responses allowed, you will have to search through the script and change each occurrence of the item separately.

In situations such as this, you may like to consider replacing these repeated items with variables. This allows you to define the variable's value once and to have Quancept substitute the appropriate value whenever the variable is used in a statement. For example, suppose you define a variable called looptop to represent the maximum number of repetitions for a set of loops:

```
substitute looptop 8
```

If you replace the upper value in each *for* statement with this variable, you can change the upper value of all those loops simply by changing value on the *substitute* statement:

```
loop1    for i = 1 to looptop
```

The same rules apply if you set a maximum number of responses that may be chosen from a multipunched list. Define the variable and its value and then use that variable in the *resp* statements:

```
substitute maxr 3
q1      ask 'Which brands can you think of?@'
INTERVIEWER: Code first three mentions only'
resp mp(maxr) allbrds
```

Another use might be to declare the number of codes to be reserved for all or a subset of defined lists within the script. This is particularly useful if your script has many defined lists whose sizes vary between waves and you always want to calculate which responses should be displayed based on the size of the list. For example:

```
substitute defsize 10
master define(defsize) 'red' / 'blue' / 'green' / 'yellow' /
                           'white' / 'black' / 'pink' / 'orange'
```

You can also use substitute for text substitutions. For example:

```
substitute advert 'SuperSuds'  
q19 ask 'Have you seen or heard any advertising for '+advert+'?'  
resp sp 'Yes' / 'No'
```

-
- ❖ Unlike the temporary variables that you define with *set*, variables defined with *substitute* cannot be redefined with different values later in the script. Once you declare a substitution variable, that variable must retain the same value for the whole of the script.
-

12.18 Sample scripts

Script A

```
comment Sample script A for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 12, Assignment (CATI/CAPI only)
comment CAPI variants using the keyword substitute are included in comment
comment statements.

allmags    define 'Cosmopolitan' / 'Woman' / 'Woman''s Own' /
            'My Weekly' / 'Woman''s Journal' / 'Options' /
            'Woman''s Realm' / 'New Woman' /
            'House and Garden' / 'Homes and Gardens' /
            'Ideal Home' / 'Beautiful Home' / 'Living' /
            'Good Housekeeping' / 'Family Circle' /
            'Traditional Homes'
womans     define 'Cosmopolitan' / 'Woman' / 'Woman''s Own' /
            'My Weekly' / 'Woman''s Journal' / 'Options' /
            'Woman''s Realm' / 'New Woman'
house      define 'House and Garden' / 'Homes and Gardens' /
            'Ideal Home' / 'Beautiful Home' / 'Living' /
            'Good Housekeeping' / 'Family Circle' /
            'Traditional Homes'

comment Key woman's magazines are Options, New Woman, Cosmopolitan
comment Key house magazines are Ideal Home, Beautiful Home,
comment Traditional Homes
dloop      for i = 1 to 3
wfirst     dummyask 'Key woman''s magazines in order of mention'
            resp sp womans in allmags
hfist      dummyask 'Key house magazines in order of mention'
            resp sp house in allmags

comment Ensure dummy variables have no value in case of snapbacks
        unset wfirst(i)
        unset hfist(i)
next
allkey    dummyask 'All key brands mentioned'
            resp mp allmags
wknow     ask 'Which woman''s magazines can you think of?'
            resp mp womans in allmags

comment Find order in which magazines were mentioned. If this
comment is a key brand, assign that response number to the
comment appropriate cell of wfirst.
        set wkeymag = 0
```

Quancept and mrInterview Scriptwriter's Manual

```
comment For CAPI, you may delete the line 'for wpos=1 to 8' and
comment uncomment the following 2 lines.
comment      substitute numwom 8
comment      wloop1 for wpos = 1 to numwom
wloop1  for wpos = 1 to 8
        set tmpmag = mention(wknow,wpos)
        if (tmpmag = 1) {
            set wkeymag = wkeymag+1
            set wfist(wkeymag) = 1
        }
        if (tmpmag = 6) {
            set wkeymag = wkeymag+1
            set wfist(wkeymag) = 6
        }
        if (tmpmag = 8) {
            set wkeymag = wkeymag+1
            set wfist(wkeymag) = 8
        }
    next
    route (wkeymag=null) go hknow
wmag      display 'The key magazines known were mentioned in the
               following order:@'
wloop2  for key = 1 to 3
        route (key>wkeymag) go nxt1
        display wfist(key)+', '
nxt1      next
        pause
hknow      ask 'Which home furnishing/management magazines can you
               think of?'
        resp mp house in allmags

comment Find order in which magazines were mentioned. If this
comment is a key brand, assign that response text to the
comment appropriate cell of hfist
        set hkeymag = 0

comment For CAPI, you may delete the line 'for hpos=1 to 8' and
comment uncomment the following 2 lines.
comment      substitute numhse 8
comment      hloop1 for hpos = 1 to numhse
hloop1  for hpos = 1 to 8
        set tmpmag = mention(hknow,hpos)
        if (tmpmag = 3) {
            set hkeymag = hkeymag + 1
            set hfist(hkeymag) = 'Ideal Home'
        }
        if (tmpmag = 4) {
            set hkeymag = hkeymag + 1
            set hfist(hkeymag) = 'Beautiful Home'
        }
```

```
        if (tmpmag = 8) {
            set hkeymag = hkeymag + 1
            set hfist(hkeymag) = 'Traditional Homes'
        }
next
route (hkeymag=null) go ctn
hmag    display 'The key magazines known were mentioned in the
following order:@'
hloop2   for key = 1 to 3
        route (key>hkeymag) go nxt2
        display hfist(key)+', '
nxt2    next
pause

comment Copy values of hfist and wfist into allkey. Unset
comment allkey first to ensure it is empty in case of snapbacks
ctn      unset allkey
all      for subs = 1 to 3
        set allkey = wfist(subs)
        set allkey = hfist(subs)
next
route (allkey=null) go endst
exit     display 'All key brands mentioned were: @'+allkey
pause
endst    end
```

Script B



```
comment Sample script B for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 12, Assignment (Web/mrInterview only)
```

comment mrInterview only: Unanswered questions to be treated the same as
comment questions at which a null response was chosen. Without this, the
comment logical expressions (variable=null) are always false.

```
set noaneqnl = 1
```

```
allmags define 'Cosmopolitan' / 'Woman' / 'Woman''s Own' /
    'My Weekly' / 'Woman''s Journal' / 'Options' /
    'Woman''s Realm' / 'New Woman' /
    'House and Garden' / 'Homes and Gardens' /
    'Ideal Home' / 'Beautiful Home' / 'Living' /
    'Good Housekeeping' / 'Family Circle' /
    'Traditional Homes'
womans  define 'Cosmopolitan' / 'Woman' / 'Woman''s Own' /
    'My Weekly' / 'Woman''s Journal' / 'Options' /
    'Woman''s Realm' / 'New Woman'
house   define 'House and Garden' / 'Homes and Gardens' /
    'Ideal Home' / 'Beautiful Home' / 'Living' /
    'Good Housekeeping' / 'Family Circle' /
    'Traditional Homes'
```

Quancept and mrlInterview Scriptwriter's Manual

```
comment Key woman's magazines are Options, New Woman, Cosmopolitan
comment Key house magazines are Ideal Home, Beautiful Home,
comment Traditional Homes
wfirst    dummyask 'Key woman''s magazines mentioned'
          resp mp womans in allmags
hfirst    dummyask 'Key house magazines mentioned'
          resp mp house in allmags

comment Ensure dummy variables have no value in case of snapbacks
unset wfist
unset hfist

allkey   dummyask 'All key brands mentioned'
          resp mp allmags
wknow    ask 'Which woman''s magazines can you think of?'
          resp mp womans in allmags

comment If a key brand was mentioned, assign that response
comment number to wfist
wloop1   for wpos = 1 to 8
          set tmpmag = bit(wknow/wpos)
          route (.not. tmpmag) go nxt1
          if (wpos = 1) {
              set wfist = 1
          }
          if (wpos = 6) {
              set wfist = 6
          }
          if (wpos = 8) {
              set wfist = 8
          }
nxt1     next
          route (wfist=null) go hknow
wmag     display 'The key magazines known were mentioned '+wfist

hknow    ask '@@Which home furnishing/management magazines can you
think of?'
          resp mp house in allmags

comment If a key brand was mentioned, assign that response
comment number to hfist
hloop2   for hpos = 1 to 8
          set tmpmag = bit(hknow/hpos)
          route (.not. tmpmag) go nxt2
          if (hpos = 3) {
              set hfist = 'Ideal Home'
          }
          if (hpos = 4) {
              set hfist = 'Beautiful Home'
          }
```

```
if (hpos = 8) {
    set hfirst = 'Traditional Homes'
}
nxt2    next
route (hfirst=null) go ctn
hmag    display 'The key magazines known were '+hfirst
comment Copy values of hfirst and wfirst into allkey. Unset
comment allkey first to ensure it is empty in case of snapbacks
ctn      unset allkey
        set allkey = wfirst
        set allkey = hfirst
        route (allkey=null) go endst
exit    display 'All key brands mentioned were: @'+allkey
        pause
endst   end
```


13 Logical expressions

Logical expressions test whether a variable has a given value. If the variable has the stated value, the expression is true; if not, the expression is false. Logical expressions are often used as part of routing statements. New keywords introduced in this chapter are:

.and.	test that two logical expressions are both true
.not.	negate logical expressions
.or.	test that at least one of a pair of logical expressions is true
.xor.	test that only one of a pair of logical expressions is true
logical	assign a logical (true/false) value to a variable

13.1 Syntax of an expression

Quick Reference

The syntax of any logical expression is:

variable operator value

A logical expression is the name given to a statement which may be true or false. For example, you may have a statement to the effect that p1=10. If this is correct and p1 is equal to 10, then this statement is true. If p1 is not equal to 10, the statement is false.

You use statements such as this when you want to check whether a particular answer has been given. For example, you might want to check whether the respondent bought peas. You can do this using a logical expression. If the question label is q1, you could then write a statement to check whether the response to q1 is peas. If the respondent bought peas, the statement would be true. Otherwise it would be false. The logical expression to perform this check would look like this:

(q1='peas')

Remember how you used this sort of statement with *route*. When you say:

```
route (q1='peas') go ctn1
```

you are saying, check whether the response to q1 is peas. If the statement is true (the answer is peas) then route to the label ctn1; however, if the statement is false (the answer is not peas) then do not route to ctn1.

13.2 Symbols used with logical expressions

A logical expression may include one or more of the following symbols:

>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
<>	less than or greater than; not equal to
=	equal to

Some of these symbols may only be used with certain types of responses, as shown in the table below. Note that none of them may be used to check open-ended (*resp coded*) or database (*resp dbase*) responses.

operator	num / real	sp	mp	ref / null / dk	list
>	✓	✗	✓	✗	✗
<	✓	✗	✓	✗	✗
<=	✓	✗	✗	✗	✗
>=	✓	✗	✗	✗	✗
<>	✓	✓	✗	✓	✓
=	✓	✓	✗	✓	✓

The response type of the names on either side of a symbol must be the same, for example, both integer, both texts. Illegal or mixed-type statements are always false. For example,

```
x > 10
```

is false if x is not numeric. It is, of course, also false if x is numeric but not greater than ten. Here are some further examples of statements using these symbols:

```
x >= 10
```

This expression is true if x is numeric and has a value of 10 or greater. It is false if x is less than 10 or if it is non-numeric.

```
tran <> 'train'
```

For this to be true tran must not be equal to 'train'. tran must previously have been defined as question with a *sp* response list, and one of the possible responses must have been 'train'.

```
cars <= 5
```

For this to be true cars must be less than or equal to 5 and cars must have been defined as *num* with values of 5 or less being valid responses.

Symbols with multipunched responses

Quick Reference

To check whether a response was chosen with any responses which appear before it in the response list, type:

```
mp_variable = '<resp_text'
```

To check whether a response was chosen with any responses which appear after it in the response list, type:

```
mp_variable = 'resp_text>'
```

To check whether a response was chosen with any other response in the list, type:

```
mp_variable = '<resp_text>'
```

The symbols =, <, and > work slightly differently with multipunched responses than with any other type of response. An expression containing the = symbol will be true if the named response is the only response given to the question. Thus,

```
color = 'yellow'
```

is true if the respondent chooses yellow and nothing else from the response list. If the respondent also chooses other responses, the expression will always be false.

The < and > operators, as used with multipunched responses, mean ‘before’ and ‘after’, and are not the same as the < and > operators that you may use with numeric responses. When you use one of these operators, you are writing an expression which will be true only if the respondent chooses the given response either by itself or with one or more of the responses which come before (after) it in the response list. For example, if the response list is:

```
color      ask 'Which colors do you like?'
            resp mp 'red' / 'blue' / 'yellow' / 'green' / 'black'
```

the expression:

```
color = '<yellow'
```

will be true only if the respondent chooses yellow by itself or with blue and/or red. It will be false if the respondent does not choose yellow, or the respondent chooses it with green or black. On the other hand:

```
color = 'yellow>'
```

will be true only if yellow is chosen by itself or with green or black. It will be false if yellow is chosen with red or blue.

From these examples, you will have noticed four things:

- That the < or > sign is enclosed in the single quotes with the response text.
- That < precedes the response text and > follows the response text.
- That there is no space between the < or > sign and the response text.
- That expressions with < are always false if any of the responses after the named response are chosen (for example, any responses which appear in the list after yellow), and that expressions with > are always false if any of the responses before the named responses are chosen.

The point about spacing is especially important. If you leave a space between the response text and the < or > symbol, the interviewing program will expect to find the quoted response text in the list preceded or followed by a space, exactly as you have written it in the expression. Compare the two examples:

```
color ask 'You had three packets of the test product each with a  
different color scheme. Which one or ones did you think suited  
the product?'  
      resp mp 'blue and pink' / 'red and yellow' / 'yellow and green'  
      route (color = '< yellow') go y1  
      route (color = 'yellow >') go y2
```

The first routing statement will be executed if the response containing _yellow (the _ represents the space character) is chosen with or without any which are defined before it — that is, ‘red and yellow’ by itself or with ‘blue and pink’. It will be false if ‘red and yellow’ is not chosen, or if it is chosen with ‘yellow and green’, or if all three responses are chosen.

The second routing statement will be executed if the response containing yellow_ (the _ represents the space character) is chosen, namely, ‘yellow and green’. Since this is the last response in the list, all responses in which this item is chosen with another color will be false.

If you wish, you may use < and > with the same response text to check whether a response containing that word has been chosen. If we write:

```
color = '<red>'
```

then in the previous example, the expression will be true if the response containing ‘red’ is chosen by itself or with any response on either side of it.

If you have long response texts you may abbreviate them. For example:

```
color ask 'Which colors do you like?'  
      resp mp 'blue and pink' / 'red and yellow' / 'yellow and green'  
      route (color = 'b>') go blue
```

Here we have used the letter b as an abbreviation for ‘blue and pink’. The expression will be true if that response is chosen by itself or with either or both of the other two responses. It will be false if ‘blue and pink’ is not chosen.

If you abbreviate responses, make sure that the abbreviations are not ambiguous; otherwise Quancept may make the wrong assumptions when checking responses. In our example, using the letter y as an abbreviation for yellow (or just the word yellow without any spaces) would be ambiguous because yellow appears in two different response texts.

13.3 Logical operators used with logical expressions

The expressions you have seen so far have tested the value of one variable in isolation. The interviewing program is also able to process more complex expressions which test the combined values of two or more variables. For example, you may test whether the respondent is a woman over the age of 65, which involves testing the respondent's gender and age in the same expression.

Logical expressions using *num*, *real* or *sp* responses allow the use of four logical operators for this type of test:

- .and.** Both/all.
- .or.** One or the other or both.
- .xor.** One or the other, but not both (CATI only)
- .not.** Negates an expression.

.and., *.or.* and *.xor.* are used to compare two or more logical expressions. The format of such statements is:

log.exp log.op log.exp

where the logical expression is checking the value of a numeric or single-punched response.

☞ Similar tests for multipunched variables are available but they use a different syntax. See section 14.2, Checking multipunched responses, for more information.

The .and., .or. and .xor. operators

Quick Reference

Use **.and.** to join two expressions such that both must be true for the expression as a whole to be true:

log_exp .and. log_exp

Use **.or.** to join two expressions such that at least one must be true for the expression as a whole to be true:

log_exp .or. log_exp

In Quancept CATI, use **.xor.** to join two expressions such that only one must be true for the expression as a whole to be true:

log_exp .xor. log_exp

The *.and.* operator signals that both sections of the expression must be true in order for the expression as a whole to be true. The example below checks whether the answer to q6 was ‘peach’ and also whether the answer to q7 was not ‘plum’.

`q6='peach' .and. q7<>'plum'`

If either or both of q6 and q7 have different answers, this test will be false.

An expression containing *.or.* is true if **one or the other or both** of the sub-expressions is true. Only if both expressions are false is the whole expression false. The expression:

`age>60 .or. wstat='Retired'`

will be true unless the respondent is 60 or younger and is not retired.

 Expressions with *.xor.* are true if **one or the other** of the subexpressions is true. If both or neither are true then the whole expression is false. The expression below is true if the answer to trans is train or the answer to q6 is train, but not if both questions are set to train, nor if neither question is set to train:

`trans='train' .xor. q6='train'`

The .not. operator

Quick Reference

To negate or reverse the meaning of an expression, type:

.not. *log_exp*

.not. always precedes the statement which it negates. For instance:

```
color    ask 'You had three packets of the test product each
with a different color scheme. Which one or ones did you think best suited
the product?'
      resp mp 'blue and pink' / 'red and yellow' /
             'yellow and green' / 'none were suitable' sp
      route (.not. color = 'none were suitable') go nxtcol
```

The *route* statement says that if the response to *color* is not ‘none were suitable’ then go to the label *nxtcol*.

If an expression consists of two or more sub-expressions and the *.not.* refers to the second sub-expression, the *.not.* and that sub-expression must be enclosed in parentheses:

```
tried='Brand A' .and. (.not. prefer='Brand A')
```

This expression will be true for all respondents who tried brand A and did not prefer it to the other brands.

Precedence with more than one operator

Quick Reference

When expressions contain a mixture of operators, the order of priority is:

.and.	.or.	.xor.	.not.
-------	------	-------	-------

You may use parentheses to impose a different order.

In the examples so far we have compared two expressions. The interviewing program allows you to compare any number of sub-expressions in a single expression. For instance:

```
tested='Product A' .or. tested='Product B' .or. tested='Product C'
```

will be true for all respondents who tried at least one of products A, B or C.

Where an expression contains a combination of operators, the expression will be evaluated according to a set order of precedence. This deals with *.and.* first, followed by *.or.*, then *.xor.*, and finally *.not.* If you wish to enforce a different order of evaluation, you may use parentheses to group the expressions which need to be evaluated together. Without any parentheses, the test:

```
unaid='Brand A' .or. aided='Brand B' .and. advert='Brand B'
```

follows the standard rules for precedence; that is the *.and* clause takes precedence over the *.or*. The expression as a whole will be true if the respondent:

- mentioned Brand A at the unaided awareness question; or
- was aware of Brand B when prompted and also remembered seeing the advertisement for it; or
- falls into both the previous categories

It is as if the second and third expressions were enclosed in parentheses:

```
unaid='Brand A' .or. (aided='Brand B' .and. advert='Brand B')
```

If you want the *.or.* to take precedence over the *.and.*, you must enclose the two expressions on either side of *.or.* in parentheses:

```
(unaid='Brand A' .or. aided='Brand B') .and. advert='Brand B'
```

Now the expression as a whole will be true if the respondent:

- mentioned Brand A unaided and also remembered seeing the advertisement for Brand B; or
- was aware of Brand B when prompted and also remembered seeing the advertisement for it; or
- was aware of both brands and remembered seeing the advertisement for Brand B

Notice here that respondents must always remember the advertisement if the expression is to be true.

.not. with other operators

The *.not.* operator is evaluated last in the sub-expression in which it occurs. If the *.not.* clause contains two or more sub-expressions, the *.not.* is applied after all other sub-expressions have been evaluated. For example:

```
.not. unaid='Brand A' .and. advert='Brand A'
```

In this example, the *.and.* is evaluated first. That expression will be true if the respondent mentioned Brand A unaided and remembered the advertisement for that brand. The *.not.* reverses this so that the expression as a whole is true if the respondent:

- mentioned brand A unaided but did not remember the advertisement, or
- did not mention brand A but did remember the advertisement, or
- did not mention brand A and did not remember the advertisement.

If you use `.not.` in the middle of an expression to refer just to the expression which follows it, you must enclose it and the expression to which it refers in parentheses, so that the interviewing program knows which expression to reverse. For example:

```
tested='Product B' .and. .not. advert='yes'
```

is illegal, but the parser and the interviewing program will both accept:

```
tested='Product A' .and. (.not. advert='yes')
```

The expression as a whole will be true if the respondent tested Product A and did not know the name of the product in the advertisement.

An alternative way to deal with this type of expression is to create a variable which has a true or false value based on the `.not.` part of the expression and then to substitute this variable in place of the `.not.` expression. This is described in the section below.

13.4 Assigning logical values to variables

Quick Reference

Use the **logical** keyword to assign a logical value to a variable:

```
set variable = logical (log.exp1 [log.oper log.exp2 ...] )
```

where *variable* is the variable which is to be assigned the value true or false, *log.exp1* and *log.exp2* are the logical expressions to be evaluated, and *log.oper* is one of the logical operators *.and.*, *.or.* or *.xor*.

The logical expressions we have looked at so far have all been relatively short and simple. In the sample scripts in earlier chapters, logical expressions have always been used with *route* or *if* and have tested the value of a variable defined immediately before that statement. If you are testing the values of a number of questions scattered through the script, the logical expression may become quite long and it will become increasingly difficult to be sure of exactly what you are testing. If the expression also requires parentheses to force a particular order of evaluation, things become even more confusing.

In cases such as these, you may find it simplifies things if you break the expression down into smaller expressions. You can then evaluate these expressions individually and set variables to true or false according to the values of those expressions. When you are ready, you can then write your final expression which compares the values of the logical variables you have created.

Now let us look at an example. In this script we have a section which is to be asked of men aged 18 to 24 and women aged 18 to 30. The expression to select these respondents with *route* or *if* would be:

```
(gender='male' .and. age>=18 .and. age <=24) .xor.  
(gender='female' .and. age>=18 .and. age <=30)
```

The parentheses to separate the expression for men from the expression for women are optional in this example because the operators we have used mean this happens automatically. The sub-expressions for men and women are each made up of *.and.* operators, so these are evaluated first. This yields two expressions joined by *.xor.*, which is what we want. Nevertheless, we recommend that you use parentheses whenever you have expressions with mixed logical operators, to ensure that the expression is evaluated in the way you want. Without parentheses, the validity of the expression is dependent on you remembering correctly the order of precedence among logical operators. Also, it makes the logic of the expression immediately clear to anyone else needing to work on the script.

We can simplify this expression, even though it involves us in more typing, by creating two logical variables, one for the men we want to interview and one for the women, and then comparing the values of those variables:

```
comment Example to show use of logical variables  
gender ask 'Enter respondent''s gender'  
        resp sp 'Male' / 'Female'  
age     ask 'How old are you?'  
        resp num 16 to 65  
comment men is true if respondent is male & in required age group  
comment women is true if respondent is female & in required age group  
        set men = logical(gender='Male' .and. age>=18 .and. age <=24)  
        set women=logical(gender='Female'.and.age>=18.and.age<=30)  
comment Now test for the men and women we want  
        route (men .xor. women) go ctn  
        goto exit  
ctn    continue
```

-
- ❖ With questions where selecting one response automatically prevents any other response from being selected, you may sometimes be able to use *.or.* instead of *.xor.* and still obtain valid results. The example here is such a case, because a person cannot have two genders or two ages.
-

13.5 Sample script

```
comment Sample script for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 13, Logical expressions

tested    ask 'Which test product did you try?'
          resp sp 'Product A' / 'Product B' / 'Product C'
color      ask 'You had three packets of '+tested+' each with a
           different color scheme. Which one or ones did you think suited the
           product?'
          resp mp 'blue and pink' / 'red and yellow' /
                  'yellow and green' / 'none were suitable ^s'

comment Respondents mentioning red and yellow packaging with any others
route (color <> '<red and yellow>') go nxtcol
red       ask 'Why do you think red and yellow is effective?'
          resp coded

comment Respondent mentioning blue and pink packaging with any others
nxtcol    if (color = 'blue') {
blue       ask 'Why do you think blue and pink is effective?'
          resp coded
}

comment Respondents mentioning yellow and green packaging with any
comment others
if (color = '<green>') {
green     ask 'Why do you think yellow and green is effective?'
          resp coded
}

advert    ask 'The test product you tried is one of a range of new
           cooking sauces with a West Indian flavor. Do you remember seeing
           any advertisements for this type of product during the last week?'
           resp sp 'yes' / 'no' go ctrl1
product   ask 'What product was it advertising?'
           resp sp 'Caribbean Sizzler' / 'Don''t remember the name'
buy       ask 'Has the advertisement tempted you to try that cook-in
           sauce?'
           resp sp 'yes' / 'no'

comment People who know the name of the advertised product and
comment who are tempted to buy the product and who did not have
comment it as their test product
if (product = 'Caribbean Sizzler' .and. buy = 'yes' .and.
    tested <> 'Product B') {
free      display 'In that case, please allow me to offer you a
           sample packet absolutely free of charge'
           pause
addr1     ask 'Please let me have your name and home address'
           resp coded(0)
```

Quancept and mrInterview Scriptwriter's Manual

```
        goto ctn2
    }
ctl1      continue

comment People who did not try Product B (i.e., tried A or C)
        if (tested = 'Product A' .or. tested = 'Product C') {
trycs      ask 'Would you like to try some of the new Caribbean
Sizzler cook-in sauce free of charge?'
        resp sp 'yes' / 'no' go ctn2
addr2      ask 'Please let me have your name and home address'
        resp coded(0)
    }
ctn2      continue

gender     ask 'Are you ...'
        resp sp 'Male' / 'Female'
age        ask 'How old are you?'
        resp num 16 to 65

comment men is true if respondent is male & in required age group.
comment women is true if respondent is female & in required age group.
        set men = logical(gender='Male' .and. age>=18 .and. age <=24)
        set women = logical(gender='Female' .and. age>=18 .and. age <=30)

comment Now test for the men and women we want
comment CAPI/Web/mrInterview: replace .xor. with .or.
        if (men .xor. women) {
mags      ask 'Which magazines do you read on a regular basis?'
        resp mp 'New Woman' / 'Cosmopolitan' / 'GQ' /
                'The ES Magazine' / 'Esquire' / 'Options'
                other null
    }
exit      display 'Thank you for your help'
pause
end
```

14 Checking non-numeric responses

The Quancept language has a number of facilities for finding out about response lists and the responses chosen from a list. You will find many of these keywords useful for extracting information that you can use in logical expressions. The examples you have seen so far in earlier chapters have usually included the response text as part of the logical expression. Often, you will want to write your script in a more open fashion and you will want a general method of referring to responses in expressions. This is particularly true when you are using loops where you also need to take the question's subscript into account, or where you are using a rotated or randomized response list.

The keywords covered in this chapter are:

and	check for all named responses from a multipunched list
bit	check whether a certain response is given
editopen	edit open ends and specified other responses
elm	find the text of a selected response number
mention	find the order of mention for multipunched responses
nbit	find the response number chosen from a single-punched list
numb	count the number of responses given
numv	count the number of responses in the list
or	check for one or more named responses from a multipunched list
othertext	extract individual texts from a list of open ends entered for <i>promptoth</i> or ^o responses
strngchk	check text responses in open ends and specified other
xor	check for only one named response from a multipunched list

14.1 Checking single-punched responses with nbit

Quick Reference

Use **nbit** to check whether a particular response was chosen from a single-punched response list.

```
set variable= nbit(label)
```

Each response in a response list is identified by its position in that list. During an interview, the first response in any list is always identified by a code 1, the second by a code 2, and so on. The interviewing program allows you to use the response position as a means of checking whether a particular response has been given to a question.

For example, if April is the fourth response in the response list, rather than checking whether the response was April, you can check whether the fourth response was given:

```
set monname = nbit(month)
```

If the response to 'month' is April, monname will have the numeric value 4. Similarly, if the response is September, which is ninth in the list, monname will have the value 9.

The responses *null*, *dk* and *ref* always return a value of 0 regardless of the fact that they come at the end of the response list. Specified other is treated as an ordinary response and returns a value corresponding to its position in the list.

The next example shows how to use *nbit*.

```
whodata ask 'Thinking of the data that you tabulate with  
Tables/Qmoretabs, would you tell me whether it is data that your  
company collects itself, or is it data that you receive from other  
companies?'  
        resp sp 'Our data' / 'Other companies' data' /  
                'Both our and other companies' data'  
comment Users who tabulate their own data  
        set owndata = nbit(whodata)  
        if (owndata<>2) {  
datent      ask 'How do you enter the data onto the computer?'  
        resp mp 'In-house data entry program' /  
                'Data entry program from elsewhere ^o' other  
    }
```

This example is a simplified version of part of the sample script. It tests whether the company tabulates its own data or runs solely as a service bureau tabulating data for other companies.

The whodata question has three possible responses, which the interviewing program sees as responses 1, 2 and 3. The *set* statement checks which answer was given and assigns its response number to the temporary variable owndata. The values we are interested in are 1 and 3 for companies tabulating their own data, so we write a logical expression which tests the value of the owndata variable. If the value is not 2, we ask how data is entered onto the computer.

Another way of writing this test is:

```
if (owndata<>'Other companies' data') {
```

You may choose the method you prefer, since there is no difference as far as the interviewing program is concerned.

There is a difference, however, if you want to test whether the respondent chose specified other. You cannot write:

```
route (qname = 'other')
```

because this assumes that other is an ordinary quoted response. Instead, you must use *nbit* and test for specified other according to its position in the list:

```

color    ask 'Which color do you like best?'
        resp sp 'red' / 'blue' / 'green' / 'yellow' other
        set pos = nbit(color)
        if (pos = 5) {
            set othcol = othertext(color,0)
why      ask 'Why did you choose '+othcolor+'?'
            resp coded(9)
}

```

14.2 Checking multipunched responses

When we discussed routing in response lists with *go*, we said that you should not use this type of routing with multipunched response lists. This is because the route taken depends on the order in which the interviewer enters the response codes.

To check multipunched responses use one of the following functions:

and	all named responses
or	at least one named response
xor	only one named response
mention	order in which responses were mentioned

-
- ☞ If you're interested in the response list as a whole rather than a subset of it, use the *bit* and *elm* keywords described in section 14.3, Testing for chosen responses by their position in the list, and section 14.6, Find response text by position of response in list.
-

At least one named response chosen

Quick Reference

Use **or** to check whether at least one of a given set of responses was chosen from a multipunched response list:

```
set variable = or(label / resp1 / resp2 / ... respn)
```

where *label* is the label whose value is to be tested, and *resp1* to *respn* are the responses against which that value is to be compared. All these responses must be present in the response list defined for *label*.

If at least one of the listed responses is chosen, the test is true, otherwise, it is false.

The assignment statement will normally be followed by a routing statement based on the value of the variable. For example:

```
format ask 'How do you receive the data? Is it in the form of  
printed questionnaires, or on a tape or floppy disk, or is it in  
some other form (e.g., via a modem connection)?'  
        resp mp 'printed questionnaires' /  
                'other printed forms' / 'on tape/disk' other  
        set printq = or(format / 'printed questionnaires'/  
                'other printed forms')  
        route (printq) go datent
```

The format question allows a multipunched response. If one or more of the responses ‘printed questionnaires’ or ‘other printed forms’ is given, the temporary variable printq will be set to true and the *route* statement will be obeyed. If neither of these responses is given, printq is false and the routing is ignored.

The respondent must select at least one of the listed items for printq to be true. It does not matter what other responses are given because they have no bearing on the *or* statement.

All named responses chosen

Quick Reference

Use **and** to check whether all responses in a given set were chosen from a multipunched list:

```
set variable = and(label / resp1 / resp2 / ... .respn)
```

All responses listed with *and* must also be present in the response list defined with *label*. If all the responses listed with *and* have been given, the statement is true; otherwise, it is false. For example:

```
progs define 'Tables' / 'Call and See' / 'Desktop Tables' /  
            'Stats for MR' / 'Survey Design' / 'Basic Stats' /  
            'Qmoretabs'  
have ask 'Which ABC Inc products are used in your company?'  
       resp mp progs  
       set stats = and(have / 'Stats for MR' / 'Basic Stats')  
       route (.not. stats) go ctnl
```

checks the responses given to the question ‘have’. If both Stats for MR and Basic Stats are given, the variable ‘stats’ is true, but if only one or neither of the listed responses has been given, it is false and the routing will be obeyed.

Any of the other responses in the response list may be given because these do not affect whether ‘stats’ is true or false. The table below shows which types of answers will be accepted and which will not.

Example responses	stats true	stats false
Stats for MR/Basic Stats	✓	✗
Stats for MR/Tables/Basic Stats	✓	✗
Survey Design/Basic Stats	✗	✓

Only one named response chosen



Quick Reference



Use **xor** to check whether only one of a set of responses was chosen from a multipunched list:

```
set variable = xor(label / resp1 / resp2 / ... respn)
```

All the responses listed with *xor* must also appear in the response list defined with *label*. If only one of the responses listed with *xor* has been mentioned, the statement is true. If more than one, or none, of the listed responses has been given, the statement is false.

```
progs    define  'Tables' / 'Call and See' / 'Desktop Tables' /
           'Stats for MR' / 'Survey Design' / 'Basic Stats' /
           'Qmoretabs'
have     ask 'Which ABC Inc products are used in your company?'
         resp mp progs
comment Users who have either Tables or Qmoretabs but not both
          set tables = xor(have / 'Tables' / 'Qmoretabs')
          route (.not. tables) go ctn2
```

In this example, we are interested in companies which have either Tables or Qmoretabs but not both. These are the users for whom the temporary variable 'tables' will be true. Users at companies who have both products or neither product are not eligible to answer questions in this section, so we route them to the start of the next section.

As shown in the table below, the respondent may mention as many of the remaining products as he/she likes, since they have no bearing on *xor*.

Example responses	tables true	tables false
Tables/Qmoretabs	✗	✓
Call and See/Survey Design	✗	✓
Tables/Call and See	✓	✗
Tables/Survey Design/Qmoretabs	✗	✓

Order of mention for multipunched responses

Quick Reference

Use **mention** to store the order in which responses are selected from a multipunched list:

```
set temp_var = mention(qname, resp_num)
```

temp_var will contain the number of the response mentioned as choice number *resp_num*.

To store the response text, use a dummy question variable instead of the temporary variable:

```
set dum_qvar = mention(qname, resp_num)
```

It is possible to store the order in which responses were selected from a multipunched response list for use later in the script or for writing into the data file. For example, to find the first advertisement location that the respondent mentions, you might write:

```
comment A new product – all users answer this section
newprod ask 'The company has just launched a new product, ABC Games.
          This is a collection of games designed to distract even the most
          dedicated employee from his/her work.@@
          Have you seen or heard any advertising for this product recently?
          resp sp 'Yes' / 'No' go ctn3
where ask 'Where was that?
          resp mp 'Radio' / 'TV' / 'Newspaper' / 'Sales handout' /
          'MRS newsletter'
comment Get first advertisement mentioned
          set first = mention(where,1)
d1      display 'The first advertisement mentioned was '+first
          pause
ctn3    continue
```

If the respondent mentions the newspaper advertisement first, the value of *first* will be 3 because *Newspaper* is the third item in the list. If the respondent does not answer the question, *first* will be null.

When you want to store more than just the first mention, there are two ways of using *mention*. The first is to write as many *mention* statements as there are responses you want to store. For example:

```
where ask 'Where did you see/hear the advertisements?
          resp mp 'Radio' / 'TV' / 'Newspaper' / 'Sales handout' /
          'MRS newsletter'
          set first = mention(where,1)
          set second = mention(where,2)
d1      display 'First mentioned: '+first+
          @Second mentioned: '+second
```

The second method is to place *mention* in a loop which is repeated as many times as there are responses to be stored:

```

where    ask 'Where did you see/hear the advertisements?'
        resp mp 'Radio' / 'TV' / 'Newspaper' / 'Sales handout' /
              'MRS newsletter'
d1      display 'Order of mention is:@'
l1      for pos = 1 to 5
        set numchos = numb(where)
        route (pos > numchos) go nxt
        set chosen(pos) = mention(where,pos)
d2      display chosen(pos)
nxt     next

```

If the respondent chooses only one response, the value of the second mention will be null in both examples.

So far, we have stored the order of mention in a temporary variable with no predefined type. This causes the variable to store the positions of the responses in the response list rather than the response texts. In the examples above, the *display* statements will say that response 3 was mentioned first if the respondent mentions the newspaper advertisement first.

The same rule applies if you save the order of mention in a numeric variable — that is, a question with a numeric response list, or a temporary variable which has previously stored a numeric value.

If you want to pick up the text of the response, you will need to save the order of mention in a single-punched or multipunched variable; that is, as the response to a question with a single-punched or multipunched response list, as shown below:

```

adloc   define 'Radio' / 'TV' / 'Newspaper' / 'Sales handout' /
          'MRS newsletter'
comment Dummy question to store first three responses mentioned
comment in order of mention
l1      for i = 1 to 3
qment   dummyask 'Responses in order mentioned'
        resp sp adloc
        next
where   ask 'Where did you see/hear the advertisements?'
        resp mp adloc
comment Set first three responses into dummy question
l2      for i = 1 to 3
        set qment(i) = mention(where,i)
        next

```

The first loop defines a dummy question which we will use to store the first three responses chosen in the order in which they were mentioned. This is followed by the question whose responses we want to check. There is no restriction on the number of responses that may be chosen, and all those chosen will be stored in the data for this question.

The second loop is concerned with reading the responses given to ‘where’ and assigning the first, second and third mentions to the dummy question qment. qment(1) will store the first response mentioned, qment(2) will store the second mention and qment(3) will store the third mention. Any other advertisements which were mentioned will be ignored (they are still available in ‘where’). Since qment is a question, the values assigned to it will be treated as the answers to the question and will be placed in the data accordingly. Therefore, if the respondent has seen or heard all five advertisements, the data for ‘where’ will consist of a multipunch of five codes, and the data for qment(1), qment(2) and qment(3) will be a set of three single codes representing the first three advertisements chosen.

The notes below summarize the important points to remember when using *mention*:

- If you use *mention* with anything other than a multipunched question, the message ‘MENTION must only be used with MP questions’ will be produced when you parse the script.
- The response value used with *mention* must be in the range 1 to *n*, where *n* is the number of responses in the response list. If it is not numeric, or if it is less than 1 or greater than the number of responses in the list, the interviewing program will terminate with the message ‘MENTION argument incorrect type or value’.

-
- ❖ In Quancept Web and mrInterview, *mention* can be used only with dummy questions whose responses are set randomly within the script. Because of this, the examples shown earlier will not work in these variants. If your script contains *mention*, the parser issues a warning to this effect.

If it is important to know the order in which responses were selected, you will have to repeat the question and gather one response at a time. You could then merge those responses into a dummy question which you could then use in other ways later in the script.

14.3 Testing for chosen responses by their position in the list

Quick Reference

To check whether a particular response number was chosen from a single-punched or a multipunched list, use **bit**:

```
set variable = bit(qname / response_number)
```

If the response was chosen, *bit* returns a true value; otherwise, it returns false. For example:

```
where    ask 'Where did you see/hear the advertising?'
        resp mp 'Radio' / 'TV' / 'Newspaper' / 'Sales handout' /
              'MRS newsletter'
        set radio = bit(where/1)
        route(.not. radio) go exit
```

sets the temporary variable ‘radio’ to true if the respondent heard a radio advertisement. If radio is false, the script routes to the exit label.

If the question is not asked, *bit* returns the value false.

Specified other in a response list is treated as an ordinary response, so if it is the sixth response in the list, you would write *bit* (*qn/6*) to check whether it was selected. This statement is not valid to check for *null*, *dk* or *ref*. Instead, use statements of the form:

```
route (qname = null)
```

Note that *null*, *dk* and *ref* are not enclosed in single quotes because they are keywords rather than quoted response texts from a list.

Differences between *nbit* and *bit*

Do not confuse *bit* and *nbit*. Although they perform similar functions, they are not the same:

- *nbit* works with single-punched responses only; *bit* works with single-punched and multipunched responses.
- *nbit* looks at the response list as a whole; *bit* looks at a given response in the list.
- When the respondent answers the question, *nbit* returns the position, as a **numeric value**, of that response in the list; *bit* returns a value of true if the respondent chose the specified response, with or without any other responses in the list. If the respondent did not choose the named response at all, then *bit* returns a value of false.

Here is an example for clarification:

```
color    ask 'Which color do you like best?'
        resp sp 'red' / 'blue' / 'green' / 'yellow'
comment Find position of preferred color in the list
        set pos = nbit(color)
d1      display 'You chose response number '+pos+'. This is '+color
comment Check whether green was chosen
        set green = bit(color/3)
        if (green) {
            display 'You chose green'
        }
        else {
            display 'You did not choose green'
        }
```

14.4 Number of responses in a list

Quick Reference

To find the number of responses in a single-punched or multipunched list, use **numv**:

```
set variable = numv(question_name)
```

The question name must be that of a question allowing single-punched or multipunched responses. In the following example, the value of the variable numprog will be 7:

```
progs    define  'Tables' / 'Call and See' / 'Desktop Tables' /
           'Stats for MR' / 'Survey Design' / 'Basic Stats' /
           'Qmoretabs'
have     ask 'Which ABC Inc products are used in your company?'
resp mp progs
set numprog = numv(have)
```

If the response list includes specified other, this is included in the response count, whereas *null*, *dk* or *ref* are ignored.

The question need never be asked for *numv* to be valid; all the interviewing program does is look at the response list as it appears in the script file and count the number of responses in the list. It is not concerned with the number of responses actually given.

14.5 Number of responses chosen from a list

Quick Reference

To find the number of responses chosen from a single-punched or multipunched list, use **numb**:

```
set variable = numb(question_name)
```

If the question is not asked, or if one of the responses *null*, *dk* or *ref* is used, *numb* will return a 0 value. If the respondent selects specified other, this counts as one response from the list regardless of the number of responses the respondent then gives under specified other. (The interviewing program has no way of differentiating between individual responses entered with specified other.)

In the example below, we want to test whether a company uses all the products in a list and, if so, to check why this is. We count the number of responses in the list using the keyword *numv* as described above. We then follow this with a *numb* statement to count the number of answers chosen. If the two variables have the same value, we know that all responses were chosen.

```

progs    define  'Tables' / 'Call and See' / 'Desktop Tables' /
          'Stats for MR' / 'Survey Design' / 'Basic Stats' /
          'Qmoretabs'
have     ask 'Which ABC Inc products are used in your company?'
resp mp progs
set numprog = numv(have)
set numhave = numb(have)
if (numhave = numprog) {
decide   ask 'What were the reasons which made your company
choose so many ABC Inc products?'
resp mp 'Company policy' / 'Best products for the jobs' /
      'Nothing else caters for our type of work' /
      'No special reasons ^s' other
}

```

14.6 Find response text by position of response in list

Quick Reference

To pick up the text of a response in a single-punched or multipunched list according to its position in the list, use **elm**:

set variable = elm(*qname / response_number*)

The following example simply tells you that the third advertisement is a newspaper advertisement.

```

where   ask 'Where did you see/hear the advertising?'
resp mp 'Radio' / 'TV' / 'Newspaper' / 'Sales handout' /
      'MRS newsletter'
set advert = elm(where/3)
d1      display 'The type of advertisement was '+advert

```

If you give the number of a non-existent response (perhaps you ask for the fifth response when the list contains only three responses), the parser will issue an error message.

You cannot use *elm* to pick up the values of the *null*, *dk* and *ref* responses. It is, however, valid with specified other, which it treats as an ordinary response in the list. If you want the answers that specified other represents, you'll need to extract these by hand, using negative subscription or *othertext* in the usual way.

-
- ☞ For more information on extracting specified other responses, see section 14.7, ‘Displaying specified other answers’.
-

elm picks up a maximum of 999 characters of text from a response list. As with *numv*, the question need not be asked for *elm* to work. The interviewing program looks at the response list in the script and reads the given response from this list. This has the added advantage that it is not affected by any rotation or randomization of responses which takes place when the interview is in progress.

Using bit and *elm* together

When used together, *bit* and *elm* are a very powerful and efficient means of checking which responses were chosen from a list. Their most common use is with unaided and aided brand-awareness questions. For example:

```
comment Unaided awareness
unaided    ask 'Which brands can you name?'
            resp mp 'Brand A' / 'Brand B' / 'Brand C' / 'Brand D' /
                  'Brand E' / 'Brand F' / 'Brand G' / 'Brand H' /
                  'Brand I' other null dk
comment Aided awareness
11        for brdnum = 1 to 9
            set given = bit(unaided/brdnum)
            route (given) go getnxt
            set brd = elm(unaided/brdnum)
aided      ask 'Have you ever heard of '+brd+'?'
            resp sp 'Yes' / 'No' null dk
getnxt     next
```

The first question is the unaided awareness question — the interviewer asks the question but does not read out the possible responses. The *for/next* loop forms the aided awareness section. The *for* statement has values 1 to 9, where each value represents one of the brands listed in the response list to unaided (other, null and dk, which are also present in unaided response list, are excluded).

The next statement checks whether response *brdnum* was given to *unaided*. On the first pass through the loop, *brdnum* will be 1, so we will be checking whether the first response in the list was given — that is, did the respondent mention Brand A.

The *route* statement skips to the end of the loop if that brand was mentioned, and the interviewing program returns to the *for* statement to find the next response to check.

If the brand was not mentioned (*given* is false), we set *brd* equal to its name so that it can be substituted as part of the next question to be asked. When *aided* has been answered, the *next* statement is read and the process is repeated until each brand has been checked.

☞ You'll find a variation of this script written using defined response lists in section 7.10, Using question names with lists.

14.7 Displaying specified other answers

There are three ways of picking up answers given with specified other:

- Typing the name of the question to which the response belongs, as you do for ordinary responses.
- Using an *othertext* statement.
- Using negative subscription.

Of these, *othertext* is the most useful and the easiest to use. Typing the question name as you would for an ordinary response does work, but the interviewing program displays the response text preceded by the response serial number. Interviewers may find this confusing so you may prefer not to do this. Negative subscription is retained for backwards compatibility with older CATI scripts only.

For example:

```
mainuse ask 'How is cheese mainly used in your household?'
        resp sp 'in sandwiches' / 'with crackers' / 'with wine' /
                'for cooking' other
d1      display 'Main use chosen from the list was '+use
```

might display:

```
Main use chosen from the list was SER      15/in mousetraps
```

if specified other is chosen.

Using **othertext**

Quick Reference

To extract individual texts from a list of open ends entered for *promptoth* or *^o* responses, type:

```
set variable = othertext(qname, resp_pos)
```

where *qname* is the name of the question whose answers you want, and *resp_pos* is the position of the precoded response whose open end you wish to extract.

Use the special response position 0 for open ends entered with specified other, and -1 for the complete set of open ends belonging to the question.

Another way of picking up responses to specified other is with **othertext**. This statement is designed to deal particularly with open ends entered for responses flagged with *promptoth* (or *^o* in older scripts), but you can also use it for the standard specified other too, as shown below:

```
advert ask 'Where did you see or hear the advertising?'
      resp mp 'Radio' / 'TV' / 'Newspaper' / 'Sales handout' /
            other
      set goth = othertext(advert, 5)
d1    display 'Other advertising seen: '+goth
```

When a response list contains responses flagged with *promptoth* or *^o*, the interviewing program stores any open ends for that question as a single entry in the text file. Therefore, if the question is:

```
know ask 'Which washing powders can you think of?@
INTERVIEWER: For each unlisted powder chosen, check whether
it is automatic, biological, for woolens, or general purpose@
      resp mp 'Suds' / 'Washo' / 'Other general purpose' promptoth /
            'Suds Automatic' / 'Washo Automatic' /
            'Other Automatic' promptoth / 'Suds Biological' /
            'Washo Biological' / 'Other Biological' promptoth
            other
```

The interviewing program might store the open ends as:

```
000012000003KNOW      0011343601<Other Automatic>:Bubbles
<Other Biological>:Gleam
```

We have printed this entry on two lines, but it is stored as a single line in the text file.

With *othertext* you have the choice of extracting either an individual open end from the list or the response given to specified other itself, or of taking the list as a whole. To do this you tell the interviewing program the name of the question and the position of the response whose open end you want. In the previous question, you would ask for the open end for response number 6 if you wanted the names of other automatic brands mentioned:

```
set othauto = othertext(know,6)
```

If that response has an open end, the interviewing program will copy it into the named variable.

The values 0 and -1 have special meanings. 0 requests the open end associated with *other* itself and -1 returns the complete set of open ends for the whole question.

Here is an extension of the previous example which illustrates this:

```
know ask 'Which washing powders can you think of?@
INTERVIEWER: For each unlisted powder chosen, check whether
it is automatic, biological, for woolens, or general purpose.'
      resp mp 'Suds' / 'Washo' / 'Other general purpose ^o' /
            'Suds Automatic' / 'Washo Automatic' /
            'Other Automatic' promptoth / 'Suds Biological' /
            'Washo Biological' / 'Other Biological' promptoth
            other
comment Get open end for each promptoth response
comment Check whether response in position pos was chosen. If so,
comment get the precoded response text and the promptoth open end
```

```

11      for pos = 3 / 6 / 9
          set chosen = bit(know/pos)
          route (.not. chosen) go nxtone
          set btext = elm(know/pos)
          set brand = othertext(know, pos)
          display btext+' is '+brand
nxtone   next
comment Now get Specified Other and all Others
          set specoth = othertext(know,0)
          set alloth = othertext(know,-1)
d1       display 'Specified other is '+specoth+'@'
All others is '+alloth

```

Negative subscription



Subscription is a technical term that we use to mean typing a number after a question name. It has some important uses in Quancept, particularly when you are using loops for repetitive questions. You can also use it to pick up specified other responses.

When you want the responses that were given to a specified other, copy the value of the question into a temporary variable, and follow the temporary variable with **-1** in parentheses. To demonstrate this, we'll look at an example script:

```

mainuse  ask 'How is cheese mainly used in your household?'
          resp sp 'in sandwiches' / 'with crackers' / 'with wine' /
                  'for cooking' other
comment Check which answer was chosen
          set pos = nbit(mainuse)
comment Save the use chosen with the appropriate notation
          if (pos <= 4) {
              set dispuse = mainuse
          }
          else {
              set dispuse = mainuse(-1)
          }
d1       display 'Main use chosen from the list was '+dispuse

```

Here, we want to display the answer given to the opening question, mainuse, in the closing *display* statement. Because mainuse includes a specified other response, we need to use negative subscription to display this response, if it is selected.

Having asked the opening question, we check the position of the chosen answer in the response list. Specified other is answer number five. Next, we assign the answer chosen to the variable we will use in the *display* statement. If one of the predefined uses was chosen, the position number will be between one and four. We can therefore copy the value of mainuse into the variable dispuse straight away.

If specified other was chosen, the value of mainuse will be other and the response position will be five. We therefore need to get the answer given to the automatic ‘please specify’ prompt. We do this using the -1 notation, mainuse(-1). Now, the variable dispuse contains the response given in all cases; that is, whether a predefined response was selected or other. Therefore, our *display* statement works the same way whether or not specified other was chosen.

If the question whose value you want to display is part of a loop, the -1 value will alter according to the number of times the question was asked. Thus, if you want to retrieve the answer given the second time the question was asked, you will type -2 instead of -1 inside the parentheses; to pick up the answer the fourth time the question was asked, you will type -4. You will also need to give similar, positive numbers for answers that are not specified other.

-
- ☞ To find out more about displaying the answers to questions in loops, see section 11.2, Referring to answers given in loops.
-

14.8 Null, dk and ref

To test whether one of the special responses *null*, *dk* or *ref* was given to any question, enter a statement such as the one shown at the end of the following example:

```
pcode      ask 'Now your postcode.@INTERVIEWER: Only allow NULL as
a last resort.'
          resp coded null
          route (pcode = null) go exit
```

You can use this notation to test for *dk* and *ref* answers, too.

If you write a test of this type, the parser will issue an error message if the keyword(s) *null*, *dk* or *ref* are not present in the response list.

If an *nbit* statement encounters a null, don’t know or refusal response during an interview, it returns a value of 0 but does not differentiate between the three responses.

14.9 Checking text responses

Quick Reference

To check that a text has been entered in the correct format, use **strngchk**:

```
callfunc('strngchk', pattern, txtvar, match)
```

where *pattern* is the name of the variable defining the pattern against which the open end is to be checked, *txtvar* is the name of the open end or text variable whose value is to be checked, and *match* is the name of a variable to be set to true if the string matches the pattern, or false if it does not.

Sometimes you will want to check that the interviewer has entered an open end in the correct format; for example, you may check that an Outer London telephone number is entered in the format 0208–nnn–nnnn.

For each open end that you want to check, you need to define a pattern with which the text entered by the interviewer can be compared. Each character in a pattern matches itself. Characters that will vary between respondents (for example, the digits in the phone number) can be replaced in the pattern with ‘wild’ characters. These are:

- * any single character (letters, numbers, spaces, punctuation)
- # any single digit
- % any single letter a–z or A–Z

Thus, an Outer London telephone number would have the pattern 0208–###–#### because all Outer London numbers start with 0208 and then consist of a 3-digit code followed by a 4-digit code. In this instance, we have indicated that the interviewer must separate each group of codes with a hyphen. If you wanted only a space between the area code and the phone number, you would enter the pattern as 0208 ###–####.

The example below checks that a British National Insurance number has been entered correctly (a sample National Insurance number is AB 12 34 56 Z).

```
ninum    ask 'What is your National Insurance number?'
        resp coded nodata
        set nipatt = '%% ## ## %%'
        callfunc ('strngchk',nipatt,ninum,isok)
        route (isok) go ctn
wrong    display 'The National Insurance number that you entered@@'
        +ninum+'@@@ does not match the required format. Please re-enter it.'
        goto ninum
ctn     continue
```

If the respondent's National Insurance number is AB 12 34 56 Z, it must be entered as a list of letters, numbers and spaces, as shown here. If interviewers omit the spaces or type hyphens instead, they will be asked to re-enter the number in the required format.

If the string you want to test has several variations, write one *strngchk* statement for each variation. Here is a part of our sample script which tests British and Canadian postal codes (zip codes):

```
PCODE      ask 'Now your postcode.'
            resp coded (0)
COMMENT Test for postcodes of the form NW6 2EG and WC1A 7XX
        set pcfmt1 = '%%# %%'
        callfunc('strngchk', pcfmt1, PCODE, isok1)
        route (isok1) go exit
        set pcfmt2 = '%%## %%'
        callfunc('strngchk', pcfmt2, PCODE, isok2)
        route (isok2) go exit
WRONG      display 'Invalid postcode format'
            pause
            goto PCODE
EXIT       end
```

-
- ☞ The interviewing program can check texts with *strngchk* only when they are entered or changed during the main part of the interview. Open ends which are entered or changed at the 'Append verbatims' prompt at the end of the interview cannot be checked, even if the script contains a *strngchk* statement for that question.
 - ☞ A similar note applies when you review a completed interview. See section 34.3, Reviewing completed interviews, for further details.
-

14.10 Editing open ends and specified other responses



Quick Reference

To allow the interviewer to edit an open-ended or specified other response during the interview, type:

editopen *variable_name*

At the end of each interview, the interviewer is offered the choice of appending to or changing open-ended responses. This allows interviewers who are slow typists to record open ends on paper during the interview and then to type them in when the call is finished.

Sometimes, though, you will want to force interviewers to edit or enter open ends while the interview is in progress. For example, if the interviewer enters a post code or zip code in the wrong format, you may insist that the interviewer corrects it before continuing.

If the question is not open-ended or specified other was not chosen, the *editopen* statement is ignored. When there is a text to edit, the interviewing program copies it into a temporary file for editing. The screen clears and the message ‘Editing verbatim for question *name*’ is displayed with a prompt to press any key to continue. When you press a key, the screen clears again and the temporary file is opened for editing.

-
- ❖ Interviewers must have the environment variable QCEDITOR set to the editor program they want to use. If this variable is not defined, the editor message will be displayed but when the interviewer presses a key to continue, the interviewing program will display the next question rather than the open-ended text.
-

The interviewing program allows each open end to be edited only once on each pass through the script. After an open end has been edited, the *editopen* for that variable is ignored if it is reached again in the current pass. If the interviewer snaps back to before the *editopen* statement, the interviewing program will execute it once more when it is reached during the walk forward to the place from which the interviewer snapped back. The script segment below shows not only how to use *editopen* but also how to avoid an infinite loop if the edited text is not correct:

```

PCODE      ask 'Now your postcode'
            resp coded(0)
            set edflag = 0
comment Test postcode starts with two letters and a digit.
PCHK      set pcfmt1 = '%%# %%'
            callfunc('strngchk', pcfmt1, PCODE, isok1)
            route (isok1) go exit
BADPC     ask 'We were expecting your postcode to start with two
            letters and a digit. Yours does not. Can I just check that '
            '+PCODE+' is correct?'
            resp sp 'Yes, correct' go exit / 'No, incorrect'
comment Edit open end & set edit flag to show we have done so
            route (edflag = 1) go exit
            editopen PCODE
            set edflag = 1
            goto PCHK
EXIT      display 'Thank you for your help.'

```

The script starts by setting the edflag variable to 0. If interviewers enter a post code that does not start with two letters and a digit, they are asked to say whether or not it is correct. If the post code is what the respondent said, then the interviewer may direct the interviewing program to accept it.

If interviewers make a mistake typing the post code, they may say that it is not correct. The interviewing program then copies the post code to a temporary file for the interviewer to edit. It also sets the variable edflag to 1 to show that the variable has been edited. If the correction is still not right, the interviewing program will follow the routing and go to the end of the script. Without this routing, the script would remain in a permanent loop until a satisfactory post code was entered.

14.11 Testing for null-response and unanswered questions

Quancept CATI, Quancept CAPI and Quancept Web do not differentiate between questions that have not been answered and questions that the respondent answered by choosing *null* from the response list. Therefore, you can test whether a question is unanswered by using the logical expression:

```
(qname=null)
```

mrInterview is different, and so it is possible to treat unanswered questions differently to questions that have a null response. Only questions at which the respondent chose *null* have a null value and will pass the (*qname=null*) test. Questions that are not asked have no value at all and will always fail this test. The following script illustrates this.

```
useabc ask 'Does your company use any ABC Inc products?'
    resp sp 'Yes ('yes') / 'No ('no') go ctn
have ask 'Which products are they?'
    resp mp 'Tables' ('tab') / 'Call and See' ('cs') /
        'Desktop Tables' ('dt') / 'Stats for MR' ('smr') /
        'Survey Design' ('sd') / 'Basic Stats' ('bs') /
        'Qmoretabs' ('qm') null
ctn continue
    set number = numb(have)
ifnum if (number>0) {
d1      display 'Company uses the following ABC Inc products: '+have
        goto ctn2
}
ifhave if (have=null) {
d2      display 'Company uses ABC Inc products but the respondent
        did not say which ones.'
        goto ctn2
}
d3      display 'Company does not use ABC Inc products so the have
question was not asked.'
        }
ctn2 continue
```

There are two ways of dealing with this.

- If you want to retain the difference between unanswered and null-response questions, you can write an expression that tests whether the question has an empty response.
- You can specify that unanswered questions are to be treated the same as null-response questions, and write an expression that tests for a null response.

Testing for questions with empty responses

Quick Reference

To test whether a question is unanswered, type a logical expression of the form:

(qname=empty)

Only questions that have not been answered will pass this test. For example:

```
useabc ask 'Does your company use any ABC Inc products?'
        resp sp 'Yes' ('yes') / 'No' ('no') go ctn
have   ask 'Which products are they?'
        resp mp 'Tables' ('tab') / 'Call and See' ('cs') /
               'Desktop Tables' ('dt') / 'Stats for MR' ('smr') /
               'Survey Design' ('sd') / 'Basic Stats' ('bs') /
               'Qmoretabs' ('qm') null
ctn    continue
        if (have=empty) {
d3      display 'Company does not use ABC Inc products.'
        }
```

Treating unanswered questions the same as questions with a null response

Quick Reference

To treat unanswered questions the same as questions with a null response, type:

set noaneqn1 = 1

You can use this statement anywhere in your script, but placing it at or near the start of the script is good because it is then immediately clear how the script should work. You can use any non-zero value, not just 1.

If you need to return to the default behavior, type:

set noaneqn1 = 0

For example:

```
useabc ask 'Does your company use any ABC Inc products?'
    resp sp 'Yes ('yes') / 'No' ('no') go ctn
have ask 'Which products are they?
    resp mp 'Tables' ('tab') / 'Call and See' ('cs') /
        'Desktop Tables' ('dt') / 'Stats for MR' ('smr') /
        'Survey Design' ('sd') / 'Basic Stats' ('bs') /
        'Qmoretabs' ('qm') null
ifhave if (have=null) {
d2      display 'Company either does not use ABC Inc products or uses them
but the respondent did not say which ones.'
}
```

14.12 Sample script

```
comment Sample script for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 14, Keywords for checking non-numeric responses

progs      define  'Tables' / 'Call and See' / 'Desktop Tables' /
            'Stats for MR' / 'Survey Design' / 'Basic Stats' /
            'Qmoretabs'
have       ask 'Which ABC Inc products are used in your company?'
            resp mp progs
comment Were all products mentioned?
            set numprog = numv(have)
            set numhave = numb(have)
            if (numhave = numprog) {
decide      ask 'What were the reasons which made your company
            choose so many ABC Inc products?'
            resp mp 'Company policy' / 'Best products for the jobs' /
            'Nothing else caters for our type of work' /
            'No special reasons ^s' other
            }

comment Users with Stats for MR and Basic Stats
            set stats = and(have / 'Stats for MR' / 'Basic Stats')
            route (.not. stats) go ctrl
manual      ask 'You said your company has both Stats for MR and
            Basic Stats. Both products have manuals written in the tutorial style,
            but the way the tutorials are written is very different. Could you
            tell me which ones users prefer?'
            resp mp 'All users prefer Stats for MR' /
            'All users prefer Basic Stats' /
            'Some prefer Stats for MR and some prefer Basic Stats'
ctrl       continue

comment Users who have one of Tables or Qmoretabs and nothing else.
comment Example is for CATI/CAPI only. For Web/mrInterview, comment
comment out the xor line and uncomment the following 3 lines.
            set tables = xor(have / 'Tables' / 'Qmoretabs')
comment      set havet = bit(have/1)
comment      set haveq = bit(have/7)
comment      set tables = logical (.not. (havet .and. haveq))
            route (.not. tables) go ctn2
            route (numhave>1) go ctn2
whodata    ask 'Thinking of the data that you tabulate with '+have+' ,
            would you tell me whether it is data that your company collects
            itself, or is it data that you receive from other companies?'
            resp sp 'Our data' / 'Other companies'' data' /
            'Both our and other companies'' data'
```

Quancept and mrInterview Scriptwriter's Manual

```
format      ask 'How do you receive the data? Is it in the form of
printed questionnaires, or on a tape or floppy disk, or is it in some
other form (e.g., via a modem connection)?'
            resp mp 'printed questionnaires' / 'other printed forms' /
                  'on tape/disk' other
comment Users who tabulate data from their own printed questionnaires
        set owndata = nbit(whodata)
        set printq = or(format / 'printed questionnaires' /
                          'other printed forms')
        if (owndata<>2 .and. printq) {
datent      ask 'How do you enter the data onto the computer?'
            resp mp 'In-house data entry program' /
                  'Data entry program bought from elsewhere ^o'
                  other
awareqi     ask 'Do you know that ABC Inc markets DataIn, a
data entry program which generates data files in the Tables format?'
            resp sp 'Yes' / 'No'
}
ctn2       continue

comment A new product - all users answer this section
newprod     ask 'The company has just launched a new product,
ABC Games. This is a collection of games designed to distract
even the most dedicated employee from his/her work.@@
Have you seen or heard any advertising for this product recently?'
            resp sp 'Yes' / 'No' go ctn3
where       ask 'Where was that?'
            resp mp 'Radio' / 'TV' / 'Newspaper' / 'Sales handout' other

comment CATI/CAPI only: Get first advertisement mentioned and
comment fix it into the data
comment      set first = mention(where,1)
comment fix1    fix (1) first

ads         for advert = 1 to 5
comment Check by position number whether advertisement was seen/heard
comment and, if so, pick up the text
            set given = bit(where/advert)
            route (.not. given) go nxtad
            if (advert < 5) {
                set adtext = elm(where/advert)
}
            else {
comment Extract text from prompted other response
                set adtext = othertext(where,5)
}
adabout     ask 'What do you remember about the '+adtext+
advertisement?'
            resp coded
nxtad      next
ctn3       continue
```

```
comment Get name/address details of users wanting a demo copy
demo      ask 'Would you be interested in having a demonstration
copy of ABC Games?'
            resp sp 'Yes' / 'No' go exit
name      ask 'In that case, please let me have your name and
address for our mailing department. First, I''d like to take your name.'
            resp coded (0)
addr      ask 'And your postal address - I will ask for your
postcode in a moment?'
            resp coded (0)
PCODE      ask 'Now your postcode.'
            resp coded (0)

comment Set edit flag to 0 as open end is not yet edited
        set edflag = 0

comment Test postcode is in the correct format. Examples of valid
comment formats are NW6 2EG, WC1A 7XX, E5 9PP and S12 0QQ
pchk      set pcfmt1 = '%## %%'
            callfunc('strngchk', pcfmt1, pcode, isok1)
            route (isok1) go exit
            set pcfmt2 = '%##% %%'
            callfunc('strngchk', pcfmt2, pcode, isok2)
            route (isok2) go exit
            set pcfmt3 = '%# %%'
            callfunc('strngchk', pcfmt3, pcode, isok3)
            route (isok3) go exit
            set pcfmt4 = '%## %%'
            callfunc('strngchk', pcfmt4, pcode, isok4)
            route (isok4) go exit
badpc     ask 'We were expecting your postcode to start with one
of the following combinations:@
letter letter digit@
letter letter digit letter@
letter digit@
letter digit digit
@Yours does not match any of these formats. Can I just check that '
+PCODE+' is correct?'
            resp sp 'Yes, correct' go exit / 'No, incorrect'

comment CATI only: Edit open end & set edit flag to show we have done so
        route (edflag = 1) go exit
        editopen pcode
        set edflag = 1
        goto pchk
comment CAPI/Web/mrInterview only
comment  goto pcode
exit      display 'Thank you for your help.'
            pause
            end
```


15 Working with numeric data

This chapter explains statements that use numeric data. These include arithmetic expressions for performing simple arithmetic calculations, as well as a range of callable functions for converting between whole numbers (integers) and real numbers (decimal values). New keywords introduced in this chapter are:

longmath	arithmetic with whole numbers
realmath	arithmetic with real (decimal) numbers
setinteger	convert a real number to an integer (whole number)
setreal	convert an integer to a real value
valstring	convert numbers in text variables to numeric values

-
- ☞ In descriptions, the word ‘numeric’ refers to integer and real values equally unless otherwise noted. Where it is necessary to differentiate between integer and real values, the words integer and real are used as appropriate.
-

15.1 Arithmetic statements

Quick Reference

To assign the value of an arithmetic expression to a variable, type:

CATI: **set** *variable* = *a operator b*

CAPI/Web/mrInterview: **set** *variable* = *a operator b [operator c ...]*

In both cases *a*, *b* and *c* are numbers or variables with numeric values, and *operator* is one of the arithmetic operators * / + or -.

The simplest arithmetic statements are those involving numeric (integer or real) responses. Operators are:

+ addition	- subtraction
* multiplication	/ division

When we talk of values here we mean either an actual number (for example, 4) or the name of a question with a numeric response. In this latter case, the number is represented by a variable (the question name) and, as the interviewing program reads the statement, it replaces the variable name with the value of the response. For example, if question qp asks how many packets of frozen peas were bought, the response type will be *num*. If we then have a statement saying:

```
set tot = 2 + qp
```

the interviewing program will calculate the value of tot by substituting the value of qp and adding it to 2. If the respondent buys 5 packets of frozen peas, the statement will read tot=2+5, so tot will have a value of 7.

Another example is:

```
set total = cost * 1.175
```

which calculates the total cost of an item by multiplying the basic cost by the amount of sales tax required (17.5%).

Numbers in arithmetic expressions may also be negative:

```
set negvar = -10
```

Negative numbers must be entered by themselves and not as part of an expression containing another value. If you need to use a negative number with another value, you must place it in a variable and then use the name of that variable in the expression. For example, to divide a value by -50 you should write:

```
set top = 100  
set bottom = -50  
set result = top / bottom
```

rather than:

```
set result = top / -50
```

You can use an arithmetic statement within a loop to calculate totals. This involves the subscription of question names in the loop.

☞ An example of subscription of question names in loops may be found in section , Subscription when assigning values to repeated questions.



Quancept CAPI, Quancept Web and mrInterview can execute more than one arithmetic operation at a time. This means that you can include more than one operator in a line of code, reducing the amount of code needed to perform a mathematical operation. Use parentheses to clarify the order of calculation where necessary. (Quancept CAPI, Quancept Web and mrInterview follow the normal mathematical order of operations, with operations in parentheses being performed first followed by multiplication and division, and finally addition and subtraction.)

15.2 Variable types with arithmetic assignments

A *variable* is the name we give to a temporary storage area for a particular item of data. Questions are special types of variables. Every variable has its own data type which tells the interviewing program what type of data it can store in that variable. With question variables, the data type is determined by the response type on the *resp* statement. For example, you know that a question whose response type is *num* can only store whole numbers; it cannot store real numbers, text, or codes representing single-punched or multipunched answers.

The data types of temporary variables (that is, variables that are not questions) are determined by the type of data you place in them:

Assignment	Integer	Real	Text
set var = 'text'			✓
set var = 123	✓		
set var = 123.456		✓	

Therefore, a statement that says:

```
set total = 25.5
```

tells the interviewing program to create a real variable called *total* and to give it a value of 25.5. Similarly, the statement:

```
set name = 'Benjamin'
```

tells the interviewing program to create a text variable called *name* and to give it the text value Benjamin.

You may re-use variables within a script as long as you always place the same type of data in them. In our examples here, you may re-use *total* as long as you place another real number in it; you may re-use *name* as long as you place another text in it. If you try to store, say, a text in an integer variable, you will find that your script yields unexpected results when you use it for interviews.

A variable's data type has important implications with regard to assignments that evaluate arithmetic expressions. In this respect, Quancept CATI differs from Quancept CAPI and Web.

-
- ❖ If you are writing scripts that will be used for CATI and CAPI or Web interviewing you should test them carefully to ensure that any arithmetic expressions produce the desired results in all environments. If you find that your results are not quite what you would expected, check that you are using the right type of variables for the results of each expression.
-

In Quancept CATI



The rules for arithmetic expressions in Quancept CATI are based either on the data type of the variable in which the result will be saved or on the data type of the first item in the expression.

Saving into a question variable

When you save the result of an arithmetic expression in a question variable, the question type controls the data type of the expression. If the question is an integer (*resp num*) question, any real values in the expression are truncated at the decimal point *before* the expression is evaluated. For example:

```
total dummmmyask 'Total cost in whole units'  
      resp num 1 to 50  
cost  ask 'Exactly how much did you pay?'  
      resp real 1.0 to 20.0  
      set total = cost*1.175
```

Here ‘cost’ is a real question and the tax percentage is an explicit real value, but the total cost including tax is calculated as an integer because ‘total’ is an integer question. If the cost is 15.67 the true total would be 18.41, but because the reals are truncated before evaluation the total becomes 15.

When saving into a question variable, the only way to save a value as a real number is to save into a real question.

If you save an integer value in a real question variable, Quancept appends .000 to the value.

❖ Quancept CATI does not reject or flag results that are outside the valid range for the question in which they are saved. In the example, the ‘total’ question is defined as allowing responses in the range 1.0 to 50.0, but if the arithmetic expression produced a result outside this range Quancept would still make the assignment.

Saving into an existing temporary variable

Saving the result of an arithmetic expression in an existing temporary variable is the same as saving into a question variable: the variable’s data type controls the data type of the expression. If the variable is an integer variable, any real values in the expression are truncated at the decimal point *before* the expression is evaluated. If you are setting variables to zero before you use them, it is therefore important that you consider the values you want to place in the variables and set them to 0 or 0.0 accordingly. For example:

```
set itotal=0  
set rtotal=0.0  
cost ask 'Exactly how much did you pay?'  
      resp real 1.0 to 20.0  
      set itotal = cost*1.175  
      set rtotal = cost*1.175
```

Here, ‘itotal’ is defined as an integer variable so Quancept will truncate the value of cost and the 1.175 tax percentage at the decimal point before evaluating the expression. If the cost is 15.67, ‘itotal’ will be 15. ‘rtotal’ is defined as a real variable and will hold the true value of $15.67 * 1.175$ — that is, 18.41.

When saving in an existing temporary variable, the only way to save a value as a real number is to save into a real variable.

If you save an integer value in a real temporary variable, Quancept appends .000 to the value.

Saving into a new temporary variable

When you save the result of an arithmetic expression in a new temporary variable, the data type of the first item in the expression controls the data type of the expression. If the first item is an integer question, variable or value, any real values in the expression are truncated at the decimal point *before* the expression is evaluated. For example:

```
cost ask 'Roughly how much did you pay?'
      resp num 1 to 20
      set itotal = cost*1.175
      set rtotal = 1.175*cost
```

Here, ‘itotal’ is integer because ‘cost’ is an integer and ‘rtotal’ is real because 1.175 is a real value. If ‘cost’ is 15, ‘itotal’ will be 15 and ‘rtotal’ will be 17.625.

In Quancept CAPI, Quancept Web and mrInterview



The rules for arithmetic expressions in Quancept CAPI, Quancept Web and mrInterview are based firstly on a hierarchy of data types for evaluation and secondly on the data type of the variable in which the result is saved.

Data type hierarchy for evaluation

Quancept CAPI, Quancept Web and mrInterview use a hierarchy of data types to determine the data type of the result, as follows:

- Real question
- Real variable
- Integer question
- Real Value
- Integer variable or integer value

If the expression contains real variables and integer questions, the expression is evaluated as a real because the real variables take precedence over the integer questions. If the expression contains integer questions and real values, the expression is evaluated as an integer because the integer questions take precedence over the real values. The real values are converted to integers by truncating them at the decimal point *before* the expression is evaluated, so if q1 is 6, the expression q1*1.96 produces 6 rather than 11.76.

- ❖ If you assign the value of an integer question variable to a temporary variable, the temporary variable inherits the question's value but not its precedence within the hierarchy. This means that the question variable and the temporary variable are not interchangeable within the expression. For instance:

```
cost ask 'Roughly how much did you pay?'
      resp num 1 to 20
      set tcost = cost
      set total1 = cost * 1.175
      set total2 = tcost * 1.175
```

In the first expression, 'cost' is an integer question so it forces the expression to be evaluated as an integer. In the second expression, 'tcost' is a simple numeric variable so the real value takes precedence and forces the expression to be evaluated as a real.

Saving into question variables

Real values saved in real questions and integer values saved in integer questions are saved as reals and integers respectively. For example:

```
total dummyask 'Total cost'
      resp real 1.0 to 50.0
cost ask 'Exactly how much did you pay?'
      resp real 1.0 to 20.0
      set total = cost*1.175
```

The expression for 'total' contains a real question and an explicit real value so the result will be a real number — 18.41.

If you save an integer result in a real question variable, Quancept and mrInterview convert it into a real by appending .000 to it. For example:

```
total dummyask 'Total cost'
      resp real 1.0 to 50.0
cost ask 'Roughly how much did you pay?'
      resp num 1 to 20
      set total = cost*1.175
```

This example is the same as the previous one except that 'cost' is now an integer question. When used in the arithmetic expression, it takes precedence over the real value so Quancept and mrInterview convert the real number into an integer by truncating it at the decimal point. If 'cost' is 15, 'total' will be 15.000.

If you save a real value in an integer question, Quancept and mrInterview truncate the real value at the decimal point. For example:

```
total dummyask 'Total cost'  
    resp num 1 to 50  
cost ask 'Exactly how much did you pay?'  
    resp real 1.0 to 20.0  
    set total = cost*1.175
```

Here, ‘total’ is 18.41 but it is saved as 18 because ‘total’ is now an integer question.

When checking your script, bear in mind the following points:

- Quancept CAPI, Quancept Web and mrInterview do not display trailing zeroes in decimal places. For example, if an expression evaluates to 27.000, Quancept and mrInterview display it as 27 even though they write 27.000 into the data. If you are checking your script you should always look at the data to ensure that you are obtaining the correct results. It is very easy to assume that because you see 27 on the screen the result is integer.
- If the result of the expression is outside the valid range for the question in which it is saved, Quancept and mrInterview do not save the result and the question’s existing value remains unchanged. Quancept and mrInterview do not tell you when this happens, so it is important to make sure that all questions that have values assigned to them are defined with a range sufficient to cater for all possible values of the expression.

Saving into new or existing temporary variables

The data type of the result always controls the data type of the variable. If you save into an existing temporary variable this may mean that the variable’s data type changes. For example:

```
set itotal=0  
cost ask 'Exactly how much did you pay?'  
    resp real 1.0 to 20.0  
    set itotal = cost*1.175  
    set rtotal = cost*1.175
```

Here, ‘itotal’ is defined as an integer variable, but it becomes real when the expression is evaluated and saved in it. ‘rtotal’ is created as real because the expression has a real result.

15.3 Arithmetic assignment with question variables

Quick Reference

To assign an integer value to a question with a real response list, type:

callfunc('setreal', real_var, num_var)

To assign a real value to a question with an integer response list, type:

callfunc('setinteger', num_var, real_var)

To assign an integer value to a question with a real response list, use *setreal*. If you always want to place the same value in the question variable, you may insert a fixed value in place of the integer variable. Integers written out as reals always have three decimal places. For example:

```
price    ask 'How much did you sell your property for?  
resp num 30000 to 300000  
rate     ask 'And what percentage did the agent charge as a fee?'  
resp real 1.5 to 3.95  
fee      dummyask 'Fee value'  
resp real 1 to 300000  
set perc = rate / 100  
set fee = price * perc  
rprice   dummyask 'Price as a real value'  
resp real 30000 to 300000  
comment Copy integer price value into a real question variable  
callfunc('setreal', rprice, price)  
set rec = rprice - fee
```

This example calculates the amount of money the respondent received for his/her property once the agent's fees had been paid. The price paid by the purchaser is an integer value, whereas the fee is a real number. The dummy question 'fee' which stores the value of the agent's fees is also a real value.

To calculate the amount the respondent received, we need to subtract the fees from the sale price. Since we are saving this information in a temporary variable, qtip will create that variable with the same type as the first variable in the arithmetic expression. This is 'price' which is an integer variable, so the value of the fees will be truncated at the decimal point before the subtraction takes place. This reduces the accuracy of the calculation, so we use *setreal* to copy the integer price value into a real variable before we do the calculation. This ensures that the calculation is carried out using real values and that the result is also saved as a real value.

If the sale price was £59995 and the fees were 1.95%, the value of the fees will be £1169.903 and the amount received will be £58825.098.

To assign a real number to a question with an integer response list, use *setinteger*. The example below is an extension of the one shown above for *setreal*:

```

othexp ask 'Were there any other expenses? How much were
they in total?'
      resp real 0 to 5000
      set rec = rec - othexp
profit dummyask 'Profit = price - expenses and fees'
      resp num 1 to 300000
comment Assign real value to integer question variable
      callfunc('setinteger',profit,rec)

```

The variable ‘rec’ contains the amount the respondent received once the agent’s fees had been paid. If there are any other expenses, we subtract these from ‘rec’ to calculate the amount of profit made on the sale. Since we want to store this as a whole number to match the original sale price, we copy the real value into an integer question variable using *setinteger*. This truncates the real value at the decimal point.

If the value of ‘rec’ is £58,825.098 at the start of this script segment, and the respondent paid £3,513.06 in other expenses, the final value of ‘rec’ will be £55,312.038. When saved in the integer variable profit it will become £55,312 and will be written to the data as such.

Rounding real numbers

If you want real numbers with decimal places of 0.5 or greater to be rounded up to the nearest whole number, add 0.5 to the original number before converting it with *setinteger*. If you want 2.5 and 2.98 to be saved as 3, for example, your script would be:

```

q2 ask 'Choose a real number between 1 and 20'
      resp real 1 to 20
      set tempq2 = q2 + 0.5
      callfunc('setinteger',nvar,tempq2)
d1 display q2+' converted to an integer is '+nvar
      pause

```

Notice that we do not add the 0.5 directly to the value of q2 since this would change a variable on the interview path. This is a dangerous thing to do and is therefore discouraged. Instead, we use a temporary variable which will be created as a real because q2 (the first variable in the expression) is real.

The reason for using a temporary variable is to protect the integrity of your data. Consider the following statements:

```
q1      ask 'First question'
        resp sp 'Yes' / 'No'
q2      ask 'Choose a real number between 1 and 20'
        resp real 1 to 20
        set q1 = q1 + 0.5
d1      display 'q1 is '+q1
q3      ask 'Last question'
        resp sp 'Final response'
d2      display 'q1 is '+q1
```

Let us say that the interviewer enters 10.5 as the answer to q2. The *set* statement increments this to make it 11.0. If there are no snapbacks before the data is saved, the data for q2 will be written as 11.0.

Now, suppose that the interviewer reaches q3 and snaps back to q1 to check or change the answer. If he/she does not change the answer, the interviewer may snap forward to q3 without looking at q2 again. When qtip receives a forward snap command, it silently walks through all the statements between the current location and the snap destination until it reaches a question without an answer. In this example, it will read q2 and find that it has an answer of 11.0. It then executes the *set* statement and alters the answer to q2 to make it 11.5. Finally, it stops at q3, which has no answer.

A quick look at the script will tell you that the value saved for q2 will be 0.5 greater than the answer given, but this does not allow for snapbacks which alter the value further as we have just shown. Each time the interviewer snaps back to before q2 and then snaps forward again, qtip will silently increment the value of q2. If actions later in the script depend on the value of q2, you may find that the wrong actions are being carried out simply because of snapbacks which artificially increase the value of q2. By using a temporary variable which always takes the untouched value of q2 and adds 0.5 to that, you can avoid all these complications.

15.4 Arithmetic with text numbers

You can use text variables that contain integer or real values in arithmetic expressions in the same way that you use integer or real variables. When Quancept encounters a text variable in an arithmetic expression, it converts the text into the corresponding numeric value and then saves the result of the expression as an integer or real value according to the data type of the target variable. For backwards compatibility with very early versions, Quancept provides three functions for working with text variables that contain integer or real values:

- longmath** for arithmetic with integers
- realmath** for arithmetic with real value
- valstring** for converting a text value into an integer

Real numbers



Quick Reference

Use **realmath** for arithmetic with texts which are real numbers:

callfunc ('realmath', op, txtval1, txtval2, result)

where *op* is the type of operation and may be: 0 for addition, 1 for subtraction, 2 for multiplication or 3 for division; *txtval1* and *txtval2* are text variables storing the numbers to be used in the calculation; and *result* is the name of the text variable in which the result will be stored.

When your text or open-end variable contains a real number you may define your arithmetic expression using the *realmath* function. If either of *txtval1* and *txtval2* has a constant value for all interviews (for example, it is a conversion factor) you may enter the value directly in the *realmath* statement rather than using a variable.

The results of the calculation are shown to two decimal places so 193.677 becomes 193.68. Values of 0.005 or greater are rounded up (13.045 becomes 13.05) and values of less than 0.005 are rounded down (13.044 becomes 13.04).

For example:

```
price    ask 'How much did you sell your property for?
           resp coded
rate     ask 'And what percentage did the agent charge as a fee?
           resp coded
comment Multiply two text variables to find fee amount
           callfunc('realmath',2,price,rate,perc)
comment Divide a text variable by a constant value to find amount received
           callfunc('realmath',3,perc,100,fee)
feel     fix (8) fee
```

As the comments show, this script calculates the amount of money spent on agent's fees. The same figures apply here as in the example shown above for *setreal*; namely that fees at a rate of 1.95% on a sale price of £59,995 are £1169.90 (the calculation is $59995 * 1.95$; then $116990.25 / 100$. The decimal places of .9025 are rounded to become .90).

Integer numbers



Quick Reference

Use **longmath** for arithmetic with texts which are integers:

callfunc ('longmath', op, txtval1, txtval2, result)

where *op* is the type of operation and may be: 0 for addition, 1 for subtraction, 2 for multiplication or 3 for division; *txtval1* and *txtval2* are text variables storing the numbers to be used in the calculation; and *result* is the name of the text variable in which the result will be stored.

When the value you want is a whole number saved in a text variable, you use the *longmath* function. Here is an example which converts a real value into an integer by multiplying it by 1:

```
comment Multiply (2) prof2 which contains a real value by 1 and  
comment save it in profint  
    callfunc('longmath',2,prof2,1,profint)
```

If the values used or produced by *longmath* are real values they are truncated before the decimal point. This causes both 14.3 and 14.9 to be returned as 14.

Converting text numbers to integers

Quick Reference

Use **valstring** to place a text value in an integer variable:

CATI: **callfunc('valstring', numvar, textvar)**

CAPI/Web/mrInterview: **callfunc('valstring', textvar, numvar)**

where *textvar* is the variable containing the number as a text and *numvar* is a numeric variable which will contain the result of the conversion.

Here is a simple example written for CATI. A comment in the script shows the different valstring statement needed for CAPI, Quancept Web and mrInterview.

```
price     ask 'How much did you sell your property for?  
          resp coded  
rate      ask 'And what percentage did the agent charge as a fee?'  
          resp coded  
          callfunc('realmath',3,rate,100,perc)  
          callfunc('realmath',2,price,perc,fee)  
comment prof1=price-fee (real)  
          callfunc('realmath',1,price,fee,prof1)
```

```

othexp ask 'Were there any other expenses such as solicitor''s
fees incurred as a result of the sale? How much were they in
total?'
resp coded
comment prof2=prof1-othexp (real); profint is prof2 as an integer
    callfunc('realmath',1,prof1,othexp,prof2)
    callfunc('longmath',2,prof2,1,profint)
comment CATI: Convert a text number to a numeric (integer) value
    callfunc('valstring',profint,profint)
comment CAPI/Web/mrInterview: Convert a text number to a numeric (integer) value
comment callfunc('valstring',profint,profint)
prof fix (8) profit

```

All the calculations in this example are done using real numbers. However, we want to store the final profit in the data as an integer to match the original sale price. We therefore convert the value to integer with *longmath* and then use *valstring* to convert this from a text value to a numeric value.

valstring can cope only with integer values. If the text variable contains a real value it will be read as zero in Quancept CATI and will be truncated to an integer in mrInterview.

Testing for a real value in a given range

Here is an example which draws these three functions together in a script which checks whether a real number entered in a text variable is within a given range.

```

q1      ask 'Enter a decimal number between -10.50 and 90.00 in
one of the formats shown below:@
    ##      -##      @
    ##.#     -##.#     @
    ##.##    -##.##    @
    #.##    -#.##'
resp coded
comment Define string patterns to match the valid formats
    set patt(1) = '###.##'
    set patt(2) = '#.##'
    set patt(3) = '###.##'
    set patt(4) = '##'
    set patt(5) = '-##.##'
    set patt(6) = '-#.##'
    set patt(7) = '-##.#'
    set patt(8) = '-##'
comment Check whether answer to q1 matches one of the patterns
11      for patnum = 1 to 8
            callfunc('strngchk',patt(patnum),q1,isok)
            route (isok) go ctn
        next
wrong    display 'Value '+q1+ is not in the correct format'
pause
goto q1
ctn     continue

```

```
comment CATI only: Convert real value into integer. Multiply real by 100 to
comment ensure decimal places are 00.
    callfunc('realmath',2,q1,100,realval)

comment CATI only: Convert real value into integer. Reals truncated just
comment before decimal point.
    callfunc('longmath',2,realval,1,result)

comment Convert string variable result into numeric equivalent.
comment CATI: Save numeric value in numres.
    callfunc('valstring',numres,result)
comment CAPI/Web/mrInterview: Save numeric value in numres.
comment     callfunc('valstring',result,numres)

comment Set variables min and max to minimum and maximum decimal
comment values allowable for q1 multiplied by 100 (Quancept cannot
comment make comparisons between real and integer values).
    set min = -1050
    set max = 9000

comment Compare integer result with min/max acceptable values:
comment remember its value is 100 * q1
    if (numres < min) {
lt        display 'numres is less than '+min
        pause
    }
    if (numres > max) {
gt        display 'numres is greater than '+max
        pause
    }
```

The first step is to collect the value as text in q1. Notice how the question text includes instructions on what formats will be accepted. Next we define the acceptable formats in variables which we use with *strngchk* to test the format of the given value. If the value is in one of the valid formats, it is accepted; if not, the user is prompted to enter another value.

☞ For further information about *strngchk* see section 14.9, Checking text responses.

The next stage is concerned with converting the real value into an integer and converting the value from a text to a numeric value. First we multiply the text value by 100 to move the decimal point to the end of the value so that the decimal places are 00 (we use 100 because there were a maximum of two decimal places allowed). At this stage, the value is still stored as text even though it looks like a real number.

After this we call *longmath* to convert the real value into an integer. We multiply by one because we do not want to alter the value; we just want to get rid of the decimal point and the decimal places. Again, all values are still texts.

The final step in the conversion process is to convert the value from a text to a number. We use *valstring* for this.

Now we are ready to compare the value against the range. Because we multiplied the original value by 100, we have to apply the same rule to the range. Thus, the minimum and maximum are defined as -1050 and 9000 respectively, rather than -10.5 and 90. The range tests are just simple numeric comparisons of the value given against the minimum and maximum acceptable values.

15.5 Sample scripts

There are two scripts for this chapter. Both perform the same tasks but they use different keywords. Script A works with all variants of Quancept and with mrInterview, whereas script B works with CATI only.

Sample script A

```
comment Sample script A for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 15, Working with numeric data

agents    define  'Umbrella Estates' / 'McDougal Property Services' /
           'Green Properties' / 'High Class Estates' /
           'Harry Davies' / 'Thistlethwaite and Co.'
allag     ask 'Which agents did you approach when selling your
           property?'
           resp mp agents other
sell      ask 'And which one sold your property?'
           resp sp allag in agents other
price     ask 'How much did you sell your property for?'
           resp num 30000 to 300000
rate      ask 'And what percentage did the agent charge as a fee?'
           resp real 1.5 to 3.95
fee       dummyask 'Fee value'
           resp real 1 to 300000

comment Divide real by integer and save value in a temporary variable
comment perc is therefore real because rate is real
           set perc = rate / 100

comment Multiply integer by real and save in a predefined real variable
           set fee = price * perc
rprice    dummyask 'Price as a real value'
           resp real 30000 to 300000

comment Assign integer value to real question variable
           callfunc('setreal',rprice,price)

comment Subtract real from real, so rec is real
           set rec = rprice - fee
ok        ask 'Did you think a fee of '+fee+' was reasonable?'
           resp sp 'yes' go othexp / 'no'
why       ask 'Why not?'
           resp coded
othexp    ask 'Were there any other expenses such as solicitor''s
           fees incurred as a result of the sale? How much were they in total?'
           resp real 0 to 5000
           set rec = rec - othexp
profit    dummyask 'Profit = price - expenses and fees'
           resp num 1 to 300000
```

```
comment Assign real value to integer question variable
    callfunc('setinteger',profit,rec)
amt      display 'So the amount you were left with after all
expenses and fees were paid was '+profit+'. That is:@
Sale price = '+price+ minus@
Agent''s fees = '+fee+ minus@
Other expenses = '+othexp
    pause
end
```

Sample script B

 comment Sample script B for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 15, Working with numeric data (CATI only)

```
agents define 'Umbrella Estates' / 'McDougal Property Services' /
            'Green Properties' / 'High Class Estates' /
            'Harry Davies' / 'Thistlethwaite and Co.' other
allag ask 'Which agents did you approach when selling your
property?'
        resp mp agents other
sell ask 'And which one sold your property?'
        resp sp allag in agents other
price ask 'How much did you sell your property for?'
        resp coded
rate ask 'And what percentage did the agent charge as a fee?'
        resp coded

comment Multiply two text variables
    callfunc('realmath',2,price,rate,perc)

comment Divide a text variable by a constant value
    callfunc('realmath',3,perc,100,fee)
feel fix (8) fee
ok ask 'Did you think a fee of '+fee+' was reasonable?'
        resp sp 'yes' go ctnl / 'no'
why ask 'Why not?'
        resp coded
ctnl continue

comment Subtract one text variable from another, where one variable
comment contains a real value
    callfunc('realmath',1,price,fee,prof1)
othexp ask 'Were there any other expenses such as solicitor''s
fees incurred as a result of the sale? How much were they in total?'
        resp coded

comment Subtract one text variable from another, where one contains
comment a real value
    callfunc('realmath',1,prof1,othexp,prof2)
```

```
comment Multiply by 1 to convert a (text) real to a (text) integer
    callfunc('longmath',2,prof2,1,profint)

comment Convert a text number to a real value
    callfunc('valstring',profit,profint)
prof      fix(8) profit
amt       display 'So the amount you were left with after all expenses
and fees were paid was '+profit+'. That is:@
Sale price = '+price+' minus@
Agent''s fees = '+fee+' minus@
Other expenses = '+othexp
end
```

16 Ending the interview

Interviews may be terminated by statements in the script or by the interviewer typing or selecting a special response. When this happens, the data for that interview is, in most cases, written to the data files and in Quancept CATI accounting information is added to the act file.

There are a number of ways that interviews may be terminated by statements in the script:

- End of script — the *end* keyword is reached.
- End of interview — a *qexit* statement is read (Quancept CATI only)
- Planned stop — a *stop* keyword is read.
- Forced termination — a *signal* keyword is read.
- Quota control — the respondent falls into a quota-controlled category that is already full (Quancept CATI and Quancept Web only).

There are also several ways that an interviewer may terminate an interview:

- Temporary stop — the interviewer types *stop* or clicks the Stop button.
- Forced successful completion — the interviewer types *complete* (Quancept CATI only).
- Premature termination with data — the interviewer types *quit* (Quancept CATI only).
- Premature termination without data — the interviewer types *abandon* (Quancept CATI only).

☞ Quota control is described in chapter 30, ‘Quota control’, and chapter 31, ‘Quota maintenance’.

The special responses *quit*, *abandon*, *stop* and *complete*, which the interviewer may enter during an interview, are discussed in chapter 33, ‘Testing scripts with qtip’.

Stopping interviews in Quancept CAPI is discussed in section 35.3, Stopping and restarting interviews.

Stopping interviews in Quancept Web is discussed in section 36.3, Stopping and restarting interviews.

This chapter discusses keywords that may be used to end an interview from inside the script before all questions have been asked; for example, when the respondent is required to try the product before continuing with the interview. It also covers statements for determining whether or not data is saved for various types of uncompleted interviews. Keywords associated with these facilities are:

allow	allow early completion of the interview
chgstop	user-definable names for stopped data files
disallow	disallow early completion of the interview

qexit	terminate interview as if <i>end</i> had been reached
quitdata	save data for all terminated interviews
setdata	save data for all uncompleted interviews
signal	forced termination of an interview
stop	stop interview for later restarting
stopdata	save data for all stopped interviews

16.1 Allowing/disallowing early completion



Quick Reference

To mark the earliest point at which early completion is allowed, type:

allow

To disallow early completion once it has been allowed, type:

disallow

Early completion is the term used when the interviewer may terminate an interview before the end of the script by typing *complete* instead of a response from the response list. The data collected up to that point will be copied into the data files, and the interview will be flagged in the accounting file as successfully completed, even though some questions were not asked.

Whether or not an interviewer may terminate interviews in this way is determined by the script. If early completion is allowed, you may place the keyword **allow** in the script at the earliest point at which early completion is possible. For example, if early completion is not to be permitted until after question 10, the keyword *allow* must not appear in the script file before that point.

The default is to disallow early completion at all times. However, having allowed it for one section of the script, you may then disallow it for a later section by placing a **disallow** statement at the start of the relevant section of the script. If you never allow early completion, then you will never need to use *disallow*.

If an interviewer snaps back in an interview from a point where early completion is allowed to a point where it is not, qtip takes account of this and prevents early completion until the interviewer has again passed the *allow* statement.

☞ See section 33.18, Early completion of an interview, for notes on early completion from an interviewer's point of view.

16.2 Ending an interview without going to 'end'



Quick Reference

To terminate an interview without going to the *end* statement, type:

qexit

at the point at which the interview must end.

In large or complex scripts there may be many paths through the script and most respondents may not answer all questions. Interviews can only be successfully completed if the respondent reaches the *end* statement. One way of achieving this is to include routing statements such as *route* or *goto* that send all respondents to the end of the script. For example:

```
comment Questions for full-time workers
    Questions
    goto term
comment Questions for part-time workers
    Questions
    goto term
comment Questions for people who do not work outside the home
    Questions
term      end
```

Another way is simply to replace the routing with a *qexit* statement that tells the interviewing program to terminate the interview at this point and to flag it as successfully completed:

```
comment Questions for full-time workers
    Questions
    qexit
comment Questions for part-time workers
    Questions
    qexit
comment Questions for people who do not work outside the home
    Questions
term      end
```

In large or complex scripts this makes it immediately obvious that the interview is terminated and saves someone who may not be familiar with the script from scrolling through the script to see what happens next.

(A third option is to allow early completion by the interviewer as described in the previous section.)

16.3 Stopping and restarting interviews

The Quancept language allows you to stop an interview at a given point. You might need to do this if you want to conduct part of an interview and then continue it at a later date; for example, when the respondent has had a chance to try the test product. The keyword associated with this facility is **stop**.

- ☎ When an interview is stopped by a *stop* statement, the Quancept CATI, Quancept CAPI and Quancept Web interviewing programs create a file for the interview in a subdirectory of the project directory. The directory may contain a number of files, one for each stopped interview. The file is in a user-readable form, named with the respondent number of the interview whose data it contains. We call these files the *stopped data files*.

Interviewers may also stop interviews temporarily if asked to do so by the respondent. The interviewer may reschedule the call for a more convenient time and restart the interview at the point at which it was stopped when the callback is made. Data is saved in the stopped interview directory and can be restarted in the same way as interviews stopped by the script.

- ☎ The name of the CATI stopped interview directory is *script_stop*, so the file cars/cars_stop/132 will contain the data for respondent 132.

When the first interview in a project is restarted, qtip creates a subdirectory in the stopped data directory called USED and moves the file of stopped data from the stopped data directory into the USED subdirectory. For example, when the interview for respondent 132 is restarted, qtip moves the stopped data file cars/cars_stop/132 to cars/cars_stop/USED/132. This prevents two interviewers restarting the same interview.

If the interview is stopped again, a new 132 file is placed in the cars/cars_stop directory. The original 132 file is now in the USED subdirectory and is no longer accessible.

Options are available to create stopped data directories with names other than stop and for saving stopped data in files with names other than the respondent number. These facilities are described below.

- 💻 The name of the Quancept CAPI and Quancept Web stopped interview directories is ‘stopped’, so the file cars\stopped\132.stp will contain the data for respondent 132. The interviewing programs delete this file when the interview is restarted.
 - 💻 mrInterview always writes data to the case data file as soon as it is entered, regardless of whether or not the data is associated with a stopped interview. It does not use stopped data files. The case data variable DataCollection.Status stores the status of each interview so you can select only completed interviews from the data file at a later stage.
-
- ☞ Refer to section 33.3, Starting a session; section 35.3, Stopping and restarting interviews; and section 36.3, Stopping and restarting interviews, for further information about restarting interviews.

Stopping an interview from the script

Quick Reference

To stop an interview, type:

CATI: **stop 'dirname' [decrm='quota_suffix[,quota_suffix2,...]']**

CAPI/Web/mrInterview: **stop**

where *dirname* is the name of the directory in which the data gathered so far is to be saved. The optional *decrm* parameter for Quancept CATI identifies one or more quotas to be decremented at this point.

- ☎ The first time you stop an interview, qtip creates a subdirectory called *project__dirname* in which to place the files containing the data for the stopped interviews. The statement below requests that data for stopped interviews be saved in the directory *project__trial*:

```
stop 'trial'
```

If you want to use the same directory that is used for interviews stopped by the interviewer, type:

```
stop 'stop'
```

This will save the data in the default *project__stop* directory.

-
- ❖ Don't choose too long a name for the stopped directory if you are running Quancept on an operating system with a limitation on filename length. qtip takes two additional characters for the underscores that it inserts when it builds the directory name.
-

When a script contains quotas, they are normally decremented at the end of the interview. However, by identifying the quota file on the *stop* statement, you can have them decremented when the interview is stopped. For example:

```
stop 'trial' decrm='a1'
```

indicates that the quotas defined in the file suffixed a1 should be decremented when the interview is stopped rather than at its end.

-
- ☞ For further information on quotas in the script, see section 30.2, How the CATI and Web interviewing programs deal with quotas.
-

- ☎ If the *quota* statement has not been executed when the interview is stopped, it will remain available in the restarted portion of the interview and will be decremented when the interview is successfully completed. If the interview is not stopped (that is, the *stop* statement is skipped), the quota is decremented at the end of the interview as usual.
- ☒ If an mrInterview project uses sample management, the project's authentication page will ensure that the interview restarts just after the *stop* statement in the script. If the project does not use sample management the interview cannot be restarted.

Rotation and randomization in stopped interviews

If a script contains rotation or randomization, the random number used for this is saved for when the interview is restarted. This means that if an interviewer snaps back in a restarted interview to a question that was answered before the interview was stopped, any rotations or randomizations will appear as they did when the question was originally presented.

Changing the names of the stopped data files



Quick Reference

To define your own names for stopped data files, type:

callfunc ('chgstop',*file_var*)

where *file_var* is a variable containing the file name to be used.

When an interview is stopped, either by the interviewer or by a statement in the script, qtip normally saves the data in a file named according to the respondent number of the interview. The **chgstop** function allows you to define your own names for these data files.

This variable that defines the name of the stopped data files will usually be the name of a question with an open-ended response. If a file of this name already exists in the stopped data directory, a message to this effect is displayed and the data is saved in a file named with the interview serial number as before.

Here is a sample script to illustrate this:

```
stpname    ask 'INTERVIEWER: Please enter respondent''s surname  
and the last 4 digits of the telephone number.'  
        resp coded nodata  
        callfunc('chgstop',stpname)
```

This script fragment does not affect the data in any way. It simply sets up a filename to be used if the interview is stopped. If the respondent's name is Peter Smith and his phone number is (512) 1234-5678, the name of the stopfile will be smith5678. If you had this information available in your SMS sample files, you could build the file name automatically.

When the interview is stopped, qtip will display the name of the stopped data file. If the interviewer has entered spaces in the name, these will be converted to underscores — for example, smith_5678.

qtip keeps track of the respondent IDs associated with each stopped data file. When the interview is restarted, qtip reads the data from the stopped data file and allocates to it the respondent ID which it had when the interview was originally started. For example, if Peter Smith is respondent number 613, the interview stopped in the file smith5678 will be restarted as respondent 613.

-
- ❖ When deciding how to name the stopped data files, bear in mind any restrictions on the lengths of file names and the characters they may contain. For example, the file name Hamilton-Smythe9876 contains 19 characters and is therefore invalid on most systems.
-

16.4 Forced termination of an interview

Quick Reference

To terminate an interview with a predefined termination code, type:

signal *n*

where *n* is a positive whole number.

The most common way of terminating an interview when a respondent is not eligible to answer any other questions in the script is to route directly to the *end* statement. Any interviews which reach the *end* statement by whatever means are flagged in the accounting file as successfully completed. An alternative is to write a statement which terminates the interview immediately and flags it as having been terminated in particular way — for example, if a certain set of responses has been given, you may want the interview to terminate as if by quota control.

- ☎ The interviewing program accepts all positive whole numbers as signal values. Those with special meanings are:

Signal	Meaning	Acts as if
1	successful completion	the end of script had been reached
2	premature termination with data	the interviewer typed <i>quit</i>
5	terminated by quota control	a quota had been exceeded
6	early completion	the interviewer typed <i>complete</i>
8	premature termination without data	the interviewer typed <i>abandon</i>
9	interview stopped with no data written	the interviewer typed <i>stop</i> and answered no when asked whether to write data now
10	interview stopped with data written	the interviewer typed <i>stop</i> and answered yes when asked whether to write data now

- ☎ Of the unspecified signals, some write data collected so far to the data file, while others do not, and some signals offer the option of inspecting and/or changing that data before it is written. The table below summarizes the behavior of valid signals:

Signal	Meaning	Inspect data?	Write data?
0	—	✓	✓
1	successful completion	✓	✓
2	quit	✓	✓
5	quota	✓	✓
6	early completion	✓	✓
8	abandon	✗	✗
9	stopped interview with no data written	✗	✗
10	stopped interview with data written	✗	✓

Numbers outside the values shown in the table have similar but unspecified results (a number of them allow inspection of the data but do not save it). If you use them, be sure to check their effects prior to live interviewing.

- ⌚ In Quancept Web and mrInterview, all signal values have the same result as a *stop* statement in the script. Also, in mrInterview only, if the survey uses sample management, the signal value is passed to the sample management system as a SampleRecReturnCode.

☞ See the Sample Management section of the online *mrInterview User's Guide* for further details.

- ☎ The functions *quitdata* and *setdata*, which are covered in the next section, have no effect on signals. For example, if you have a *quitdata* statement in the script which causes all data for both quit and abandoned interviews to be written, terminating an interview with *signal 8* overrides this so that no data is saved for that interview.

This facility provides a useful alternative to routing; where you might otherwise use the keyword *route* to skip to the end of the script, you can use *signal* if you want. With *route*, the interview will be terminated immediately and flagged as successfully completed even though some questions have not been answered. This is perfectly correct, since respondents answer only questions which are relevant to them.

With *signal*, the interview will again be terminated immediately, but instead of being flagged as successfully completed, it will be flagged according to the code given with *signal*. For example, if the respondent does not buy yogurt, you may want to terminate the interview with a code 2 — premature termination with data. All data up to that point is saved in the data file exactly as it would be if *route* were used to skip to the end of the script. The only difference is the termination code in the accounting file.

16.5 Saving data for manually terminated interviews

-  When an interviewer stops a Quancept CATI interview by typing *stop* instead of entering a response from the response list, the CATI interviewing program checks with the interviewer whether to write the data collected so far to the data files. If the interviewer answers ‘n’, the data will be saved in the stopped data directory until the interview is restarted. At the end of the restarted interview, the data for the whole interview will then be written out to the data files.

If the interviewer answers ‘y’ to write data, the data collected so far will be written to the data files. When the full record is written at the end of the restarted interview, the data files will contain two records for that respondent. (You should remove these duplicate records from the data files before coding open ends or tabulating the data.)

 See the *Quancept Utilities Manual* for more information about deduplicating data.

When a respondent refuses to continue an interview, the interviewer again has control over whether or not any data collected so far is saved. If the interviewer terminates the interview with the *quit* response, data will be saved; if he/she uses *abandon*, no data is saved.

-  In Quancept CAPI, the interviewer stops an interview by clicking the Stop button. The CAPI interviewing program checks that the interviewer wants to stop the interview and, if so, asks whether the interview may be restarted later on. If the respondent does not wish to continue the interview, the interviewer will answer ‘no’ and the data will be written to the dat and drs files in the usual way.

If the respondent is willing to continue the interview later on, the interviewer will answer ‘yes’ to the restart question and the data collected so far will be written to the dat, drs and stp files. When the full record is written at the end of the restarted interview, the dat and drs files will contain two records for that respondent. (You should remove these duplicate records from the data files before coding open ends or tabulating the data.)

-  In Quancept Web, the respondent stops the interview by clicking the Stop button. Data collected so far is written to the dat, drs and stp files. If the respondent restarts and completes the interview, the dat and drs files will contain two records for that respondent. (Remove these duplicate records from the data files before coding open ends or tabulating the data.)

-  In mrInterview, the respondent stops the interview by clicking the Stop button. Data collected so far is written to the case data file. When the respondent restarts the interview, the data from the first portion of the interview is retrieved so that the respondent can check and, if necessary, change earlier answers. When the interview ends, the new data for the whole interview overwrites the respondent’s original record in the case data file.

You may prefer it if interviewers do not have this type of flexibility, so three statements are provided which may be used in the script to determine whether or not data should be saved. These statements do not take effect until they are executed during an interview, so if you want them to

apply to the whole interview, you should place them near the start of the script. On other occasions, you may want to save data only once a certain point has been reached. You may even repeat the statements with different values to alternate between saving and not saving data.

The table below summarizes these keywords. You will find full details of how to use them in the following sections.

Keyword	Action	Flags
stopdata	Save data for stopped interviews	0=do not save data, 1=save data
quitdata	Save data for quit or abandoned interviews	0=do not save data, 1=save data
setdata	Save data for stopped, quit or abandoned interviews	0=do not save data, 1=save data

Saving data for stopped interviews



Quick Reference

To save data when an interviewer stops an interview, type:

callfunc('stopdata', flag)

where *flag* is 1 to save data or 0 not to save data.

- ☎ In Quancept CATI, data is not normally written to the main data files when the interviewer stops the interview, instead, it is held only in the stopped data file until the interview is restarted. If you want to write the data for these interviews to the main data files you must insert the statement *callfunc(stopdata,1)* in the script. Using stopdata in a CATI script always suppresses the question about writing data that appears when the interview is stopped.
- ☎ In the example below, data will be saved automatically if the interview is stopped at the question *howm* or later:

```
trial    ask 'Would you be prepared to take part in a trial of the
        new healthwise bio yogurt? We would ask you and your household
        to eat one yogurt a day for a week and to watch out for and
        record all the advertising you see for the yogurt.'
        resp sp 'yes' go stop1 / 'no' go endst

comment Save data for all stopped interviews
        callfunc('stopdata',1)
comment Stop the interview
stop1    stop 'trial'
```

```
howm      ask 'I will need to talk to each member of your household  
        who took part in the trial by eating Healthwise Bio yogurt for a  
        week. How many people took part in the trial?'  
        resp num 1 to 10 nodata  
endst    continue
```

-  With Quancept CAPI and Quancept Web, any data gathered before the interviewer stops the interview is always written to the main data files, so you would normally use *stopdata* only when you want to prevent this happening. You might wish to do this if you do not want partially complete interviews appearing in the data files.

Saving data for terminated interviews



Quick Reference

To remove the difference between the *quit* and *abandon* command, type:

```
callfunc('quitdata',flag)
```

If *flag* is 1, data is always saved; if it is 0, data is never saved.

If the respondent refuses to continue with the interview, interviewers may type *quit* or *abandon* to terminate it. *quit* saves any data collected so far, whereas *abandon* does not. Sometimes you will want to save the data even if the interview was abandoned or to reject it for a quit.

The change to treating *quit* and *abandon* as the same thing applies only to interviews which are terminated after *quitdata* has been executed; if the interviewer terminates the interview before that point in the script, whether or not data is saved will depend on what keyword the interviewer uses.

Saving data for all uncompleted interviews



Quick Reference

To save or reject data automatically for all types of uncompleted interview, type:

```
callfunc('setdata',flag)
```

If *flag* is 1, data is always saved; if it is 0, data is never saved.

The **setdata** function is the equivalent of calling *quitdata* and *stopdata* both with the same argument; it allows you to save or reject data for uncompleted interviews regardless of whether they were stopped or terminated with *quit* or *abandon*.

If you wish, you may call *setdata* more than once within the script to switch data collection on or off at appropriate points in the script. For example:

Quancept and mrlInterview Scriptwriter's Manual

```
comment Do not save data for uncompleted interviews
      callfunc('setdata',0)
adv2    ask 'Are you aware that Healthwise are producing a new
      bio yogurt?'
      resp sp 'yes' / 'no' go trial
trial   ask 'Would you be prepared to take part in a trial of the
      new Healthwise Bio yoghurt? We would ask you and your household
      to eat one yogurt a day for a week and to watch out for and
      record all the advertising you see for the yogurt.'
      resp sp 'yes' go stop1 / 'no' go endst

comment Stop the interview
stop1   stop 'trial'

comment Save data for all stopped/quit/abandoned interviews
      callfunc('setdata',1)
howm   ask 'I will need to talk to each member of your household
      who took part in the trial by eating Healthwise Bio yogurt for a
      week. How many people took part in the trial?'
      resp num 1 to 10 nodata
endst   continue
```

16.6 Sample scripts

Script A



To use this script as it stands, you will need to create quota files with the identifier a1. Comments in the script explain how to modify it if you do not want to use quotas.

```

comment Sample script A for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 16, Ending the interview (under control of the script -
comment CATI only)

yflav    define  'apple and coconut' / 'banana' / 'cherry' /
           'damson' / 'nectarine' / 'plain'
ads      define  'Local tv station' / 'Channel 4' / 'Sky' /
           'Other satellite' / 'Radio' / 'Newspaper' /
           'Billboard' other

comment User-defined names for stopped data files
stpname  ask 'INTERVIEWER: Please enter respondent''s surname
           and the last 4 digits of the telephone number. This will be used
           as the name for the stopped data file. Spaces in the name will be
           converted to underscores.'
           resp coded nodata
           callfunc('chgstop',stpname)

comment Do not save data for quit/abandoned interviews
           callfunc('quitdata',0)
gender   ask 'Is the respondent ...'
           resp sp 'male' / 'female'
age      ask 'Which of the following age groups do you belong to?'
           resp sp '17 and under' / '18 to 25' / '26 to 45' /
           '46 to 65' / '66 and over'
yog      ask 'Do you personally eat yogurt at least once a week?'
           resp sp 'yes' / 'no' go endst

comment If you are not using quotas, comment the next three lines out
qot1    quota gender / age file='a1' pass=ok
           set fx1 = 1
           route (.not. ok) go final
           set fx1 = 2

advl    ask 'Have you seen advertising for the following brands of
yogurt. INTERVIEWER: code all seen'
           resp mp 'arctic' / 'cherry ripe' / 'edelweiss' /
           'healthwise' / 'igloo' / 'everest'
           set hw = bit(advl/4)
           route (.not. hw) go endst

```

```
when      ask 'You say you have seen advertising for Healthwise
yogurt. When was this?'
        resp sp 'today' / 'yesterday' / 'this week' /
                  'last week' / 'this month'
think     ask 'What can you tell me about the new Healthwise Bio
yogurt from the advertising you have seen?'
        resp coded
adv2      ask 'Are you aware that Healthwise are producing a new
bio yogurt?'
        resp sp 'yes' / 'no'
trial     ask 'Would you be prepared to take part in a trial of
the new Healthwise Bio yoghurt? We would ask you and your household to
eat one yogurt a day for a week and to watch out for and record all
the advertising you see for the yogurt.'
        resp sp 'yes' go stop1 / 'no'
        set fx1 = 3
        goto endst

comment Save data for all stopped interviews
callfunc('stopdata',1)

comment Stop the interview and decrement quotas. If you are not
comment using quotas, remove decrm='a1'
stop1    stop 'stop' decrm='a1'

comment Allow early completion.
allow

howm     ask 'I will need to talk to each member of your household
who took part in the trial by eating Healthwise Bio yogurt for a
week. How many people took part in the trial?'
        resp num 1 to 10 nodata
pers      for person = 1 to 10
        route (person > howm) go nxt1
getp      ask 'Please may I speak to person number '+person+'?@@
INTERVIEWER: If not all people are available, speak to those
who are and then type STOP and arrange an appointment to call
back. If this is not convenient, you may terminate the interview by
typing COMPLETE.'
        resp sp 'someone available to speak to' nodata
class     ask 'What is your position in the household?'
        resp sp 'husband/wife' / 'child' / 'grandparent' /
                  'house/flat sharer' / 'other'
likely    ask 'Did you like the Healthwise Bio yogurt?'
        resp sp 'yes' go flav / 'no'
whynot   ask 'Why not?'
        resp coded
        goto adverts
flav      ask 'Which of the flavors you tried did you like the
best?'
        resp sp yflav
```

```

nflav      ask 'And which did you like the least?'
            resp sp not flav(*) in yflav

adverts    ask 'How many advertisements did you see for
            Healthwise Bio yogurt?'
            resp num 0 go nxt1 / 1 to 50
media      ask 'Where did you see the majority of those adverts?'
            resp sp ads other
omedia     ask 'And where else did you see advertisements?'
            resp mp not media(*) in ads other null
nxt1      next
            goto endst

comment Set termination flag for respondents failing the quota
final     signal 5

endst     display 'Thank you for your help'
f1        fix (1) fx1
            end

```

Script B

```

comment Sample script B for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 16, Ending the interview (under interviewer/respondent
comment control)

yflav      define 'apple and coconut' / 'banana' / 'cherry' /
            'damson' / 'nectarine' / 'plain'
ads       define 'Local tv station' / 'Channel 4' / 'Sky' /
            'Other satellite' / 'Radio' / 'Newspaper' /
            'Billboard'

gender     ask 'Is the respondent ...'
            resp sp 'male' / 'female'
age       ask 'Which of the following age groups do you belong to?'
            resp sp '17 and under' / '18 to 25' / '26 to 45' /
            '46 to 65' / '66 and over'
yog       ask 'Do you personally eat yogurt at least once a week?'
            resp sp 'yes' / 'no' go endst

            set fx1 = 2
advl      ask 'Have you seen advertising for the following brands of
            yogurt?'
            resp mp 'arctic' / 'cherry ripe' / 'edelweiss' /
            'healthwise' / 'igloo' / 'everest'
            set hw = bit(advl/4)
            route (.not. hw) go endst

```

Quancept and mrInterview Scriptwriter's Manual

```
when      ask 'You say you have seen advertising for Healthwise
yogurt. When was this?'
        resp sp 'today' / 'yesterday' / 'this week' /
                'last week' / 'this month'
think     ask 'What can you tell me about the new Healthwise Bio
yogurt from the advertising you have seen?'
        resp coded
adv2      ask 'Are you aware that Healthwise are producing a new
bio yogurt?'
        resp sp 'yes' / 'no'
trial     ask 'Would you be prepared to take part in a trial of
the new Healthwise Bio yoghurt? We would ask you and your household to
eat one yogurt a day for a week and to watch out for and record all
the advertising you see for the yogurt.'
        resp sp 'yes' go stop1 / 'no'
        set fx1 = 3
        goto endst

stop1    continue

comment CATI/CAPI/Web: Save data for all stopped interviews.
comment Ignored by mrInterview which always saves data.
        callfunc('stopdata',1)

namadd   ask 'Please type your name and address in the box so that
we can send you the trial samples of Healthwise Bio yogurt.'
        resp coded nodata

comment Question text is for Web users.
comment For CATI, the text should be changed to instruct the interviewer
comment to type stop, and to select the single response to continue.
comment For CAPI, just remove the text to do with the interview code.
comment For mrInterview, remove the text to do with the interview code
comment and uncomment the 'set hidestop=0' statement to display the
comment Stop button.
comment      set hidestop=0
qstp     ask 'Please click the Stop button to suspend the interview,
and make a note of the interview code that is displayed on the
screen. You will need this code to continue the interview after you
have tried the test product.@@
If you have just restarted your interview, select the radio button
and click Next to continue.@@'
        resp sp 'I have tried the test products and want to continue
the interview' nodata

howm    ask 'We need to record the opinions of each member of
your household who took part in the trial by eating Healthwise Bio
yogurt for a week. How many people took part in the trial?'
        resp num 1 to 10 nodata
pers     for person = 1 to 10
        route (person > howm) go nxt1
```

```

getp      display 'Enter opinions for person number '+person
class     ask 'What is your position in the household?'
          resp sp 'husband/wife' / 'child' / 'grandparent' /
                  'house/flat sharer' / 'other'
likely    ask 'Did you like the Healthwise Bio yogurt?'
          resp sp 'yes' go flav / 'no'
whynot   ask 'Why not?'
          resp coded
          goto adverts
flav      ask 'Which of the flavors you tried did you like the best?'
          resp sp yflav
nflav    ask 'And which did you like the least?'
          resp sp not flav(*) in yflav
adverts  ask 'How many advertisements did you see for
          Healthwise Bio yogurt?'
          resp num 0 go nxtl / 1 to 50
media    ask 'Where did you see the majority of those adverts?'
          resp sp ads other
          route (adverts=1) go nxtl
omedia   ask 'And where else did you see advertisements?'
          resp mp not media(*) in ads other null
nxtl    next
endst   display 'Thank you for your help'
f1      fix (1) fx1
end

```

Notes for Quancept Web users

In order to see the interview ID that is assigned to the stopped portion of the interview, you will need to create a file called *project_name.ini* in the project directory on the Quancept server containing the following lines:

```
[HTML]
StopButtonEnabled=Yes
ShowStopStringAtEnd=The code for restarting the interview is <BR>
```

The Quancept script displays instructions on how and when to stop the interview. It is important that you make a note of the interview ID that is displayed when you stop the interview as you will need this code to restart the interview. To restart the interview, type the following command in your browser:

http://server_name/scripts/qcweb.dll?starthtml&server_name&project_name&interview_ID

where *server_name* is the name of your Quancept server, *project_name* is the project name, and *interview_ID* is the code that was displayed when you stopped the interview. (If qcweb.dll is not installed in the scripts directory, you will also need to replace the word *scripts* with the appropriate directory name.)

Notes for mrInterview users

In order to restart stopped interviews, you must set up the project to use Sample Management. Refer to the Sample Management section of the online *mrInterview User's Guide* for details.

17 Data format

When a script is error free, the parser generates a map file showing which columns have been allocated to each question in the script. Quancept uses the information in the map file, which is also known as the ***data map***, to write the responses to questions into the data files. Data is stored (for historical reasons) in a file that simulates a stack of punched cards, where each ‘card’ contains 80 columns.

-
- ☞ See section 38.2, Files created by the CATI, CAPI and Web interviewing programs, for more information about the data files.
-

The allocation of columns can be automatic, based primarily on the number of responses in the response lists and the order of questions in the script. Automatic allocation of columns is applied when you parse your script. There will, however, be times when you want some control over the data format.

By default, columns are allocated sequentially so that all responses to a particular question appear consecutively in the data file. Using *grouped* on a *for* statement, you can control the order in which columns are allocated to questions in loops.

-
- ☞ See section 10.6, Column allocation with loops, for more information.
-

In Quancept CATI, you also have the following options for data mapping. You can:

- Allocate questions to columns using your own predefined map file (by parsing with *qparse -u*).
- Allocate specific cards and columns to hold the responses to some or all questions.

-
- ☞ See ‘Using predefined data map files’ in chapter 32, ‘Parsing and compiling scripts’, for information about *qparse -u*.
-

mrInterview does not automatically save data in card-column format. Instead, it writes the data gathered during each interview as a separate record in a relational database file. It is then up to your company to choose how to export the data from the database. Most of the keywords covered in this chapter therefore do not apply to mrInterview.

This chapter introduces keywords that you can use to determine the position of data in the record and account for data as it is collected:

codscheme	choose the format in which single-punched and multipunched data is written to the dat file
col xxyy	specify a question’s location in the data file

column xxxy	specify the location in the data file of the first question in a block of questions
column 100	switch off the effect of an earlier <i>column</i> statement and revert to the standard data layout
datamap	define the number of columns for the serial number and card type in the dat file
fix	copy texts and values of variables into the data
force	force columns for extra questions to the end of the record
mcodscheme	choose the format in which single-punched and multipunched data is written to the dat file
newcard	start a new card
noblank	suppress blank cards
noclean	save all data regardless of changed routing patterns
privsig	store a customized private status code in the interviewing program's accounting file
scodscheme	choose the format in which single-punched and multipunched data is written to the dat file

17.1 Adding extra questions to the script



Quick Reference

When extra questions are added to the script once interviewing has begun, add **force** to the end of the response lists to have the data for those questions added to the end of the record:

resp *resp_type* [*resp_list*] **force**[(*n*)]

n is the wave number and is required when questions are added at different times.

From time to time, you may need to add extra questions to the script. For example, you may decide you need more detailed information about why the respondent liked or disliked the product, or what he/she considers a reasonable price to pay for it. If the script has not been used — that is, no production interviews have been conducted yet — then you can add the questions at the relevant places in the script and reparse it.

Once the script has been used for an interview, the data file will contain specific data in specific columns so that the addition of extra questions in the middle of the script will cause any new data from further interviews to be incompatible with the old.

Suppose that the original script allocates the responses to questions 1 to 10 to columns 11 to 20. If we amended the script by inserting additional questions between questions 6 and 7, the responses to questions 7, 8, 9 and 10 would no longer be stored in columns 17 to 20. They would be shifted to the right so that the responses to the extra questions could be accommodated.

The parser offers a way around this problem by allowing the extra questions and their response lists to be typed at the appropriate places in the script, but flagged so that the responses are coded at the end of the original data. If you add questions at different times, you may flag each batch of questions so that each new batch is coded at the end of the existing data.

force by itself is the same as *force(1)*. The effect is that all *force* or *force(1)* responses are added to the end of the record first, followed by all *force(2)* responses, and so on. This allows you to force new questions in waves rather than as a single group, preserving the previous layout each time.

To illustrate this, we will take a basic script and add questions to it in two phases. Each time, we will look at the map file to see how the parser has allocated columns to the additional questions. The basic script is:

```

gender    ask 'INTERVIEWER: Is the respondent ...'
          resp sp 'male' / 'female'
age       ask 'Which age group do you belong to?'
          resp sp '18 and under' / '19 to 25' / '26 to 35' /
                  '36 to 45' / '46 to 55' / '56 to 65' /
                  '66 and over'
school    ask 'What level of education have you achieved?'
          resp sp 'O level/GCSE' / 'A level' / 'HNC' / 'HND' /
                  'degree' / 'post graduate'
numwage   ask 'How many people in your household receive a wage,
          salary or some type of benefit?'
          resp num 1 to 10
income    ask 'What is the approximate total income for your
          household?'
          resp sp '10,000 or less' / '10,001-20,000' /
                  '20,001-30,000' / '30,001-40,000' /
                  '40,001-50,000' / '50,001 or more'
sweets    ask 'Do you eat sweets?'
          resp sp 'yes' / 'no' go bye
prefer    ask 'Which sort of sweets do you prefer?'
          resp sp 'boiled fruit sweets' / 'soft fruit sweets' /
                  'hard mints' / 'soft mints' / 'chewing gum' /
                  'toffee' / ('chocolate' / 'chocolate snack bars' /
                  'chocolate cookies') go ctrl1
buychoc   ask 'Do you ever buy chocolate?'
          resp sp 'yes' / 'no' go bye
ctrl1     continue
cpref     ask 'Which sort of chocolate do you prefer?'
          resp sp chocs
bye      display 'Thank you for your help.'
          end

```

The data map for the script is:

qcp_ser_	Numeric	(1)	1	1	5
qcp_crd_	Numeric	(1)	1	6	7
cardtype 1 ...	type	iter	crd	frm	to
gender	Single	(1)	1	8	8
age	Single	(1)	1	9	9
school	Single	(1)	1	10	10
numwage	Numeric	(1)	1	11	12
income	Single	(1)	1	13	13
sweets	Single	(1)	1	14	14
prefer	Single	(1)	1	15	15
buychoc	Single	(1)	1	16	16
cpref	Single	(1)	1	17	19

After some interviewing has taken place, the project manager asks you to add some extra questions as follows:

- After the preference question, ask those not preferring any type of chocolate why they prefer their choice to chocolate.
- At the end of the script, ask if respondents have tried the new plain and milk slice bars and, if so, what they thought of the bar size.

To implement these changes, you enter the questions and response lists at the points in the script where the questions are to be asked. Since you want the new data to be compatible with the old, you add the keyword *force(1)* to each response list so that the data for the extra questions is written to the end of each record.

The script is now as follows (we've omitted the demographic questions from the example to save space, since they are unchanged):

```

comment Demographic questions here
sweets    ask 'Do you eat sweets?'
            resp sp 'yes' / 'no' go bye
prefer    ask 'Which sort of sweets do you prefer?'
            resp sp 'boiled fruit sweets' / 'soft fruit sweets' /
                  'hard mints' / 'soft mints' / 'chewing gum' /
                  'toffee' / ('chocolate' / 'chocolate snack bars' /
                  'chocolate cookies') go ctrl1
why       ask 'Why do you prefer '+prefer+' to chocolate?'
            resp coded force(1)
buychoc   ask 'Do you ever buy chocolate?'
            resp sp 'yes' / 'no' go bye
ctrl1     continue
cpref     ask 'Which sort of chocolate do you prefer?'
            resp sp chocs
tried     ask 'Have you tried the new Plain and Milk Slice
            chocolate bars?'
            resp sp 'yes' / 'no' go bye force(1)

```

```

size      ask 'What did you think of the size of bar it comes in?@'
INTERVIEWER: Press respondent to say why they chose a particular
answer.'
      resp sp 'too big ^o' / 'just right ^o' / 'too small ^o'
      other force(1)
bye       display 'Thank you for your help.'
end

```

and the data map is:

qcp_ser_	Numeric	(1)	1	1	5
qcp_crd_	Numeric	(1)	1	6	7
cardtype 1 ...	type	iter	crd	frm	to
gender	Single	(1)	1	8	8
age	Single	(1)	1	9	9
school	Single	(1)	1	10	10
numwage	Numeric	(1)	1	11	12
income	Single	(1)	1	13	13
sweets	Single	(1)	1	14	14
prefer	Single	(1)	1	15	15
buychoc	Single	(1)	1	16	16
cpref	Single	(1)	1	17	19
why	Coded	(1)	1	20	22
tried	Single	(1)	1	23	23
size	Single	(1)	1	24	24
size[other]	Other	(1)	1	25	27

After this version of the script has been used for a while, the project manager requests another change:

- Cancel the question about plain and milk slice bars and replace it with one about chocolate with mint chip wafers. Ask respondents who tried this product what they thought about the mint flavor.

You cannot delete the question about plain and mint slice bars because this would alter the column allocations of all questions after it in the script. Instead, you insert some routing before that question, which takes all respondents on to the new question about the chocolate wafers, which comes after it. To maintain the integrity of the existing data, you add the keyword *force(2)* to the additional questions you insert. The final version of the script is as follows. Again, we've shown only the part affected by the changes; the earlier sections are as shown in the previous examples.

```

cpref    ask 'Which sort of chocolate do you prefer?'
        resp sp chocs
        goto tried2
tried   ask 'Have you tried the new Plain and Milk Slice chocolate bars?'
        resp sp 'yes' / 'no' go bye force 1)
tried2  ask 'Have you tried Chocolate and Mint chip wafers?'
        resp sp 'yes' / 'no' go bye force(2)
taste   ask 'What did you think of the mint flavor?'
        resp coded force(2)
        goto bye

```

```

size      ask 'What did you think of the size of bar it comes in?@'
INTERVIEWER: Press respondent to say why they chose a particular
answer.'
      resp sp 'too big' / 'just right' / 'too small'
      other force(1)
bye       display 'Thank you for your help.'
end

```

The data map for this version of the script is:

qcp_ser_	Numeric	(1)	1	1	5
qcp_crd_	Numeric	(1)	1	6	7
cardtype 1 ...	type	iter	crd	frm	to
gender	Single	(1)	1	8	8
age	Single	(1)	1	9	9
school	Single	(1)	1	10	10
numwage	Numeric	(1)	1	11	12
income	Single	(1)	1	13	13
sweets	Single	(1)	1	14	14
prefer	Single	(1)	1	15	15
buychoc	Single	(1)	1	16	16
cpref	Single	(1)	1	17	19
why	Coded	(1)	1	20	22
tried	Single	(1)	1	23	23
size	Single	(1)	1	24	24
size[other]	Other	(1)	1	25	27
tried2	Single	(1)	1	28	28
taste	Coded	(1)	1	29	31

17.2 Forcing data onto a new card



Quick Reference

To start a new card in the data, type:

newcard

If you want to start a new card in the data, place a **newcard** statement at that point in the script. Any unused columns on the current card will be left blank and the interviewing program will put the data for the next question on the new card. You cannot choose which card number to use because these are allocated automatically by the parser.

You may place two or more *newcard* statements consecutively in the script. The effect of two consecutive *newcards* is to generate an ‘empty’ data card in between two cards of data. Quancept CAPI and Quancept Web both suppress this card, but in Quancept CATI the ‘empty’ card appears in the data file with just the respondent serial number and a unique card type at the start of the card.

The exception in all Quancept variants is when you have two *newcard* statements separated only by a forced question. In this case only one new card is generated. The data for the forced question appears at the end of the data record as usual.

If you place *newcard* inside a standard ungrouped loop, the parser will start a new card the first time the statement is read. It is then ignored for the second and subsequent iterations. The following example helps to explain why this is:

<pre>for i = 1 to 3 q1 newcard q2 next</pre>	<p>produces:</p>	<p>card 1: q1(1) q1(2) q1(3) card 2: q2(1) q2(2) q2(3)</p>
--	------------------	---

When the parser reads the script and assigns columns to questions, the *for* statement tells the parser that each question will be asked three times. When the parser reaches q1, it therefore allocates space in the data file for three sets of answers for that question. The next statement is *newcard*, so the parser starts a new card as requested. Then it reaches q2 and allocates columns for three sets of answers for that question. At the end of the loop, the parser knows that it allocated columns for all iterations of all questions, so it continues with the next statement in the script.

Grouped loops behave differently:

<pre>for j = 1 to 3 grouped q1 newcard q2 next</pre>	<p>produces:</p>	<p>card 1: q1(1) card 2: q2(1) q1(2) card 3: q2(2) q1(3) card 4: q2(3)</p>
--	------------------	---

The *grouped* keyword tells the parser that columns are to be allocated in the order in which the questions are asked, so it steps through the loop exactly as it will during an interview. This means that it reads the statements in the order in which they appear in the loop as many times as the *for* statement directs. Every time the *newcard* statement is read, a new card is started in the data file.

Here is an example of how to write a script which gives each adult in the household a separate card:

```
adults    ask 'How many wage earning adults are there in your household?'
          resp num 0 go none / 1 to 5
d1       display 'May I speak to all the wage earning adults?'
pers     for person = 1 to 5 grouped
          newcard
          route (person > adults) go none
q1       ask 'Which age group applies to you?'
          resp sp 'Under 21' / '21-30' / '31-40' / '41-50' /
                  '51-60' / 'Over 60' ref
```

```
q2           ask 'Are you .... '
             resp sp 'In full-time employment' /
                     'In part-time employment' / 'A pensioner'
nxt1        next
none         continue
```

Data for questions up to and including ‘adults’ is stored on card 1. If there are two wage earners in the household, the data for the first one is stored on card 2 and the data for the second one is stored on card 3. The interviewing program returns to the start of the loop and begins a new card, card 4. However, the count of adults is now less than the value of person, so the interviewing program skips out of the loop. All data from this point on is stored on card 4 onward.

17.3 Specifying the card and columns for one question



Quick Reference

To write data for a question into a specific location in the data file, type:

```
resp resp_type [resp_list] col xxxy
```

where *xx* is the card number, and *yy* is the column number up to column 80. A leading zero is required for column numbers less than 10.

For example.

```
films   define 'Snow White' / 'The Lion King' / 'Dumbo' /
             '101 Dalmations' / 'Jungle Book' / 'Toy Story'
seen    ask 'Which films have your children seen recently?'
         resp mp films
buy     ask 'And which of these films would you buy on video?'
         resp mp films col 208
best    ask 'Which film did your children like best?'
         resp sp seen in films
```

Assuming that this is the start of the questionnaire, ‘seen’ will be coded into card 1, column 8. ‘buy’ would normally be coded into card1, column 9, but the script forces it to be placed in card2, column 8 instead. ‘best’, which would normally be allocated to card 1, column 10, is now coded into card 1, column 9.

If the response list contains *other*, *null*, *dk* or *ref*, *col* must come after *other* but before *null*, *dk* and *ref*.

17.4 Specifying card and columns for blocks of questions



Quick Reference

To specify card and column locations for a group of questions, type:

column xxyy

on the line before the first question in the group. *xx* is the card number and *yy* is the column number up to column 80. A leading zero is required for column numbers less than 10.

If the group does not extend to the end of the script and you want to revert to the standard column allocations for later questions in the script, type:

column 100

on the line after the last statement in the group.

Data is written sequentially, filling up columns and cards in the normal way, but the interviewing program will skip over any columns that have been designated for other questions. Using *newcard* after a *column 100* statement causes data to be written into the next available (default) column on the following card. You can use multiple *newcard* statements to advance by more than one card.

Note the comments in the following example.

```

films    define 'Snow White' / 'The Lion King' / 'Dumbo' /
          '101 Dalmations' / 'Jungle Book' / 'Toy Story'
comment q1, q2 and q3 are coded in card 1, columns 8, 9 and 10
q1      ask 'Which films have your children seen at the
        cinema recently?'
        resp mp films
q2      ask 'And of those films, which one or ones was the most
        popular?'
        resp mp q1 in films
q3      ask 'If you could buy just one of these films on video
        which one would it be?'
        resp sp films

comment q4 and q5 are coded in card 1, columns 30 and 31
column 130
q4      ask 'How often do you take your children to the cinema?'
        resp sp 'Once a week' / 'Several times a month' /
                  'Once a month' / 'Several times a year'
                  'Once a year' / 'Less than once a year'
q5      ask 'And on these visits, do you buy anything from the
        in-cinema shops?'
        resp mp 'Sweets' / 'Drinks' / 'Popcorn' /
                  'Hot dogs/burgers' / 'Nothing ^s' other

```

```
comment End of block, continue coding on next free column
comment after q3; that is, column 11
    column 100

howmany ask 'How many children do you have under the age of 13?'
    resp num 1 to 9
ages   ask 'And how old are they?'
    resp mp 'Under 1 year' / '1 to 3 years' / '4 or 5 years' /
        '6 to 10 years' / '11 to 12 years'
```

In this example, the data for questions q1 to q3 goes into columns 8, 9 and 10 of card 1 respectively. The *column 130* statement places data for all subsequent questions into columns 130 onwards. The *column 100* statement resets card and column mapping, so the data for the howmany question will go back onto card 1, column 11, immediately following the data for q3.

Avoiding data map conflicts



-
- ☞ You may not assign multiple questions into a single set of columns. If you want to combine the answers from multiple questions into one or more columns, use the keyword *set* to assign the answers into the columns allocated to a dummy question.
 - ☞ See section 12.4, Assigning responses to questions, for more information.
-

There are two situations where *qparse -u* and the keywords *col* and *column* can come into conflict:

- attempting to put the same question into different sets of columns, or
- attempting to put more than one question into a single set of columns

Behavior in these circumstances is as follows:

- *qparse -u* overrides *col* and *column*, and
- the parser issues an error saying that data for the second and subsequent questions cannot be allocated to the specified column.

If *col* and *column* specify the same location in the data map, even for the same question, the parser will issue an error, such as:

```
script_name:9 Unable to place data for question q2 in column 315
Quancept Parse - 1 error, 0 warnings
```

The message above was generated for the following script fragment:

```
    column 315
q1      ask 'text for q1'
        resp sp 'a'/'b'/'c'
q2      ask 'text for q2'
        resp sp '1'/'2'/'3' col 315
```

If *col* and *column* specify different locations, *col* will override *column*. For example, this next script fragment will allocate column 240 to question qx, column 315 to qy and column 316 to qz.

```
        column 315
qx      ask 'text for question x'
        resp sp 'a'/'b'/'c' col 240
qy      ask ' text for question y'
        resp sp '1'/'2'/'3'
qz      ask 'text for question z'
        resp sp 'x'/'y'/'z'
```

17.5 More columns for serial number and card type



Quick Reference

To define the number of columns to reserve for the respondent serial number, type:

datamap serialcols = *number_of_columns*

To define the number of columns to reserve for the card type, type:

datamap cardcols = *number_of_columns*

If used, these variables must be defined once only at the beginning of the script. For example:

```
datamap serialcols=6
datamap cardcols=3
```

-
- ↖ Each card in the dat file can hold up to 80 columns of data. Increasing the number of columns allocated to the serial number or card type reduces the amount of data you can store on each card and may increase the overall size of the dat file.
-

17.6 Data mapping for single-punched and multipunched responses



Quancept normally writes data to the dat file in card-column format. However, it provides several keywords that give you the flexibility to select a different format for single-punched and multipunched data, to switch between formats during a questionnaire or have one question in a particular format. Possible output formats are:

- In standard card-column format
- As numerics — for example, response 1 in a response list of 100 responses is written as 001

- In exploded binary format — for example, response 2 in a response list of 10 responses is written as 0100000000, and response 6 is written as 0000010000.

Typically, you might stick to the standard card-column format for single-punched responses and use one of the other formats for multipunched data.

» These facilities apply only to precoded single-punched and multipunched responses; they do not apply to specified other or open-ended responses.

The utilities qct, qcum and qcprt work only with surveys whose data is written in card-column format. Use QDItum to create Quantum axes for surveys that format data using *codscheme*, *scodscheme* or *mcodscheme*.

Changing the data format for a block of questions



Quick Reference

To define the data format for single-punched responses type:

scodscheme *code_type*

To define the data format for multipunched responses type:

mcodscheme *code_type*

where *code_type* defines the type of coding:

- 0 to use the standard card-column format
 - 1 to write out numeric codes based on responses' positions in the response list
 - '10' to write an on/off switch for each response, where *1* is the code to write if the response is chosen and *0* is the code to write if the response is not chosen
-

If you use *mcodscheme 1*, you must define the maximum number of responses that may be chosen from each multipunched list so that the parser knows how many columns to allocate to the question. If you forget to do this, the parser will fail with the message ‘*codscheme 1* requires maxpunch for mp’.

These changes apply from the point at which the statement appears until another *scodscheme* or *mcodscheme* statement is read, or until the end of the script. You can switch coding methods as many times as you wish and wherever you wish. The specified format can also be overridden for an individual question in isolation.

Here is an example in which single-punched responses are written in card-column format and multipunched responses are written as numeric codes. Notice how the maximum number of responses allowed has been defined for every multipunched list:

```
films    define 'Snow White' / 'The Lion King' / 'Dumbo' /  
           '101 Dalmations' / 'Jungle Book' / 'Toy Story'  
comment Write multipunches out as numeric codes  
mcodscheme 1  
q1      ask 'Which of the following films have your children seen  
at the cinema recently?'  
        resp mp 6 films null go ctn dk go ctn  
q2      ask 'And of those films, which one or ones was the most  
popular (up to 3 choices allowed).'  
        resp mp 3 q1 in films  
ctn     continue
```

-
- ☞ For detailed examples of the way data is written out, see ‘What does the data look like?’ later in this section.
-

Setting the data format for individual questions



Quick Reference

To override the current data format for one question only, type:

```
resp sp codscheme code_type [rot/ran/atoz] responses  
resp mp [max] codscheme code_type [rot/ran/atoz] responses
```

The maximum number of answers is required in multipunched lists only if *code_type* is 1; that is, you want to use numeric codes for responses.

For example, to override the current multipunched layout for q2 with a string of on/off switches, you would type:

```
q2      ask 'And of those films, which one or ones was the most  
popular.'  
        resp mp codscheme '10' q1 in films
```

What does the data look like?

Type 0 — card-column format

Quancept normally writes data out in card-column format. If a script contains the following:

```
q1      ask 'Which of the following films have your children seen  
at the cinema recently?'  
       resp mp 'Snow White' / 'The Lion King' / 'Dumbo' /  
              '101 Dalmations' / 'Jungle Book' / 'Toy Story'
```

and the respondent chooses Snow White, The Lion King and Toy Story, the data will be as follows (the first two lines are a ruler marking the column numbers):

```
1  
----+---0  
00089011  
2  
6
```

Type 1 — numeric codes

If you choose to write the data out as numeric codes, Quancept numbers quoted responses sequentially from 1 according to their positions in the list and writes the appropriate codes to the data file one after the other. The number of columns allocated to each code depends on the number of responses in the list:

- If the list contains between 1 and 9 responses, Quancept allocates one column to each response and assigns responses codes in the range 1 to 9.
- If the list contains between 10 and 99 responses, Quancept allocates two columns to each response and assigns codes in the range 01 to 99.
- If the list contains between 100 and 999 responses Quancept allocates three columns to each response and assigns codes in the range 001 to 999.

All data for a single iteration of a question must be held on the same card. This inevitably means that you will be restricted in the number of responses you can store for each question if you use this coding method.

In Quancept, each card has 73 columns available for data. If responses are coded using a single numeric code you can store 73 responses for one question on a single card. If the response list contains between 10 and 99 responses coded with two-digit codes, you can store 36 responses. If the response list contains between 100 and 999 responses coded with three-digit codes you can store 24 responses.

When you code data in this way Quancept insists that you specify the maximum number of responses that may be chosen from each multipunched list. The data map allocates sufficient columns to hold these responses, thus reducing the number of blank columns that are reserved for nothing.

In the following example there are six responses in the list so responses will be numbered 1, 2, and so on up to 6. The respondent may choose up to six answers so Quancept will allocate 6 columns to this question altogether.

```
mcodscheme 1
q1      ask 'Which of the following films have your children seen
        at the cinema recently?'
        resp mp 6 'Snow White' / 'The Lion King' / 'Dumbo' /
                '101 Dalmations' / 'Jungle Book' / 'Toy Story'
```

If the respondent chooses Snow White, The Lion King and Toy Story the data file will look like this:

1	2
-----0-----0	
0008901126	

Each column starting at column 8 contains a numeric code representing a quoted response: 1 (Snow White), 2 (The Lion King), 6 (Toy Story). Columns 11 to 13 are blank because the respondent's children did not see all the films.

Specified other, null, dk and ref

Under the standard format, *null*, *dk* and *ref* are coded as –, & and { respectively in the first column of a response field. Specified other is either coded according to its position in the response list or, if you have used *mapothzero*, as 0 in the first column of the field.

With type 1 coding, Quancept allocates one extra column at the start of the question's data field specifically for these responses, even if they do not appear in the current response list. Quancept uses the standard codes for *null*, *dk* and *ref*, but for specified other it behaves as if *mapothzero* were set and always codes specified other as a zero in this extra column. Three additional columns are still reserved at the end of the field for coding the open-ends. If we modify the example script so that it becomes:

```
mcodscheme 1
q1      ask 'Which of the following films have your children seen
        at the cinema recently?'
        resp mp 6 'Snow White' / 'The Lion King' / 'Dumbo' /
                '101 Dalmations' / 'Jungle Book' / 'Toy Story'
                other null dk
```

and the respondent chooses Dumbo, Toy Story and Bugs (via *other*), the record will be:

1	2
-----0-----0	
00089010306	

where the 0 in column 8 represents specified other with which Bugs was recorded for later coding.

Type '10' — on/off switches

With this type of coding you use just two characters that act like on/off switches indicating whether or not a response was chosen. One column is allocated to every response in the list and each of those columns will contain either an on or an off code. You can choose which characters you use; the example uses 1 if a response was chosen and 0 if it was not.

Specified other, *null*, *dk* and *ref* are coded into an extra column at the start of the question, as described for type 1 coding. The question is defined as:

```
mcodscheme '10'
q1      ask 'Which of the following films have your children seen
        at the cinema recently?'
        resp mp 6 'Snow White' / 'The Lion King' / 'Dumbo' /
              '101 Dalmations' / 'Jungle Book' / 'Toy Story'
              other null dk ref
```

and the respondent chooses Dumbo, Toy Story and Bugs (via *other*), the record will be:

1	2
-----0-----0	
00089010001001	

The string of seven codes in columns 8 to 14 represents the answer to the question. Code 0 in column 8 represents specified other (Bugs). Code 0 in columns 9 and 10 indicates that neither Snow White or The Lion King was chosen. Code 1 in column 11 indicates that response 3 (Dumbo) was chosen. Code 1 in column 14 represents Toy Story.

17.7 Inserting non-response data in the data file

Literal text strings and values of temporary variables, or both, may be inserted into the data for each respondent with **fix**. Each *fix* statement may insert a maximum of 72 characters.

User-defined text

Quick Reference

To insert a fixed text in the data, type:

CATI/CAPI/Web **label fix (nn) 'text' [force (n)]**

where *nn* is the number of columns required. If the *fix* statement is added to a Quancept script after interviewing has begun, use the keyword *force* to write the text at the end of the record.

The text must be enclosed in single quotes and the field width should be at least as long as the text it is to contain. Texts shorter than the field are left-justified in the field with the remaining columns left blank; texts longer than the field are truncated.

The text appears in the data file in the same relative position as the *fix* statement occupies in the script. You cannot tell Quancept which columns to use.

If you add a *fix* statement to the script after some interviewing has taken place, you may force the text to be written to the end of the record by placing the keyword *force* on the *fix* statement. For example:

```
jnum    fix (5) 'k1826'  force
```

☞ For further information on *force*, see section 17.1, Adding extra questions to the script.

Variables

Quick Reference

To copy the value of a temporary variable into the data file, type:

CATI/CAPI/Web **label fix (nn) variable [force (n)]**

mrInterview **label fix (nn) variable**

where *nn* is the number of columns required. If the *fix* statement is added to a Quancept script after interviewing has begun, use the keyword *force* to write the text at the end of the record.

For example:

```
set intvr = intname
iname    fix (10) intvr
```

Here, we are saving the interviewer's name (as reported by the keyword *intname*) in the variable *intvr*. Then *fix* inserts it in a 10-column field in the data record. You may use this type of *fix* statement with rotated response lists to copy the rotation figure into the data so that you test can whether the respondent's answer was influenced by the position of responses in the list.

-
- ☞ For further information about the use of rotated response lists, refer to section 5.7, Rotation and randomization of responses.
-

Text and variables together

Quick Reference

To insert a combination of text and variable values into the data file, type:

CATI/CAPI/Web *label fix (nn) 'text' +variable [force (n)]*

mrInterview *label fix (nn) 'text' +variable*

where *nn* is the number of columns required. If the *fix* statement is added to a Quancept script after interviewing has begun, use the keyword *force* to write the text at the end of the record.

You may also insert a combination of text and variable values into the data file. The example below copies the job number (J123) and the interviewer's name into a 15-character field in the data file at the point at which the *fix* statement appears in the script:

```
set intvr = intname
jobinfo  fix (15) 'J123'+intvr
```

You will notice that the combination of text and variable name follows the standards for question texts and displayed texts containing variable information — namely, that text is enclosed in single quotes with variable names separated from the text by a plus sign.

Other points to be aware of when using *fix* are:

- If you place *fix* in a loop, the text will be placed in the data file each time the statement is read.
- Variables on *fix* statements in loops are exactly the same as question names and must be subscripted if necessary.

Inserting open ends in the precoded data file

fix may be used to copy open-ended responses into the data. You might do this to copy the respondent's name and address directly into the data rather than placing it in the file of open-ended responses for later recoding. For example:

```

name      ask 'Please could I have your name?'
          resp coded(0) nodata
addr     ask 'And your address.@
INTERVIEWER: Do not enter town or postcode here'
          resp coded(0) nodata
town     ask 'Enter town name here'
          resp coded(0) nodata
PCODE    ask 'Enter postcode here'
          resp coded(0) nodata

comment Place name and address data at the end of the record
          newcard
rname    fix (50) name
          newcard
raddr    fix (70) addr
          newcard
rtown   fix (40) town
          newcard
rpcode  fix (20) pcode

```

In this example, you will notice that we have fixed each item on a new card. This enables us to edit the data file later to extract all cards containing name and address information in a format suitable for producing sticky labels or for reading into a direct mail program. By defining the original open-ended responses as *coded(0) nodata*, we have prevented columns being allocated in the data file where we did not want them, and we have also prevented the open ends from being written to the open-end text file.

☞ The labels of the *fix* statements count toward the limit of 1000 temporary variables per script.

17.8 Saving 'dirty' data



Quick Reference

To save data related to questions that were asked once but then became irrelevant after snapbacks changed the routing, type:

noclean

This statement affects the dat and tex files only; the drs and ddr files hold all data even if snapbacks make it irrelevant.

During an interview the interviewer may need to skip back to a previous question and change the original answer given because, for example, the respondent remembers trying a different product. Depending on how your script is written, backtracking through a script to re-ask questions or check responses, even if nothing is then changed, can affect your data. Backtracking and jumping around in a script is called **snapping**.

First, let's talk about how the interviewing program deals with snapping. As questions are answered the first time, a flag is set for each one, indicating that it is on the path through the interview. If the interviewing program encounters dummy questions, *fix* statements, or other variables whose values are set with *set* statements, it flags these as on the interview path too.

When an interviewer snaps from, say, q6 to q4, the interviewing program checks the position of q4 relative to q6. If q4 comes before q6, then the interviewing program returns to the **beginning of the script** and unsets all the flags. It then walks silently through the script, resetting the flags until it reaches q4; this is invisible to the interviewer. The interviewer may then accept the current answer to q4 or change it. Accepting it means that he/she may retrace the original path through the script; changing q4 may make this impossible if the script contains routing based on the answer to q4.

Once the interviewer has answered q4, he/she may decide to skip forward to his/her previous place in the script, that is, to q6. The interviewing program will continue walking through the script until it reaches q6 or an unanswered question, whichever comes first. The interviewing program will reach q6 either if the answer to q4 was not changed or if it was changed and the new answer did not affect any routing between q4 and q6. It will stop at an unanswered question if the answer to q4 is changed and the new answer invokes different routing before q6 is reached. For example:

```
q4    ask 'Which brands did you buy?'
      resp mp 'Brand A' / 'Brand B' / 'Brand C' null
      set brda = bit(q4/1)
      route (.not. brda) go q10
q5    ask ....
q6    ask ....
```

If the original answer to q4 is Brands A and B, the *route* statement is false, so the script continues on to questions 5 and 6. If the interviewer backtracks at q6, for example, and changes the answer to q4 to Brand B only, the *route* statement becomes true, so the next question to ask is q10, which has not yet been answered.

At the end of the interview, all questions and variables (for example, from *fix* or *set* statements) whose flags are set as ‘on the path’ are written out to the data file. Other variables which have lost their data due to changes in routing are not written out (for example, q5 and q6 in our example above). If you wish, you may force the interviewing program to write out this ‘dirty’ data as well by placing a **noclean** statement somewhere in the script.

When *noclean* is present anywhere in a script, all data collected during the interview will be written out, regardless of whether the statement was actually executed during the interview or whether it was bypassed with routing. Suppose that we had placed a *noclean* statement between the *route* statement and q5 in the example above. Whenever the response to q4 is not Brand A, the interviewing program skips straight to q10, missing out *noclean* and questions 5 to 9. However, because *noclean* is present in the script, any dirty data will still be written out, just as if the answer to q4 had been, say, Brand D and the interviewing program had then read the *noclean* statement before continuing with q5.

In our example as it stands, data for q4 and q10 will be written out, but with *noclean* data for questions 4, 5, 6 and 10 would be written out.

17.9 Set statements with snapbacks

When the interviewer snaps back during an interview and then returns to the original position in the interview, the interviewing program silently re-executes each statement in the script as it walks through it, so changing answers may have an effect on routing. This is true not just for *route*, but for *if*, *go* and *goto* as well. *set* statements are also critical, especially if you are using them arithmetically. Consider the script and the table below:

```

sheep    ask 'How many sheep do you own?'
        resp num 0 to 100
        set animals = animals + sheep
cows     ask 'How many cows do you own?'
        resp num 0 to 100
        set animals = animals + cows
pigs     ask 'How many pigs do you own?'
        resp num 0 to 100
        set animals = animals + pigs
all      fix(4) animals

```

The table below assumes that the interviewer asks all the questions shown in this example and then snaps back some time later to the sheep question. It shows, in the second column, the values of the variables as they are first entered or calculated. The third column shows their values after the snapback to the sheep question and the subsequent alteration of the number of pigs kept. The last column shows what the values should have been (we explain below how to get these values).

Variables	Before snap	After snap	Should be
animals	0	140	0
sheep	50	50	50
animals+sheep	50	190	50
cows	80	80	80
animals+cows	130	270	130
pigs	10	20	20
animals+pigs	140	290	150

The same thing happens if the interviewer snaps back to a point before the *set* statements and then either passes through them again keeping the original values or skips forward over them. You can verify this by displaying the number of animals after each *set* statement in the previous script.

To remedy situations of this kind, you should always set numeric variables to 0 or unset them before you use them. (Setting to zero is the preferred method as it ensures that the variable has a numeric value. Unsetting a variable gives it no data type at all.)

```

        set animals = 0           (or unset animals)
sheep    ask ....
        set animals = animals + sheep
cows    ask ....
        set animals = animals + cows
pigs    ask ....
        set animals = animals + pigs

```

-
- ☞ The same applies to dummy questions whose responses are assigned by statements in the script, especially if they are multipunched. With multipunched assignments, each assignment is additional to the current value of the question.
 - ☞ See also section 9.8, Testing for snapbacks, for information on *ifsnap*.
For further information on assignment with multipunched question variables, see section 12.4, Assigning responses to questions.
For information about *unset*, see section 12.13, The *unset* statement.
-

17.10 Private status codes to collect accounting data



Quick Reference

Use **privsig** to define your own status codes for use in the qtip accounting (act) file:

privsig *number*

where *n* is any number that you choose between 0 and 99.

Private status codes are one method of keeping track of how data was collected. Whenever the interviewing program encounters a private status code in the script, it remembers that code, either until it is overwritten by another code later in the script or until the end of the interview. When the interview finishes for whatever reason, the private status code is written out with other accounting information to the interviewing program accounting file (*project.act*). By setting different status codes at the crucial points in the script, you are able to retain information for reporting on the different routes taken through the script.

Here is an outline script to illustrate how *privsig* might be used:

```
comment First section - set accounting flag to 1 when completed
q1a      ....
q1b      ....
    privsig 1
    route (....) go ex
comment Second section - set accounting flag to 2 when completed
q2a      ....
q2b      ....
    privsig 2
    route (....) go ex
comment Third section - set accounting flag to 3 when completed
q3a      ....
q3b      ....
    privsig 3
    route (....) go ex
ex      end
```

In this example, any interview which reaches the statement labeled 'ex' will be flagged as a completed interview in the accounting file. Without the *privsig* statements, you would have no easy way of checking how the interview reached that stage, since any analysis of the accounting file will treat all interviews which reach the ex label as identical. For example, without looking at the name of the last question asked, you will not know whether the respondent answered all questions or was routed to the end of the script by the *route* statement in section 1.

The use of *privsig* enables you to place your own accounting code in the accounting file to mark more precisely which sections of the script were answered. If we assume that there is no other routing in the script other than the three statements shown, you will know that any respondent who has a private accounting code of 0 (the default) did not complete the first section. Any respondent with a private accounting code of 1 completed the first section but did not complete the second. Respondents whose accounting flag is 2 completed sections 1 and 2 but did not complete section 3. Finally, respondents whose accounting code is 3 completed all three sections.

17.11 Suppressing blank cards



Quick Reference

If you are using *col* or *column* and want to prevent blank cards from being written to the data file, type:

noblank

on a line somewhere before the first occurrence of either *col* or *column*.

For example, if you are using *col* or *column* is used to write data to cards 1 and 7 only, you can use *noblank* to prevent cards 2 to 6 being written with just a serial number and card type on them.

-
- ❖ If the script contains questions with open-ended responses that will be coded later on, you will need to consider carefully whether using *noblank* is a good idea. If a card contains nothing but questions with open-ended responses, *noblank* treats the card as blank even though it will contain data once the open-ends have been coded. If you later use qccon to combine the coded open-ends held in the acd file with the precoded data from the ddt file, problems may arise if the coded open-ends need to be merged into cards that were suppressed by the *noblank* keyword.
-

17.12 Sample script

Script A



This script runs under CATI only and is designed to illustrate the keywords controlling the layout and coding of your data. Comments in the script state the columns in which the data will appear in the dat file.

```
comment Sample Script A for Quancept & mrInterview Scriptwriter's Manual  
comment Chapter 17, Data format (CATI only)
```

```
films define 'Snow White' / 'The Lion King' / 'Dumbo' /  
           '101 Dalmations' / 'Jungle Book' / 'Toy Story'
```

```
comment Write multipunches out as strings of 0's and 1's.
```

```
comment 1 means the corresponding response was chosen;
```

```
comment 0 means it was not
```

```
    mcodscheme '10'
```

```
comment Card 1, columns 8-14
```

```
q1      ask 'Which films have your children seen at the  
         cinema recently?'  
        resp mp films
```

```
comment Card 1, columns 20-23
```

```
comment Override multipunch format for current question so that  
comment responses are assigned numeric codes according to their  
comment positions in the response list. Also, place data in  
comment column 20 onwards of card 1
```

```
q2      ask 'And of those films, which one or ones was the most  
         popular (up to 3 choices allowed).'  
        resp mp 3 codscheme 1 q1 in films col 120
```

```
comment Data for this question goes in the next free column
```

```
comment after q1; that is card 1, column 15. Columns 16-18
```

```
comment are reserved for coding specified other responses
```

```
q3      ask 'If you could buy just one children's film on video  
         which one would it be?'
```

```
        resp sp films other
```

```
comment Code the next block of questions in card 1, columns  
comment 30 onwards. q4 is coded in column 30, q5 is coded in  
comment columns 31-36, and columns 37-39 are reserved for  
comment coding specified other responses  
comment column 30
```

```
q4      ask 'How often do you take your children to the cinema?'  
        resp sp 'Once a week' / 'Several times a month' /  
               'Once a month' / 'Several times a year' /  
               'Once a year' / 'Less than once a year'
```

```
q5      ask 'And on these visits, do you buy anything from the
in-cinema shops?'
        resp mp 'Sweets' / 'Drinks' / 'Popcorn' /
              'Hot dogs/burgers' / 'Nothing ^s' other

comment End of block, continue coding on next free column
comment after q3. howmany is coded in column 19; ages is
comment coded in columns 24-29. Columns 30 to 39 are
comment already used so area and occup are coded into
comment columns 40 and 41-43 respectively
comment column 100

howmany ask 'How many children do you have under the age of 13?'
        resp num 1 to 9
ages    ask 'And how old are they?'
        resp mp 'Under 1 year' / '1 to 3 years' / '4 or 5 years' /
              '6 to 10 years' / '11 to 12 years'
area    ask 'Where do you live?'
        resp sp 'City' / 'Large town' / 'Small town' / 'Village' /
              'Out in the country'
occup   ask 'What is the occupation of the main wage earner?'
        resp coded
end
```

Script B



This script runs under Quancept CATI, Quancept CAPI and Quancept Web and is the final script in a series of three. The first script (called ch17b1) is the basic script, the second (ch17b2) is with *force(1)* and the third (ch17b3) is with *force(1)* and *force(2)*. To see the effect of *force*, run all three scripts in order.

```
comment Sample Script B for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 17, Data format (CATI, CAPI and Web only)

chocs   define (20)  'plain chocolate' / 'milk chocolate' /
          'white chocolate' / 'chocolate with nuts' /
          'chocolate with fruit and nuts' /
          'chocolate with nougat'

gender  ask 'INTERVIEWER: Is the respondent ...'
        resp sp 'male' / 'female'
age     ask 'Which age group do you belong to?'
        resp sp '18 and under' / '19 to 25' / '26 to 35' /
              '36 to 45' / '46 to 55' / '56 to 65' /
              '66 and over'
school  ask 'What level of education have you achieved?'
        resp sp 'O level/GCSE' / 'A level' / 'HNC' / 'HND' /
              'degree' / 'post graduate'
```

```

numwage ask 'How many people in your household receive a wage,
salary or some type of benefit?'
      resp num 1 to 10
income   ask 'What is the approximate total income for your
household?'
      resp sp '10,000 or less' / '10,001-20,000' /
             '20,001-30,000' / '30,001-40,000' /
             '40,001-50,000' / '50,001 or more'

comment CATI only: Accounting flag for don't eat sweets
privsig 1

sweets   ask 'Do you eat sweets?'
      resp sp 'yes' / 'no' go bye
prefer   ask 'Which sort of sweets do you prefer?'
      resp sp 'boiled fruit sweets' / 'soft fruit sweets' /
             'hard mints' / 'soft mints' / 'chewing gum' /
             'toffee' / ('chocolate' / 'chocolate snack bars' /
             'chocolate cookies') go ctrl
why      ask 'Why do you prefer '+prefer+' to chocolate?'
      resp coded force (1)

comment CATI only: Accounting flag for eat sweets but never buy chocolate
privsig 2
buychoc  ask 'Do you ever buy chocolate?'
      resp sp 'yes' / 'no' go bye
ctrl     continue
cpref    ask 'Which sort of chocolate do you prefer?'
      resp sp chocs

comment CATI only: Accounting flag for didn't try plain & milk slice
privsig 3
tried    ask 'Have you tried the new Plain and Milk Slice
chocolate bars?'
      resp sp 'yes' / 'no' go sample force (1)

comment CATI only: Accounting flag for didn't try chocolate & mint chip wafers
tried2x  privsig 4
tried2   ask 'Have you tried Chocolate and Mint chip wafers?'
      resp sp 'yes' / 'no' go sample force (2)
taste    ask 'What did you think of the mint flavor?'
      resp coded force (2)

size     ask 'What did you think of the size of bar it comes in?@'
INTERVIEWER: Press respondent to say why they chose a particular
answer.'
      resp sp 'too big ^o' / 'just right ^o' / 'too small ^o'
              other force (1)

```

Quancept and mrInterview Scriptwriter's Manual

```
sample    ask 'Would you like a free sample of this product to try?'
          resp sp 'yes' / 'no' go bye nodata
name     ask 'Please could I have your name?'
          resp coded nodata
addr     ask 'And your address'
          resp coded nodata

comment Place name and address data at the end of the record
          newcard
rname    fix (50) name
          newcard
raddr    fix (70) addr

comment CATI only: Accounting flag for answered all questions
          privsig 5
bye      display 'Thank you for your help.'
comment CATI/CAPI only: Copy interviewer's name into the data
          set intvr = intname
iname   fix (10) intvr
          end
```

18 Report files

You can write out the data that is collected during interviewing to report files in a predefined format. Report writing statements in the script enable you to create up to four different reports per script and to write selected texts and responses to those files in any format you want.

The statements described in this chapter are:

rfclose	close a report file
rfopen	open a report file
rfprint	write to a report file

18.1 Filenames



Quick Reference



In Quancept CATI, report files are called:

`REP[1-4]int_num`

where `int_num` is the interviewer's Quancept user number as defined in the file in `/ip/users`. For example, REP454321.

In Quancept CAPI, report files are called:

`r[1-4]i0`

or have a user-defined name.

In CATI, the data written to each report is subdivided by interviewer so that for each type of report, you will have as many files as there are interviewers working on the project. The report filenames are therefore made up of the report name and the interviewer's Quancept user number (as a five-digit number) as it is defined in the file `/ip/users`. Therefore, if Barbara's Quancept user number is 4 and the script writes to report file 2, Barbara's file will be called REP200004. It is then the supervisor's job to combine the various files that make up a report (for example, all the REP2 files) at the end of an interviewing session.

In CAPI, each interviewer works on a separate PC so the possibility of two interviewers wanting to write to the same file at the same time does not arise. The interviewer ID is therefore always i0. If the survey data is uploaded onto the Quancept CAPI server, it is then the supervisor's responsibility to merge all report files of the same name into a single report.

You also have the option of creating report files with names of your choice rather than using the default names.

18.2 Opening a report file



Quick Reference

To open a report file for writing, type:

CATI: **callfunc('rfopen',file_num)**

CAPI: **rfopen file_num[, 'filename']**

where *file_num* is a number or a numeric variable defining the report file to be opened. In CATI, this number must be between 1 and 4. In CAPI, *file_num* may be any number greater than zero, but if it is greater than 4 a *filename* must also be defined.

In order to write to a file, you must first open it using the **rfopen** callfunc or keyword. If the number you include in the *rfopen* statement is that of a report file that already exists, the file is opened ready for Quancept to append the new entries. If the file does not exist, it is created.

- If you open a file whose file number is in the range 1 to 4, Quancept will create a file called r1i0, r2i0, r3i0 or r4i0 as appropriate. If you open a file with a number greater than 4, Quancept will open a file with the name given on the *rfopen* statement. For example:

```
rfopen 6, 'names'
```

creates a report file called names whose reference number is 6.

When referring to report files on *rfprint* and *rfclose* statements always use the file number only.

18.3 Closing a report file



Quick Reference

To close a report file, type:

CATI: **callfunc('rfclose',file_num)**

CAPI: **rfclose file_num**

where *file_num* identifies the file to be closed.

When you have finished with a file, close it with an *rfclose* statement. If you want to write several lines for each respondent, you need only open the file before the first line is written and close it after the last line has been written — there is no need to open and close it for each entry.

18.4 Writing to a report file



Quick Reference

To write to a report file, type:

CATI: **callfunc('rfprint',file_num,format,[var1,... var18])**

CAPI: **rfprint file_num,format,[var1,... var18]**

where *file_num* is a number or a numeric variable identifying the report file, *format* is a text variable that defines the output format, and *var1* to *var18* are the names of any variables to be printed in the text.

In Quancept CATI, the output format must be defined in a variable using a *set* statement. As it is a text, it must be enclosed in single quotes. For example, to print a respondent's name and age to a CATI report file, you might write:

```
name    ask 'What is your name?'
       resp coded nodata
age     ask 'How old are you?'
       resp num 18+
comment Define format of line to be written
        set fmt='%s is %ld years old'
comment Use file number 1
        callfunc('rfopen',1)
        callfunc('rfprint',1,fmt,name,age)
        callfunc('rfclose',1)
```

In Quancept CAPI, the output format may be defined in a variable using a *substitute* statement, or it may be placed in the *rfprint* statement itself. As it is a text, the format string must be enclosed in single quotes. For example, to print a respondent's name and age to a CAPI report file, you might write:

```
name    ask 'What is your name?'
       resp coded nodata
age     ask 'How old are you?'
       resp num 18+
comment Define format of line to be written
        substitute fmt=''%s is %ld years old''
comment Use file number 1
        rfopen 1
        rfprint,1,fmt,name,age
        rfclose 1
```

In these examples, you can see that the output for printing to the report file is defined in a text variable. The variable information to be printed to the file is represented by a % sign followed by one or two letters. These indicate the type of information being written to the file — such as a decimal integer or a text string — and are known as **conversion characters**. In the example, the respondent's name is a text string and his/her age is a number. So the conversion character %s, specifying a text string format, is used to write the respondent's name to the report file. The character %ld, which specifies a long decimal number, is used to write his or her age to the file.

Defining the output



Quick Reference



To start a new line in a report file, include the following in the format string:

CATI: {#0a}

CAPI: \n

You define the output that will be written to a report file in a text variable. This definition can include text to be printed to the file as well as conversion characters for the variables to be printed.

Quancept does not start a new line in a report file unless you tell it to do so. Instead, it prints the texts from all *rfprint* statements in the script as one continuous line. To force a new line in the report file, you must place a special code at the point at which you want the new line to start. For example:

```
name      ask 'What is your name?'
          resp coded nodata
comment  For CATI
          set fmt1 = '%s{#0a}'
comment  For CAPI:
comment substitute fmt1 = '%s\n'
addr      ask 'What is your full postal address?'
          resp coded nodata
comment CATI syntax
          callfunc('rfprint',2,fmt1,name)
          callfunc('rfprint',2,fmt1,addr)
comment CAPI syntax
          rfprint 2,fmt1,name
          rfprint 2,fmt1,addr
```

-
- ☞ In versions of Quancept CATI earlier than v7e.4, each *rfprint* statement prints up to 200 characters.
 - ☞ For information on conversion characters, see 'Using conversion characters in an output definition' below.
-

Using conversion characters in an output definition



You can use the following conversion characters in an output definition to print the information held in a variable to a report file.

-
- ❖ If you have used printf in the C programming language, these conversion characters and their modifications may already be familiar to you.
-

Type	To print
%d	A signed decimal integer in as many columns as there are digits in the number. If an integer has a negative value, its sign occupies one column.
%ld	An integer in as many columns as there are digits in the number (the letters stand for 'long decimal'); for example, 123 takes three columns, while 12345 takes five columns.
%u	An unsigned decimal integer in as many columns as there are digits in the number.
%o	An unsigned octal integer in as many columns as there are digits in the number.
%x	An unsigned hexadecimal integer in as many columns as there are digits in the number using the digits 0 to 9 and the lower-case letters a, b, c, d, e and f.
%X	An unsigned hexadecimal integer in as many columns as there are digits in the number using digits 0 to 9 and the upper-case letters A, B, C, D, E and F.
%f	A real value in as many columns as required.
%s	A string (text) left-justified in the field.
%%	A % sign (note that there are no spaces between the two percent signs).

Defining the field size

You can modify the conversion character and specify the size of the field to be written to. The syntax for specifying the field size is:

`%Min_ColsChar`

where *Min_Cols* defines the minimum number of columns in the field, and *Char* is any of the conversion characters listed above.

If you specify a minimum number of columns for the field and the number or text string being written has fewer digits or characters than the number of columns available, the information is right-justified in the field and padded on the left with blanks. If there are more digits or characters than the minimum number of columns, the first one is displayed in the leftmost column of the field, and the field is expanded to the right to accommodate the additional digits/characters.

For example:

5	printed with	%3ld	is	5
15		%3ld		15
150		%3ld		150
1500		%3ld		1500

Right-justifying numeric values

In the case of numeric conversion characters, you can also specify that numbers are right-justified in the field and padded on the left with zeroes using the following syntax:

%0Min_ColsChar

where *Char* is the d, ld, u, o, x, X or f character. For example:

5	printed with	%03ld	is	005
15		%03ld		015
150		%03ld		150
1500		%03ld		1500

Specifying the number of digits or characters to write

You can specify the minimum number of digits or maximum number of characters to be written to a field by typing:

%.NChar

where *N* is the minimum number of digits or maximum number of characters to be written, and *Char* is the d, ld, u, o, x, X or s conversion character.

If you specify a minimum number of digits to be written to the field, numbers with fewer than *N* digits are expanded to *N* digits by padding them on the left with zeroes. If a number has more than *N* digits, the field is expanded to the right. For example:

5	printed with	%.2d	is	05
15		%.2d		15
150		%.2d		150
1500		%.2d		1500

In the case of %s, the expression %.Ns specifies the maximum number of characters to be written to the field. Characters are right-justified in the field and padded on the left with blanks. Any text string longer than *N* is truncated. For example:

Mary	printed with	%.10s	is	Mary
Mary Farmer	printed with	%.10s	is	Mary Farme

You can also use this syntax with the %f conversion character to specify the number of digits to be printed after the decimal point. In the expression %.Nf, *N* specifies the number of digits to be printed after the decimal point.

Specifying the column width and the number of digits or characters to write

You may specify both the minimum number of columns and the number of digits or characters by typing:

`%Min_Cols.NChar`

If there are fewer digits or characters than *Min_Cols* (including numbers that have been padded with zeroes), the number or text is right-justified in the field and padded on the left with blanks. For example:

5	printed with	%3.2d	is	05
15		%3.2d		15
150		%3.2d		150
1500		%3.2d		1500

Applying left-justification to a field

When you write information to a report file using an unmodified conversion character, the information is always left-justified in the field. If you modify a conversion character in any of the ways described above, the information is right-justified in the field by default. You can specify that information is left-justified in the field by applying a ‘-’ flag to the expression. This flag must appear before the conversion character or any modifications you have applied to it. For example:

5	printed with	%-3.2d	is	05
15		%-3.2d		15
150		%-3.2d		150
1500		%-3.2d		1500

Conversion characters and variable types

The following table summarizes conversion characters and the variable types with which you may use them:

Variable type	Conversion characters	Prints as	Notes
sp	%d, %ld, %u, %o, %x, %X	Integer	The number is the response's position in the response list.
mp	%s	Text	The text is a string of integers separated by commas showing the responses' positions in the response list.
num	%d, %ld, %u, %o, %x, %X	Integer	

Variable type	Conversion characters	Prints as	Notes
real	%s, %f	Real	With %s, the real is converted to a string and printed with at least one digit before the decimal point and three after it.
			For greater precision, use %f which allows you to specify a field width and the number of columns to appear after the decimal point.
coded	%s	Text	
list	%d	Integer	
dbase	%d, %ld, %u, %o, %x, %X	Integer	
logical	%d, %ld, %u, %o, %x, %X	Integer	0=false, 1=true
text	%s	Text	
null/dk/ref	%s	Text	
specified other	%s	Text	

Example usage



Let's look at some examples.



The first is a Quancept CATI script that saves names and addresses in a file that could then be manipulated to produce sticky labels. The script for Quancept CAPI is the same apart from the differing syntax for *rfopen*, *rfprint* and *rfclose*, and the newline notation.

```
comment Use report file 2
      callfunc('rfopen',2)
name   ask 'What is your name?'
      resp coded nodata
      set fmt1 = '%s{\#0a'
      callfunc('rfprint',2,fmt1,name)
addr   ask 'What is your full postal address?'
      resp coded nodata
      callfunc('rfprint',2,fmt1,addr)
      callfunc('rfclose',2)
```

In CATI, this will produce REP2int_ID files containing two lines per respondent as follows:

```
Violet Fisher
Ace Removals|17 High St|Smalltown GU32
Michael Brown
137 High Street|New Town|Scotland
```

The different sections of the address are listed consecutively on the same line regardless of how they are entered by the interviewer. The | symbols represent points at which the interviewer pressed the Return key to start a new line. You can edit the file to replace these characters with new line characters and maybe insert a blank line between each respondent.

In CAPI this example will create files called r2i0. If the interviewer types CTRL+ENTER to start a new line for each address line, the files will contain varying numbers of lines for each respondent:

```
Violet Fisher
Ace Removals
17 High St
Smalltown GU32
Michael Brown
137 High Street
New Town
Scotland
```

The next example is a Quancept CAPI script that illustrates how the interviewing program deals with single-punched and numeric information. The CATI script is the same apart from the different syntax for *rfopen*, *rfprint* and *rfclose* and the newline notation.

```
brd      ask 'Which brand did you buy?'
        resp sp 'Suds' / 'Bubbles' / 'Gleam' / 'Sparkle' /
              'Fresh and Clean' other null go ctn
              dk go ctn ref go ctn
cost     ask 'How much did you pay?'
        resp real 0 to 4.00 null go ctn dk go ctn ref go ctn
        set rid = respondent
        substitute fmt '%05ld : Brand %ld : Cost %s\n'
        rfopen 4
        rfprint 4,fmt,rid,brd,cost
        rfclose 4
ctn      continue
```

This script is designed to produce a list of respondents showing their respondent ID, the brand bought and the price paid. Anyone giving a *null*, *dk* or *ref* answer to either question is excluded from the report. Since we want the output to be printed in columns, we have used numbers with %ld to control the width of the first two fields and the positions of the numbers in those fields. The respondent ID will occupy five columns and will, if necessary, be padded on the left with zeroes as in the data file. The brand code is always a single digit so %ld by itself is sufficient. The cost is a real number and must be printed with the %s notation. It will be left-justified after the last colon in the format statement. The three columns are separated by colons.

Quancept and mrlInterview Scriptwriter's Manual

The report generated might look like this:

```
00001 : Brand 3 : Cost 0.750  
00007 : Brand 2 : Cost 0.810  
00010 : Brand 5 : Cost 1.150  
00011 : Brand 2 : Cost 0.82
```

18.5 Sample script

Sample scripts A and B perform identical tasks. Script A uses CATI syntax whereas script B uses CAPI syntax.

Script A

```
comment Sample script A for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 18, Report files (CATI syntax)

brd      ask 'Which brand did you buy?'
         resp sp 'Suds' / 'Bubbles' / 'Gleam' / 'Sparkle' /
                  'Fresh and Clean' other null go ctn
                  dk go ctn ref go ctn
cost     ask 'How much did you pay?'
         resp real 0 to 4.00 null go ctn dk go ctn ref go ctn
opin     ask 'What did you think of '+brd+ '?'
         resp coded
         set rid = respondent

comment Open report file 4 and define entry formats for cost
comment and opinion entries
         callfunc('rfopen',4)
         set cfmt = '%05ld : Brand %ld : Cost %s{#0a}'
         set ofmt = '%s{#0a}'
         callfunc('rfprint',4,cfmt,rid,brd,cost)
         callfunc('rfprint',4,ofmt,opin)

comment Close report file
         callfunc('rfclose',4)
ctn      continue

comment Open report file 2 and define the format for this file
         callfunc('rfopen',2)
         set fmt1 = '%s{#0a}'
name     ask 'What is your name?'
         resp coded nodata

comment Write name to report file 2
         callfunc('rfprint',2,fmt1,name)
addr     ask 'What is your full postal address?'
         resp coded nodata

comment Write address to report file 2 then close it
         callfunc('rfprint',2,fmt1,addr)
         callfunc('rfclose',2)
end
```

Script B

```
comment Sample script B for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 18, Report files (CAPI syntax)

brd      ask 'Which brand did you buy?'
         resp sp 'Suds' / 'Bubbles' / 'Gleam' / 'Sparkle' /
                  'Fresh and Clean' null go ctn
                  dk go ctn  ref go ctn
cost     ask 'How much did you pay?'
         resp real 0 to 4.00 null go ctn  dk go ctn  ref go ctn
opin    ask 'What did you think of '+brd+'?'
         resp coded
         set rid = respondent

comment Open report file 4 and define entry formats for cost
comment and opinion entries
         rfopen 4
         substitute cfmt '%05ld : Brand %ld : Cost %s\n'
         substitute ofmt '%s\n'
         rfprint 4,cfmt,rid,brd,cost
         rfprint 4,ofmt,opin

comment Close report file
         rfclose 4
ctn      continue

comment Open report file 2 and define the format for this file
         rfopen 2
         substitute fmt1 '%s\n'
name     ask 'What is your name?'
         resp coded nodata

comment Write name to report file 2
         rfprint 2,fmt1,name
addr     ask 'What is your full postal address?'
         resp coded nodata

comment Write address to report file 2 then close it
         rfprint 2,fmt1,addr
         rfclose 2
         end
```

19 Very long response lists in CATI (response list databases)

In some types of studies you may have very long response lists which are really too large to define in the normal way. To deal with this situation, Quancept CATI and Quancept CAPI have a database response type which allows you to set up the response list as an external file which the interviewing program can read when necessary. This external file is known as a *response list database*.

-
- ❖ Response list databases are stored in a ‘flat file’ format. If you are using Quancept CATI, you can also store and manipulate records using relational database software from the Free Software Foundation.
 - ☞ See chapter 21, ‘GNU GDBM relational databases in Quancept CATI’, and chapter 20, ‘ODBC databases in Quancept Web and mrInterview’, for more information.
-

Besides allowing you to handle responses neatly and with minimal memory usage, response list databases let you:

- Display responses (records) on the screen and prompt the interviewer to select the item chosen by the respondent. The interviewer may scroll backwards and forwards through the list before making a selection, and may filter the list so that only responses containing a certain text are displayed.
- Create a hidden look-up table from which whole records or fields from records can be selected. Selections may be based on responses given by the respondent or may be controlled entirely by parameters defined by the script writer, so there is never any need to display the response list database unless you wish to do so.
- Store selected records from one or more response list databases for manipulation during the interview. Any number of records may be held during an interview and information may be selected from those records in any order, as required by the script.

The statements described in this chapter are:

dbrecno	get the record number of the selected database record
nslrec	select a record from a response list database
resp dbase	display the response list database and store the response selected
resp multidbase	allow selection of more than one response from a database response list
selfld	select a field from a record
selrec	select a record from a response list database

This chapter also explains how to use the **shm**, **rmshm** and **listshm** programs for sharing CATI response list databases.

19.1 Response list database format



A response list database is a text file that contains one record per line. When you use a database as the response list to a question, the interviewing program treats each record as a separate response. Depending on the way shared memory is configured on your system, a database may usually contain up to 29999 records. In Quancept CATI each record may be a maximum of 2047 characters long; in Quancept CAPI the maximum length is 80 characters.

» Screen restrictions mean that qtip displays only the first 72 characters of each record.

Records may be divided into fields by separating the information in each field by a semicolon. For example:

```
Volkswagen;Scirocco GT II;6;0265  
Volkswagen;Scirocco Scala;6;0265  
Volkswagen;Corrado 16V;6;0641  
Volkswagen;Corrado G60;6;0641
```

These records contain four fields each. The first is the name of the manufacturer, the second is the car's make and model, the third is the insurance group, and the fourth is a reference code which we will talk more about later on.

If you need to enter a blank field in the middle of a record, type a space in between two semicolons. If the blank field is at the end of the record, type a blank space and then a semicolon (normally, records need not end with semicolons). If you type a blank field in any other way, the interviewing program will either ignore it or fail completely. If the interviewing program ignores an incorrectly specified blank field, the script will still run but it will not work as you expected, since any fields after the blank field will be shifted to the left.

19.2 Where to store response list databases

Databases you can store response list databases in the project directories of the projects that use them, or in a location of your choice.

If you store the database in the project directory, then all you need to do is enter its name in questionnaire script. Similarly, if you place the database in a different directory but you give the absolute pathname of the database in the questionnaire script, the interviewing program will find it.

If you store databases elsewhere than in the project directory, either in a central location or in a number of different directories, all scriptwriters and interviewers must have the path or paths to the databases named in the QCSHMPATH environment variable. This variable is like standard Unix PATH variables, which means that:

- Its value can be a string of one or more pathnames separated by colons.
- The pathname '.' or a blank entry represents the current directory.

- Directories are searched for databases in the order they appear in the path string. If the database is not found in one directory, the next path in the string is searched, and so on until either the database is found or the end of the string is reached. Once a database has been found, no further searching takes place.
- If the database cannot be found in any of the directories named with QCSHMPATH (or the variable is not defined), qtip looks for the database in the current directory.

If the database is named in the questionnaire script using an absolute pathname (one that starts with /) then qtip searches for the database in that directory only, regardless of QCSHMPATH.

-
- » On Unix systems, response list databases must also be placed in shared memory with the program *shm* as described in section 19.7, Making the response list database available to interviewers.
-

19.3 Displaying the response list database



Quick Reference



To display a database as a response list type:

```
label  ask 'question text'
      resp dbase('dbase_name') [null] [dk] [ref]
```

where *dbase_name* is either just the name of the database or an absolute path for the database.

As far as the script is concerned, a response list database is just another type of response list. It is named on a *resp* statement using the special response type **dbase** and is displayed during an interview just like any other response list. The respondent may select only one record and the data is saved as a numeric value in the dat file. The numeric value and the response text are written to the drs file. The text is assigned to the question variable so that it may be displayed or otherwise manipulated during the rest of the script.

For example, if the first three records in our database are:

```
Ford;Escort 1.3;3;0123
Ford;Escort 1.4;4;0123
Ford;Escort 1.6;4;0270
```

and the respondent has a Ford Escort 1.3, *dbase* will place the number 00001 in the data and will assign the complete record text to the question variable. We will explain later how you can use other keywords to extract information such as the make and model from a particular field in the record.

Here is a simple script designed to call up a response list database of cars:

```
cartype ask 'Which make/model of car do you own?'
        resp dbase('carsdb')
new      ask 'Did you buy your '+cartype+' new or was it second hand?'
        resp sp 'new' / 'second hand'
```

During the interview, the question text is displayed as usual and the first ten records in the database are listed on separate lines below it. Since most response list databases are quite long, the interviewing program provides facilities for the interviewer to scroll up and down through the database and to apply filters so that only records satisfying the filter condition are displayed.

☞ For further details of filtering database response lists, see section 33.10, Selecting answers from a response list database.

☎ The CATI interviewing program gives each record in the database a number and displays this at the start of the record. The interviewer selects a record by typing this number. The record text is assigned to the question variable, and the line number of the record in the database as a whole is placed in the data.

The interviewing program allows only one selection to be made from a *resp dbase* database at a time. If you want the interviewer to select more than one item you will need to place the question in a loop.

💻 The CAPI interviewing program has a graphical user interface so does not normally allocate record numbers. Instead it simply lists the contents of the database in alphabetical order on the screen. The interviewer clicks on a record to select it, and the line number of the record in the database as a whole is placed in the data. The exception is when the interviewer is using the keyboard interface, in which case record numbers are displayed and the interviewer selects a response by typing its number.

The interviewing program allows only one selection to be made from a *resp dbase* database at a time. If you want interviewers to be able to select more than one response at a time, display the database using *resp multidbase* rather than *resp dbase*, as described in the following section.

All response list database questions have five columns allocated to them in the data file. Line numbers of fewer than five digits are right-justified in the field and padded on the left with zeros. So, if the respondent owns a Scirocco Scala which is line number 24 in the database, the interviewing program will place the code 00024 in the data file. If this is the only text in the record, the interviewing program will display the next question as 'Did you buy your Scirocco Scala new or was it second hand?'

-
- ❖ Because data values are assigned based on responses' positions in the database file, altering the file once interviewing has begun may affect the validity of your data. If you need to add records to the database, you should add them at the end of the file. Deleting records is not recommended.
-

If the respondent chooses *null*, *dk* or *ref*, these are coded using the standard notation in the first column of the first field for the question and the remaining columns in that field are left blank.

The record that the respondent chooses becomes the current or active record for that database, and fields may be extracted from it individually with *selfld* as described below. If your script uses more than one response list database, then you will have one current record for each database. (Note also that any record selected by a *selrec* statement in the script becomes the current record for that database).

- ☎ Quancept CATI can display up to 50 databases with *resp dbase*. However, you may find that the maximum for your system is lower due to your system's hardware and operating system configuration.

All Unix operating systems have an upper limit on the number of shared memory segments that can be used at one time. They also have a limit on the number of shared memory segments that can be attached to one process. These limits can also be configured on individual systems.

Running a Quancept script always uses one shared memory segment, and each database shared using shm also uses one segment. The maximum number of databases you can access in a script is therefore one less than the maximum number of attached shared memory systems your system allows per process. For example, on a system with a limit of 32, you cannot access more than 31 databases in a single script. If you are running interviews on a number of projects that use databases, the limit per script will be further reduced.

Allowing multiple selections from a database



Quick Reference

To allow selection of more than one response from a database response list, type:

```
for variable =1 to max_resps
label      ask 'question_text'
            resp multidbase ('dbase_name') [null] [dk] [ref]
next
```

where *max_resps* is the maximum number of responses that can be selected and is in the range one to fourteen, and *dbase_name* is either just the name of the database or an absolute path for the database.

Unlike other *for* statements, the *for* statement in this situation does not repeat the question a fixed number of times. Instead, it determines the number of responses that may be selected from the database. Once the interviewer leaves the database question, control of the interview passes out of the loop to the statement immediately after *next*.

For example:

```
comment Display the database and allow up to 7 items to be chosen
comment from it.
11      for i = 1 to 7
extras    ask 'Which of the following extras does your car have?'
            resp multidbase('extras') null
next
```

Here, the records in the extras database will be displayed in alphabetical order. The interviewer selects up to seven responses by clicking on each one in turn, or clicks the No answer button. The line numbers of these records in the database are written to the data files. If the respondent chooses fewer than the maximum number of responses allowed, the columns for those responses contain the code 00000.

❖ Do not nest *multidbase* questions.

The CAPI keyboard interface does not allow multiple selections from *multidbase* questions.

19.4 Saving the record number in a variable



Quick Reference

To save the database record number of the respondent's answer in a variable, type:

```
set variable = dbrecno(qname)
```

where *qname* is the name of the database question.

When a record is selected from the database, it is coded in the data file using the record's line number in the database. Sometimes it can be useful to have this record number available in the script. If we extend the cars example above, we might write:

```
cartype  ask 'Which make/model of car do you own?'
            resp dbase('carsdb')
            set rec = dbrecno(cartype)
d1        display cartype+' is record number '+rec
```

In this example, cartype will place the number of the selected database record in the data file, but its value when displayed is the complete text of the selected record as it appears in the database. To allow the number of the selected database record to be displayed, it must be set into a variable using *dbrecno*.

« In Quancept CAPI, *dbrecno* can be used with *resp dbase* but not with *resp multidbase*.

19.5 Selecting a field from a record



Quick Reference



To copy a field from a selected database record into a variable, type:

```
set variable = selfld(db_qname,field_num)
```

where *db_qname* is the name of the database question at which the record was selected, or the name of the variable which stores a record selected with *selrec*, and *field_num* is the position of the field you want to copy.

If a record is broken down into fields, you may wish to pull out information from a particular field. This data is copied into a temporary variable and may be displayed in texts, used in routing statements, and so on.

For example, suppose the cars database contains a unique car code in the fifth field of the record:

```
Ford;Escort 1.3;3;0123;1001
Ford;Escort 1.4;4;0123;1002
Ford;Escort 1.6;4;0270;1003
Volkswagen;Scirocco Scala;6;0265;4002
Volkswagen;Corrado 16V;6;0641;4003
Volkswagen;Corrado G60;6;0641;4004
```

You can retrieve this code from the record which the respondent selects and code it into the data file instead of using the record's line number as before. (Alternatively, you could use the group code in field 4 as a key to selecting records from a second database which you do not wish to display on the screen. There is an example of how to do this in the notes on *selrec* below.)

Now let us look at the script which will place the car codes in the data instead of their line numbers.

```
comment Use nodata since we don't want to store the record number
cartype ask 'Which make/model of car do you own?'
        resp dbase('carsdb') nodata
comment Read the car code from field 5
        set carcode = selfld(cartype,5)
cfix      fix (4) carcode
```

If the respondent says he/she owns a Ford Escort 1.6, the code 1003 will be fixed into the data, whereas if we had used the record's line number, the data would contain a code 3.

¶ In Quancept CAPI, *dbrecno* can be used with *resp dbase* but not with *resp multidbase*.

19.6 Selecting a record from a response list database



Quick Reference

Use **selrec** (Quancept CATI and Quancept CAPI) or **nselrec** (Quancept CATI, Quancept CAPI and mrInterview) to select a record from a database. If you know the text in the first field of the record, type:

```
set variable = selrec(dbvar,field_txt)
```

where *dbvar* is a text variable containing the name of the database and *field_txt* is the text in the first field of the required record.

If you know the line number of the record, type:

```
set variable = nselrec(dbvar,rec_num)
```

where *dbvar* is a text variable containing the name of the database and *rec_num* is the position of the required record in the database.

It is possible to select records from response list databases using commands in the script rather than displaying the database as a response list. You might do this when you want to read information from a look-up database which you do not want to display on the screen.

Any record selected with *selrec* or *nselecr* becomes the current record for that database, and fields may be extracted from it using *selfld*.

Consider the following example which illustrates how you might use this statement.

We have two response list databases about cars. Records in the first database contain four fields: the manufacturer, the make and model, the insurance group, and a group code. The first two fields are those on which the interviewer selects a record; the fourth is the key to the second database which contains more specific information about each type of car. The first three records in this database are:

```
Ford;Escort 1.3;3;0123  
Ford;Escort 1.4;4;0123  
Ford;Escort 1.6;4;0270
```

Records in the second database start with the group code in the first field. The interviewing program compares the group code of the record chosen from the first database with the group codes in the first field of the records in the second database to find the record it needs. The remaining fields (2 to 8) list extras the car may have. The records for cars in groups 0123 and 0270 are as shown below.

```
0270; ;catalytic converter;central locking; ;sun roof;metallic paint; ;
0123; ;catalytic converter; ; ;sun roof; metallic paint; ;
```

Notice the blank fields in both records, particularly those at the ends of the records. If you need to enter blank fields in a record, you must ensure that they consist of at least one space if you want to be able to check for blank fields in your script. The interviewing program does not recognize two consecutive semicolons as having the same meaning as ‘; ;’.

The script below uses these databases. First, it displays the database and waits for the interviewer to code the type of car the respondent owns. Then it selects the group code from the fourth field ready to be used as a pointer into the second database which the interviewer does not see. Once the appropriate record has been found in the second database, the interviewing program checks which of the available extras the respondent’s car has and codes them into the data file accordingly. A second version of the script for Quancept CAPI uses the *resp multidbase* keyword to display a list of all possible extras on the respondent’s car.

```
comment Dummy question for collecting extras purchased
comment This is the full list of extras for all cars
allext    dummyask 'Dummy list of all extras on all cars'
            resp mp 'power steering' / 'catalytic converter' /
                  'central locking' / 'air conditioning' /
                  'sun roof' / 'metallic paint' / 'anti-lock brakes'

comment Find exact type of car owned from the first database.
comment Record selected becomes current record for that database.
cartype   ask 'Which make/model of car do you own?'
            resp dbase('carsdb')
comment Select car name from 2nd field & group code from 4th field
            set car = selfld(cartype,2)
            set carcod = selfld(cartype,4)
comment Select record starting with this code from second database.
comment This record becomes current record for that database.
            set second = 'carextra'
            set extra = selrec(second,carcod)
comment Deal with each extra in turn. ex points to a cell in
comment allext; field is an offset to ensure we skip the 1st
comment field
            unset allext
11       for ex = 1 to 7
            set field = ex + 1
            set item = selfld(extra,field)
```

```
comment Ignore unavailable extras
    route (item=' ') go nex1
qd      ask 'Does your '+car+ have '+item+'?
    resp sp 'yes' / 'no' go nex1 nodata

comment Net extras into dummy response list
    set allext = ex
nex1    next
```

- In mrInterview you can also use nselrec to select individual records from a subset of records that have already been selected from a database. In this instance, nselrec selects records using their line numbers in the subset of selected records rather than their line numbers in the main database.

19.7 Making the response list database available to interviewers



Quick Reference

To copy a response list database into shared memory, run the **shm** program:

shm database_name or **shm ***

On some systems you may need to be logged in as root to run shm.

Before you can use a response list database in Quancept CATI, the database must be copied into shared memory. As the name suggests, shared memory is an area of the computer's memory which is shared between a number of different processes. When you run a program, the computer normally keeps your work in an area of memory which is unique to you. If someone else runs the same program, they are allocated their own memory area.

A simple example is when you are editing a script file over a network. The computer keeps a copy of the script in memory and updates it as you make changes. If someone edits the file at the same time, the computer will place a second copy of the file in a different area of memory and will update that copy with any changes made by the second person. Your changes will not affect the second user's view of the script, and his/her changes will not affect your version.

Placing a file in shared memory means that many users can look at that file by accessing the same area of memory. In the case of response list databases, this means that you copy the database into memory once and it becomes available to any interviewer who needs it.

Once the database has been copied into shared memory, shm creates a file containing a reference to that area of memory. The file is called *dbname.shm* and is written in computer readable form. If you list the file on your screen it will appear as a blank line.

While this is going on, `shm` displays messages on your screen:

```
Welcome to Quancept: shm v7.8 (1.1)
database information:
database name /usr/barbara/qctest/carsdb
No. of lines 26
Completed successfully
```

Once you have shared the database, you can leave it in shared memory until the project is finished.

If you forget to share a database you will still be able to run the script but when you reach questions that use the database, only the question text and the base prompt will be displayed. (Type \$ to continue with the next question.)

-
- ❖ If you change **any** information in the response list database or add any items to it, you must remove the old version from shared memory and then reshare the database before the changes will appear in the interviewing program. Similarly, if the system crashes or is shut down you will need to reinstate the database in shared memory before interviewing starts.
-

19.8 Checking for response list databases in shared memory



Quick Reference

To see which shared memory segments have been allocated to which response list database, run the **listshm** program:

listshm

This produces a report of the form:

```
Welcome to Quancept: listshm v7.8 (1.1)
listshm lists all shared memory segments it finds.
```

ID	Size	
0		
196612	55284	/usr/ben/qcdemo/shops
3768333	1000072 *	/usr/barbara/qctest/shopdb

The asterisk in the second line of the example indicates that the database is currently in use.

If *listshm* does not work for any reason, you can obtain similar information with the system program ipcs.

19.9 Removing response list databases from shared memory



Quick Reference

To delete a response list database from shared memory, type one of the following:

rmshm *database_name*

rmshm id *n*

rmshm *

where *n* is the database's shared memory identifier.

For example, to delete the carsdb database from shared memory you'd type:

`rmshm carsdb` or `rmshm id 12345`

The screen dialog is:

```
Welcome to Quancept: rmshm v7.8 (1.1)
Removing shared memory id 12345
Shared segment size: 242 removed
```

Removing the database from shared memory with *rmshm* automatically deletes the shm file from the project directory.

If *rmshm* fails to work, you can remove response list databases from shared memory with the system program *ipcrm*.

19.10 Sample scripts

Script A



Before you run this script, copy the carsdb and carextra databases into shared memory with *shm*. When you have finished, remove them from shared memory with *rmshm*.

```

comment Sample script A for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 19, Very long response lists (response list
comment databases - CATI)

allext      dummyask 'Dummy list of all extras on all cars'
             resp mp 'power steering' / 'catalytic converter' /
                     'central locking' / 'air conditioning' /
                     'sun roof' / 'metallic paint' / 'anti-lock brakes'

comment Response list comes from a response list database
cartype     ask 'Which make/model of car do you own?'
             resp dbase('carsdb')

comment Select the car name from 2nd field of chosen record and the
comment group code from 4th field of chosen record
             set car = selfld(cartype,2)
             set carcod = selfld(cartype,4)

comment Select record starting with this code from second database.
             set second = 'carextra'
             set extra = selrec(second,carcod)
             unset allext
l1          for ex = 1 to 7
             set field = ex + 1

comment Select next extra from record in second database
             set item = selfld(extra,field)
             route (item=' ') go nex1
qd          ask 'Does your '+car+' have '+item+'?'
             resp sp 'yes' / 'no' go nex1 nodata
             set allext = ex
nex1        next

d1          display 'The respondent''s '+car+' has the following
extras:@@'+allext
             end

```

The cars response list database, carsdb, contains 26 records. Field 1 contains the manufacturer's name; field 2 contains the make and model; field 3 contains the insurance group; and field 4 contains a reference code to a related record in the carextra response list database:

```
Ford;Escort 1.3;3;0123
Ford;Escort 1.4;4;0123
Ford;Escort 1.6;4;0270
Ford;Sierra 1.6;4;0270
Ford;Sierra 1.8;4;0265
Ford;Sierra 2.0;5;0265
Ford;Sierra XR4x4;6;0641
Porsche;944 S2;9;0641
Porsche;944 Turbo;9;0641
Porsche;911 Carrera;9;0641
Porsche;911 Turbo;9;0641
Porsche;928S;9;0641
Porsche;928 Turbo;9;0641
Vauxhall;Astra 1.4;3;0123
Vauxhall;Astra 1.6;4;0270
Vauxhall;Astra 1.8;5;0265
Vauxhall;Astra GTE;6;0265
Vauxhall;Cavalier 1.4;4;0123
Vauxhall;Cavalier 1.6;4;0123
Vauxhall;Cavalier 1.8;4;0270
Vauxhall;Cavalier 2.0;5;0265
Vauxhall;Cavalier GSi;6;0265
Volkswagen;Scirocco GT II;6;0265
Volkswagen;Scirocco Scala;6;0265
Volkswagen;Corrado 16V;6;0641
Volkswagen;Corrado G60;6;064
```

The carextra response list database contains four records. Each record starts with a numeric cross reference code and continues with seven fields of possible extras the car may have.

```
0641;power steering;catalytic converter;central locking;air conditioning;sun
roof;metallic paint;anti-lock brakes
0265; ;catalytic converter;central locking; ;sun roof; metallic paint;anti-lock
brakes
0270; ;catalytic converter;central locking; ;sun roof;metallic paint; ;
0123; ;catalytic converter; ; ;sun roof; metallic paint; ;
```

Script B

```
comment Sample script B for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 19, Very long response lists (response list
comment databases - CAPI)

comment Response list comes from a response list database
cartype ask 'Which make/model of car do you own?'
    resp dbase('carsdb')

comment Select the car name from 2nd field of chosen record and the
comment group code from 4th field of chosen record
    set car = selfld(cartype,2)
    set carcod = selfld(cartype,3)

comment Display the database and allow up to 7 items to be chosen
comment from it.
11      for i = 1 to 7
extras      ask 'Which of the following extras does your car have?'
            resp multidbase('extras') null
next
end
```

The cars response list database is the same as that used in sample script A. The extras response list database contains a list of seven possible extras the car may have:

```
power steering
catalytic converter
central locking
air conditioning
sun roof
metallic paint
anti-lock brakes
```


20 ODBC databases in Quancept Web and mrInterview

Quancept Web and mrInterview support ODBC (Open Database Connectivity) databases as a means of selecting information from an external database, and updating and adding to that database based on information gathered during the interview.

-
- ↖ ODBC databases are not the same as Quancept CATI response list databases. The response list database is a special Quancept CATI tool for dealing with very long response lists, and is just a collection of text files from which responses lists can be built rather than a true relational database. Quancept Web and mrInterview do not support this feature.
-

The statements and variables described in this chapter are:

connect	establish a connection to the database
dbclose	close the connection to the database
dbquery	define a query to be passed to the database to extract records from it, update an existing record or add a new record
dbsqlcrs	set the SQL cursor type
nselrec	extract a record into a variable
rplaycnt	number of times to replay script after an ODBC error
selrec	extract a record into a variable
selfld	extract a field from a record and place it in a variable

20.1 What database formats can I use?

Quancept Web and mrInterview database facilities have been designed primarily to work with Microsoft Access and Microsoft SQL Server databases. We do not currently recommend using Excel databases directly. Instead, we suggest that you export the data to Access for use during the survey.

Licensing issues

The software license for your database software may set a limit on the number of people who can access the database at any one time. You should make sure that any such limit is large enough to handle many simultaneous interviews. Requests to access the database that are above this limit will often fail to retrieve data or update the database (the exact outcome depends on your licence mode).

20.2 Making the database public

Databases that are to be accessed via Quancept Web and mrInterview must be made public as a system DSN, and must have the appropriate ODBC driver installed if they are to work as intended. The exact details of how this is done may vary between installations depending on the databases used and the way the system is set up. Your system or database administrator will do this for you using the 32-bit ODBC tool in the Control Panel.

For simplicity and ease of understanding, the DSN (or domain system name) is referred to as the database name in the remainder of these notes.

20.3 Connecting to the database

Quick Reference

To connect to the database, type:

```
label  connect('sql : database_name : user_name : password ')
```

where *database_name* is the name of the ODBC database. *user_name* and *password* are the user name and password to be used to open the database; if there is no password enter a single space instead.

If the connection fails, *connect* returns a value of -1.

The following example connects to the carsdb database using the Admin user name which has no password. If the connection fails a suitable error message is displayed.

```
cdbopen connect('sql:carsdb:admin: ')
    if (cdbopen=-1) {
dberr        display 'Unable to open carsdb database'
        pause
    }
```

Once a respondent is connected to the database, that connection lasts for the whole interview.

-
- ↖ You must open a connection to the database before you query it. If you have a number of queries to make, you can either leave the connection open until after the last query or, if the queries are scattered through a large script, close the connection with *dbclose* and then re-open it the next time you need it.
-

20.4 Closing the connection to the database



Quick Reference

To close the connection to the database, type:

```
set variable = dbclose(connect_label)
```

where *connect_label* is the label of the *connect* statement that opened the connection to the database, and *variable* is the name of a temporary variable that will be set to 0 if the database is closed successfully or to -1 if not.

For example:

```
cdbopen connect('sql:carsdb:admin: ')
    if (cdbopen = -1) {
dberr1      display 'Unable to open carsdb'
    }
    ...
    set cdbclose = dbclose(cdbopen)
    if (dbcclose = -1) {
dberr1      display 'Unable to close carsdb'
    }
```

If you do not close the database connection in the script it is still closed automatically at the end of the interview. Any errors associated with closing the database are noted in the log file.

20.5 Querying the database

Quick Reference

To query a database, type:

```
set varname = dbquery(db_lab, 'sql_query')
```

where *db_lab* is the label of the *connect* statement that opened the database and *sql_query* is a standard SQL query for the type of action you want to take. *varname* is the name of a temporary variable that is set to 0 if the query is successful or to -1 if it fails.

Once you have connected to the database you can query it. In simple terms, this means issuing commands that either select records from the database, update existing records in the database, or add new records to it.

The following sections explain how to select, update and add records.

-
- ❖ The notes in these sections give examples of basic queries only. For more detailed information on SQL queries please refer to your database manuals.
-

20.6 Selecting records from the database

Quick Reference

To select records from an ODBC database, type:

```
set varname = dbquery(db_lab, 'select * from table_name')
```

where *db_lab* is the label of the *connect* statement that opened the database, and *table_name* is the name of the database table from which to select the records. *varname* is set to 0 unless the query fails due to an error.

Before you can use information from the database in an interview, you need to select the records you want from the database. You do this using an SQL *select* statement. For example:

```
set makes = dbquery(cdbopen, 'SELECT * FROM carmake')
```

selects all records from the carmake table that is part of the database opened by the *cdbopen connect* statement. The ‘makes’ variable will be set to 0 if the query runs successfully. (In this example, SQL keywords are shown in upper case to distinguish them from Quancept keywords and database names, but you can type them in lower case if you wish.)

-
- ❖ It is not an error for a query to select no records; *varname* is returned as zero as long as the query runs without errors whether or not it selects any records.
-

At the most basic level shown here, the *select* statement selects all records in the database, but it is possible to narrow the selection down so that only records that meet certain criteria are selected. For instance, you could select only records that refer to Ford cars, or only records for low calorie fizzy drinks.

Extracting a single record from the temporary variable

Quick Reference

To extract a single record from those selected and place it in a temporary variable, type

```
set variable = selrec(db_lab, 'fieldname = value')
```

where *db_lab* is the name of the *connect* statement that opened the database, *fieldname* is the name of a field in the record and *value* is a value that may appear in that field.

If you know the line number of the record, type:

```
set variable = nselrec(db_lab, rec_num)
```

where *db_lab* is the name of the *connect* statement that opened the database and *rec_num* is the position of the required record in the database.

With *selrec*, Quancept Web and mrInterview will locate a record that contains *value* in *fieldname* and will copy it into *variable*. If the command fails, *variable* will contain the words ‘Not found’ instead.

There are two ways of defining field values. If you always want to extract the same record you can type an explicit value. For example, to extract the record that contains the words ‘Porsche’ in the make field you would type:

```
set cmake = selrec(cdbopen, 'make=Porsche')
```

Usually, however, you’ll want your script to be more flexible and to extract a record whose field contents vary between respondents. For instance, you may want to ask the respondent what type of car he/she drives and then extract the record for that car from the database. In this case, you enter the value as [+*variable_name*+], as shown below:

```
q19 ask 'What type of car do you drive?'
resp sp 'BMW' / 'Ford' / 'Porsche' / 'Toyota' / 'Volkswagen'
set cmake = selrec(cdbopen, 'make=[+q19+]')
```

Extracting data from a field

Quick Reference

To copy information from a database field into a temporary variable, type:

```
set variable = selfld(tempvar,field_num)
```

where *tempvar* is the name of the temporary variable containing a database record and *field_num* is the number of the field you want to extract.

Some records consist of a number fields. Once you have a single record available in a temporary variable you can use *selfld* to copy information from a field into a temporary variable. For example, if the fifth field in the Porsche record contain the car's insurance group, you could extract it by typing:

```
set insgrp = selfld(cmake,5)
```

-
- ☞ All values are returned as text strings even if the fields are defined as integers in the database. If you want to use these values in expressions or calculations, you should convert them to numeric format using *valstring* first.
 - ☞ See 'Converting text numbers to integers' in chapter 15, Working with numeric data.
-

20.7 Updating records in the database

Quick Reference

To update records in a database, type:

```
set varname = dbquery(db_lab, 'update table_name  
    set field1="val1"[, ...fieldn="valn"]  
    where this_field="value";')
```

where *db_lab* is the label of the *connect* statement that opened the database, and *table_name* is the name of the database table containing the records to be updated. The *set* clause inside the parentheses names the fields to be updated and the new value each one is to have; the *where* clause defines which records are to be updated. *varname* is a variable which Quancept Web and mrInterview set to -1 if the update fails.

The statement is shown on three lines and with the continuation lines indented for convenience and ease of reading only.

The *set* clause inside the parentheses names the fields to be updated and the new value each one is to have. You can update as many fields in a record as you wish simply by separating each field and value pair with a comma. Notice that the values are enclosed in two sets of single quotes (in the standard SQL statement, values are enclosed in one set of single quotes, but this standard is overridden by the Quancept requirement of two single quotes whenever one is required). If the value to be placed in a field is the answer to a question or the value of a Quancept variable, type **[+variable_name+]** inside the quotes instead of a fixed value.

The *where* clause defines which records are to be updated. It says that only records in which *this_field* contains *value* should be updated. If you want to update a single record, the value that you specify here must be unique.

Finally, notice that the entire update command is enclosed in single quotes, and that there is a semicolon just before the terminating quote. If you omit any of this punctuation your update will fail.

As an example of when to use this statement, suppose that you regularly interview the same set of respondents, perhaps in a monthly panel. Your database is a respondent list containing background details about each respondent, in which each person is identified by a unique reference code. From time to time, this data may need updating, as the respondent buys a new car, has more children or changes jobs. If you check this information in the script and it is out of date, you can update the database directly from the script.

Here is the section that deals with new car purchases. If the customer has bought a new car since the last interview, his/her record in the database is updated with the revised information:

```
newmake ask 'What make is your new car?'
resp sp allmake
set upd1=dbquery(cdbopen,'UPDATE carmake SET make=''+[+newmake+]''
WHERE cust_id=''[+cref+'';']')
```

It opens the database that was connected at label *cdbopen* and locates the record for the current customer (this has been extracted into the Quancept variable '*cref*' earlier in the script). The statement updates the *carmake* table by placing in the 'make' field the response held in the Quancept variable '*newmake*'. All other fields in the respondent's record are ignored and retain their existing values.

20.8 Adding new records to the database

Quick Reference

To add a completely new record to the database, type:

```
set varname = dbquery(db_lab, 'insert into table_name(field1, ..., fieldn)
                                values (value1, ..., valuen)')
```

where *db_lab* is the label of the connect statement that opened the database; *table_name* is the name of the table in which you want to insert the new record; *field1* to *fieldn* are the names of the fields that are to contain data; and *value1* to *valuen* are the data to be placed in those fields.

The statement is shown on two lines and with the continuation line indented for convenience and ease of reading only.

You must declare the same number of fields and values. Any fields that exist in the table but that are not named on the *insert* statement are left blank.

If the addition fails, Quancept Web and mrInterview set the variable to -1.

☞ When ODBC error occurs during an interview, usually because the connection to SQL is broken, the interviewing program replays the interview twice (this is configurable) in an attempt to re-establish the connections to SQL and to the ODBC database. If the connections can be re-established, this is normally invisible to the user and the interview continues as normal. However, if the replayed portion of the interview contains statements that insert records in the database, this generates a duplicate record in the database for each replay.

☞ Interviews are also replayed when the user continues an interview that has timed out and when the user clicks Previous to return to the previous question.

To avoid inadvertently creating duplicate records in these circumstances, it is a good idea to check whether a record exists before inserting it. For example, instead of simply writing:

```
set insok = dbquery(cdbopen,'insert into mytable(name,age)
                           values ('[+qcname+]',qcage)')
```

you might write:

```
if (cdbopen=0) {
    if (done<>1) {
        set insok = dbquery(cdbopen,'insert into mytable(name,age)
                           values ('[+qcname+]',qcage)')
        set done = 1
    }
}
```

20.9 Number of script replays after an ODBC error

Quick Reference

To set the number of times a script will be replayed after an ODBC error, type:

```
set rplayent = value
```

where *value* is the number of replays allowed.

When ODBC error occurs during an interview, the interviewing program replays the interview twice in an attempt to re-establish the connections to SQL and to the ODBC database. If the connections can be re-established, this is normally invisible to the user and the interview continues as normal. You can set a different number of replays using the variable *rplaycnt*.

The interviewing program logs each ODBC error, including any error information from the ODBC driver, and reports the number of ODBC errors that have been encountered so far. If *rplaycnt* is non-zero and the ODBC error count is not zero, the interviewing program also logs an error just before an *ask* or *pause* statement with saying either ‘odbc errors found, replaying script’ or ‘odbc errors found, no more script replaying’. This log entry includes the number of ODBC errors found, the number of replays so far, and the number of replays allowed. If the number of replays so far is smaller than the number allowed, then a replay is done, otherwise the interview continues despite any remaining ODBC errors.

To be sure that this feature works for your script, be sure to place all ODBC statements before an *ask* or *pause* statement, because the interviewing program will not do this check when the script terminates. Failing ODBC statements between the final *ask* or *pause* statement and termination will not result in a replay.

20.10 Setting the SQL cursor type

Quick Reference

To set the SQL cursor type, place the following statement in your script:

```
set dbsqlcrs = value
```

where *value* is 1 for a keyset driven cursor, 2 for a static cursor, or 3 for a forward only cursor.

If *dbsqlcrs* has any other value or is not set, a value of 0 is assumed which equates to the default dynamic cursor.

This is an advanced feature and is designed for users with a reasonable knowledge of SQL. The notes that follow provide only as much information as is required to implement this feature in an mrInterview script. For further information on SQL cursors, refer to your SQL documentation (if you have SQL Server installed, you can find out about SQL cursors by selecting SQL Server Books Online from the Start menu).

Cursors are an SQL feature that allow *selrec* and *nselelrec* statements in a questionnaire script to function correctly. In particular, they are used in positioning at (or, in mrInterview terms, selecting) a particular record from a set of records that have been selected from an ODBC database by means of an SQL SELECT statement defined in a *dbquery* statement in the questionnaire script.

The default cursor type used for mrInterview ODBC operations is the dynamic cursor, but this may not be the most efficient one to use if you have large databases or if your selection criteria select many records from the database. In these cases you may prefer to use a different type of cursor.

As an example, a simple test that used *selrec* to extract a field from the last row of a 100000-row database produced the following timings:

Cursor type	Time in seconds
Dynamic (default)	75
Keyset driven	35
Static	25
Forward only	1 to 2

When choosing a cursor type, bear in mind the following points:

- Dynamic cursors are the default because they provide an automatic updating feature.
- It is your responsibility to test that the cursor you choose provides the functionality you expect from your script. Testing has shown that the static cursor type does not support the SQL_FETCH_FIRST, SQL_FETCH_NEXT or SQL_FETCH_ABSOLUTE fetch orientations for SQL Server.
- Performance improvements can often be achieved by using SQL_FETCH_ABSOLUTE.
- *nselelrec* currently fails when used with a forward only cursor. The error in the log file is 'odbc nselrec failed, fetchscroll error'.
- Some combinations of cursor type and SQL_FETCH orientation (for example, forward only cursor with SQL_FETCH_ABSOLUTE) may fail.
- Some ODBC functions will always work slowly; for example, *nselelrec* always steps through records one at a time until it finds the requested record.

20.11 General notes about working with ODBC databases

- Information extracted from databases using *nsele**rec*, *selrec* or *selfld*, or related to updates made using an SQL *update* command, is not added to the respondent's data. To do this, use a *fix* statement in the script.
- Database keywords should not be used with *colgrid* or *rowgrid*.
- *nsele**rec*, *selrec*, *selfld* and the SQL *update* command can be used inside loops, but *connect* may not.
- Any SQL commands that appear as part of a Quancept statement are passed directly to the database without any checking. It is therefore entirely your responsibility to ensure that your statements are correct and have the desired effects on the database.
- If you want to access information that has been updated or inserted in the database during the interview, you must issue another SQL *select* command to force the database to be requeried.
- mrInterview has a built-in time-out of 60 seconds for ODBC queries. If the SQL server does not respond to the query within 60 seconds, the respondent will see the message 'Unable to Service Request'.
- Use the SQL *where* clause to restrict the number of records that a query selects, particularly if your database is large. Queries that are very general are more likely to result in time-outs.
- All values are returned as text strings even if the fields are defined as integers in the database. If you want to use these values in expressions or calculations, you should convert them to numeric format using *valstring* first.
- Some ODBC functions will always work slowly; for example, *nsele**rec* always steps through records one at a time until it finds the requested record.
- Dynamic cursors are the default because they provide an automatic updating feature.
- When changing the cursor type, it is your responsibility to test that the cursor you choose provides the functionality you expect from your script.
- Testing has shown that the static cursor type does not support the SQL_FETCH_FIRST, SQL_FETCH_NEXT or SQL_FETCH_ABSOLUTE fetch orientations for SQL Server.
- Performance improvements can often be achieved by using SQL_FETCH_ABSOLUTE.
- *nsele**rec* currently fails when used with a forward only cursor. The error in the log file is 'odbc nsele failed, fetchscroll error'.
- Some combinations of cursor type and SQL_FETCH orientation (for example, forward only cursor with SQL_FETCH_ABSOLUTE) may fail.

20.12 Sample script

This section contains a complete yet simple script that you may like to use for reference. The script is designed for a company that carries out regular research among a fixed panel of respondents. Information about each respondent is held in a database, and different items of information may be extracted depending on the requirements of the current survey. The database contains one table called info.

-
- ↖ Before running this script you will need to edit it to change the user name and password shown in the line labeled qdb1.
-

```
comment Sample script for the Quancept & mrInterview Scriptwriter's Manual
comment Chapter 20, ODBC databases in Quancept Web and mrInterview

carlist define 'Ford' / 'Porsche' / 'BMW' / 'Volkswagen' / 'Toyota' /
               'Peugeot' / 'Volvo' / 'Nissan' / 'Renault'

comment Connect to the paneldb database
qdb1      connect('sql:paneldb:ADMIN:topdog')
if (qdb1 = -1) {
dberr      display 'Cannot connect to the panel database'
            pause
            goto ctn
}

comment Select all records from the database
set getall = dbquery(qdb1,'SELECT * FROM info')

comment Get respondent's reference code and extract that record from
comment the database
custref ask 'Please enter your panel reference code.'
resp coded(0)
set getme = selrec(qdb1,'refcode=[+custref+]')

comment Extract current car make and model from respondent's record.
comment These are in fields 9 and 10 respectively
set carmake = selfld(getme,9)
set carmod = selfld(getme,10)

comment Check these details are correct and get new details if not.
qchk      ask 'Our records show that you own a '+carmake+' '+carmod+'.'
           Is this correct?
           resp sp 'Yes' go ctn / 'No'

newmake ask 'What make is your current car?'
       resp sp carlist nodata
newmod  ask 'And what is the model?'
       resp coded(0) nodata
```

```
comment Update the respondent's record with the new information
    set upd1 = dbquery(qdb1,'update info
        set carmake=''[+newmake+'',carmodel=''[+newmod+''
        where refcode=''[+custref+]''';')
ctn    continue
end
```


21 GNU GDBM relational databases in Quancept CATI

Quancept allows you to work with GNU GDBM external relational databases from within your scripts. GNU GDBM databases are developed for Unix by the Free Software Foundation. SPSS MR is not involved in the development of GDBM and can only offer technical support in so far as your database interacts with Quancept. See ‘About the GNU GDBM database manager’ below for information about the Free Software Foundation.

The keywords covered in this chapter are:

eximport	establish a connection to the database
export	export an existing record or create a new record in the database manually
import	import and view a record from the database
impupdate	read in, view and automatically update existing records or create new records in the database
listdbm	view the contents of the database
setupdbm	create a GNU database

About the GNU GDBM database manager

The following information was supplied by the Free Software Foundation:

GDBM, the GNU data base manager, by Philip A. Nelson. Copyright ©1990, 1991 Free Software Foundation, Inc. GDBM is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 1, or (at your option) any later version.

GDBM is distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with GDBM; see the file ‘copying’. If not, write to:

The Free Software Foundation 675 Mass. Ave. Cambridge MA 02139 USA

You may contact the author by:

<i>e-mail:</i>	<i>phil@cs.wwu.edu</i>
<i>us mail</i>	<i>Philip A Nelson Computer Science Department Western Washington University Bellingham WA 98226</i>

21.1 Defining a GNU database

A GNU database is a binary data file where the information is laid out in records in a structured way. The structure of the database is held in an associated definitions file. Records in a GNU database can take one of two forms. In a **gdbm** format database, data is held in fields that can vary in length from record to record and the fields are separated by a semicolon:

key field1;field2;...;fieldN;

In a **gdbmfix** format database, data is held in fields of fixed lengths:

key field1 field2 ... fieldN

Each line in a GNU database is a record comprised of a key field and one or more data fields. A record may not exceed 1024 characters, of which 64 characters can be allocated to the key field and 54 characters is the maximum length of any one field. Record types may not be mixed in a database.

A GNU database can start out empty or you can build one from a plain text file that already contains data.

-
- ☞ If you want to use an existing text file as a database, you must first convert the text file into a form that qtip can read. See section 21.4, Creating the GNU database.
-

Key fields

In both types of databases, the first field in every record is known as the *key* field. The key does not have to be defined in either type of description file; it is implicitly defined as being any data that appears before the first blank space in the record. In either type of database, the key field can be of variable length but must not exceed 64 characters.

The processing of record keys is case-sensitive. You may enter keys in the database in any combination of upper or lowercase characters or numbers. However, when you work with records inside your script (using one of the keywords *import*, *impupdate* or *export*), you must enter the key exactly as it appears in the database using the same combination of upper and lowercase characters.

When you want interviewers to type in specific record keys in order to work with existing records in a database, it may be helpful to print out a list of the record keys that they can use for reference.

-
- ☞ There is an anomaly between Quancept and GNU that affects the way key fields are processed. A key field in a database created from an existing text file may not contain spaces. However, a key field that is entered from within Quancept can include spaces and the GNU database will accept it. If you plan to work with the database only within Quancept, this will not be a problem and can be considered a useful feature. For example, you could have records with keys like 'Simon Bolivar' and 'Simon Evans'. However, if you plan to use the GNU database

in other applications external to Quancept, those records will be considered duplicates. In this instance, you should validate keys entered by interviewers and remove spaces from the input; an example of how to do this is given in the example script B. In either case, an original text file that will be converted using the command *setupdbm* may not contain spaces.

- ☞ The key must be unique for each record. Otherwise, data will be overwritten when you convert the database using *setupdbm*. For more information, see section 21.4, Creating the GNU database.
-

Creating original data files

Each GNU database has an associated description file that defines the structure of the records (the field layout). The description file layout is different for the two types of databases. You can reuse a description file to define multiple databases of the same type as long as the fields are the same in each database. You do not have to have a separate description file for each database.

The order in which you create the database and its associated description file is not important. You can set up a description file for an existing database or you can set up the description file first and then create the database later.

You can create databases in various ways. Although a text file must exist, it can be empty when you convert it into a GNU database .gdb file for qtip. If you prefer to use existing data, you can simply type the information into a file using any text editor. You can also export records from spreadsheets or other software packages, for example, tables from a word processor. If you export data from other packages, save the file in a plain ASCII format that can be read by Unix applications.

You can also add information gathered during a project to the database as interviewing takes place.

-
- ☞ See section 21.2, Description file for variable field length (gdbm) database, and section 21.3, Description file for fixed field length (gdbmfix) databases.
-

21.2 Description file for variable field length (gdbm) database

Quick Reference

To describe a field in a variable field length database, type:

[]fieldname fieldnum 1 field_type*

where *fieldname* is a field name of between one and eight alphanumeric characters, the first of which must be a letter. Field names beginning with * cannot be updated by interviewers. *fieldnum* is a number indicating the field's position in the record, and *field_type* is a letter defining the type of data the field contains — **t** for text, **n** for numeric or **r** for real numbers.

Every GNU database must have an associated database description file. The description file is a plain text file that defines the record layout or structure of the database. Each field is described on a separate line in the database description file.

In gdbm files, the number of characters of data held in each field can vary from record to record, but no field can be more than 54 characters (not including the delimiting semicolon). For example, the same text field could hold one word in one record and ten words in the next. If you enter more than 54 characters of information, the data will be truncated when the record is written to the database. Data in text fields can include spaces. Data in numeric fields can include a minus sign to indicate negative numbers.

-
- ☞ You are allowed to enter a decimal point in numeric fields, but qtip will not recognize it. If you need a decimal point in your data, define the field as type *r* (real) instead of *n*.
-

If you want to preserve a field from being updated accidentally, then place an * (an asterisk symbol) before the field name in the description file.

You create the description file with any text editor and save it as a plain text (ASCII) file. Remember that in the database itself, the key is the first value that appears in each record, delimited by a space. You do not enter anything in the description file to define the key; it is defined implicitly for you.

When you open the database in order to read from it or write information to it, Quancept uses the *fieldname* you define as a temporary variable.

Example data

The following is an example of a description file for a gdbm (variable field length) database:

```
artist 1 1 t
composer 2 1 t
title 3 1 t
media 4 1 t
type 5 1 t
```

A GNU gdbm database that contains data defined according to the entry above might look like the following example:

```
elton Elton John;;Goodbye Yellow Brick Road;CD;pop;
madonna Madonna;;;CD or cassette;pop;
kiri Kiri Te Kanawa;Mozart;Magic Flute;CD;class;
kylie Kylie Minogue;;;;pop;
ella Ella Fitzgerald;Porter;Cole Porter Songbook;CD;show tunes;
ludwig ;Beethoven;1st Symphony;CD;class;
```

Note that each of the explicitly defined fields (including the last field in each record) ends in a semicolon. Only the key is separated by a space. There is always one semicolon at the end of each field, even if the field is empty. If two or more empty fields appear consecutively, then you enter one semicolon for each field. You may not include a semicolon character inside a field as data.

If you count the semicolons in the example above, you will see there are five in each record. The records with the keys ella and kiri contain data in all fields. In the ludwig record, the ‘artist’ field (field 1) contains no data. Therefore, the key field ends in a space and then field 1 is represented by a semicolon alone. The elton record has no data in the ‘composer’ field (field 2). The kylie record has no data in the ‘composer’, ‘title’ or ‘format’ fields (fields 2, 3 and 4).

21.3 Description file for fixed field length (*gdbmfix*) databases

Quick Reference

To describe a field in a fixed field length database, type:

`[*]fieldname field_begin_col field_length field_type`

where *fieldname* is a field name of between one and eight alphanumeric characters, the first of which must be a letter. Field names beginning with * cannot be updated by interviewers. *field_begin_col* is the position number for the first column of that field; *field_length* is the number of characters in the field; and *field_type* is a letter defining the type of data the field contains — **t** for text, **n** for numeric or **r** for real numbers.

Every GNU database must have an associated description file that defines the layout of the database. The description file is a plain text file. Each field (except the key) is defined on a separate line in the database description file.

The first field always starts in column one, and column one always begins after the first space in the record which indicates where the key finishes. Text fields may be up to 54 characters. The number of fields is restricted by the total character limit of 1024 for a record. For numeric fields, the field length is restricted to nine digits.

In *gdbmfix* databases, each field in a record is defined to be of a fixed length. However, each field in a record can be a different length to the other fields. The data is not required to fill the entire field. Spaces are allowed inside text fields, but not the key field. The key is the first value that appears in each record and is always delimited by a space. You do not enter anything for the key as part of the definition — it is implicitly defined for you. Even though this type of database requires the data in the ordinary fields to be of fixed length, the key itself can be of variable length.

Example data

The following database definition:

```
lowrate 1 5 r  
midrate 6 5 r  
highrate 11 6 r
```

would be suitable for a GNU *gdbmfix* database containing three data fields, such as percentage interest rates as shown in the following example:

```
Bank1 9.0009.15010.900  
Bank2 1.0001.00001.000  
Bank3 4.4004.50004.600
```

In this example, the letter *r* indicates the first field is defined as a real number field and the number 5 indicates that the field can contain up to 5 digits including a decimal point and a minus sign. The second field and third field are also real number fields. The data for the second field is held in columns six to ten. The third field therefore starts in column 11. The only space in any record is the single space that delimits the key. In this example, the keys all happen to be of the same length, but a key can be up to 64 characters.

When you enter data in numeric fields, you can use a minus sign to indicate a negative number.

21.4 Creating the GNU database

Quick Reference

To read an original text file and write out a database file for qtip, go to a system prompt and type:

```
setupdbm textfile
```

where *textfile* is the name of the text file that holds the original data for your GNU database. (This file can be empty or contain data.)

This process is similar to parsing your script. The command *setupdbm* leaves your original text file unchanged and creates a binary file called *textfile.gdb*. Any changes you make to the database from within your script are written into *textfile.gdb*, not into your original text file.

-
- » In some versions of Unix, *setupdbm* will not work across NFS. If this happens at your site, you can get around the problem by copying the *setupdbm* program to your project directory. To do this, type the following command:

```
cp $QCHOME/bin/setupdbm setupdbm
```

Then, when you type *setupdbm*, it will run the local copy.

Duplicate keys

When you create the GNU database file from a text file containing data as described above, *setupdbm* does not report duplicate keys. If your original text file contains records with duplicate keys, only the last record in the file with that key is stored in the gdb file.

```

ella Ella Fitzgerald;Porter;Cole Porter Songbook;CD;show tunes;
kiri Kiri Te Kanawa;Mozart;Magic Flute;CD;class;           will be overwritten
elton Elton John;;Goodbye Yellow Brick Road;CD;pop;
ludwig ;Beethoven;1st Symphony;CD;class;
kiri Kiri Te Kanawa;Mozart;Great Mass;CD;class;           will be overwritten
kylie Kylie Minogue;;;;pop;
kiri Kiri Te Kanawa;various;Arias;CD;class;                will remain
madonna Madonna;;;CD or cassette;pop;

```

In the example above, there are three records with the key *kiri*. The first two records (the records closest to the beginning of the file) will be lost.

Other errors

If you forget to create the gdb file, the script will parse correctly. However, interviewers will see error messages such as the following when they reach a question that uses a record in the database:

```

Cannot open database near label qname
errorstop-attempting to stop interview n

```

The errorstop message is displayed when the interviewer closes the interview.

Writing data back to the original text file

Quick Reference

To add changes back into your original file (or another text file), go to a system prompt and type:

listdbm *gnudatabase > textfile*

where *gnudatabase* is the gdb file (do not type .gdb on the end of the filename).

Any changes that are made to the database from within your script will only affect the gdb database file. Changes will not be automatically reflected in your original text file.

Viewing data in a GNU database

Quick Reference

To view data inside the binary gdb file, type:

listdbm gnudatabase

where *gnudatabase* is the name of the gdb file (do not type .gdb on the end of the filename).

Database records are not sorted in the gdb file. You can use the Unix commands *sort* and *more* to view the output of *listdbm* in sorted, paged format. To do this type a command such as:

```
listdbm gnudatabase | sort | more
```

See your Unix documentation for more information on these commands.

21.5 Using GNU keywords in scripts

You use the following keywords in your script to work with records in a GNU database:

- | | |
|------------------|--|
| eximport | To establish a connection to the GNU database (the gdb file). |
| impupdate | To import, view and update existing records automatically, or to create new records in the GNU database automatically. |
| import | To import and view a record from the GNU database. Can be used with <i>export</i> to update and create new records manually. |
| export | To export (overwrite) an existing record or create a new record in the GNU database manually. |

❖ Variable names given to GNU database fields should not be used for other purposes in the qtip script. When you compile your script, qparse checks the GNU database at the same time and treats fields in the database as if they were Quancept temporary variables. If you declare dummy questions that use the same name to store field information, then the field information will be set to blank or null.

Quancept only saves one set of values at any given time for the fields in the database records. If you want to preserve values for certain fields while importing and exporting several records from the same database, you must save these values into other temporary variables then copy the values back into the database record's fields before exporting the record.

21.6 Opening a database

Quick Reference

To introduce a GNU database into a script so that you can read information from the database, type:

```
eximport(id_num, 'gnudatabase', 'gdb_desc_file', 'type', 0)
```

where *id_num* is a unique number between 1 and 8 that Quancept can use to identify the database in this script (only eight GNU databases can be accessed from a script); *gnudatabase* is the name of the gdb file (without the .gdb extension) containing the database; *gdb_desc_file* is the name of the database description file; and *type* is either **gdbm** for variable length fields delimited by semicolons or **gdbmfix** for fixed field length data.

You use a separate **eximport** statement to establish the connection between the script and up to eight GNU databases. You must place the *eximport* commands in the script before any of the record manipulation keywords, *impupdate*, *import* and *export*, can be used with that database.

The following example opens a variable field length GNU database called musicdb:

```
eximport(1, 'musicdb', 'musdesc', 'gdbm', 0)
```

You only need to enter a line such as this once in your script for each database, regardless of how many times you access records in the database.

21.7 Reading database records

Quick Reference

To introduce a particular record from a database into a script, type:

```
import(id_num, key_variable)
```

where *id_num* is the identification number of the database from which you want to import the record and *key_variable* is a Quancept variable containing the key of the record you want to access.

You use the **import** keyword when you want to display information for interviewers but do not necessarily want to update that information in the database. If you want to update the information only in certain circumstances, you use *import* with the keyword *export* to update information manually.

You must use an *eximport* statement to establish a connection to the database before you can import a record from the database. The *eximport* statement must appear before the *import* statement in the script.

The identifier given with *import* must match the identifier given for this database's *eximport* statement. If the *key_variable* isn't found in the database, that is, if the record key doesn't exist or the interviewer mistypes it, then the variables named in the database's description file will be set to NULL. Testing on NULL will allow you to route your interviewer through different paths and to update the database or not as you wish.

The following script shows an example of *import* being used to display information from the variable field length database musicdb, which was described earlier in this chapter:

```
eximport(1,'musicdb','musdesc','gdbm',0)
artkey ask 'Enter an artist''s catalog identifier'
resp coded null dk ref
import(1, artkey)
d1 display 'catalog: '+artkey+
@ artist: '+artist+
@ title: '+title+
@composer: '+composer+
@ media: '+media+
@ type: '+type
```

21.8 Manually updating records

Quick Reference

To update a particular record or introduce a new record into a database type:

export(*id_num*, *key_variable*)

where *id_num* is the identification number of the database from which you want to import the record, and *key_variable* is a Quancept variable containing the key of the record you want to update.

You must use an *eximport* statement to establish a connection to the database before you can export a record to the database. The *eximport* statement must appear before the *export* statement in the script.

The keyword **export** may be called more than once in an interview but you only need to call it once to update a particular record on a particular path through that interview. When the script reaches the *export* statement, the database is updated immediately. If the interview is stopped prior to the *export* command then the database will not be updated.

If you want to update multiple records using *import* and *export*, then you must ensure that you're exporting the data back to the record with the correct key. You can do this by saving the value of each record's key into a new temporary variable. Quancept holds only one set of variables corresponding to the fields of the database records. For this reason, if you access more than one record from a single database during the interview, you must copy the values of all the fields in the

database record into separate temporary variables before reading in the values for the next record. Similarly, when you wish to update the database records, you should copy the values stored for each record into the variables corresponding to the database record fields before you export the values to the database.

In the following example, *export* is used along with the keyword *import* to update a record when certain conditions are met. In this case, the interviewer asks for the respondent's job title. The record is only written back to the database if the answer is one of the named job titles: 'Chief executive', 'Chairman' or 'Director' (that is, if jtitle is equal to 1, 2 or 3). If the answer to the question jtitle is Other, that is, if jtitle equals 4, then the record is left unchanged.

```

eximport(2,'contdb','contdes','gdbm',0)
qorg ask 'Enter organization''s code name'
resp coded
set ratekey=qorg
comment Use import keyword to retrieve the record for this organization
comment from contdb(if the record exists) but only update the record in
comment conjunction with the keyword export
import(2,ratekey)

newname ask 'Could you please tell me your name?'
resp coded (0)

jtitle ask 'What is your job title?'
resp sp 'Chief executive''Chairman''Director''Other'

set bigjob=nbit(jtitle)
if (bigjob<4){
    set contact=newname
    export(2,ratekey)
}

```

21.9 Automatically updating records

Quick Reference

To introduce a single record from a database for automatic updating, or to add a new record to a database, type:

impupdate(*id_num*, *key_variable*)

where *id_num* is the identification number of the database from which you want to import the record, and *key_variable* is a variable containing the key of the record you want to access.

The keyword **impupdate** automatically updates the current record in the database at the end of the interview, even if it is stopped, quit or abandoned by the interviewer. You must use an *eximport* statement to establish a connection to the database before you can use *impupdate* with a record from the database. The *eximport* statement must appear before the *impupdate* statement in the script.

You should only use *impupdate* to work with one GNU database record at a time. This is because Quancept holds just one set of temporary variables for each GNU database record. For example, if you use *impupdate* to read in three different records, then all three records will be updated automatically with the values stored in the third record.

The value given for *id_num* must match the value given for the database earlier in the script (using the *import* statement). If the *key_variable* isn't found in the database, that is, if the record key doesn't exist or the interviewer mistypes it, then the variables (the fields named in the description file) will be set to NULL. A new record is added automatically to the database with a key as given in *key_variable*. If you do not want an automatic update of your database, you should use the *import* and *export* keywords instead.

In the following example, the interviewer is first prompted to enter a record key (in question qorg) to access a record in the database. The *impupdate* keyword then ensures that the answer to the question ratechk1 is always written back into the database.

```
eximport(1,'ratedb','ratedes','gdbm',0)
qorg    ask 'Enter organization''s code name'
        resp coded
        set ratekey=qorg
        impupdate(1,ratekey)
ratechk1 ask 'What interest rate are you currently offering short-term
           investors?'
        resp real 0 to 100.00
        set lowrate=ratechk1
```

21.10 Sample scripts

These scripts are designed to be used with GNU databases and will not run by themselves. You will also need other files (stored in the project directory) in order for the scripts to work properly. The additional files are listed immediately before the script that they support.

-
- » All additional files are provided as part of the software, along with the other sample scripts in the manual. The information here is for reference purposes only.
-

Additional files required for Script A

- A plain text database file called musicdb that contains the following text:

```
ella Ella Fitzgerald;Porter;Cole Porter Songbook;CD;show tunes;
elton Elton John;;Goodbye Yellow Brick Road;CD;pop;
kiri Kiri Te Kanawa;Mozart;Magic Flute;CD;class;
kylie Kylie Minogue;;;;pop;
ludwig ;Beethoven;1st Symphony;CD;class;
madonna Madonna;;CD or cassette;pop;
```

- A file called musicdb.gdb created from the text file above by entering the command:

```
setupdbm musicdb
```

- A plain text database description file called musdesc containing the following lines:

```
artist 1 1 t
composer 2 1 t
title 3 1 t
media 4 1 t
type 5 1 t
```

Script A — Displaying records in GNU databases

```
comment Sample script B for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 21, Using GNU GDBM relational databases (CATI only)

comment Variable fields from record to record database example
comment Prompts interviewer for a record key (catalog id)
comment Displays the record if it exists otherwise displays an error

comment Open the database called musicdb; this only need be done once
      eximport(1,'musicdb','musdesc','gdbm',0)

comment Prompt for the record key
artkey  ask 'Please enter an artist''s catalog identifier'
      resp coded null dk ref
```

```
comment The import keyword opens the record with that key
      import(1,artkey)

comment If there is no record with that key, OR the key
comment is misspelled, then a dummy record is created where
comment all the fields are filled with the value NULL. The next
comment line checks the first field to see if it is NULL
comment and if so, displays an error message
      if (artist = NULL) {
sorry          display 'Sorry, cannot find '+artkey+' in our database'
            pause
      }
      else {

comment Display the record that matches the key entered
d1          display 'catalog: '+artkey+
@  artist: '+artist+
@  title: '+title+
@composer: '+composer+
@  media: '+media+
@  type: '+type
            pause
}
end
```

Additional files required for Script B

- A plain text database file called contdb that contains the following lines:

```
Bank1 Mr Bob Bank;
Bank2 Mrs Ursula Insurance;
Bank3 Lester Investor;
```

- A plain text database file called ratedb that contains the following lines:

```
Bank1 9.0009.15010.900
Bank2 1.0001.00001.000
Bank3 4.4004.50004.600
```

- Two gdb files called contdb.gdb and ratedb.gdb created from the text files above by entering the commands:

```
setupdbm contdb
setupdbm ratedb
```

- A plain text database description file called contdes containing the following line:

```
contact 1 1 t
```

- A plain text database description file called ratedes containing the following lines:

```
lowrate 1 5 n
midrate 6 5 n
highrate 11 6 n
```

Script B — Updating records in GNU databases

```
comment Sample script B for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 21, Using GNU GDBM relational databases (CATI only)

comment Prompts for respondent's name and job title
comment and saves those details to database 'contdb' but only
comment if the respondent is a Very Important Person.
comment Also asks respondent for interest rates for investors
comment and always saves the response to database 'ratedb'

comment Open the two databases used in this script
    eximport(1,'ratedb','ratedes','gdbmfix',0)
    eximport(2,'contdb','contdes','gdbm',0)

qorg      ask 'Enter organization''s code name'
resp coded

comment The loop below checks the value entered at qorg for validity
comment as a key in a gnu database record. Valid keys do not contain
comment spaces and are up to 64 characters in length.

comment If text entered at qorg is longer than 64 characters or
comment contains spaces, qorg is cleared and question is asked again.
comment When the character "checking" is blank it means that we
comment have reached the end of a string which is shorter than 65
comment characters and does not contain any spaces, so we skip
comment out of the loop and assign the value of qorg to the key
comment variable ratekey.

11       for i = 1 to 65
        if (i > 64) {
            display 'Only 64 characters allowed in database
key variable. Please re-enter.'
            unset qorg
            pause
            goto qorg
        }
        set checking=substr(qorg,i,1)

comment Next line checks for a single space
if (checking=' ') {
    display 'No spaces allowed in database key variable.
Please re-enter'
    pause
```

```
        unset qorg
        goto qorg
    }
comment Next line checks for blank (null)
    if (checking=='') {
        goto setkey
    }

nextchar next

setkey    set ratekey=qorg
import(2,ratekey)

comment If contact is null, we assume there is no record for the
comment organization named. We ask interviewers if they made a
comment typing error and wish to re-enter the name, OR do they wish
comment to add this organization to the databases OR do they wish
comment to terminate the interview without adding anything

    if (contact=NULL) {
missing      ask 'INTERVIEWER: YOU HAVE ENTERED '+ratekey+
AS THE NAME OF THE ORGANIZATION AND THERE IS NO CONTACT NAME
AVAILABLE. DO YOU WISH TO ...?@@'
        resp sp 'Re-enter name of organization' /
        'Add a new organization' /
        'Terminate interview'
        set react = nbit(missing)

        if (react=1) {
comment Allow re-entry of the organization's name
            unset qorg
            goto qorg
        }

        if (react=3) {
comment Go to the end of the interview and update nothing
            goto goodbye
        }
    }

comment If contact is NULL here it means we are creating new records
comment so there is no need to check whether the name we have is
comment correct
    route(contact=NULL) go newname

namechk  ask 'Is your name '+contact
        resp sp 'Yes'/'No'

comment If the name we have is correct, skip over the new name section
        route(namechk='Yes') go skipname
```

```
comment We want their names for GNU database but not in survey data
comment so keyword nodata suppresses normal qtip data collection
newname ask 'Could you please tell me your name? '
        resp coded nodata

comment Ensure that the name entered is not longer than 54 characters
comment which is the maximum length of a gnu database field
        set toolong=substr(newname,55,1)
        if (toolong<>'') {
            display 'The name you entered is more than 54 characters.
Please re-enter@@'
            pause
            unset newname
            goto newname
        }
jtitle ask 'What is your job title?'
        resp sp 'Chief executive' / 'Chairman' / 'Director' / 'Other'
        set bigjob=nbit(jtitle)

comment Update contact (add name to GNU database) if one
comment of the following is true:
comment (1) the respondent is not the named person and
comment is either Chief Executive, Chairman or Director, OR
comment (2) there is no name currently in the database
        if (bigjob<4 .or. contact=NULL) {
            set contact=newname
            export(2,ratekey)
        }
skipname continue
comment Always update the interest rate
        impupdate(1,ratekey)

comment Don't ask "is NULL still the correct rate of interest?"
        route (lowrate=NULL) go ratelchk
changer ask 'We have your interest rates listed as
@'+lowrate+'%' for short term investors
@'+midrate+'%' for medium-term investors
@'+highrate+'%' for long-term investors
@@ Are these figures still correct?@@'
        resp sp 'Yes'/'No'

comment All rates are still correct, terminate interview.
        if (changer='Yes') {
            display 'In that case ...'
            goto goodbye
        }

comment Otherwise, ask for all three types of interest rates
ratelchk ask 'What interest rate are you currently offering short-term
investors?'
        resp real 0.00 to 100.00
```

```
set lowrate=rate1chk

rate2chk ask 'What interest rate are you currently offering medium-term
investors?'
    resp real 0.00 to 100.00
    set midrate=rate2chk

rate3chk ask 'What interest rate are you currently offering long-term
investors?'
    resp real 0.00 to 100.00
    set highrate=rate3chk

goodbye display 'Thank you for your time. I have no further
questions, goodbye'
    pause
end
```

22 Screen management with graphical user interfaces

In Quancept CAPI, Quancept Web and mrInterview, the interviewing program has a graphical user interface (GUI) that takes advantage of the computer's graphics capabilities. The Quancept scriptwriting language therefore includes facilities for formatting the text that appears on the interview screen and setting an image, color and style for the screen's background. This chapter describes those facilities.

-
- ☞ For information on moving and resizing elements such as response buttons or blocks of text using QCompile, see chapter 42, 'Quancept CAPI design mode'.
-

22.1 Formatting instructions in CAPI scripts



Quick Reference

To define the format for a piece of text, type a formatting tag enclosed in angle brackets before the text and, if necessary, another one after it.

For example:

```
d1 display '<p:10>Text in 10-point type'  
d2 display '<b+>Bold text<b-> and ordinary text'
```

Instructions, such as the point size instruction, that do not have a closing tag apply either until another instruction of the same type is read in the current text, or until the end of the current text. If you want a question text and its response list to be displayed in the same way, you must specify the formatting instructions at the start of the question text and at the start of each response in the response list. Alternatively, use one of the global formatting variables described later in this chapter.

-
- ☞ Text formatting in response lists becomes an integral part of the response texts. This is particularly important if you are using defined lists. In order for Quancept to match responses correctly across lists, responses must have identical formatting in all lists, since *Brand A* in one list is not the same as *<b+>Brand A<b->* in another.
-

There may be times when you want to change the default appearance of text by applying more than one formatting instruction at once; for example, you may want to change both the size and the color of the text. When this is the case, there is no need to create a separate tag for each instruction. Instead, you can list all of the instructions in one tag, separating each one by a semicolon. In the following example, the font, size and style of the question text are changed from the default with the instructions listed in one tag:

```
value      ask '<f:Arial;p:12;b+>Which one brand offered the  
best value over all?'
```

If you want to apply the same formatting to a number of texts, you can assign those formatting tags to a variable of your choice and then refer to those variables whenever you want to use a particular formatting command. For example:

```
setfmt qemphon = '<c:red';b+>'  
setfmt qemphoff = '<c:blue;b->  
everfly ask 'Which airlines have you [+qemphon+]ever[+qemphoff+]  
flown with?'  
        resp mp atoz airline  
recent   ask 'And which was the one you flew with [+qemphon+]most  
recently[+qemphoff+]?'  
        resp sp everfly in airline
```

Quancept CAPI provides a number of special temporary variables for defining global formats for question texts, response texts, displayed texts and protected texts. You may prefer to use these variables rather than your own variables if your specification refers to all texts within the questionnaire.

☞ For information on setting up global formats see 'Global text formats' later in this chapter.

Bold, italics, underlining and strikethrough



Quick Reference

To display text in bold, type: **<b+>text <b->**

To display text in italics, type: **<i+>text <i->**

To underline text, type: **<u+>text <u->**

To strike through text, type: **<st+>text<st->**

For example:

```
flavor   ask 'Which <b+>one</b-> flavor did you like the best?'  
        resp sp 'Strawberry' / 'Raspberry' / 'Peach' / 'Pineapple' /  
        'Apricot' / 'Mango' / 'None of them' dk ref
```

When this question and its response list are displayed, the word 'one' in the question text will be displayed in bold.

Changing the font



Quick Reference

To display text in a specific font, type:

`<f:font_name>text`

where *font_name* is the name of the character set. Font names that contain spaces must be enclosed in double quotes.

The Quancept CAPI interviewing program displays text in a font called MS Sans Serif. If you wish, you can display text in a different font by specifying the new font name in a tag. For example:

```
heard      ask 'Have you heard of a product called <f:Courier>Jingle?'
           resp sp 'Yes' / 'No' dk ref null
```

Here, the Courier font instruction will be applied until the end of the question's text. The list of possible responses will appear in MS Sans Serif.

-
- 〝 When choosing fonts, try to choose ones that are present on all interviewing machines. If you choose an unusual font that is not present on an interviewing machine, it is possible that the text will not appear as you expected. Common fonts are Arial, Courier, Helvetica and TimesNewRoman.
-

Changing the text size



Quick Reference

To specify the size of display text, type:

`<p:number>text`

where *number* is the point size at which you want to display the text.

The interviewing program displays text at a default size of eight points, but you can increase or decrease the point size as you wish. For example:

```
everfly ask '<p:10>Which airlines have you flown with in the last
         three years?'
         resp mp 'British Airways' / 'KLM' / 'Transavia' / 'TWA' /
                 'Virgin Atlantic' / 'Singapore Airlines' / 'Quantas' /
                 'Air France' / 'Japan Airlines' / 'Air India' /
                 'Lufthansa' / 'Malaysian Airlines' / 'Aeroflot' /
                 'Royal Jordanian' / 'Alitalia' / 'Air Canada'
```

displays the question text in ten-point type. The response list is treated as a separate text and is displayed at the default size.

If you find that you need to scroll down the screen to see the full set of responses, you may find that reducing the size of the text by one or two points makes everything visible without scrolling. If not, you may need to consider displaying the response list in multiple columns.

Changing the text color



Quick Reference

To apply color formatting to text, type:

`<c:fore [,back]>`

where *fore* defines the text's color and *back* defines the background color.

By default, the interviewing program displays black text on a white background, but you can use any colors from the following list:

black	lblue (light blue)	white
blue	purple	yellow
green	red	trans (background only)

For example:

```
everyfly ask '<c:blue>Which airlines have you ever flown with?'
          resp mp atoz airline
```

will display the question's text in blue against the default white background.

To display colored text against a background other than white, you need to specify the background color in the tag. For example:

```
everyfly ask '<c:blue,green>Which airlines have you ever flown with?'
          resp mp atoz airline
```

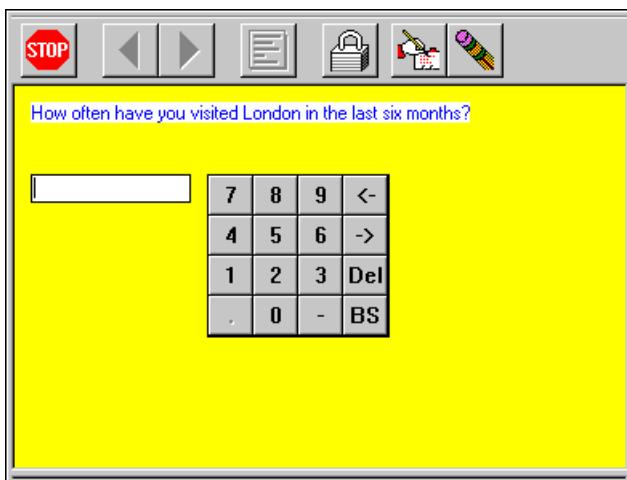
will display the question's text in blue against a green background. The green background sits behind the question text but does not alter the background color of the rest of the screen.

☞ For information on changing the background color for the whole interview screen, see 'Background colors and images' later in this chapter.

If you apply color formatting to a text that will be displayed against a background image or color that you have defined for the whole screen, you need to specify the background color as transparent. Colored text that does not have a transparent background color set is displayed against the default white background and so will appear to have a white surrounding border. For example, the following question:

```
visit      ask '<c:blue>How often have you visited London in the
           last six months?'
           resp num 0 to 20
```

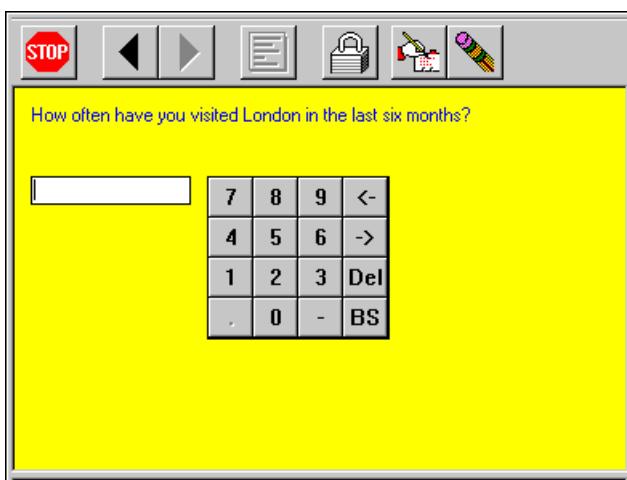
looks like this when displayed against a colored screen background:



To remove this border, you must specify the text's background color as transparent using the *trans* color keyword as follows:

```
visit      ask '<c:blue,trans>How often have you visited London
           in the last six months?'
           resp num 0 to 20
```

results in the following display:



Color instructions are applied either until the end of the current text or until a new color tag is found within that text.

Global text formats



Quick Reference

To define default characteristics for questions, responses and displayed and protected text, type:

All text: **setfmt winfmt='format[;format;...]'**

Question text: **setfmt winqfmt='format[;format;...]'**

Response text: **setfmt winrfmt='format[;format;...]'**

Displayed text: **setfmt windfmt='format[;format;...]'**

Protected text: **setfmt winpfmt='format[;format;...]'**

where *format* is any of the formatting characteristics described in the following sections.

Quancept CAPI has a number of built-in temporary variables that are associated with the appearance of text on the interview screen. The *winfmt* variable defines the basic characteristics of all texts in the questionnaire, whereas the other variables define characteristics that override or are additional to those defined with *winfmt*. For example:

```
setfmt winfmt='f:Arial; p:10'  
set winqfmt='c:red'  
q1 ask ...  
resp ...  
set winqfmt = 'f:TimesNewRoman'
```

specifies that all texts will be displayed in a nine-point arial font; question q1 will be displayed in a red, nine-point arial font; subsequent questions will be displayed in a black nine-point Times New Roman font. Notice that the red color does not carry forward from q1; instead, the font modification for subsequent questions is based on the original *winfmt* setting and the underlying system defaults.

When using these variables, bear in mind the following points:

- The variables are not parsed, so if you mistype a variable's name it will be treated like an ordinary variable and you will not obtain the desired behavior.
- For clarity, we recommend that you use *setfmt* to define these special variables and *set* for other temporary variables. (The script will still work if you define the formatting variables with *set* but it may not be so easy to follow.)

- All rules that apply to temporary variables also apply to these variables. For example, if you define a background color, the background remains set to that color until it is unset, even if the interviewer snaps back. If you intend to use these variables, you should therefore initialize them at the start of the interview by setting each variable to a blank string. For example:

```
setfmt wingfmt=''
```

Background colors and images



Quick Reference

To define the screen's background color, type:

```
setfmt bkcolor = 'color'
```

To use a picture rather than a color as the screen background, type:

```
setfmt bkground = '[path/]filename'
```

To define the way in which a background image is to be displayed, type:

```
setfmt bkstyle = 'style'
```

where *style* is **stretch** to increase the size of the image so that it fills the screen, **center** to display the image at its original size in the center of the screen, or **tile** to fill the screen with multiple copies of the image at its original size. The default is *tile*.

You can specify a background color or image for a script and a display style for background images using three predefined temporary variables. You can assign a value to these variables at any point in a script, enabling you to select a different background color or image for different sections of the interview.

Color

You can define a background color for a script either by assigning one of Quancept CAPI's predefined colors to the *bkcolor* variable or, by assigning an RGB (Red Green Blue) value to the variable, define your own color. RGB values are the values that computers use to define color.

The following predefined colors are available: white, black, red, green, yellow, lblue (light blue), purple and blue. For example:

```
setfmt bkcolor = 'blue'
```

sets the interviewing screen's background color to blue.

To create your own background color by setting an RGB value, type:

```
setfmt bkcolor = '#rrggbb'
```

where *rrggbb* are three hexadecimal numbers representing the RGB value, with the first pair of digits representing the red value, the second pair representing the green value, and the third pair, the blue. (Hexadecimal is a base-16 numbering system that computers use to represent bytes of information.) For example:

```
set bkcolor = '#FE64B4'
```

creates a pink background color for the interview screen whose red hexadecimal value is FE (=254 in base 10), green value is 64 (=100) and blue is B4 (=180). If you are using a Windows 95 or Windows NT computer, you can find the RGB value you need by looking at the decimal RGB values for your desired screen color on the Color dialog box. Then convert these decimal values to hexadecimal using the scientific calculator included with your Windows 95/NT system. Here are a few hexadecimal values to get you started:

black	000000
blue	0000FF
cyan	00FFFF
red	FF0000
green	00FF00
yellow	FFFF00

Images

To specify a background image for a script, set the *bkground* variable to the image's filename. For example:

```
setfmt bkground = 'gonefishing.bmp'
```

If the image is not in the same directory as the script you must enter the file's full pathname.

By default, background images are tiled to fill the interview screen:



You can specify a different style for a background with the *bkstyle* variable. For example:

```
setfmt bkstyle = 'centre'
```

displays the image in the center background of the interview screen (notice the British spelling of the word 'centre'), while

```
setfmt bkstyle = 'stretch'
```

displays the image in the center of the screen but stretches it so that it fills the whole interviewing screen.

- ☞ If you display color formatted text against a background image or color, you need to format the text with the *trans* keyword to ensure that the text does not appear with a white surrounding box.
- ☞ You can also set a default background color or image by entering a custom option on QCompile's Special table. For more information, see 'The Special tab — additional runtime options' in chapter 32, 'Parsing and compiling scripts'.

22.2 Screen design using HTML code



Because they run from your Web page, Quancept Web and mrInterview provide you with a wide range of controls over the appearance of text on the interviewing screen. This means that you can increase or decrease the size of characters in the text and change their style and color. For example, you can display text in bold or italics if you want to add emphasis. Alternatively, you may wish to set up a standard where instructions for the user are always displayed in italics or in a contrasting color. HTML also allows you to display pictures either as part of the question text or as part of individual response texts.

- ❖ Formatting in Quancept Web and mrInterview scripts is done using the standard HTML tags used in the creation of any Web page. You'll find full details of these commands in any HTML reference guide. The notes in this section introduce only the most common requirements for scripts.
-

How to use HTML tags in scripts



Quick Reference

- ❖ To define the format for a piece of text, type a formatting tag enclosed in angle brackets before the text and another one after it.
-

For example:

```
d1 display '<font size="4">Text slightly larger than normal</font>'  
d2 display '<b>Bold text</b> and ordinary text'
```

When you use the ** control you can specify more than one format at a time by listing your requirements separated by spaces and enclosing the whole specification in a single set of angle brackets. For example:

```
value ask '<font face="Arial" size="4">Which <b>one</b> brand  
offered the best value over all?</font>'
```

displays the whole question text in size 4 Arial characters, with the word 'one' in bold.

-
- ❖ Text formatting in response lists becomes an integral part of the response texts. This is particularly important if you are using defined lists. In order for Quancept to match responses correctly across lists, responses must have identical formatting in all lists, since *Brand A* in one list is not the same as *Brand A* in another.
-

Some formatting strings can be quite long and can easily be mistyped if you're retyping the same strings many times. The Quancept Web and mrInterview compilers treat all HTML tags as simple text strings and places them in the qti (Quancept Web only) and sif files ready for interpretation by the browser. This is an advantage for you, since it means that you can assign the HTML tags you commonly use to variables and then refer to those variables whenever you want to use a particular formatting command.

For example, suppose your company standard is to display all question texts in red Arial characters with a blank line between the question text and the response list. A quick way to achieve this is as follows:

```
set s = '<font color="red" face="Arial">'  
set e = '</font><p>'  
everfly ask s+'Which airlines have you ever flown with?'+e  
resp mp atoz airline  
recent ask s+'And which was the one you flew with most recently?'+e  
resp sp everfly in airline
```

-
- ☞ Although this works in mrInterview, it is better to use mrInterview's templates and cascading stylesheets capabilities to define formatting, since this separates the formatting from the script content.
 - ☞ See chapter 23, Using page layout templates in mrInterview, and chapter 24, Templates for grids, for further information.
-

Bold, italics and underlining



Quick Reference



To display text in bold, type: '**b** text **/b**'

To display text in italics, type: '*i* text */i*'

To underline text, type: 'u text /u'

The letters b, i and u may be entered in upper or lower case.

To display text in bold or italic characters or underlined, use the b, i or u code pairs. Type the first code in front of the text to switch on the format you require, and type the second code at the end of the text to switch off the special format. For example:

```
favor ask 'Which <b>one</b> flavor did you like the best?'  
resp sp 'Strawberry' / 'Raspberry' / 'Peach' / 'Pineapple' /  
'Apricot' / 'Mango' / 'None of them' dk ref
```

When this question and its response list are displayed, the word 'one' in the question text will be displayed in bold.

-
- ☞ The Web uses underlining to mark hyperlinks so you may prefer not to use underlining in scripts.
-

Line breaks and paragraphs



Quick Reference



To start a new line in a text, type **
**.

To start a new paragraph, type **<p>**.

HTML has its own codes for line breaks and paragraphs. These work rather like similar commands in word processing packages, where a line break starts a new line with no space between the two lines, and a paragraph mark starts a new line with some amount of spacing between the two paragraphs.

The following examples show two methods of achieving the same visual effect. The first example shows the standard Quancept way of starting a new paragraph, while the second example shows how to do it using HTML code:

```
know1   display 'The sports you mentioned were:@@'+sports  
know2   display 'The sports you mentioned were:<p>'+sports
```

Color, typeface and character size changes in texts



Quick Reference



To define the color, typeface and/or character size for a text, type:

```
'<font [color="color"] [face="typeface"] [size="number"]> text </font>'
```

where *number* is an integer between 1 and 7. The usual default is 3.

Web interviewing gives you almost unlimited options with regard to the color, typeface and size you use for text. Not only do these types of changes make the screen visually more interesting, they also allow you to make the questionnaire easier to follow. As an example, you may want to implement a standard where instructions to the respondent are displayed in red or in a different typeface, so that they are easily distinguishable from question and response texts.

You make all changes to color, typeface and size using the **font** command. This is an on/off switch rather like **** and **** that you saw earlier, so you'll need a **** command for each **** specification you make.

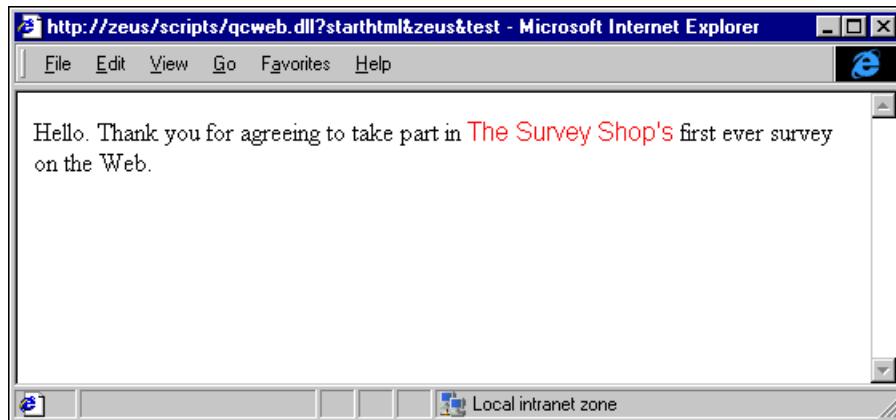
The **font** command can be followed by one or more parameters, depending on the types of changes you want to make. You may type these parameters in any order as long as you separate them from each other with spaces. Keywords for use with **font** are as follows:

Type	To
color="color"	define the color for the following text.
face="typeface"	define the typeface for the following text.
size="number"	set the character size to <i>number</i> . The default size is 3 and values between 1 and 7 are valid. The larger the number, the less text you'll fit on the screen.

Here is an example that uses color and typeface to highlight the company name:

```
intro    display 'Hello. Thank you for agreeing to take part in
<font color="red" face="Arial">The Survey Shop's</font> first ever
survey on the Web.'
```

This displays as:



-
- ❖ Changes of this type are supported by all recent Microsoft and Netscape browsers. If you are using an old version of a browser, you may find that these changes are not visible even though you have programmed them correctly. To allow support for all Web browsers, you may have to use the RGB values for colors; that is, you'll need to specify colors by defining the hexadecimal values of their Red, Green and Blue content. For example:

```
<font color="#ffffff> instead of <font color="white">
<font color="#000000> instead of <font color="black">
```

Default characteristics for questions, responses, display and protect texts



Quick Reference



To define default characteristics for questions, responses and displayed and protected text, type:

Question text: **setfmt qfmt='<start_tag>%s[<end_tag>]'**

Response text: **setfmt rfmt='<start_tag>%s[<end_tag>]'**

Displayed/protected text: **setfmt dfmt='<start_tag>%s[<end_tag>]'**

start_tag is the standard HTML tag for the effect you want. *end_tag* is the end tag that is sometimes needed to switch off the effect. *%s* represents the text and must be entered exactly as shown here.

In mrInterview, to define default global formatting for *null*, *dk*, *ref* and *other*, type:

null/dk/ref: **setfmt fmt='<start_tag>%s[<end_tag>]'**

other: **setfmt ofmt='<start_tag>%s[<end_tag>]'**

(A variable called *pfmt* has been reserved for defining formatting characteristics for protected texts but it is currently unimplemented. This means that you cannot use this name for a question or other variable in the script.)

Most browser screens are not particularly visually appealing and can often be made to look more interesting by the careful use of color, fonts and different character sizes. Many companies already have Web pages and are implementing standards that project the company image. You can implement the same or similar standards in Web surveys.

Normally, all font and color changes have to be specified separately each time, but Quancept Web and mrInterview provides a set of special variables that allow you to set up defaults for an entire survey. For example, if your company standard is to display all question texts in red Arial characters with a blank line between the question text and the response list, you simply assign these formatting commands to the appropriate variable and the interviewing program will apply them to every question it displays.

Here is how to set up the defaults that were used as an example earlier, where all question texts were displayed in red Arial characters with a blank line between the question text and the response list:

```
        set qfmt = '<font color="red" face="Arial">%s</font><p>'  
everfly ask 'Which airlines have you ever flown with?'  
        resp mp atoz airline  
recent  ask 'And which was the one you flew with most recently?'  
        resp sp everfly in airline
```

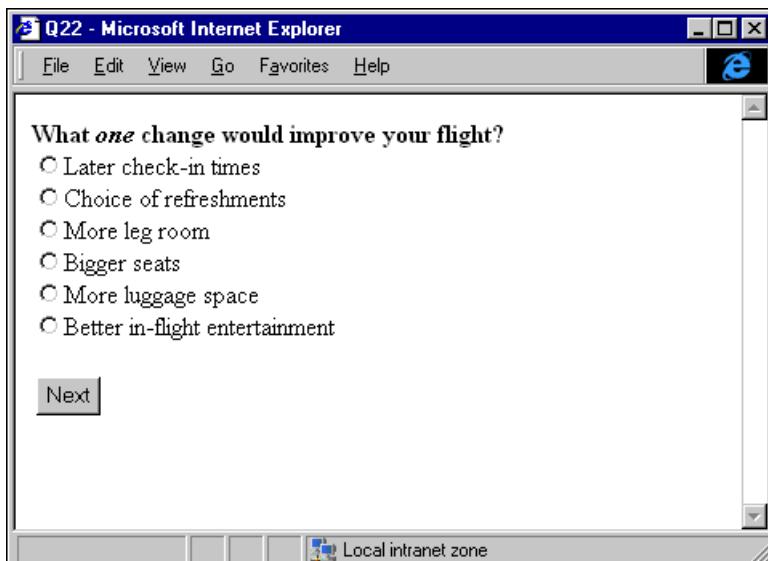
The next example shows how to display the survey title in bold and question texts in blue:

```
setfmt dfmt = '<b>%s</b>'  
setfmt qfmt = '<font color="blue">%s</font>'  
top protect 'SPSS Airline Study@@'  
q1 ask 'Was your last flight for business or pleasure?'  
resp sp 'Business' / 'Pleasure' / 'Both'
```

Defining default settings for a text type does not prevent you adding to or overriding those settings for a complete text or for portions of a text. In the following example, the question text is displayed in bold but the word 'one' has an additional italic setting applied:

```
setfmt qfmt = '<b>%s</b>'  
q22 ask 'What <i>one</i> change would improve your flight?'  
resp sp 'Later check-in times' / 'Choice of refreshments' /  
      'More leg room' / 'Bigger seats' /  
      'More luggage space' / 'Better in-flight entertainment'
```

The effect on the screen is as follows:



Positioning text on the page



Quick Reference



To center text within the line, type:

```
'<center> text </center>'
```

For example, you may want to display a reminder to the respondent at the top of a screen. You could type:

```
resid    dummyask 'What type of residence do you live in?'
                  resp sp  'Flat' / 'Terraced house' / 'Semi-detached house' /
                           'Detached house' / 'Bungalow'
own      dummyask 'And is this residence ....'
                  resp sp  'Owned outright' / 'Mortgaged' / 'Leased' /
                           'Rented' / 'Other'
both     multiask '<center>Please be sure to answer both questions
                  </center><p>' resid / own
```

In this example, the two questions are displayed on the same screen with the text 'Please be sure to answer both questions' displayed centrally on the line above the first question.

Tabular layouts



Quick Reference



To display text in tabular format, use the following commands at the relevant points in the text:

<table>...</table>	to mark the start and end of the text to be treated as a table
<td>...</td>	to mark the start and end of a column
<tr>...</tr>	to mark the start and end of a row

Tabular or columnar layouts are generally appropriate only for response lists, and Quancept provides its own keywords for these layouts. You have *colgrid* and *rowgrid* for displaying responses as grids and *hsetcols/vsetcols* for specifying the number of columns to use for long response lists.

One situation that Quancept does not provide for directly is displaying the questions belonging to a *multiask* side-by-side rather than one under the other. Often, this means that a response list is only partially visible or that a question and its response list are not visible unless the respondent scrolls down the screen. You may be able to make this part of the questionnaire more user friendly by displaying the questions and their response lists side by side.

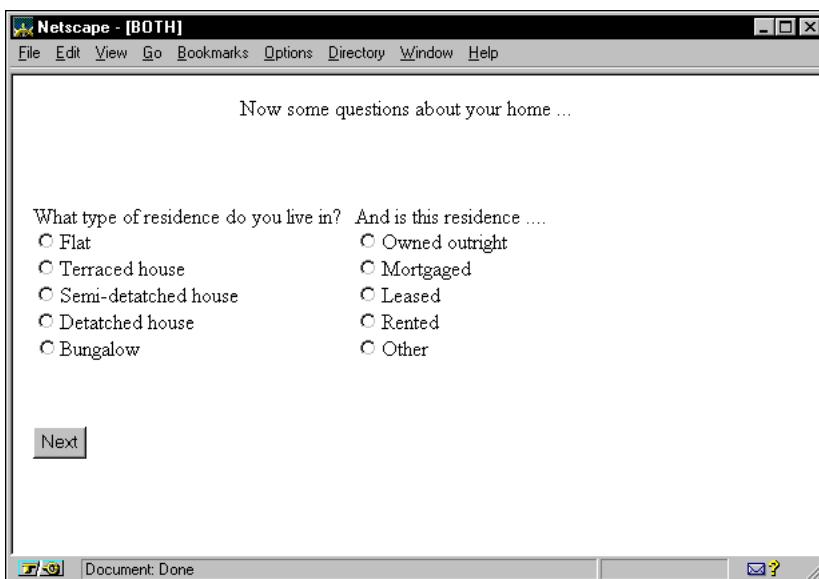
Here's the *multitask* example you saw earlier, but this time it is defined so that each question and its response list form one column of a table:

```

resid      dummyask '<table><tr><td>
    What type of residence do you live in?'
        resp sp  'Flat' / 'Terraced house' / 'Semi-detached house' /
                    'Detached house' / 'Bungalow</td>'
own       dummyask '<td>And is this residence ....'
        resp sp  'Owned outright' / 'Mortgaged' / 'Leased' /
                    'Rented' / 'Other</td></tr></table>'
both      multitask '<center>Now some questions about your home ...</center>'
resid / own

```

The screen that the respondent will see looks like this:



-
- ☞ If you are creating a new script for mrInterview, define the page layout using a template rather than placing the HTML code in the script itself.
 - ☞ See chapter 15, Working with numeric data, for details.
-

The table tags enclose the whole text that belongs in the table. The *<table>* tag is at the start of the first dummy question text. The *</table>* tag is at the end of the last response in the second dummy question's response list. Notice that the *multitask* text is not part of the table. Instead, it is centered above the table.

The *multitask* has two questions, so the table has two columns. The first column is the first dummy question and its response list. The *<td>* at the start of the question text marks the beginning of the column, and the *</td>* at the end of the last response in the response list marks the end of the column. The second column is tagged in a similar manner in the second dummy question.

The most important thing to notice is that each column is defined as having only one row even though the question and response texts span several lines. This is because Quancept has its own built-in method of displaying questions and response lists, so there is no need for you to specify anything different. However, since a table must have at least one row, the *<tr>* and *</tr>* tags enclose both questions and response lists in the same way as the table tags.

Indenting response lists



Quick Reference



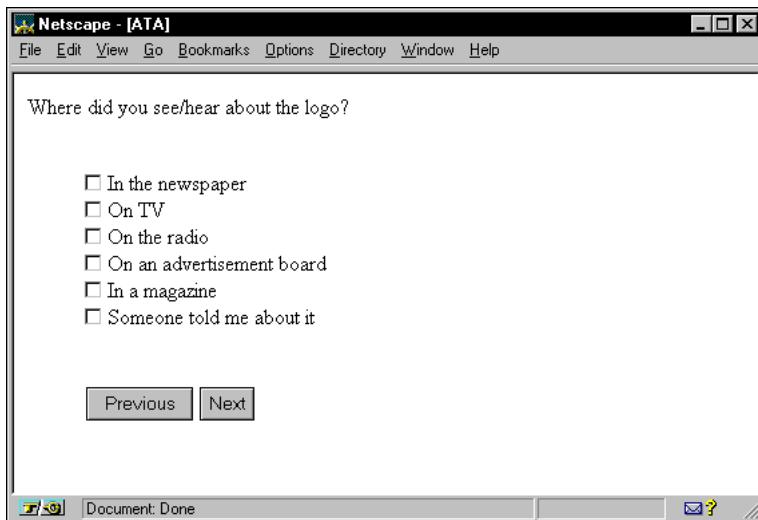
To indent a response list, type:

```
label ask 'question text<ul>
          resp resp_type 'resp1' / 'resp2' / ... / 'respn</ul>'
```

Quancept Web normally starts printing response lists in the first position on the line. If you want all response lists in the script to be indented, just type ** at the end of the first question text. For example:

```
ata      ask 'Where did you see/hear about the logo?<ul>
           resp mp 'In the newspaper' / 'On TV' / 'On the radio' /
                  'On an advertisement board' / 'In a magazine' /
                  'Someone told me about it'
```

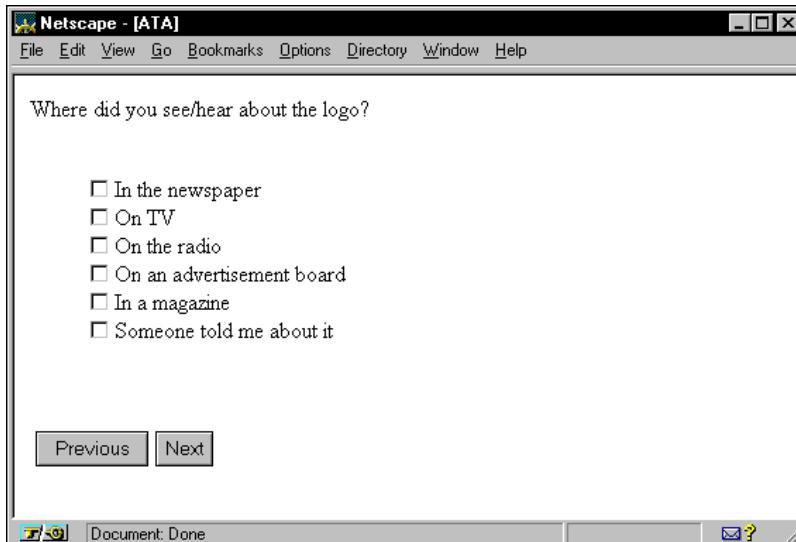
produces:



Notice that the Previous and Next buttons are treated as part of the response list and are indented. If you would prefer to have the buttons printed on the left as usual, you'll need to terminate the indentation by placing a tag at the end of the last response text in the list.

```
ata      ask 'Where did you see/hear about the logo?<ul>'  
resp mp 'In the newspaper' / 'On TV' / 'On the radio' /  
       'On an advertisement board' / 'In a magazine' /  
       'Someone told me about it</ul>'
```

The display will then look like this:



Choosing the background and default text colors



Quick Reference

To set the background and foreground (text) colors for a script, place the statement:

Header=<body [bgcolor="color"] [text="color"]>

in the [HTML] section of project's ini file.

Quancept scripts are displayed using the default colors set for the browser, but you can change this on a script by script basis if you wish. Unlike other color changes, these ones are made in the project's ini rather than in the script itself. This is a text file that you create in the project's home directory on the Quancept Web server. For example, if your project is called qctest, its ini file will be *pathname\projects\qctest\qctest.ini*.

To define a background color for the script, type:

Header=<body bgcolor="*color*">

in the [HTML] section of the file. To define a default text color, type:

Header=<body text="*color*">

If you're defining both colors, a single Header line naming both parameters is sufficient:

```
[HTML]  
Header=<body bgcolor="White" text="23238E">
```

This example displays navy blue text on a white background.

☞ The default text color can be overridden by a ** tag in a question or response text.

22.3 Input boxes for open ended and numeric responses



When they reach a numeric or open-ended question, Quancept Web and mrInterview display the question text followed, on the next line, by an input box in which the respondent may type an answer. The input box for numerics is one line high and is as wide as is necessary to hold the highest possible response, whereas the input box for open ends is 40 characters wide by six rows deep. The boxes are preceded by the word Answer: in a bold font.

You can change the width of numeric boxes and both the height and width of boxes for open ends. You can also suppress the Answer: prompt.

Multi-line boxes for open ends



Quick Reference

To display the standard 40-character by six-row input box, but without the Answer prompt, type:

set htmlfmt='textarea, default, noprompt'

To define the size of the box displayed for open ends, type:

set htmlfmt='textarea, *numcols*, *numrows*[, *noprompt*]'

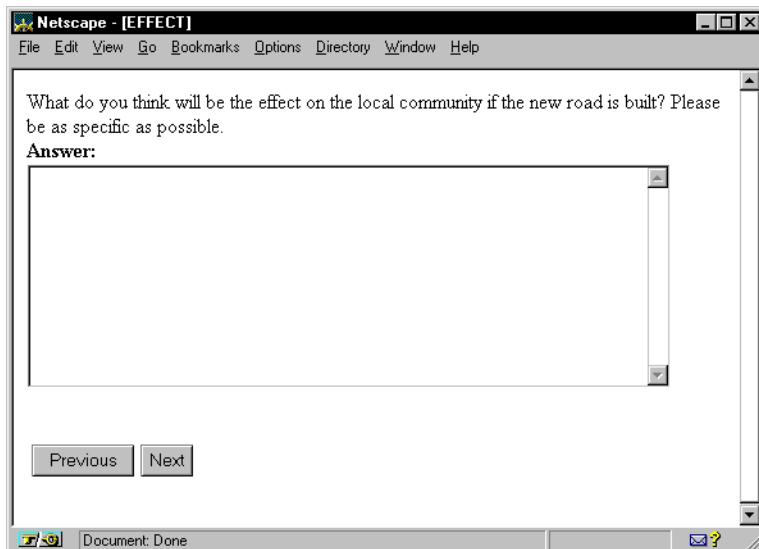
for a box *numcols* wide by *numrows* high.

When they encounter a question with an open-ended response list, Quancept Web and mrInterview display a box that is 40 characters wide by six rows deep for the respondent to type in an answer. Text is wrapped onto the next line as soon as it reaches the end of the current line, so there is no need for the respondent to press Return at the end of each line. Also, the display scrolls if the respondent fills the box so that there is always a free line left on which to type.

With Netscape, any text the respondent types is passed to the server as one long string of characters with no line breaks in it. With Internet Explorer, texts are passed with line breaks represented by \n.

You may find that the size of the box limits the amount of text that respondents will enter, and you'll want to increase the size of the box to encourage them to write in more detail. You can do this using the *htmlfmt=textarea* assignment statement. The size you make the box will depend on the size of the question text and the size of the respondent's screen.

```
set htmlfmt='textarea,60,10'
effect ask 'What do you think will be the effect on the local community
if the new road is built? Please be as specific as possible.'
resp coded
```



This example uses a box width that fits into the screen width; if the box had been wider, there would have been a horizontal scroll bar too.

-
- ❖ The keyword *htmlfmt* should appear before the first open end that requires the new setting. This then applies to all open ends until the end of the questionnaire or until another *mt* statement is read.

The widths of the open-end text boxes displayed with *multiask* are set at the time the *multiask* statement is executed, not the time that the *dummyask* statements are executed.

22.4 One-line boxes for numerics and open-ends



Quick Reference

To display a one-line box for numerics or open-ends, type:

```
set htmlfmt='textline, default[, noprompt]'
```

For numerics, the width of the box is determined by the maximum value that may be given in response to the question, whereas for open ends the box is always 40 characters wide. The boxes are preceded by the word Answer: Use *noprompt* to display an input box with no prompt.

To change the width of the input box, type:

```
set htmlfmt='textline, width[, noprompt]'
```

for a one-line box that is *width* characters wide.

You may want to alter the size of the open-end text box to display the equivalent of a form that the user fills in, perhaps with name and address information that you use later for sending out trial samples. Since the information you're gathering here is relatively short, it makes sense to display all the questions on one screen using *multitask* but with a reduced text area for the responses.

When you want an open-end text box to be only one line high, you use an *htmlfmt=textline* statement that defines just the number of columns required. Here's what you might type to create a name and address form for the respondent to complete. Notice how the box titles have been defined on the *dummyask* statements.

```
set htmlfmt='textline, 45, noprompt'
name    dummyask 'Name' resp coded
addr1   dummyask 'Address1' resp coded
addr2   dummyask 'Address2' resp coded
city    dummyask 'City' resp coded
PCODE   dummyask 'Postcode' resp coded
info    multitask 'Please enter your name and address in the boxes below
                  to receive a voucher for 250 airmiles as a token of our appreciation.'
name / addr1 / addr2 / city / PCODE
```

The screen that the respondent sees is as follows:

INFO - Microsoft Internet Explorer
File Edit View Go Favorites Help

SPSS Airline Survey

Please enter your name and address below to receive a voucher for 250 air miles as a token of our appreciation.

Name

Address1

Address2

City

Postcode

If you would prefer to display the text boxes without labels just omit the text from the dummy questions.

In this example, the text boxes are 45 characters wide. Texts longer than this are accepted but the display scrolls to the left as the respondent types. He/she can move backward and forward through long texts using the arrow keys.

22.5 Controlling the display of the Previous and Stop buttons



Quick Reference

To control the display of the Previous and Stop buttons from within the script, type:

set hiderev = 1

to hide the Previous button, and:

set hidestop = 1

to hide the Stop button.

To reinstate the display of these buttons, type:

set hiderev=0 or set hidestop=0

Quancept Web and mrInterview always display a Next and Previous button if appropriate, but do not display a Stop button unless specifically requested to do so. In Quancept Web you request a Stop button by placing a statement in the project's initialization file, and then control its display using *hidestop* in the script. In mrInterview you can request a Stop button by placing *set hidestop=0* in the script.

-
- ☞ See section 40.3, The project initialization file, for information about Quancept Web's initialization file.
-

22.6 Sample scripts

Script A



```
comment Sample script A for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 22, Screen management with graphical user interfaces
comment (Web/mrInterview only)

airline define 'British Airways' / 'KLM' / 'Transavia' / 'Garuda' /
    'Virgin Atlantic' / 'Singapore Airlines' / 'Quantas' /
    'Air France' / 'Japan Airlines' / 'Air India' /
    'Lufthansa' / 'Malaysian Airlines' / 'Aeroflot' /
    'Royal Jordanian' / 'Alitalia' / 'American Airlines' /
    'Air Canada' / 'DanAir' / 'Delta Airlines' / 'Sabena' /
    'Emirates Airlines' / 'Aeromexico' / 'Finnair' / 'SAS' /
    'BWIA' / 'Maersk Air' / 'Air Lanka' / 'Swiss Air' /
    'Pakistan Intern'l Airways' / 'Royal Brunei' /
    'Royal Nepal' / 'Air China' / 'Air New Zealand' /
    'Air Zimbabwe' / 'Cathay Pacific' / 'Aer Lingus' /
    'Austrian Airlines' / 'United Airlines' / 'Iberia' /
    'Mexicana Airlines' / 'Air Madagascar' / 'Other'

comment Display all question texts in a bold font
setfmt qfmt = '<b>%s</b>'

comment Centered page header displayed in large green Arial
comment font. Header is followed by two blank lines.
hdr protect '<center><font size="5" color="green" face="Arial">
SPSS Airline Survey</font></center><p><p>'

everfly ask 'Which airlines have you flown with in the past
three years?'
resp mp atoz airline vsetcols=4

recent ask 'And which was the one that you flew with most recently?'
resp sp everfly in airline

comment Indent response lists but not the Next/Previous buttons
club ask 'Are you a member of any airline''s privilege schemes?<ul>'
resp sp 'Yes' / 'No</ul>' go ctrl
level ask 'Which level of privileges do you have?<ul>'
resp sp 'Silver' / 'Gold</ul>'
ll for colgrid '' i = 'Available' / 'I use'
privs ask 'What privileges does your '+level+' membership give you,
and which of them do you use?'
resp mp 'Extra baggage allowance' / 'Fast check-in' /
    'Executive lounge' / 'Priority booking' /
    'Cheaper flights' / 'Free flights / Air miles' /
    'Airport shopping discounts' null dk ref
next
```

Quancept and mrInterview Scriptwriter's Manual

```
comment Set size of open-end text box to 50 characters wide by 15
comment lines long to encourage detailed responses
    set useexec = bit(privs(2)/3)
    if (useexec)
    {
        set htmlfmt = 'textarea,50,15'

lŋlike      ask 'What are the things you like most about the
executive lounge?'
            resp coded
lŋhate      ask 'And what are the things that you like the least?'
            resp coded
    }

ctn1      continue
whyfly    dummyask 'Are most of the flights you take made for ... ?'
            resp sp 'Business' / 'Pleasure'

comment Set size of open-end text box to one line of 50 characters
        set htmlfmt = 'textline, 50'

comment Forced line break between sentences. Override default
comment font color for special instructions.
airport   dummyask '<p>From which airport do you normally fly?<br>
<font color="red">Please enter both the name and the city, if
appropriate. For example, London, Gatwick or Amsterdam, Schipol</font>'
            resp coded

m1       multiask '' whyfly / airport

comment Display an image
logo      ask 'Here is the logo of the newly formed Air Travellers'
          Association.<p>
<center></center><p>
Have you see this logo anywhere or heard about this association
within the last three months?<ul>
            resp sp 'Yes' / 'No</ul>' go ctn2
comment Forced line break in text and show some text in italics
ata       ask 'Where was that?<br><i>Please select all that
apply.</i><ul>'
            resp mp 'In the newspaper' / 'On TV' / 'On the radio' /
                  'On an advertisement board' / 'In a magazine' /
                  'Someone told me about it</ul>'

ctn2      continue

comment Tabular layout for multiask questions
age       dummyask '<table><tr><td>How old are you?'
            resp num 18 to 99
gender    dummyask '<td>Are you .... ?'
            resp sp 'Male' / 'Female'
```

```
mstat  dummyask '</tr><tr><td>Are you .... ?'
      resp sp 'Single' / 'Married/Living with partner' /
              'Divorced' / 'Separated' / 'Widowed'
income dummyask '<td>What is your annual household income?'
      resp sp 'Under 20,000' / '20,000 to 29,999' /
              '30,000 to 39,999' / '40,000 to 49,999' /
              '50,000 or more</tr></td></table>'

demog  multiask 'And now a few demographic questions to help us
categorize your responses' age / gender / mstat / income

name   dummyask '' resp coded(0)
addr1  dummyask '' resp coded(0)
addr2  dummyask '' resp coded(0)
addr3  dummyask '' resp coded(0)

comment Set up open-end text box to look like a form by
comment making it one line long and suppressing the prompt
      set htmlfmt = 'textline, 60, noprompt'

details multiask 'Thank you for your time in completing this
questionnaire. Please enter your name and address below if you would
like us to send you a voucher for 250 air miles as a token of our
appreciation.<p><p><p>' name / addr1 / addr2 / addr3

end
```

Notes for Quancept Web users

This script comes with an initialization file, ch22a.ini, and an image file, atalogo.gif. The initialization file contains the following lines:

```
[HTML]
Header=<body bgcolor="White" text="23238e">
```

This sets the default background and text colors to be white and navy blue respectively.

Copy ch22a.ini into the project directory.

Copy atalogo.gif into the project directory and amend the reference to this file in the script to show the full pathname of this file (use / rather than \ in the pathname).

Notes for mrlInterview users

Edit the script and change the reference to the image file atalogo.gif so that it includes the full pathname of the project's working directory (use / rather than \ in the pathname). After you have activated the script, copy atalogo.gif into the project's working directory.

Script B



This script uses an image file, atalogo.bmp, which should be copied into the project directory.

```
comment Sample script B for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 22, Screen management with graphical user interfaces
comment (CAPI only)

airline define  'British Airways' / 'KLM' / 'Transavia' / 'Garuda' /
                 'Virgin Atlantic' / 'Singapore Airlines' / 'Quantas' /
                 'Air France' / 'Japan Airlines' / 'Air India' /
                 'Lufthansa' / 'Malaysian Airlines' / 'Aeroflot' /
                 'Royal Jordanian' / 'Alitalia' / 'American Airlines' /
                 'Air Canada' / 'DanAir' / 'Delta Airlines' / 'Sabena' /
                 'Emirates Airlines' / 'Aeromexico' / 'Finnair' / 'SAS' /
                 'BWIA' / 'Maersk Air' / 'Air Lanka' / 'Swiss Air' /
                 'Pakistan Intern'l Airways' / 'Royal Brunei' /
                 'Royal Nepal' / 'Air China' / 'Air New Zealand' /
                 'Air Zimbabwe' / 'Cathay Pacific' / 'Aer Lingus' /
                 'Austrian Airlines' / 'United Airlines' / 'Iberia' /
                 'Mexicana Airlines' / 'Air Madagascar' / 'Other'

comment Interview defaults are blue text on a white background
setfmt bkcolor = 'white'
setfmt winfmt = 'c:blue'

comment Display all question texts in a 10-point bold font
setfmt wingfmt = 'p:10; b+'

comment Page header displayed in 12-point green Arial
comment font. Header is followed by two blank lines.
hdr      protect '<p:12><c:green><f:Arial>SPSS Airline Survey@@'

everfly ask 'Which airlines have you flown with in the past
            three years?'
            resp mp atoz airline vsetcols=4
recent   ask 'And which was the one that you flew with most recently?'
            resp sp everfly in airline
club     ask 'Are you a member of any airline''s privilege schemes?'
            resp sp 'Yes' / 'No' go ctn1
level    ask 'Which level of privileges do you have?'
            resp sp 'Silver' / 'Gold'

11      for colgrid '' i = 'Available' / 'I use'
privs    ask 'What privileges does your '+level+' membership give you,
            and which of them do you use?'
            resp mp 'Extra baggage allowance' / 'Fast check-in' /
                  'Executive lounge' / 'Priority booking' /
                  'Cheaper flights' / 'Free flights / Air miles' /
                  'Airport shopping discounts' null dk ref
next
```

```
        set useexec = bit(privs(2)/3)
        if (useexec)
        {
            lnglike    ask 'What are the things you like most about the
                         executive lounge?'
                        resp coded
            lnghate    ask 'And what are the things that you like the least?'
                        resp coded
        }

        ctn1    continue
        whyfly   dummyask 'Are most of the flights you take made for ... ?'
                    resp sp 'Business' / 'Pleasure'

comment Override default font color for special instructions.
airport  dummyask '@From which airport do you normally fly?@
<c:red>Please enter both the name and the city, if
appropriate. For example, London, Gatwick or Amsterdam, Schipol'
        resp coded

m1      multiask '' whyfly / airport

comment Display an image
        display 'Here is the logo of the newly formed Air Travellers'
                Association.@' mmpicture='atalogo.bmp'
logo    ask 'Have you seen this logo anywhere or heard about this association
within the last three months?'
        resp sp 'Yes' / 'No' go ctn2

comment Show some text in italics
ata     ask 'Where was that?@<i+>Please select all that
apply.<i->'
        resp mp 'In the newspaper' / 'On TV' / 'On the radio' /
                'On an advertisement board' / 'In a magazine' /
                'Someone told me about it'

ctn2    continue

age     dummyask 'How old are you?'
        resp num 18 to 99
gender  dummyask 'Are you .... ?'
        resp sp 'Male' / 'Female'
demogl  multiask 'And now a few demographic questions to help us
categorize your responses' age / gender

mstat   dummyask 'Are you .... ?'
        resp sp 'Single' / 'Married/Living with partner' /
                'Divorced' / 'Separated' / 'Widowed' hsetcols=5
income   dummyask 'What is your annual household income?'
        resp sp 'Under 20,000' / '20,000 to 29,999' / '30,000 to 39,999' /
                '40,000 to 49,999' / '50,000 or more' hsetcols=5
```

Quancept and mInterview Scriptwriter's Manual

```
demog2 multitask '' mstat / income

details ask 'Thank you for your time in completing this
questionnaire. Please enter your name and address below if you would
like us to send you a voucher for 250 air miles as a token of our
appreciation.'

resp coded(0)

end
```

23 Using page layout templates in mrInterview

Page layout in mrInterview is controlled using a combination of standard HTML tags and customized tags that are unique to mrInterview. You can specify formatting and other characteristics for individual questions or parts of questions simply by placing HTML tags at the relevant places in the script, but the real power of mrInterview's page layout facilities comes when you use templates.

Templates allow you to define the layout of the questionnaire page separate from the page content, and to re-use the same template for a number of projects. You can use any number of templates in a project, switching between them according to the layout required for a particular page. Typically, you use three templates, one for standard pages, one for pages that display multitask questions, and one for pages that display grids. You switch to the multitask or grid template only when there is a multitask or grid to display.

mrInterview also supports cascading style sheets. These allow you to define global formatting characteristics that apply to all templates, so that, for example, you can always have dark blue, 12-point question texts regardless of how or where the question text is displayed by your templates.

This chapter describes the following keywords for use in the mrInterview script.

curpgstyle	Name the page layout to use
pgstyle()	Name a page layout file
pagstyle()	Name a page layout file
ptitle	Assign a value to <i><mrPageTitle></i>

It also describes the following tags for use in templates:

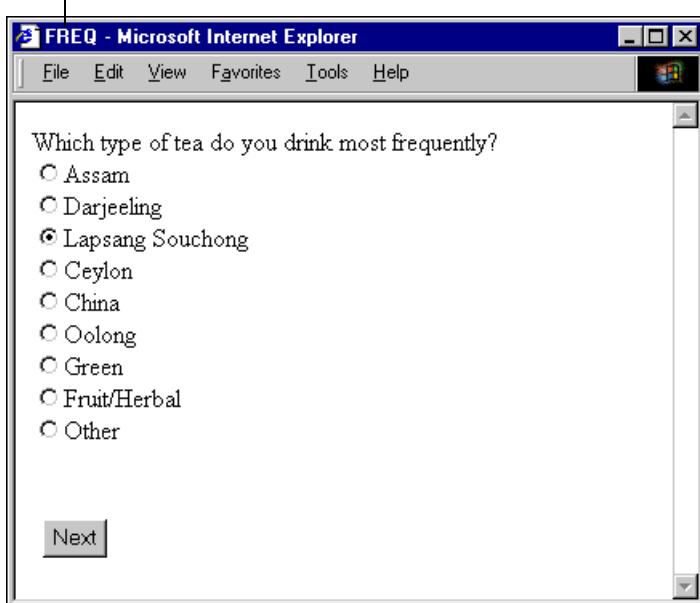
<mrData>	Print the question text, response block and error message, if any
<mrData /TextField>	Print the question text
<mrData /Controls>	Print the response block
<mrData /ErrorField>	Print error messages when necessary
<mrDisplayText>	Print text from <i>display</i> statements
<mrNavbar>	Print the standard navigation buttons
<mrNavbutton>	Print customized navigation buttons
<mrPageTitle>	Print the question name
<mrProtectText>	Print text from <i>protect</i> statements

« This chapter assumes basic familiarity with HTML.

23.1 Templates — an overview

Template files define the page layouts for mrInterview questionnaires. The standard page layout if you do not use templates is as follows:

Name of current question



If the script contains *protect* or *display* statements, these texts are displayed above the question text in the order they occur in the script. If the user selects or enters invalid responses, an error message is displayed in red at the top of the page.

This page layout is functional but uninteresting. You can produce layouts that are much more attractive and easy to read by defining your own templates.

Contents of a template

The page layout template defines some or all of the following:

- The page title.
- The position and appearance of texts and controls (responses lists, for example) on the page.
- The position in which error messages are to be displayed and their appearance.
- The position and appearance of *displayed* and *protected* texts.

- Which navigation buttons are to be displayed and their positions on the page.
- Any fixed text, such as the webmaster's mail address.

You define the position of these items on the page using special *mr* tags enclosed in angle brackets, in the same way that you use standard HTML tags. These tags are described in the rest of this chapter.

Unlike word processing or desktop publishing template files which define a number of styles, each mrInterview template file defines only one page layout. If you want to use different page layouts within a questionnaire you need to create a separate template file for each one.

Creating templates

Template files are text files that have a .htm extension. You can create a very basic template file using a text editor such as Notepad or Wordpad; just enter the special *mr* tags that define the items you want to appear on the interview page and save the file with a .htm extension.

-
- ¶ If the template contains non-English characters such as characters with umlauts, you must save the template in UTF-8 or Unicode format otherwise these characters will be displayed as ?.
-

For example, a template that produces the same page layout as the default is:

```
<div><mrProtectText></div>
<div><mrDisplayText></div>
<mrData>
<br><br>
<mrNavbar>
```

The first and second lines define the positions of *protected* and *displayed* text respectively. The third line displays the question text and response list or response box, and any error message that is required. The last line displays Previous and Next buttons for navigating through the questionnaire.

As you have seen, this does not produce very interesting pages, so you will probably want to include standard HTML formatting tags to insert font, point size and color changes and to define the exact position of the components on the page. You can do this either by typing the HTML tags into a standard text file, or by using a program such as FrontPage that converts your formatting requests into HTML code. For example:

```
<html>
  <head></head>
  <body>
    <center><b><font color="blue" size="4" face="Arial">
      SPSS MR -- Tea Survey
    </font></b></center>
```

This displays the project name in large, bold, blue text in the center of the first line on the page. (The indentation is used solely to make the example easy to follow).

Types of templates

mrInterview has two types of templates: global and project. Global templates are held in the main Projects directory (usually c:\Program Files\SPSS MR\mrInterview\Projects) and can be used by any project. These include the standard templates provided as part of the mrInterview installation package, as well as any other templates that your company creates to implement its current standards for questionnaire design and layout.

Project templates exist in the project directory and apply only to that project. You use project-specific templates either for special projects that have a different overall layout or for particular questions in the questionnaire that require a non-standard layout. You can create project-specific templates in the project's source directory (usually c:\mrInterviewSource\project_name for desktop projects or c:\Program Files\SPSS MR\mrInterview\Users\user_name\project_name for DimensionNet projects) and the activation process will copy them to the project's working directory (usually c:\Program Files\SPSS MR\mrInterview\Shared\project_name).

-
- ☞ See section 23.9, Choosing a page template for a question, for information on naming the templates to be used by a questionnaire.
-

23.2 Line Breaks and Blank Space

mrInterview does not insert line breaks or blank space between the elements of a page. If you want line breaks or blank space you may insert `
` tags or similar at the appropriate points in your template. In cases where `
` precedes or follows a tag that may not always generate a text on the page, you may prefer to enclose the `mr` tag in a `<div>` tag instead. This forces a line break if a text is displayed or ignores the `mr` tag if there is no text to display. For example:

```
<div><mrProtectText></div>
<div><mrDisplayText></div>
<mrData>
<br><br>
<mrNavbar>
```

Here, the tags for protected and displayed text are enclosed in `<div>` tags because not every page will have these texts and you will not want to see blank lines in their place. Every page will have a question and response list section and a navigation bar so the tags for these items are separated by two `
` statements to display blank space between the navigation bar and the response list.

23.3 Single questions and response lists

Quick Reference

If the page displays only one question and its response list, and the response list is to be displayed directly below the question text, type:

<mrData>

in the page template at the point you want the question text to start. Any error messages are displayed above the question text.

If you want to print or format the question, response list and error message differently, type:

<mrData /TextField>

for the question text

<mrData /Controls>

for the response list, and

<mrData /ErrorField>

for the error message line.

If you use **<mrData>** without the additional flags, mrInterview automatically insert line breaks between the error message, the question text, and the response text. If there is no error message, the ErrorField tag is ignored; mrInterview does not display a blank line when there is no error message.

-
- ❖ mrInterview's default template displays error messages in red. If you define your own template, error messages will be displayed in black unless you specify a different color.
-

If you use **<mrData>** with the additional flags it is your responsibility to deal with line breaks.

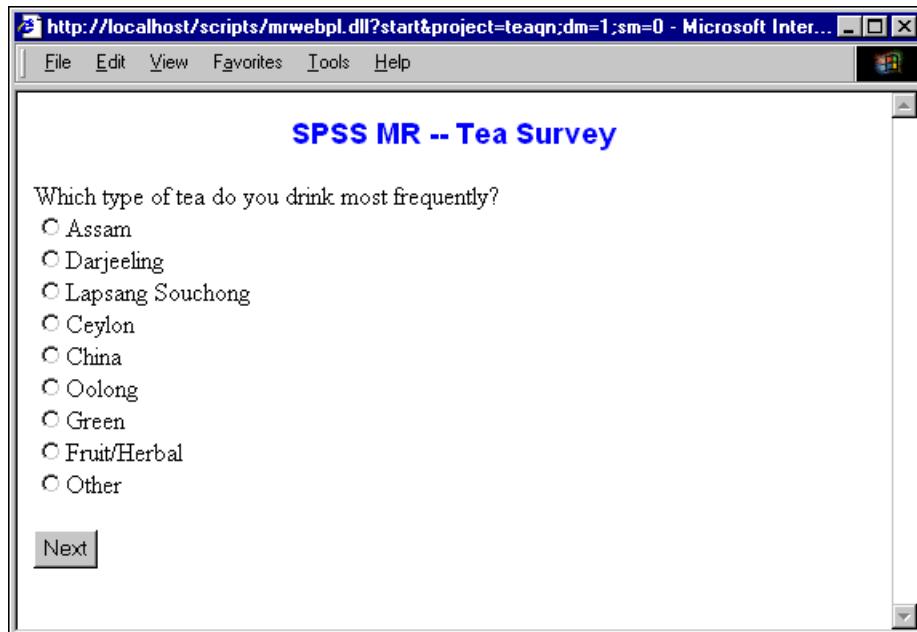
Here are two examples of how you can control the position of a question and its response list. Both examples use the following question:

```
freq      ask 'Which type of tea do you drink most frequently?'
resp sp 'Assam' ('assam') / 'Darjeeling' ('darjeeling') /
        'Lapsang Souchong' ('lapsang') / 'Ceylon' ('ceylon') /
        'China' ('china') / 'Oolong' ('oolong') / 'Green' ('green') /
        'Fruit/Herbal' ('fruit') / 'Other' ('other')
```

If the template contains:

```
<html>
  <head></head>
  <body>
    <center><b><font color="blue" size="4" face="Arial">
      SPSS MR -- Tea Survey
    </font></b></center>
    <br><br>
    <mrData>
    <br><br>
    <mrNavBar>
  </body>
</html>
```

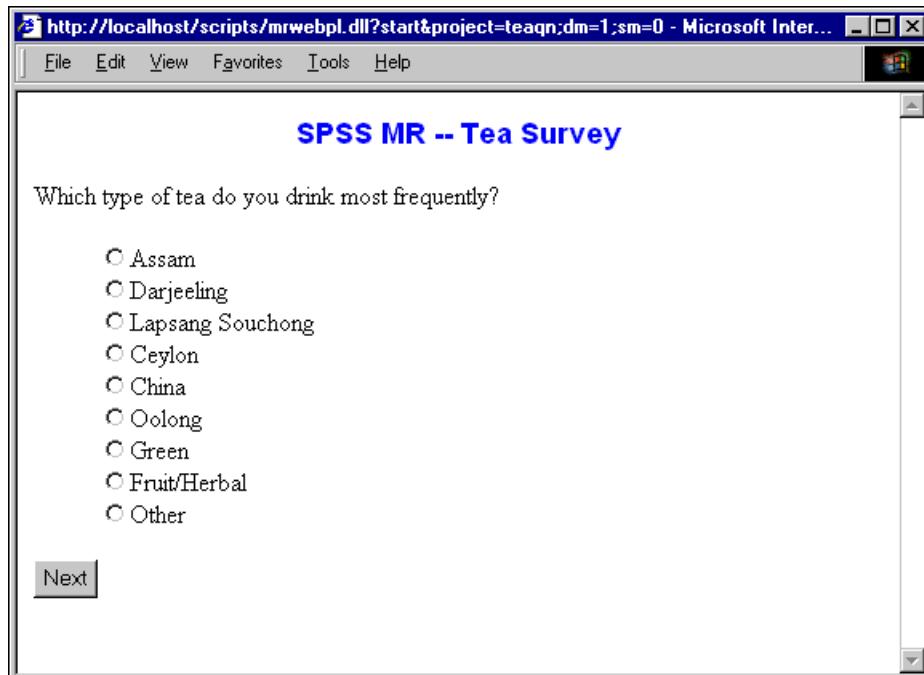
the question is displayed as:



If the template contains:

```
<html>
  <head></head>
  <body>
    <center><b><font color="blue" size="4" face="Arial">
      SPSS MR -- Tea Survey
    </font></b></center>
    <br><br>
    <mrData /TextField>
    <ul>
      <mrData /Controls>
    </ul>
    <div><mrData /ErrorField></div>
    <br>
    <mrNavBar>
  </body>
</html>
```

the question is displayed with the response list indented and separated from the question text by a blank line, and any error message is displayed below the response list:



23.4 Multiask questions

Quick Reference

In the page template, define the position of the *multiask*'s text with:

<mr Data0>

and then define the positions of the questions, response lists and error messages with:

<mrData*n* /TextField>
[<mrData*n* /Controls>]
[<mrData*n* /ErrorField>]

where *n* is a unique number for each question in the multiask group, starting at 1 for the first question on the statement. The numbers in each TextField, Controls and ErrorField group must be the same.

To display error messages for the questions in the multiask at the top of the page rather than with the individual question, type:

<mrData0 /TextField>
<mrData0 /ErrorField>

Here is the question text and controls section of a page layout for a multiask with four questions. The first line displays the text on the *multiask* statement. The second line reserves space for error messages. The remaining lines define the position of the questions. The first two questions on the multiask are displayed one below the other; the third and fourth questions are displayed one below the other to the right of the first two questions.

```
<mrData0 /TextField>
<mrData0 /ErrorField>
<br>
<table>
  <tr>
    <td width="50%" valign="top"><mrData1 /TextField></td>
    <td width="50%" valign="top"><mrData3 /TextField></td>
  </tr><tr>
    <td valign="top"><mrData1 /Controls></td>
    <td valign="top"><mrData3 /Controls></td>
  </tr><tr>
    <td valign="top"><mrData2 /TextField></td>
    <td valign="top"><mrData4 /TextField></td>
  </tr><tr>
    <td valign="top"><mrData2 /Controls></td>
    <td valign="top"><mrData4 /Controls></td>
  </tr>
</table>
```

There are two things to notice in this example:

- The use of the *width* tag to set the column widths in proportion to the page width. This is optional, but without it each column will be only as wide as the longest text it contains, making the page look cramped and hard to read. By defining column widths you ensure that there is ample space between the columns.
- The use of the *valign* tag to force all texts to start at the top of their cells. This ensures that the start of each question and response list in the second column lines up with the corresponding items in the first column.

Here is an example of a four-question multitask laid out using these specifications. The *multiask* statement in the script is:

```
m1  multiask 'And now a few demographic questions to help us categorize  
your responses'  age / gender / mstat / income
```

The screenshot shows a Microsoft Internet Explorer window with the title bar "http://localhost/scripts/MRWEBPL.DLL?ACTION - Microsoft Internet Explorer". The menu bar includes File, Edit, View, Favorites, Tools, and Help. The window content is titled "SPSS MR -- Drinks Survey". It displays the following text:
And now a few demographic questions to help us categorize your responses.....
The form consists of several groups of questions:
1. Age: "How old are you?" followed by "Answer: Are you ?" with radio buttons for Single, Married/Living with partner, Divorced, Separated, and Widowed.
2. Gender: "Are you ?" with radio buttons for Male and Female.
3. Income: "What is your annual household income?" with radio buttons for Under 20,000, 20,000 to 29,999, 30,000 to 39,999, 40,000 to 49,999, and 50,000 or more.
At the bottom of the form are three navigation buttons: Previous, Stop, and Next.

When you create a multiask template, it is important that the number of Datan blocks is the same as the number of questions to be displayed. If there are more or fewer Datan blocks, the result is undefined. If you have multiasks with different numbers of questions you must define a separate template for each type.

23.5 Display and protect statements

Quick Reference

To display text from a *display* statement, type:

<mrDisplayText>

in the page template at the point you want the text to appear on the page.

To display text from a *protect* statement, type:

<mrProtectText>

in the page template at the point you want the text to appear on the page.

If the current page does not have a displayed or protected text, these tags are ignored. This means that there is no need to create special templates for pages that have displayed or protected text unless you want the rest of the page to look different from your standard page. For example:

```
<div><mrProtectText></div>
<div><mrDisplayText></div>
<mrData>
```

allocates space for both texts to be printed above the question and response text.

23.6 Page titles

Quick Reference

To display the question name or the value of the *pgtitle* variable on the page, type:

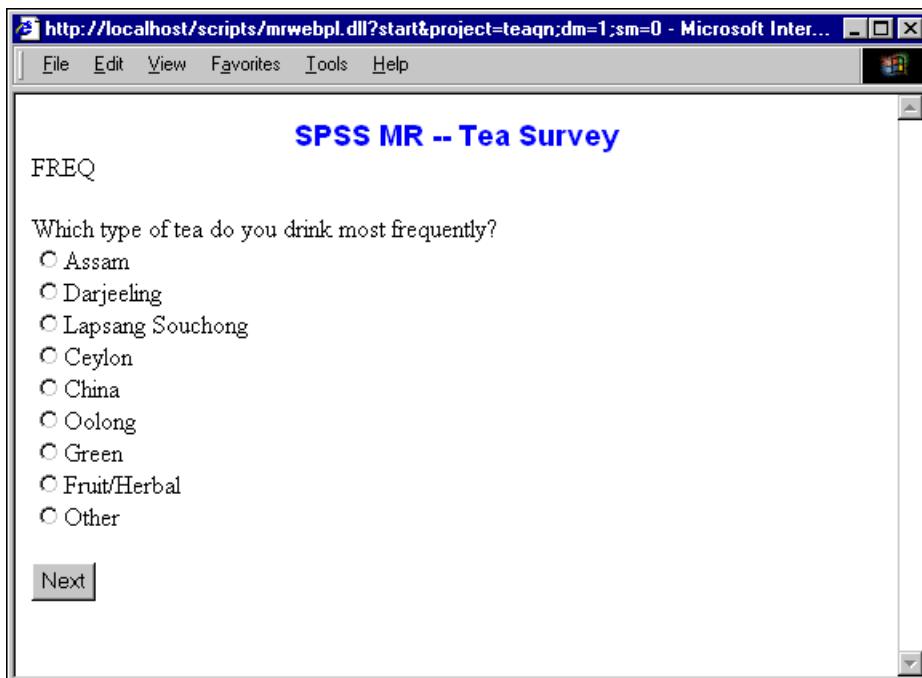
<mrPageTitle>

in the page template at the point you want it to appear.

For example:

```
<html>
<head></head>
<body>
    <center><b><font color="blue" size="4" face="Arial">
        SPSS MR -- Tea Survey
    </font></b></center>
    <br>
    <mrPageTitle>
    <br><br>
    <mrData>
    <br><br>
    <mrNavBar>
</body>
</html>
```

displays the question name in upper case on a line between the survey title and the question text as shown below:



If you want the question name to appear in the page's title bar rather than on the page itself, enclose the *PageTitle* tag in the HTML `<title>...</title>` tags:

```
<title><mrPageTitle></title>
```

You can replace the question name with a text of your choice by assigning that text to the *pgtitle* variable in the script.

☞ See section 23.10, Defining your own page titles, for further details.

The page title marker can appear anywhere in the template, not just on a line by itself at the top of the page. You can include it in the survey title by typing, for example:

```
<center><b><font color="blue" size="4" face="Arial">
    SPSS MR -- Tea Survey -- <mrPageTitle>
</font></b></center>
```

☞ With multiasks, `<mrPageTitle>` displays the name of the first question on the *multiask* statement. With grids, it displays the name of the *rowgrid* or *colgrid* statement.

23.7 Navigation buttons

Quick Reference

To display the standard navigation bar, type:

```
<mrNavbar>
```

in the template at the point you want the navigation bar displayed.

To display an individual navigation button, optionally with a label of your choice, type:

```
<mrNavbutton=type [text="button name"]>
```

in the page template, where *type* defines the button's function and is one of back, next or quit.

To display a Stop button, place the following statement in the questionnaire script at the point you want the button to become visible:

```
set hidestop=0
```

To hide the Stop button again, place the following statement in the questionnaire script:

```
set hidestop=1
```

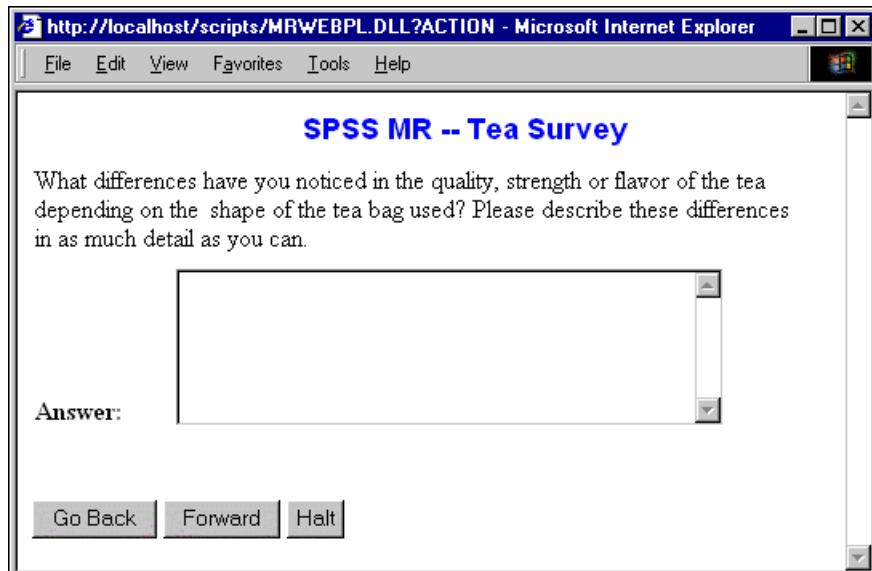
The standard mrInterview navigation bar has a Next and a Previous button and is always displayed at the foot of the page, after the last response line or other text. mrInterview hides the Previous button on the first question. To display this navigation bar, use the `<mrNavbar>` tag at the point you want the navigation bar displayed.

If you would like more control over the navigation buttons, you can use the `<mrNavbutton>` tag to specify which buttons you want to display and where, and the labels to use for each button.

To rename the buttons, use `<mrNavbutton>` with the *text* option. For example:

```
<mrNavbutton=back text="Go Back">
<mrNavbutton=next text="Forward">
<mrNavbutton=quit text="Halt">
```

displays the following buttons. Notice that not only are the texts different, but also the order has changed to match the order in which the buttons were renamed:



If you are familiar with HTML code you can take more control over the navigation bar, for example, by using a `<form>...</form>` section to display the navigation bar at the side of the page. If used, the form section must come within the body section of the code.

23.8 Automatic completion of data entry fields

Quick Reference

To switch field auto-completion on or off, place the following statement in the page template:

<mrAutoComplete>="*value*"

where *value* is either *on* or *off*.

Internet Explorer 5's AutoComplete feature remembers texts and other values that you enter in Web pages and then offers selected items as possible input for fields on other Web pages.

As you type information into a field, Internet Explorer compares the characters you type with the stored values and displays any entries that match. If the list contains the information you have started typing, you can select it from the list. If not, simply continue typing as normal.

« For further information about AutoComplete, see your Internet Explorer online help.

23.9 Choosing a page template for a question

Page templates control the layout of the interview pages and cover such things as the positions of the question and response texts, the position of any error messages that need to be displayed, and which navigation buttons are to be displayed and with what labels. Each questionnaire must have at least one template associated with it; some questionnaires may require more than one template. If a script has no templates associated with it, mrInterview will use its default layout.

The way a question or text is displayed during the interview is determined by the page template in force at that time. You can switch page templates as many times as you want, but will normally do so to accommodate question types such as multiasks or grids that cannot be displayed using the standard layout.

There are two ways that you can specify which template is to be used at a given point in the questionnaire.

- Using **pagstyle**. This method is available with version 2.1 and above of mrInterview. The name of the template to use is assigned to the *pagstyle* variable, which allows you to select different templates according to the characteristics of the current interview. For example, you could use different fonts or colors depending on the respondent's age or gender.
- Using **pgstyle** and **setfmt curpgstyle**. This method is retained for backwards compatibility with versions of mrInterview earlier than version 2.1. With this method, you write a *pgstyle* statement for each template file that will be used in the script, and then use *setfmt curpgstyle* to select the template to be used at a particular point in the script.

Whichever method you use for selecting templates, if you snap back to a previous question during an interview, mrInterview checks which template applies to the question and displays the page using the appropriate template.

mrInterview looks for templates in two locations. First, it checks the project's working directory for a template specific to the project; if the template is not found there, it then looks in the main Projects directory. If you find that your script is not being formatted as you expected, check that you have typed the filename correctly. Also check that the project template has a different name to the global templates unless you want the project template to take precedence over the global one of the same name.

Using pagstyle

Quick Reference

To define the page template for the next question or set of questions, type:

set pagstyle=*template*

where *template* is the template's filename enclosed in single quotes, or a variable containing the filename.

To use the default mrInterview template, type:

set pagstyle=default

For example:

```
age      ask 'How old are you?'
        resp num 13 to 19
gender  ask 'And are you ...?'
        resp sp 'Male' ('male') / 'Female' ('female')
        if (age<16 .and. gender='Female') {
            set pagstyle='youngfem.htm'
            goto ctn
        }
        if (age<16 .and. gender='Male') {
            set pagstyle='youngmale.htm'
            goto ctn
        }
        set pagstyle='olderteen.htm'
ctn     continue
```

This simple example illustrates how to select the template based on the characteristics of the current respondent. In all cases, the template filename is specified as a fixed text. Here is another simple example that uses text substitution to create the template filename based on the respondent's gender.

```
gender ask 'Are you ...?'
      resp sp 'Male' ('male') / 'Female' ('female')
      set pagstyle='[+gender+].htm'
```

If a script uses *pagstyle* mrInterview ignores any *pgstyle* or *setfmt curpgstyle* statements that may be present in the script. If you want to these statements to be recognized at certain points in the script (for example, because you are extending an existing questionnaire but do not want to change any existing code unnecessarily) you can either unset *pagstyle* or set it to an empty string.

Using **pgstyle** and **curpgstyle**

Quick Reference

To associate a page template with a script, type:

```
label pgstyle (template='filename.htm')
```

in the questionnaire script.

To define the page template for the next question or set of questions, type:

```
setfmt curpgstyle=style_label
```

where *style_label* is the label of the *pgstyle* statement that names the template, or a variable containing the label name.

To use the default mrInterview template, type:

```
setfmt curpgstyle=pgdefault
```

This method of defining templates is used in questionnaire scripts created by Build. It uses a two-step process whereby the template files used in the script are named with a labelled *pgstyle* statement, and then the template to use for the next question or set of questions is selected using *setfmt curpgstyle*.

For example:

```
standard pgstyle (template='tmp_std.htm')
mask     pgstyle (template='tmp_mask.htm')

comment Use the standard template
        setfmt curpgstyle=standard

freq    ask 'Which type of tea do you drink most frequently?'
        resp sp 'Assam' ('assam') / 'Darjeeling' ('darjeeling') /
               'Lapsang Souchong' ('lapsang') / 'Ceylon' ('ceylon') /
               'China' ('china') / 'Oolong' ('oolong') / 'Green' ('green') /
               'Fruit/Herbal' ('fruit') / 'Other' ('other')
```

```

comment Demographics - use the multiask template
setfmt curpgstyle=mask
age    dummyask 'Are you aged...?'
      resp sp '18-24' ('y24') / '25-34' ('y34') / '35-44' ('y44') /
             '45-54' ('y54') / '55-64' ('y64') / '65 or older' ('y65')
gender  dummyask 'Are you ...?'
      resp sp 'Male' ('male') / 'Female' ('female')
mstat   dummyask 'Are you ...?'
      resp sp 'Single' ('sine') / 'Married/Living with partner' ('marr') /
             'Divorced' ('div') / 'Separated' ('sep') / 'Widowed' ('wid')
income   dummyask 'What is your annual household income?'
      resp sp 'Under 20,000' ('u2k') / '20,000 to 29,999' ('u3k') /
             '30,000 to 39,999' ('u4k') / '40,000 to 49,999' ('u5k') /
             '50,000 or more' ('g5k')
ml      multiask '' age / gender / mstat / income

```

This questionnaire extract has two templates. The standard template used for most questions is defined in the template file tmp_std.htm and can be referred to in the script as 'standard'. The template for multiask questions is defined in the template file tmp_mask.htm and can be referred to in the script as 'mask'.

The main template for the script is the standard template, and this is set using the statement:

```
setfmt curpgstyle=standard
```

This applies to all questions until the value of *curpgstyle* is reset. This happens just before the questions that make up the multiask are defined, when the page style is changed to that defined in the mask template.

- ☞ If the questionnaire contains routing, you may need to route to a *setfmt curpgstyle* statement rather than a question if the next question uses a different template.

23.10 Defining your own page titles

Quick Reference

To define your own page title, type:

```
set pgttitle='title'
```

before the question to which the title refers. The text may contain text substitutions or expressions.

To revert to the default of displaying the question name as the page title, type:

```
unset pgttitle
```

If the interview template contains the `<mr PageTitle>` marker, mrInterview substitutes the name of the current question in place of the marker when it displays the interview pages. You can replace the question name with a title of your choice by assigning the title to the `pgtitle` variable. This title is then displayed for all questions until another `set pgtitle` statement is read, or until an `unset pgtitle` statement is read. For example, suppose the questionnaire contains the following statements:

```

teas      define 'Assam' ('assam') / 'Darjeeling' ('darjeeling') /
          'Lapsang Souchong' ('lapsang') / 'Gunpowder' ('gunpowder') /
          'Ceylon' ('ceylon') / 'China' ('china') / 'Oolong' ('oolong') /
          'English Breakfast' ('engbrek') / 'Black' ('black') /
          'Rooibos' ('rooibos') / 'Puerh' ('puerh') / 'Earl Grey' ('eg') /
          'Green' ('green') / 'Orange Pekoe' ('opekoe') /
          'Fruit/Herbal' ('fruit') / 'Other' ('other')

comment These questions have their names printed at the top of the page
freq     ask 'Which type of tea do you drink most frequently?'
         resp sp teas
oth      ask 'And which other types of tea do you drink?'
         resp mp not freq in teas null
takewith ask 'How do you take your '+freq+' tea?'
         resp mp 'With milk' ('milk') / 'With lemon' ('lemon') /
                 'With nothing' ('nothing')

comment The demographic questions all share the same title - Demographics
set pgtitle='Demographics'
age      ask 'How old are you?'
         resp num 18 to 99
gender   ask 'Are you .... ?'
         resp sp 'Male' ('male') / 'Female' ('female')
mstat    ask 'Are you .... ?'
         resp sp 'Single' ('sing')/ 'Married/Living with partner' ('marr') /
                 'Divorced' ('div') / ' Separated' ('sep') / 'Widowed' ('wid')

comment Revert to displaying the statement name in the title
unset pgtitle

```

The questions freq, oth and with will have their question names displayed at the point the `<mrPageTitle>` tag appears in the template. The questions age, gender and mstat will have the title Demographics displayed instead of their question names. The `unset` statement removes the value from `pgtitle`, so mrInterview reverts to displaying question names.

The text that you define for the page title can contain text substitutions or expressions. Text substitution is useful for CATI interviews, where you may want interviewers to write down open-ended responses rather than typing them as the respondent speaks. When open-ends are recorded in this way, interviewers will need to write down the full name of the question, including the iteration number, if any, to ensure that when the responses are typed in, they are entered against the appropriate questions. The `qnameful` variable provides the full name of the question, and you can set this as the page title as follows:

```
set pgtitle = '[+qnameful+]'
```

23.11 <meta> and <!doctype> tags in templates

HTML authoring tools such as FrontPage often insert <meta> tags in HTML templates to override HTTP information that should generally be set by the IIS server. mrInterview takes care of setting the correct HTTP information, so there is a risk that these <meta> tags could confuse some browsers. mrInterview therefore removes all <meta> and <!doctype> tags that it finds in page templates, and then inserts its own <meta> tags. These show the template's name and the version of the mrInterview component that edited it, and instruct a proxy server or firewall to obtain the most recent version of a survey from the Web server.

23.12 Cascading style sheets

Cascading style sheets (CSS) are a standard feature of HTML that allow you to define global formatting parameters that can be applied to any number of templates. It is therefore possible, for example, to have a set of completely different page layouts that all have green, 12-point Arial question texts, and to change them all to have blue, 14-point Times New Roman question texts simply by changing the specification for the question text in the CSS.

The benefit of using cascading style sheets with mrInterview is that it allows you to simplify the content of your templates by placing tags that are common to all templates in the main Projects directory, rather than having to repeat them in every template. This, in turn, makes it very easy to keep the templates up to date if you need to change the color, font or point size of the text.

-
- ❖ The notes in this section refer mainly to the mrInterview-specific keywords that you can use in cascading style sheets. Other keywords may appear in the examples, but these are described only so far as is necessary for you to be able to understand the examples. For complete documentation on the general use of cascading style sheets, refer to your HTML manuals.
-

Format and location of a CSS

A CSS is a text file containing lines that define the appearance of different components of the questionnaire. Each line starts with a tag that is followed by a number of formatting instructions enclosed in braces (curly brackets). For example:

```
.mrQText { font-family:Tahoma; color:blue }
```

Here, the tag is .mrQText, which tells us that the instruction refers to question texts. The instructions inside the braces define the appearance of the question text: it will be blue and the font will be Tahoma.

You can type these instructions in whatever format you like. For example, you could type the mrQText definition as:

```
.mrQText {  
    font-family:Tahoma;  
    color:blue  
}
```

if you find this easier to follow.

You may call your CSS anything you like, although it is best to give it a .css extension so that it is easily recognizable. Place the file in the directory /InetPub/wwwroot.

If you are testing your CSS and need to change it, you must close and re-open your browser in order for the changes to be recognized.

Tags for questionnaire items

Quick Reference

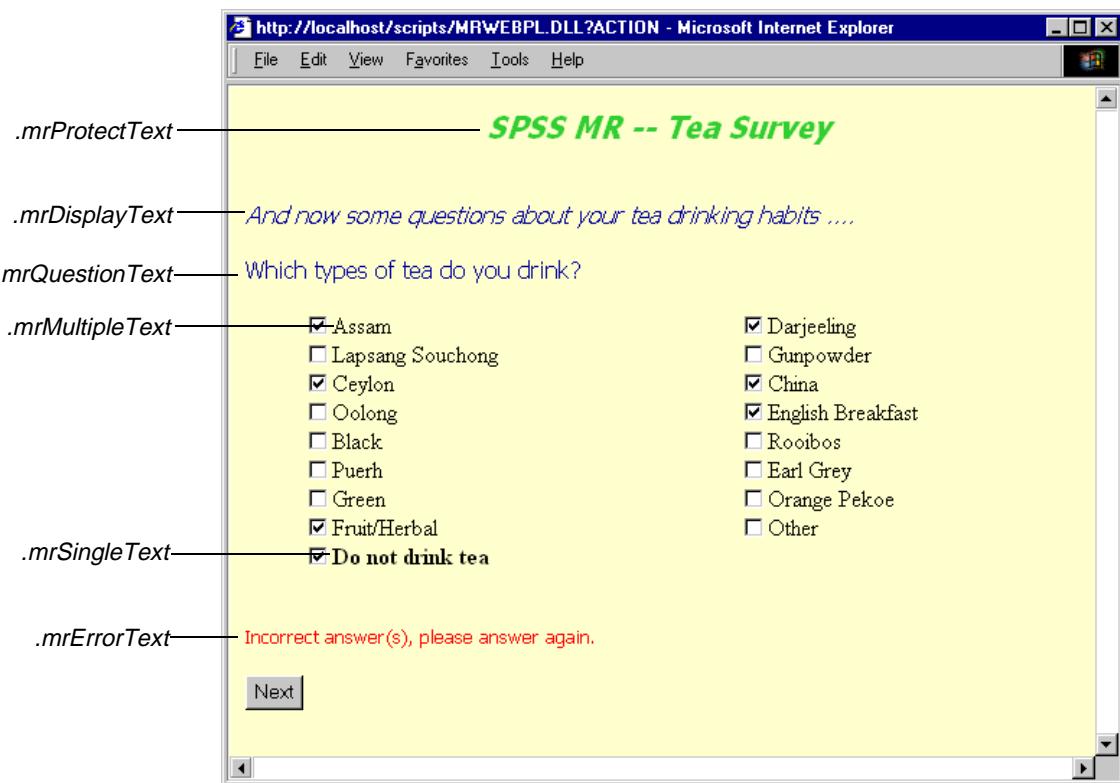
Use the following tags to define the appearance of items in the questionnaire:

.mrQText	Question texts
.mrDisplayText	Displayed texts
.mrProtectText	Protected texts
.mrErrorText	Error messages
.mrSingle	Single response buttons
.mrMultiple	Multiple response buttons
.mrSingleText	Single response texts
.mrMultipleText	Multiple response texts
.mrText	Numeric grid cells
.mrEdit	Answer boxes for numeric and open-ended responses
.mrDropdown	Dropdown list boxes
.mrListbox	List boxes

For example:

```
body {background-color:#FFFFCC}
.mrQText {font-family:Tahoma; font-size:12pt; color:#000099}
.mrSingleText {font-family:Times; font-size:12pt; color:black}
.mrMultipleText {font-family:Times; font-size:12pt; color:black}
.mrProtectText {font-family:Tahoma; font-size:16pt; font-weight:bold;
    font-style:italic; color:#32CD32; text-align:center}
.mrDisplayText {font-family:Tahoma; font-size:12pt; font-style:italic;
    color:#000099}
.mrErrorText {font-family:Tahoma; font-size:10pt; color:red}
```

produces the following output (the bold text for 'Do not drink tea is' the result of the ^s flag on that response in the questionnaire script, rather than a setting in the CSS file):



Naming the CSS in the template

Quick Reference

To name the CSS you want to use, place the following statement in the Head section of the template:

```
<link rel="stylesheet" type="text/css" href="http://URL/css_name" title="Style">
```

where *URL* is the URL of your internet server, and *css_name* is the pathname of the CSS file relative to /InetPub/wwwroot or the equivalent.

For example, if your internet address is www.abc.co.uk and the pathname of the CSS file is c:\InetPub\wwwroot\spssmr\css\mrint.css, you would type:

```
<link rel=stylesheet type="text/css"  
      href=http://www.abc.co.uk/spssmr/css/mrint.css title="Style">
```

23.13 Editing the page's form definition

When an interview is running, mrInterview builds the pages that the browser displays by merging the question and response texts for the current question into the page template for the current page and combining them with other HTML code that is generated by the interview process itself.

If you wish to customize these pages above and beyond what is possible using page templates, you can do so by writing Java script to define your requirements and editing the HTML page to include this code. For example, you can hide the standard response block containing the selection buttons and response texts and replace it with images that the respondent clicks to select a response. This is particularly useful for advertisement awareness questions, where you want to display pictures of advertisements and ask respondents to select those that they have seen before. Another example is where you want to display the next page automatically as soon as the question or questions on the current page have been answered, without waiting for the respondent to click Next. In this situation, you can edit the form so that it checks that each question has an answer and, if so, displays the next page.

Customizations of this type generally require you to refer to the form by name. The name of the mrInterview page is mrFORM.

23.14 Accessing an interview while it is running

Quick Reference

To access the connection ID for an interview, place the following tag in a standard <input> tag:

```
<mrConnectionID>
```

Each interview that takes place has a connection ID that identifies the connection between the respondent's browser and the interview. If you need to access an interview while it is in progress, perhaps because the respondent has run into difficulties and has requested help, you may do so as long as you know the interview's connection ID.

In order to do this, you will need to create a template with an <input> tag that references the special <*mrConnectionID*> tag. mrInterview substitutes the current interview's connection ID at this point, but performs no checking of any sort.

23.15 Sample script

```
comment Sample script for Quancept and mrInterview Scriptwriter's manual
comment Chapter 23, Using page templates in mrInterview

comment Display Stop button
set hidestop=0

comment Name the page templates this script will use
comment These statements can be dropped if templates are named with
comment pagestyle instead (see below)
std      pgstyle(template='standard.htm')
mask    pgstyle(template='multiask.htm')

teas     define 'Assam' ('assam') / 'Darjeeling' ('darjeeling') /
          'Lapsang Souchong' ('lapsang') / 'Gunpowder' ('gunpowder') /
          'Ceylon' ('ceylon') / 'China' ('china') / 'Oolong' ('oolong') /
          'English Breakfast' ('engbrek') / 'Black' ('black') /
          'Rooibos' ('rooibos') / 'Puerh' ('puerh') / 'Earl Grey' ('grey') /
          'Green' ('green') / 'Orange Pekoe' ('opekoe') /
          'Fruit/Herbal' ('fruit') / 'Other'('other') /
          'Do not drink tea ^s' ('donotdrink')

comment Use the standard template for the following questions.
comment An alternative is to use the following statement:
comment      set pagstyle='standard.htm'
setfmt curpgstyle = std

title   protect 'SPSS MR -- Tea Survey'

freq    ask 'Which type of tea do you usually drink?'
resp sp teas
set notea = nbit(freq)
route (notea=17) go exit
oth     ask 'And which other types of tea do you drink?'
resp mp not freq in teas null

takewith ask 'How do you take your '+freq+' tea?'
resp mp 'With milk'('milk') / 'With lemon' ('lemon') /
      'With nothing' ('nothing')
set withmilk = bit(takewith/1)
route (.not. withmilk) go ctrl
whenadd  ask 'When you take tea with milk, when do you add the milk?'
resp sp 'Before the tea' ('before') / 'After the tea' ('after')

ctrl    continue
```

Quancept and mrInterview Scriptwriter's Manual

```
uses      ask 'Do you use teas bags, loose tea or both?'
resp mp 'Tea bags' ('bags') / 'Loose tea' ('loose')
set tb = bit(uses/1)

if (tb) {
shape      ask 'Which shape tea bags do you use or have you used in the past?'
resp mp 'Round' ('round') / 'Square/oblong' ('sq') / 'Pyramid' ('pyr')

set gtl = numb(shape)
if (gtl > 1) {
tbdiff     ask 'Have you noticed any difference in the quality, strength
or flavor of the tea depending on the type of tea bag you use?'
resp sp 'Yes' ('yes') / 'No' ('no') go ctn2
differ     ask 'What are those differences? Please describe them
in as much detail as you can.'
resp coded
}
}

ctn2      continue

comment  Switch to the multiask template and set the page title
comment  for the multiask question to be Demographics.
comment  An alternative is to use the following statement:
comment      set pagstyle='multiask.htm'

setfmt curpgstyle = mask
set pgttitle = 'Demographics'

age       dummyask 'How old are you?'
resp num 18 to 99
gender    dummyask 'Are you .... ?'
resp sp 'Male' ('male') / 'Female' ('female')
mstat     dummyask 'Are you .... ?'
resp sp 'Single' ('sing') / 'Married/Living with partner' ('marr') /
'Divorced' ('div') / 'Separated' ('sep') / 'Widowed' ('wid')
income    dummyask 'What is your annual household income?'
resp sp 'Under 20,000' ('u2k') / '20,000 to 29,999' ('u3k') /
'30,000 to 39,999' ('u4k') / '40,000 to 49,999' ('u5k') /
'50,000 or more' ('m5k')
demog     multiask 'And now a few demographic questions to help us
categorize your responses' age / gender / mstat / income

comment Revert to the default of using the question name as the
comment page title, and switch back to the standard template
unset pgttitle
setfmt curpgstyle = std
```

```

d1      display 'Finally, a couple of questions to test your general
knowledge about tea ....'
country  for rowgrid '' teatype =
        'Assam' ('assam') / 'Black' ('black') / 'Ceylon' ('ceylon') /
        'Darjeeling' ('darjeeling') / 'Green' ('green') /
        'Oolong' ('oolong') / 'Gunpowder' ('gunpowder') /
        'Lapsang Souchong' ('lapsang') / 'Orange Pekoe' ('opekoe') /
        'Puerh' ('puerh') / 'Rooibos' ('rooibos')
prod     ask 'Which country or countries produces ....?'
resp mp  'China' ('china') / 'India' ('india') / 'Japan' ('japan') /
        'Kenya' ('kenya') / 'South Africa' ('safrica') /
        'Sri Lanka' ('srilanka') dk
next

places   ask 'A number of other countries also grow tea on a much
smaller scale. Which of the following countries do you think these are?'
resp mp  'Cameroon' ('cam') / 'Burundi' ('bur') / 'Ethiopia' ('eth') /
        'Madagascar' ('mad') / 'Mauritius' ('mau') /
        'Mozambique' ('moz') / 'Rwanda' ('rwa') / 'Uganda' ('uga') /
        'Zimbabwe' ('zim') / 'Argentina' ('arg') / 'Brazil' ('bra') /
        'Ecuador' ('ecu') / 'Peru' ('per') / 'The Azores' ('azo') /
        'Australia' ('aus') / 'Bangladesh' ('ban') / 'Iran' ('ira') /
        'Malaysia' ('mal') / 'Nepal' ('nep') /
        'Papua New Guinea' ('png') / 'Turkey' ('tur') /
        'Vietnam' ('vie') vsetcols=4 null dk

exit      end

```

CSS and template files

The script uses a cascading style sheet called ch23.css. Copy this file to /InetPub/wwwroot (or your site's equivalent). The file contains the following statements:

```

body {background-color:#FFFFCC }
.mrProtectText {font-family:Tahoma; font-size:16pt; font-weight:bold;
    font-style:italic; color:#32CD32; text-align:center}
.mrErrorText {font-family:Tahoma; font-size:10pt; color:red}
.mrDisplayText {font-family:Tahoma; font-size:12pt; font-style:italic;
    color:#000099}
.mrQText {font-family:Tahoma; font-size: 12pt;color: #000099}
.mrSingleText {font-family:Times; font-size:12pt; color:black}
.mrMultipleText {font-family:Times; font-size:12pt; color:black}

```

The script uses two template files, both of which should be placed in the project's source directory. They will be copied to the project's working directory as part of the activation process. The standard template, standard.htm, is as follows:

```

<html>
    <head><link rel="stylesheet" href="http://localhost/ch23.css"></head>
    <body>
        <center><mrProtectText> -- <mrPageTitle></center>

```

```
<br>
<div><mrDisplayText></div>
<br>
<mrData /TextField>
<ul>
    <mrData /Controls>
</ul>
<div><mrData /ErrorField></div>
<br>
<mrNavbar>
</body>
</html>
```

The multiask template, multiask.htm, is as follows:

```
<html>
    <head><link rel="stylesheet" href="http://localhost/ch23.css"></head>
    <body>
        <center><mrProtectText> -- <mrPageTitle></center>
        <br>
        <mrData0 /TextField>
        <div><mrData0 /ErrorField></div>
        <br><br>
        <table>
            <tr valign="top">
                <td width="50%"><mrData1 /TextField></td>
                <td width="50%"><mrData3 /TextField></td>
            </tr>
            <tr valign="top">
                <td><mrData1 /Controls></td>
                <td><mrData3 /Controls></td>
            </tr>
            <tr valign="top">
                <td><mrData2 /TextField></td>
                <td><mrData4 /TextField></td>
            </tr>
            <tr valign="top">
                <td><mrData2 /Controls></td>
                <td><mrData4 /Controls></td>
            </tr>
        </table>
        <br><br>
        <mrNavBar>
    </body>
</html>
```

24 Templates for grids

When you define the page layout for a grid, you create two files: a page template and a grid template. The page template file, like those for standard questions and multiasks, defines the position of the grid, the navigation buttons and any other text that you want to appear on the page. The grid template file defines the structure and appearance of the grid.

You can call these files anything you like, but you may find them easier to manage if you give them the same basic name—rating.htm for the page template and rating.xml for the grid template, for example.

This chapter describes the statements that you use in the page template file to request a grid layout, as well as those you can use in the grid template for defining the grid's appearance.

24.1 Naming grid template files

Quick Reference

To name a grid template for a project, place the following statement in the page template file:

```
<mrGridFormatn = "[location]filename.xml">
```

where *n* is a number that uniquely identifies this grid template within the current page template, *location* defines the location of the grid template file. This can be a drive letter and pathname or a URL. If the template is in the project's source directory or in the main Projects directory (usually, c:\Program Files\mrInterview\Projects) then you can enter just the file's name.

When you activate the project, any grid templates in the project's source directory (usually c:\mrInterviewSource\Projects\project_name for desktop projects, or c:\Program Files\SPSS MR\mrInterview\FMRoot\Users\user_name\project_name) are copied to the project's working directory (c:\Program Files\SPSS MR\mrInterview\FMRoot\Shared\project_name) with the .sif file and the page template files.

In the following example, the first template is located in a directory that is not the Projects directory, so the template's full pathname is given. The second template is named using a simple filename, so mrInterview will look first in the project's subdirectory within the Projects directory and then in the Projects directory itself.

```
<mrGridFormat1="file:///c:\mrInterview\templates\rating.xml">
<mrGridFormat2="CgridAltCol.xml">
```

Notice that the filenames describe the types of grids they define. The first one defines the structure of a rating grid, while the second one defines the structure of a column grid that has different formats for alternate columns.

-
- ❖ Don't forget to include a *pagestyle* statement in the questionnaire for the page template, as for other page templates.
-

24.2 Positioning the grid on the page

Quick Reference

To position the grid on the page, place the following statement in the page template:

```
<mrData /Controls /GridFormat=tpl_number>
```

where *tpl_number* is the number of the *<mrGridFormat>* statement that names the grid template file to be used to format the grid.

For example, if the grid template file is named as:

```
<mrGridFormat1="file:///c:/mrInterview/templates/rating.xml">
```

the *<mrData>* tag must be:

```
<mrData /Controls /GridFormat=1>
```

24.3 The grid template file

The grid template file is an XML file that determines how a grid will appear on the interview page. XML files are text files that you can create and edit using any text editor. The statements or tags that you can use in your grid templates are defined in an XML schema file called *mrigf.xsd* that is usually installed in *c:\mrInterviewSource* (the main project source directory). If you have an XML authoring tool such as XML Spy that supports schemas, you can validate your grid template against the schema to ensure that the XML code you have written is valid. If you do not have an XML authoring tool, you can work with any text editor of your choice, but you will be responsible for ensuring that the code you write is correct. If you want to validate your grid templates, you can download a free XML schema validator from <http://www.w3.org/XML/Schema>.

mrInterview comes with a number of predefined grid templates. You may find that you can use these templates as they are or with a little alteration, in which case you will not need to build your own templates.

XML tags look and behave the same as HTML tags. In fact, if you are familiar with XML or HTML you may recognize many of the tags. All tags in a grid template are case sensitive and must be entered exactly as shown in this chapter. Mistyping tags in any way causes the grid to be displayed in the default format. If you have a schema validator, it will flag any statements where the case of a tag is incorrect as errors.

24.4 Starting a grid template

Quick Reference

To define a grid template, enclose the definition in the following tags:

<gridFormat>...</gridFormat>

To define the grid's name, type:

<name>text</name>

on the line immediately below the **<gridFormat>** marker. *text* is the name or description of the template.

To display a caption centrally below the grid, type:

<comment>caption_text</comment>

If you are using an XML authoring tool and want to be able to validate your template against the grid schema, you will need to include information about the schema on the opening **<gridFormat>** marker. You can copy this information from one of the example templates.

24.5 Levels in a grid template

A grid template can contain formatting information at the following levels (listed in ascending order of precedence):

- grid level
- all cells
- alternate rows
- alternate columns
- heading row (column headings)
- heading column (row headings)

If the same tag appears at two or more levels in the template, the tag at the lower level overrides the same tag at the higher level for that particular aspect of the grid only. This allows you to define global characteristics for all cells in the grid and then to override certain of those characteristics for, say, the row headings. For example, you can have centered, bold, red text in all cells except the row headings which are left-aligned but still have bold, red text.

The grid and all-cells levels refer to the grid as a whole, and you'll normally define global specifications as relating to all cells rather than the grid itself. There are, however, some tags such as `<name>` and `<colWidths>` that cannot be defined at the all cells level and must be defined at the grid level. There are also some tags that mean different things when used at these levels. For example, `<width>` at grid level defines the width of the whole grid whereas at all-cells level it defines the widths of the individual columns.

To enter tags at the grid level, type them inside the `<gridFormat>` markers.

To define global characteristics for all cells in the grid, place the appropriate tags within the following markers:

<allCells>...</allCells>

For example, to give all cells a pale green background, you can type:

```
<gridFormat>
  <allCells>
    <bgColor>
      <colorFormat>numeric</colorFormat>
      <color>#8FBC8F</color>
    </bgColor>
  </allCells>
</gridFormat>
```

In this particular instance, this has the same effect as typing:

```
<gridFormat>
  <bgColor>
    <colorFormat>numeric</colorFormat>
    <color>#8FBC8F</color>
  </bgColor>
</gridFormat>
```

The indentation makes the hierarchy of the example easy to follow, but it is not a requirement.

To specify formatting for alternate rows and columns of the grid, place the necessary tags within the following markers:

<altBodyRow>...</altBodyRow>
<altBodyCol>...</altBodyCol>

☞ See section 24.16, Different formats for alternate rows or columns, for further information.

To define the characteristics of the heading row and heading column, enclose the appropriate tags in the following markers:

```
<headRow>...</headRow>
<headCol>...</headCol>
```

☞ See section 24.15, Row and column headings, for further information.

24.6 Background color

Quick Reference

To define the background color for a grid or part of a grid, place the following statements at the appropriate place in the grid specification:

```
<bgColor>
  <colorFormat>type</colorFormat>
  <color>color_ref</color>
</bgColor>
```

where *type* defines how the *color* will be specified, and is either *string* if the color will be entered as a text, or *numeric* if the color will be entered as a hexadecimal value. *color_ref* is the name or hexadecimal value of the color to be used.

For example, a grid specification that contains only the following formatting statements produces a standard grid on a pale yellow background:

```
<bgColor>
  <colorFormat>numeric</colorFormat>
  <color>#FFFFCC</color>
</bgColor>
```

You can change the background color of certain cells in the grid by placing *<bgcolor>* within the tags that define those cells. For example, you can use different colors for alternating rows or columns, or for the heading row or column. The following specification creates a grid with green column heading cells and pale yellow cells elsewhere:

```
<allCells>
<bgColor>
  <colorFormat>numeric</colorFormat>
  <color>#FFFFCC</color>
</bgColor>
</allCells>
<headRow>
<headerCell>
<bgColor>
  <colorFormat>numeric</colorFormat>
  <color>#32CD32</color>
</bgColor>
</headerCell>
</headRow>
```

The grid displays as follows:

The screenshot shows a Microsoft Internet Explorer window with the title bar "http://localhost/scripts/MRWEBPL.DLL?ACTION - Microsoft Internet Explorer". The main content area displays a survey titled "SPSS MR -- Tea Survey" with the subtitle "RGRID". The question is "Which country or countries produces?" Below the question is a grid table with 11 rows (tea types) and 7 columns (countries). The columns are labeled: China, India, Japan, Kenya, South Africa, Sri Lanka, and Don't know. The rows are labeled: Assam, Black, Ceylon, Darjeeling, Green, Gunpowder, Lapsang Souchong, Oolong, Orange Pekoe, Puerh, and Rooibos. Each cell in the grid contains a small square checkbox.

	China	India	Japan	Kenya	South Africa	Sri Lanka	Don't know
Assam	<input type="checkbox"/>						
Black	<input type="checkbox"/>						
Ceylon	<input type="checkbox"/>						
Darjeeling	<input type="checkbox"/>						
Green	<input type="checkbox"/>						
Gunpowder	<input type="checkbox"/>						
Lapsang Souchong	<input type="checkbox"/>						
Oolong	<input type="checkbox"/>						
Orange Pekoe	<input type="checkbox"/>						
Puerh	<input type="checkbox"/>						
Rooibos	<input type="checkbox"/>						

At the bottom left of the grid, there are two buttons: "Previous" and "Next".

24.7 Wrapping text within a column

Quick Reference

To switch text wrapping on or off, type:

`<nowrap>value</nowrap>`

where *value* is *true* to switch off line wrapping or *false* to switch it on.

When mrInterview displays a grid, it tries to fit the whole grid within the current width of the browser page. If there are texts that are longer than the column width, mrInterview wraps them over two or more lines so that they fit. You can switch line wrapping off so that all columns are as wide as is necessary to display the full text all on one line, and the respondent may need to scroll to see the full grid.

You can use `<nowrap>` in any cell-format section of the grid specification, so you could have the column heading texts wrapped and the row heading texts not wrapped.

24.8 Text alignment

Quick Reference

To define the horizontal position of text (or other cell contents) within a grid cell, type:

`<align>position</align>`

where *position* is *left*, *right*, *center*, *justify*, or *char*. The default is left alignment.

To define the vertical position of text within a grid cell, type:

`<valign>position</valign>`

where *position* is *top*, *center*, *bottom*, or *baseline*. The default is central alignment.

Most of the alignment parameters apart from *char* and *baseline* are self explanatory. *char* aligns the cell contents on a given character. The default is the user's decimal point character. *baseline* aligns the cell contents so that the first line of text in all cells is aligned on a baseline that is common to all first-line texts in the row. You can find further information on these parameters in your HTML guides.

You can set different alignments for different parts of the grid. The example centers all cell content vertically within the row height and horizontally within the column width. The exception is the row heading column which is left-aligned.

```
<allCells>
  <align>center</align>
  <valign>center</valign>
</allCells>
<headCol>
  <headerCell>
    <align>left</align>
  </headerCell>
</headCol>
```

The screenshot shows a Microsoft Internet Explorer window with the title bar "http://localhost/scripts/MRWEBPL.DLL?ACTION - Microsoft Internet Explorer". The main content area displays a survey titled "SPSS MR -- Tea Survey". The survey is titled "CGRID" and asks "Which country or countries produces?" Below this is a grid table with rows for countries and columns for tea types.

	Assam	Black	Ceylon	Darjeeling	Green	Gunpowder	Lapsang Souchong	Oolong	Orange Pekoe	Puerh	Rooibos
China	<input type="checkbox"/>										
India	<input type="checkbox"/>										
Japan	<input type="checkbox"/>										
Kenya	<input type="checkbox"/>										
South Africa	<input type="checkbox"/>										
Sri Lanka	<input type="checkbox"/>										
Don't know	<input type="checkbox"/>										

At the bottom left of the grid, there are two buttons: "Previous" and "Next".

You can set the horizontal alignment of the grid within the width of the browser page by placing the following tags at the top level of the grid:

```
<tableDef>
  <align>center</align>
</tableDef>
```

This is not the same as:

```
<allCells>
  <align>center</align>
</allCells>
```

which causes the texts in all cells to be centered.

24.9 Bold and italic text

Quick Reference

To display bold or italic text, type:

```
<font>
  <fontStyle>style</fontStyle>
</font>
```

where *style* is either *bold*, *italic*, or *bold italic*. The name of the standard text style is *regular*.

24.10 Typeface and text size

Quick Reference

To set the typeface to be used for grid text, type:

```
<font>
  <fontFace>name</fontFace>
</font>
```

where *name* is the name of the typeface.

To specify the size of text, type:

```
<font>
  <fontPoint>number</fontPoint>
</font>
```

where *number* is a number in the range 1 (small) to 7 (large). The default size is 3.

In the following example, text is displayed in the Tahoma font and, because the grid has many columns, a smaller text size has been used to fit all the columns on the screen in one go:

The screenshot shows a Microsoft Internet Explorer window with the title bar "http://localhost/scripts/MRWEBPL.DLL?ACTION - Microsoft Internet Explorer". The menu bar includes File, Edit, View, Favorites, Tools, and Help. The main content area displays a survey titled "SPSS MR -- Tea Survey" with the subtitle "CGRID". The question is "Which country or countries produces?" Below this is a grid table:

	Assam	Black	Ceylon	Darjeeling	Green	Gunpowder	Lapsang Souchong	Oolong	Orange Pekoe	Puerh	Rooibos
China	<input type="checkbox"/>										
India	<input type="checkbox"/>										
Japan	<input type="checkbox"/>										
Kenya	<input type="checkbox"/>										
South Africa	<input type="checkbox"/>										
Sri Lanka	<input type="checkbox"/>										
Don't know	<input type="checkbox"/>										

At the bottom left are "Previous" and "Next" buttons.

The statements that define the font characteristics are as follows:

```

<allCells>
  <font>
    <fontFace>Tahoma</fontFace>
    <fontPoint>2</fontPoint>
  </font>
</allCells>
<headRow>
  <headerCell>
    <font>
      <fontStyle>bold</fontStyle>
    </font>
  </headerCell>
</headRow>

```

24.11 Text color

Quick Reference

To set a default color for all text in the grid, place the following statements inside a *gridFormat* or *allCells* section:

```
<textColor>
  <colorFormat>type</colorFormat>
  <color>color_ref</color>
</textColor>
```

To override the default setting for a row or column, place the following statements inside a *headRow*, *headCol*, *altBodyRow* or *altBodyCol* section:

```
<font>
<fontColor>
  <colorFormat>type</colorFormat>
  <color>color_ref</color>
</fontColor>
</font>
```

In both cases the variable parameters are as follows. *type* defines how the color will be specified, and is either *string* if the color will be entered as a text, or *numeric* if the color will be entered as a hexadecimal value. *color_ref* is the name or hexadecimal value of the color to be used.

If you use *<textColor>* at both the grid level and the all-cells level, the setting at the all-cells level overrides the setting at the grid level. For example, specifying either:

```
<allCells>
  <textColor>
    <colorFormat>string</colorFormat>
    <color>green</color>
  </textColor>
</allCells>
<headRow>
  <headerCell>
    <font>
      <fontColor>
        <colorFormat>string</colorFormat>
        <color>blue</color>
      </fontColor></font>
    </headerCell>
  </headRow>
```

or:

```
<gridFormat>
  <textColor>
    <colorFormat>string</colorFormat>
    <color>green</color>
  </textColor>
<headRow>
  <headerCell>
    <font>
      <fontColor>
        <colorFormat>string</colorFormat>
        <color>blue</color>
      </fontColor>
    </font>
  </headerCell>
</headRow>
```

produces the following effect:

The screenshot shows a Microsoft Internet Explorer window with the title bar "http://localhost/scripts/MRWEBPL.DLL?ACTION - Microsoft Internet Explorer". The menu bar includes File, Edit, View, Favorites, Tools, and Help. The main content area displays a survey titled "SPSS MR -- Tea Survey". The survey is titled "CGRID" and asks "Which country or countries produces?" Below this, there is a grid table with rows for countries and columns for tea types. The countries listed are China, India, Japan, Kenya, South Africa, Sri Lanka, and "Don't know". The tea types listed are Assam, Black, Ceylon, Darjeeling, Green, Gunpowder, Lapsang Souchong, Oolong, Orange Pekoe, Puerh, and Rooibos. Each cell in the grid contains a small square checkbox. At the bottom of the grid, there are "Previous" and "Next" navigation buttons.

	Assam	Black	Ceylon	Darjeeling	Green	Gunpowder	Lapsang Souchong	Oolong	Orange Pekoe	Puerh	Rooibos
China	<input type="checkbox"/>										
India	<input type="checkbox"/>										
Japan	<input type="checkbox"/>										
Kenya	<input type="checkbox"/>										
South Africa	<input type="checkbox"/>										
Sri Lanka	<input type="checkbox"/>										
Don't know	<input type="checkbox"/>										

24.12 Special text effects

Quick Reference

To apply effects, type:

```
<font>
  <fontEffects>effect1</fontEffect>
  <fontEffects>effect2</fontEffect>
</font>
```

where *effect1* and *effect2* are each one of the following:

underline	strikethrough	overline	blink	superscript	subscript
smallcaps	allcaps	capitalize	hidden	strong	emphasis
sample	definition	citation	variable	code	keyboard
abbreviation	acronym	big	small	quotation	typewriter

Refer to your HTML documentation for further details on these effects.

If you want to emphasize a text you can apply special effects such as underlining. These effects work in addition to any styles such as italics, so by combining the two you can place increased emphasis on particular words or phrases in the text.

24.13 Row height

Quick Reference

To set the row height, place the following tag inside any of the cell-formatting sections:

```
<height>value</height>
```

where *value* is the row height in pixels.

Without this, mrInterview makes each row as tall as is necessary to contain the row text.

24.14 Column widths

Quick Reference

To define column widths, place the following statements in the appropriate section of the grid specification:

```
<width>
  <widthFormat>type</widthFormat>
  <value>number</value>
</width>
```

where *type* defines how to interpret the *value* parameter, and is either *absolute* if *value* is the width of the column in pixels, or *percent* if *value* is the column width as a percentage of the total grid width. *number* is the width of the column in the given format.

If you want to vary the column widths across the grid, place the following statements in the top level of the grid specification:

```
<colWidths>
  <width>
    <widthFormat>type</widthFormat>
    <value>number</value>
  </width>
</colWidths>
```

Repeat *<widthFormat>* and *<value>* in pairs for each column in the grid.

Note that *<colWidths>* overrides any column widths specified elsewhere in the template.

mrInterview normally makes each column as wide as the contents of the widest column (usually the column heading) and separates each column with one pixel of space. You can specify your own column widths in various ways depending on whether all the columns are the same width.

If all columns are the same width, place the *<width>* tags in the *<allCells>* section of the grid template. You can then override this specification for the column of row headings, for example, by placing these statements inside the *<headCol>* definition as well. For example:

```
<headCol>
  <headerCell>
    <width>
      <widthFormat>percent</widthFormat>
      <value>15</value>
    </width>
  </headerCell>
</headCol>
```

-
- ❖ When column widths are specified in different sections of the template, mrInterview chooses the width for a particular column based on a fixed order of precedence among the sections. For further information see “Precedence between specifications at different levels” on p. 560.
-

Here is an example that uses `<colWidths>` to set specific widths for each cell in the table. The column width specification is:

```
<colWidths>
  <width>
    <widthFormat>absolute</widthFormat>
    <value>100</value>
  </width>
  <width>
    <widthFormat>absolute</widthFormat>
    <value>40</value>
  </width>
  <width>
    <widthFormat>absolute</widthFormat>
    <value>60</value>
  </width>
  <width>
    <widthFormat>absolute</widthFormat>
    <value>80</value>
  </width>
  <width>
    <widthFormat>absolute</widthFormat>
    <value>100</value>
  </width>
  <width>
    <widthFormat>absolute</widthFormat>
    <value>120</value>
  </width>
  <width>
    <widthFormat>absolute</widthFormat>
    <value>140</value>
  </width>
  <width>
    <widthFormat>absolute</widthFormat>
    <value>40</value>
  </width>
  <width>
    <widthFormat>absolute</widthFormat>
    <value>50</value>
  </width>
</colWidths>
```

This produces the following table:

The screenshot shows a Microsoft Internet Explorer window with the title bar "http://localhost/scripts/MRWEBPL.DLL?ACTION - Microsoft Internet Explorer". The menu bar includes File, Edit, View, Favorites, Tools, and Help. The main content area is titled "SPSS MR -- Tea Survey" and contains a grid titled "RGRID" with the question "Which country or countries produces?". The grid has columns for China, India, Japan, Kenya, South Africa, Sri Lanka, and "Don't know". Rows list tea types: Assam, Black, Ceylon, Darjeeling, Green, Gunpowder, Lapsang, Souchong, Oolong, Orange Pekoe, Puerh, and Rooibos. Each cell in the grid contains a small square checkbox. At the bottom of the grid are "Previous" and "Next" buttons.

	China	India	Japan	Kenya	South Africa	Sri Lanka	Don't know
Assam	<input type="checkbox"/>						
Black	<input type="checkbox"/>						
Ceylon	<input type="checkbox"/>						
Darjeeling	<input type="checkbox"/>						
Green	<input type="checkbox"/>						
Gunpowder	<input type="checkbox"/>						
Lapsang	<input type="checkbox"/>						
Souchong	<input type="checkbox"/>						
Oolong	<input type="checkbox"/>						
Orange Pekoe	<input type="checkbox"/>						
Puerh	<input type="checkbox"/>						
Rooibos	<input type="checkbox"/>						

24.15 Row and column headings

Quick Reference

To enter specifications for row and column headings, enclose the statements in the following tags:

<headRow>...</headRow>

<headCol>...</headCol>

Specifications for row and column headings have two parts: cell formatting and row or column repetition. Cell formatting describes the characteristics of the row or column headings and covers such things as the cell color, the text style, and the column width. Row or column repetition specifies how frequently the heading row or column should be repeated in the grid.

Cell formatting

Quick Reference

To define cell requirements, place the appropriate statements inside the following tag:

<headerCell>...</headerCell>

For example:

```
<headCol>
  <headerCell>
    <font>
      <fontStyle>italic</fontStyle>
      <fontFace>Tahoma</fontFace>
    </font>
    <align>right</align>
    <valign>center</valign>
  </headerCell>
</headCol>
```

Repeating the row or column headings

Quick Reference

To repeat row or column headings, place the following tag inside the *headCol* or *headRow* section:

<repeat>number</repeat>

where *number* is the number of rows or columns after which the heading row or column is to be repeated. 0 displays the heading row at the top and bottom of the grid, or the heading column on the left and right of the grid.

With large grids, the respondent may need to scroll to see the full grid. When this is likely, you can make the grid easier to follow by repeating the heading row or column so that the row or column headings are still visible after scrolling. For example:

```
<headCol>
  <headerCell>
    <font>
      <fontStyle>italic</fontStyle>
    </font>
    <align>right</align>
    <valign>center</valign>
  </headerCell>
  <repeat>0</repeat>
</headCol>
```

displays the heading column on the left and right of the grid as follows:

The screenshot shows a Microsoft Internet Explorer window with the title bar "http://localhost/scripts/MRWEBPL.DLL?ACTION - Microsoft Internet Explorer". The menu bar includes File, Edit, View, Favorites, Tools, and Help. The main content area displays a grid titled "SPSS MR -- Tea Survey". The grid is titled "CGRID" and asks "Which country or countries produces?" The columns represent tea types: Assam, Black, Ceylon, Darjeeling, Green, Gunpowder, Lapsang Souchong, Oolong, Orange Pekoe, Puerh, and Rooibos. The rows list countries: China, India, Japan, Kenya, South Africa, Sri Lanka, and Don't know. Each cell contains a small square checkbox. The grid is styled with alternating row colors and bolded column headers.

	Assam	Black	Ceylon	Darjeeling	Green	Gunpowder	Lapsang Souchong	Oolong	Orange Pekoe	Puerh	Rooibos	
China	<input type="checkbox"/>	China										
India	<input type="checkbox"/>	India										
Japan	<input type="checkbox"/>	Japan										
Kenya	<input type="checkbox"/>	Kenya										
South Africa	<input type="checkbox"/>	South Africa										
Sri Lanka	<input type="checkbox"/>	Sri Lanka										
Don't know	<input type="checkbox"/>	Don't know										

24.16 Different formats for alternate rows or columns

Quick Reference

To specify the changes for alternate rows and columns of the grid, place the necessary tags within the following markers:

<altBodyRow>...</altBodyRow>

<altBodyCol>...</altBodyCol>

Large grids can be made easier to read if you use different formatting on alternate rows or columns of the grid.

When mrInterview builds the grid, it uses the characteristics you have defined for the grid as a whole, and then reads the alternate specifications and updates just those aspects of the grid before displaying it. For example, if the template contains the statements:

```
<allCells>
  <bColor>
    <colorFormat>numeric</colorFormat>
    <color>#FFFFCC</color>
  </bColor>
</allCells>
```

```

<altBodyRow>
  <bgColor>
    <colorFormat>numeric</colorFormat>
    <color>#93DB70</color>
  </bgColor>
</altBodyRow>

```

the rows of the grid will be alternately pale yellow and bright green, as shown below:

The screenshot shows a Microsoft Internet Explorer window with the title bar "http://localhost/scripts/MRWEBPL.DLL?ACTION - Microsoft Internet Explorer". The menu bar includes File, Edit, View, Favorites, Tools, and Help. The window content displays a title "SPSS MR -- Tea Survey" and a subtitle "RGRID Which country or countries produces?". Below this is a table representing a row grid. The columns are labeled "China", "India", "Japan", "Kenya", "South Africa", "Sri Lanka", and "Don't know". The rows list tea types: Assam, Black, Ceylon, Darjeeling, Green, Gunpowder, Lapsang Souchong, Oolong, Orange Pekoe, Puerh, and Rooibos. Each row has a checkbox in each column. The rows alternate between light green and pale yellow backgrounds.

	China	India	Japan	Kenya	South Africa	Sri Lanka	Don't know
Assam	<input type="checkbox"/>						
Black	<input type="checkbox"/>						
Ceylon	<input type="checkbox"/>						
Darjeeling	<input type="checkbox"/>						
Green	<input type="checkbox"/>						
Gunpowder	<input type="checkbox"/>						
Lapsang Souchong	<input type="checkbox"/>						
Oolong	<input type="checkbox"/>						
Orange Pekoe	<input type="checkbox"/>						
Puerh	<input type="checkbox"/>						
Rooibos	<input type="checkbox"/>						

Previous Next

-
- ❖ If the template contains only `<altBodyRow>` or only `<altBodyCol>`, the settings in that section are applied to both rowgrids and colgrids. If the template contains both specifications, mrInterview uses the settings in the specification that matches the grid's type and ignores settings in the other alternate specification. Therefore, a row grid is formatted using the `<altBodyRow>` specification and the `<altBodyCol>` specification is ignored.
-

24.17 Precedence between specifications at different levels

When a grid template contains specifications at more than one level, mrInterview uses the following rules to determine which specifications take precedence over others:

- In the top left cell of the grid, *colWidths* takes overall precedence, followed by *headCol*, *headRow*, *allCells* and *textColor* in that order.
- For all other cells in the lefthand column, *colWidths* takes overall precedence, followed by *headCol*, *altBodyRow* (for odd-numbered rows only), *allCells* and *textColor* in that order.
- For all other cells in the top row, *colWidths* takes overall precedence, followed by *headRow*, *altBodyCol* (for odd-numbered columns only), *allCells* and *textColor* in that order.
- For all other cells (the body of the grid), *colWidths* takes overall precedence, followed by *altBodyRow* (odd-numbered rows in a row grid or where *altBodyCol* is not defined), *altBodyCol* (odd-numbered columns in a column grid or where *altBodyRow* is not defined), *allCells* and *textColor* in that order.

Here is an example that uses color to illustrate these rules. Note that for easy reference, the *<color>* tags are followed by a description of the color in parentheses. These texts are not part of the specification. The grid template is:

```
<allCells>
  <bgColor>
    <colorFormat>numeric</colorFormat>
    <color>#FFFFCC</color>      (pale yellow)
  </bgColor>
</allCells>

<headRow>
  <headerCell>
    <bgColor>
      <colorFormat>numeric</colorFormat>
      <color>#C0D9D9</color>      (blue-gray)
    </bgColor>
  </headerCell>
</headRow>

<headCol>
  <headerCell>
    <bgColor>
      <colorFormat>numeric</colorFormat>
      <color>#93DB70</color>      (light green)
    </bgColor>
  </headerCell>
</headCol>
```

```

<altBodyRow>
  <bgColor>
    <colorFormat>numeric</colorFormat>
    <color>#FF6EC7</color>      (pink)
  </bgColor>
</altBodyRow>

<altBodyCol>
  <bgColor>
    <colorFormat>numeric</colorFormat>
    <color>#70DBDB</color>      (turquoise)
  </bgColor>
</altBodyCol>

```

With this template, row grids look like this:

The screenshot shows a Microsoft Internet Explorer window with the title bar "http://localhost/scripts/MRWEBPL.DLL?ACTION - Microsoft Internet Explorer". The menu bar includes File, Edit, View, Favorites, Tools, and Help. The toolbar has icons for Back, Forward, Stop, Refresh, Home, and Favorites. The main content area displays a grid titled "SPSS MR -- Tea Survey" with the subtitle "RGRID Which country or countries produces ?". The grid has a green header row and a pink header column. The rows represent different tea types: Assam, Black, Ceylon, Darjeeling, Green, Gunpowder, Lapsang Souchong, Oolong, Orange Pekoe, Puerh, and Rooibos. The columns represent countries: China, India, Japan, Kenya, South Africa, Sri Lanka, and Don't know. The cells in the grid are colored according to the template rules: green for the top-left cell, pink for the rest of the first row and first column, and pale blue for the remaining cells.

	China	India	Japan	Kenya	South Africa	Sri Lanka	Don't know
Assam	<input type="checkbox"/>						
Black	<input type="checkbox"/>						
Ceylon	<input type="checkbox"/>						
Darjeeling	<input type="checkbox"/>						
Green	<input type="checkbox"/>						
Gunpowder	<input type="checkbox"/>						
Lapsang Souchong	<input type="checkbox"/>						
Oolong	<input type="checkbox"/>						
Orange Pekoe	<input type="checkbox"/>						
Puerh	<input type="checkbox"/>						
Rooibos	<input type="checkbox"/>						

Previous | Next

If you follow the rules, you will see that the cells are assigned colors as follows:

- The top left cell takes its green color from *headCol*.
- The remaining cells in the lefthand column take their color from *headCol* too.
- The pale blue cells in the top row come from *headRow*.

- The pink and pale yellow cells form the body of the table and are controlled by *altBodyRow* and *altBodyCol*, but because this is a row grid only *altBodyRow* applies. Odd-numbered body rows take their pink color from *altBodyRow*, while the even-numbered rows take their pale yellow from *allCells*.

Column grids are formatted as follows:

The screenshot shows a Microsoft Internet Explorer window with the title bar "http://localhost/scripts/MRWEBPL.DLL?ACTION - Microsoft Internet Explorer". The menu bar includes File, Edit, View, Favorites, Tools, and Help. The window content is titled "SPSS MR -- Tea Survey" and contains the following text: "CGRID" and "Which country or countries produces?" Below this is a table with columns labeled: Assam, Black, Ceylon, Darjeeling, Green, Gunpowder, Lapsang Souchong, Oolong, Orange Pekoe, Puerh, and Rooibos. The rows represent countries: China, India, Japan, Kenya, South Africa, Sri Lanka, and "Don't know". The first cell (China, Assam) is green, followed by a row of pale yellow cells (Black, Ceylon, Darjeeling). The next row (India) has the first three cells (Assam, Black, Ceylon) in pale blue, followed by a row of turquoise cells (Darjeeling, Green, Gunpowder). The next row (Japan) has the first three cells in pale yellow, followed by a row of turquoise cells. This pattern continues for the remaining rows. At the bottom of the table are "Previous" and "Next" buttons.

	Assam	Black	Ceylon	Darjeeling	Green	Gunpowder	Lapsang Souchong	Oolong	Orange Pekoe	Puerh	Rooibos
China	<input type="checkbox"/>										
India	<input type="checkbox"/>										
Japan	<input type="checkbox"/>										
Kenya	<input type="checkbox"/>										
South Africa	<input type="checkbox"/>										
Sri Lanka	<input type="checkbox"/>										
Don't know	<input type="checkbox"/>										

The cell colors are assigned as follows:

- The top left cell takes its green color from *headCol*.
- The remaining cells in the lefthand column take their color from *headCol* too.
- The pale blue cells in the top row come from *headRow*.
- The turquoise and pale yellow cells form the body of the table and are controlled by *altBodyRow* and *altBodyCol*, but because this is a column grid only *altBodyCol* applies. Odd-numbered body columns take their turquoise color from *altBodyCol*, while the even-numbered columns take their pale yellow from *allCells*.

24.18 Overall characteristics of the grid table

Quick Reference

To define the overall grid characteristics, place the necessary statements within the following tags:

<tableDef>...</tableDef>

mrInterview converts the formatting instructions in the grid template into standard HTML code which it then uses to lay out the grid on the interview page. You can specify the following characteristics that will affect the overall appearance of the grid:

- The width of the table relative to the page width.
- The horizontal alignment of the table on the page.
- The width of the table border, if any.
- The amount of spacing between the columns of the grid.
- The amount of spacing between the contents of a cell and the borders of the cell.

Grid width

Quick Reference

To set the width of the grid relative to the page width, type:

```
<width>
  <widthFormat>type</widthFormat>
  <value>number</value>
</width>
```

where *type* defines how to interpret the *value* parameter, and is either *absolute* if *value* is the width of the table in pixels, or *percent* if *value* is the table width as a percentage of the total grid width.

If the respondent resizes the browser window, the grid's width and the column widths increase or decrease proportionally.

Grid alignment

Quick Reference

To define the horizontal position of the grid on the page, type:

```
<align>position</align>
```

where *position* is *left*, *right* or *center*. The default is left alignment.

Borders

Quick Reference

To draw a border around the grid and each cell in the grid, type:

<borderSize>number</borderSize>

where *number* is the width in pixels of the border. A value of 0 suppresses the border.

mrInterview does not set the border color so the browser's default applies.

-
- 〝 Drawing a border around a grid forces borders around the cells of that grid. You cannot have a grid that has a border but whose cells have no borders.
-

Space between rows and columns

Quick Reference

To set the amount of horizontal and vertical space between each cell, type:

<cellSpacing>number</cellSpacing>

where *number* is the number of pixels of space to leave between the rows and columns of the grid.

mrInterview uses the standard HTML rules for table layout when displaying grids. Depending on the number of columns in the grid, the width of the columns and the amount of column heading text to be displayed in each column, this may or may not result in a layout that is attractive and easy to follow. If you find that the column headings seem to be running into one another, you can increase the amount of space between the columns. Not only will this make the columns easier to see, but it may also reduce the column widths slightly which may improve usability.

The same problem can apply to rows. If the grid is small, you may prefer to increase the space between the rows so that the grid fills more of the page.

If the grid has different colors defined for the grid background and the cell background, and *<cellSpacing>* is greater than 0, the spaces between the cells will be shown in the grid background color.

-
- 〝 Cell spacing applies to all aspects of each cell of the table. You cannot increase the horizontal or vertical spacing in isolation, nor can you specify different spacing for rows and columns.
-

Space around cell contents

Quick Reference

To increase the space around cell texts, type:

<cellPadding>number</cellPadding>

where *number* is either the number of pixels or the percentage of space to leave above, below and on each side of the cell contents.

If you do not want to alter the space between rows and columns, another way of improving readability is to increase the amount of space between the cell margins and the cell contents. By increasing the amount of white space here, you may force the column or row text to be spread over more lines so that it no longer runs into the text for the next row or column.

Example

This section illustrates how you can change the appearance of a grid simply by changing its overall characteristics. The grid specification is as follows:

```

<gridFormat>
  <name>Table Characteristics</name>
  <tableDef>
    <width>
      <widthFormat>percent</widthFormat>
      <value>80</value>
    </width>
    <borderSize>1</borderSize>
    <cellSpacing>3</cellSpacing>
  </tableDef>
  <bcolor>
    <colorFormat>string</colorFormat>
    <color>#8FBC8F</color>
  </bcolor>
</gridFormat>

```

The grid is 80% of the width of the browser window, and will reduce or increase in size proportionally as the respondent resizes the browser window. The grid's background is green and there is a border one pixel wide around the whole grid and around each cell, and three pixels of space between each row and column.

http://localhost/scripts/MRWEBPL.DLL?ACTION - Microsoft Internet Explorer

File Edit View Favorites Tools Help

SPSS MR -- Tea Survey

RGRID
Which country or countries produces?

	China	India	Japan	Kenya	South Africa	Sri Lanka	Don't know
Assam	<input type="checkbox"/>						
Black	<input type="checkbox"/>						
Ceylon	<input type="checkbox"/>						
Darjeeling	<input type="checkbox"/>						
Green	<input type="checkbox"/>						
Gunpowder	<input type="checkbox"/>						
Lapsang Souchong	<input type="checkbox"/>						
Oolong	<input type="checkbox"/>						
Orange Pekoe	<input type="checkbox"/>						
Puerh	<input type="checkbox"/>						
Robbos	<input type="checkbox"/>						

Previous Next

The *<cellPadding>* tag increases the amount of blank space around the contents of each cell. If the specification contains the line:

```
<cellPadding>3</cellPadding>
```

the grid becomes:

The screenshot shows a Microsoft Internet Explorer window with the title bar "http://localhost/scripts/MRWEBPL.DLL?ACTION - Microsoft Internet Explorer". The main content area displays a survey grid titled "SPSS MR -- Tea Survey". The grid is titled "RGRID" and asks "Which country or countries produces?" Below the title, there is a table with rows for tea types (Assam, Black, Ceylon, Darjeeling, Green, Gunpowder, Lapsang Souchong, Oolong, Orange Pekoe, Puerh, Rooibos) and columns for countries (China, India, Japan, Kenya, South Africa, Sri Lanka, Don't know). Each cell contains a small square checkbox. At the bottom of the grid, there are "Previous" and "Next" navigation buttons.

	China	India	Japan	Kenya	South Africa	Sri Lanka	Don't know
Assam	<input type="checkbox"/>						
Black	<input type="checkbox"/>						
Ceylon	<input type="checkbox"/>						
Darjeeling	<input type="checkbox"/>						
Green	<input type="checkbox"/>						
Gunpowder	<input type="checkbox"/>						
Lapsang Souchong	<input type="checkbox"/>						
Oolong	<input type="checkbox"/>						
Orange Pekoe	<input type="checkbox"/>						
Puerh	<input type="checkbox"/>						
Rooibos	<input type="checkbox"/>						

Using a different background for the cells of the grid makes the cells look more box-like. If the following statements are added to the example specification:

```
<tableDef>
  <borderSize>2</borderSize>
</tableDef>
<allCells>
  <bgColor>
    <colorFormat>numeric</colorFormat>
    <color>#FF6EC7</color>
  </bgColor>
</allCells>
```

the grid has pink cells on a green background:

http://localhost/scripts/MRWEBPL.DLL?ACTION - Microsoft Internet Explorer

File Edit View Favorites Tools Help

SPSS MR -- Tea Survey

RGRID
Which country or countries produces?

	China	India	Japan	Kenya	South Africa	Sri Lanka	Don't know
Assam	<input type="checkbox"/>						
Black	<input type="checkbox"/>						
Ceylon	<input type="checkbox"/>						
Darjeeling	<input type="checkbox"/>						
Green	<input type="checkbox"/>						
Gunpowder	<input type="checkbox"/>						
Lapsang Souchong	<input type="checkbox"/>						
Oolong	<input type="checkbox"/>						
Orange Pekoe	<input type="checkbox"/>						
Puerh	<input type="checkbox"/>						
Rooibos	<input type="checkbox"/>						

Previous Next

25 Using multiple script files (subsurveys)

Quancept does not require the whole script to be defined in one file. Where a script divides neatly into sections, you may prefer to create some sections as separate scripts and then include them in the main script at the appropriate point. This is particularly helpful for omnibus studies and for scripts which share the same set of questions.

Scripts for omnibus studies often consist of many sections which could stand as scripts in their own right. You will run the same main script all the time but may want to vary the sections used or the respondents of whom a section is asked. Splitting the script into separate scripts makes it easy to manipulate the various sections without having to make major changes to the overall layout of the main script. We call these variable scripts *subsurveys*.

Some companies like to use the same demographic questions or introductory texts for all their interviews. Rather than retype this information each time, you may build a library of standard questions or texts which can be loaded into other scripts as and when required. We call these general script sections *library files*.

The keywords associated with these facilities are:

ask '#SUB'	load a subsurvey
include	load a library file

This chapter also covers the callable functions **getsmvar** and **putsmvar**, but only as they are used with subsurveys. These functions are primarily used for passing information between the script and the sample management system and are described in detail in chapter 29, Using information in sample records.

25.1 Loading library files

Quick Reference

To include the contents of a library file in your script, type:

include [path/]filename

If the file is in the project directory, the *path* is optional; you may enter just the file's name.

Library files are segments of a script, never complete scripts which can stand on their own. They must always end with a *continue* statement directing the interviewing program to continue with the next statement in the main script, rather than with an *end* statement. When the parser reaches an *include* in the script, it stops reading the main file and reads and checks the file named with *include*. When the end of the library file is reached, the interviewing program skips back and reads the rest of the main script file.

Similarly, when the interviewing program reaches the *include*, it executes every statement in the library file before continuing with the rest of the questions in the main file. Any data gathered by questions in the library file is written to the main data file at the point at which the questions were asked.

-
- ❖ In early versions of Quancept, the file name had to be typed in single quotes, and later versions will accept old scripts with filenames in single quotes.

If you will be using the Quancept CATI qcset utility you must still type filenames in single quotes.

25.2 Running one script from inside another



Quick Reference

To run a subsurvey from within the main script, type:

```
label  ask '#SUB=script'  
      resp num 1
```

The advantages of using subsurveys is that:

- you have a choice of subsurvey scripts and a flexible method of selecting the one to use
- you can keep everything about the subsurvey script separate from the main script

The question that calls the subsurvey is not displayed during the interview; it is simply a means of telling the interviewing program to call another script.

Because the script name is part of a text, you may set it as a variable rather than hard-coding it into the question line. This allows you to call up one of a number of subsurvey scripts based on the answers given to previous questions. For example:

```
tc      ask 'Do you drink tea or coffee most frequently?'  
      resp sp 'tea' / 'coffee' go cof /  
          'Don''t drink them at all' go ctn  
      set script = 'tea'  
      goto subrun  
cof     set script = 'coffee'  
comment Run the subsurvey  
subrun  ask '#SUB='+script+' '  
      resp num 1  
ctn     continue
```

Parsing subsurvey scripts

Scripts used with the #SUB facility are not integrated into the main script when you parse the main script, as those used with *include* are. You must parse all subsurvey scripts separately before they can be used, so each subsurvey must finish with an *end* statement rather than a *continue* statement. Therefore, it does not matter whether the subsurvey script exists when you parse the main script. This is particularly useful with omnibus studies where subsurvey scripts are being added and deleted on a daily basis.

-
- ❖ *mapothzero* in the main script does not carry through to the subsurvey scripts. If you want specified other to be coded as 0 for all scripts, you must insert this keyword as the first line of every script to which it applies.
-

Subsurvey scripts may exist in the main project directory or in a subdirectory of the project directory with the same name as the subsurvey script. The location of the script's qoc file determines where the interviewing program will create the files associated with that script. For example, if the tea script in our example is located in the subdirectory ./subtea, the interviewing program will create subtea.dat, subtea.tex, and so on, in that directory.

When the interviewing program reaches a #SUB statement, it searches for the qoc file first in a subdirectory with the same name as the subsurvey script (for example, ./subtea/subtea.qoc), and then in the current (that is, main project) directory. If the qoc file is found in the subdirectory, it is executed from there and the main project directory is not searched.

If the interviewing program does not find the qoc file in either directory, it looks instead for a file called *subsurvey.dum* in either the current directory (for example, ./subtea.dum) or the named subdirectory (for example, ./subtea/subtea.dum). The *subsurvey.dum* file allows the main project script to be run regardless of whether or not the *subsurvey* scripts are ready to be run. If neither the named qoc file nor the *subsurvey.dum* file exists, the interviewing program terminates with the message 'Cannot access *subsurvey.qoc*'.

Storing subsurvey data

Data and other files are created for subsurveys as if they were completely separate projects. In the example given earlier, the interviewing program creates three sets of date files: one set for the main script and one set each for the tea and coffee subsurveys.

-
- ❖ Data for the subsurveys is related to the main script's data by the respondent ID; all data for one respondent has the same serial number across all the scripts involved. (This is why you cannot snap back into a subsurvey, as the mechanism that guarantees the same serial number can only do so if you enter the subsurvey once only.)
-

When a subsurvey starts, a message naming the subsurvey flashes briefly on the screen. At the end of the subsurvey, the message 'End of subsurvey *name*. Hit any key to continue' is issued. Interviewers can suppress these messages by setting the environment variable QCSEGUE to 1.

At the end of the subsurvey, interviewers are offered the chance to inspect or change answers in the subsurvey and to append verbatims if necessary. Interviewers may suppress these prompts by setting the environment variables QCNOAPPEND and QCNOCHANGE to 1 or 3 (1 suppresses the prompts in the subsurvey only, whereas 3 suppresses them in the subsurvey and the main script).

☞ For further information see section 39, Managing CATI projects.

Once the interviewer leaves these questions, the subsurvey data is written out and the interviewer is returned to the next question in the main script. The question which called the subsurvey (subrun in the example above) will have one of three values:

- 2 The subsurvey script was successfully completed or was terminated with *signal 1*. This is the usual result.
- 1 The interviewer typed *stop* during the subsurvey script, or the interview was stopped by a *signal 9* or *signal 10* statement in the script, or by a *stop* statement. This stops the subsurvey script and the master script. (This response value is also set if the subsurvey fails to start, or if the subsurvey's qtip process crashes part-way through the subsurvey, or if the qtip process returns an errorstop message.)
- 0 The interviewer typed *quit* or *abandon* during the subsurvey script. The subsurvey script is terminated, with or without data as appropriate, but the interview continues with the next question in the master script.

If a subsurvey completes successfully and the interviewer snaps back and then steps forward through the script, the interviewing program will ignore the question that calls the subsurvey.

Scripts with more than one subsurvey

If the main script calls more than one subsurvey, it is possible that if the interviewer snaps back and changes a response, it will be necessary to run a different subsurvey to the one that was originally run. What happens during the interview depends on the way you have written the script. If you have used a single #SUB statement with the script name defined as a variable, changing the name of the script variable will not force the interviewing program to run that subsurvey. If the script contains the statements:

```
tc      ask 'Do you drink tea or coffee most frequently?'
        resp sp 'tea' / 'coffee' go cof
        set script = 'tea'
        goto subrun
cof     set script = 'coffee'
subrun  ask '#SUB='+script+' '
        resp num 1
```

and the respondent drinks tea most frequently, the interviewing program will execute the tea subsurvey. If the respondent later changes his/her mind and says he/she drinks coffee instead, the interviewing program will see that 'subrun' already has a value and will skip to the next statement in the script.

The alternative is to write a separate #SUB statement for each subsurvey. Our example would become:

```
tc      ask 'Do you drink tea or coffee most frequently?'
        resp sp 'tea' / 'coffee' go cof
tea     ask '#SUB=tea'
        resp num 1
        goto ctn
coffee  ask '#SUB=coffee'
        resp num 1
ctn    continue
```

When the interviewer snaps back and changes the answer to the tc question from tea to coffee, the interviewing program will run the coffee script because the coffee question has no value. It will not, however, change the value of the tea question nor will it delete the data for this respondent from the tea data files. At the end of the interview, you will have all the data you need, plus the data for the tea subsurvey, which you can ignore.

Because of this, you may wish to use *echo* on the response list to the tc question so that the interviewer can verify that the response is correct before the subsurvey is called.

Recording subsurveys

If you have the Quancept Telephony System (QTS) installed, you can record subsurveys.

☞ See 'Recording subsurveys' in chapter 6, Writing multimedia scripts, for details.

Passing data between the main and subsurvey scripts



Quick Reference

To pass information from the main survey to the subsurvey, use **putsmvar** in the main script and **getsmvar** in the subsurvey script. In the main script, type:

```
callfunc('putsmvar', hold_varname, msdata_varname)
```

where *hold_varname* is the name of a variable in which the data will be held until it is read by the subsurvey, and *msdata_varname* is the name of the variable whose value is the data to be passed from the main survey to the subsurvey.

To pick up the data in the subsurvey script, type:

```
callfunc('getsmvar', hold_varname, ssdata_varname)
```

where *hold_varname* is the name of the variable holding the data passed from the main script, and *ssdata_varname* is the name of the variable in which that data from the main survey will be placed.

These statements may be used in reverse to pass data from the subsurvey to the main script. In this case, a notional database record key must be set in the main script and passed to the subsurvey:

```
set dbrecval = respondent
set vname = 'dbkey'
callfunc('putsmvar', vname, dbrecval)
```

An empty directory called 'database' is also required.

Although you can run a subsurvey script from within the main script as if it were part of the main script, Quancept still sees the two scripts as completely separate things. The two scripts must be parsed individually, and data for the two scripts is then written to separate files. This means that you cannot pass information between the two scripts or access variables in one script from the other simply by naming the variables you wish to use. For example, if the question 'maincar' appears in the main script, you cannot access it in the subsurvey script by typing the name 'maincar'. In fact, if the subsurvey script also contains a question called maincar, the interviewing program will assume you mean this variable rather than the one in the main script.

There is, however, a way to get around this using keywords that are normally associated with SMS sample files and databases. To pass information from the main script to the subsurvey script, you send it with *putsmvar* and pick it up in the subsurvey script with *getsmvar*. To pass data from the subsurvey script to the main script, you send it with *putdbvar* and pick it up in the main script with *getdbvar*.

-
- ❖ There is no need to set up any SMS sample files or start the SMS server unless your script requires them for other purposes. However, you must create a directory called database for *putdbvar* and *getdbvar* to use. If you find that variables in the main script which receive data from the subsurvey script are null, check that you have created the database directory.
 - ☞ For full documentation on these keywords, see chapter 29, Using information in sample records.
-

You can test this facility by running the sample script for this chapter. For the moment, we will look at a variation of that script. The main script contains the statements:

```

caff      ask 'Are you interested in low caffeine drinks?'
          resp coded
          set script = 'tea'

comment Place answer to 'caff' in the variable 'frommain' and pass
comment to the subsurvey
          set vname = 'frommain'
          callfunc('putsmvar',vname,caff)

comment Set up database record key so subsurvey script can pass
comment info back to main. Pass variable to subsurvey script
comment for use. Not necessary if data is not passed from the
comment subsurvey script.
          set dbrecval = respondent
          set vname = 'dbkey'
          callfunc('putsmvar',vname,dbrecval)

comment Run the subsurvey
subrun    ask '#SUB='+script+
          resp num 1

comment Pick up data from subsurvey. The subsurvey will have placed
comment data in a variable called 'fromsub'
          set vname = 'fromsub'
          set dcbrand = ''
          callfunc('getdbvar',vname,dcbrand)
display '@@BACK IN MAIN SCRIPT.@@Data passed from subsurvey
script is '+dcbrand
          pause

```

The subsurvey script, sub1, is as follows:

```

d1      display '@@STARTING SECONDARY SCRIPT@@'
comment Pick up interest in caffeine-free drinks from main script
          set vname = 'frommain'
          set gottxt = ''
          callfunc('getsmvar',vname,gottxt)
d2      display 'Text from main script is '+gottxt
          pause

```

```
brand    ask 'Which brands of low caffeine teas have you tried?'
         resp coded
comment Send data back to main script
         set vname = 'fromsub'
         callfunc('putdbvar',vname,brand)
d3      display '@@ABOUT TO LEAVE SECONDARY SCRIPT@@'
         pause
end
```

putsmvar, *getsmvar*, *putdbvar* and *getdbvar* all pass data as a text. Our example uses open ends, which makes things straightforward. The interviewing program converts numeric values to texts automatically, but if you want to pass single-punched or multipunched data, you must convert it to text before you send it, and then convert it back to its proper type in the destination script.

Here is an example which passes the value of a single-punched variable from the main script to the subsurvey. Notice that we have passed the response's position as a text value rather than using the response text itself. This is because, although the response text looks like a text to us, the interviewing program sees it as a single-punched value which it treats differently to a text. Notice also that we do not define a database key variable because we are not passing anything from the subsurvey script to the main one.

The main script is:

```
caff    ask 'Are you interested in low caffeine drinks?'
         resp sp 'yes' / 'no'
         set pos = nbit(caff)
         set vname = 'frommain'
         callfunc('putsmvar',vname,pos)
```

The subsurvey script is:

```
subcaff  dummyask 'Interested in low caffeine drinks?'
         resp sp 'are interested in low caffeine drinks' /
                 'not interested' nodata
         set vname = 'frommain'
         callfunc('getsmvar',vname,txtvar)
         if (txtvar = '1') {
             set subcaff = 1
         }
         if (txtvar = '2') {
             set subcaff = 2
         }
         route (subcaff = 'not interested') go ctnt2
dcaff    ask 'You said that you '+subcaff+'. Have you tried any
         of the low caffeine teas marketed by The Healthy Drinkers Company?'
         resp sp 'yes' / 'no'
ctn      continue
```

25.3 Sample script

 To run this sample script, you will need to parse the subsurvey scripts as well as the main script, and also create a database directory called ‘database’.

```
comment Sample script for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 25, Using multiple script files (CATI only)
caff    ask 'Are you interested in low caffeine drinks?'
        resp sp 'yes' / 'no' / 'never thought about it'
tc      ask 'Do you drink tea or coffee most frequently?'
        resp sp 'tea' / 'coffee' go cof /
                'Don''t drink them at all' go ctn
        set script = 'tea'
        goto setup
cof     set script = 'coffee'

comment Pass information to subsurvey
setup   set pos = nbit(caff)
        set vname = 'frommain'
        callfunc('putsmvar',vname,pos)

comment Set up and pass database record key to subsurvey
        set dbrecval = respondent
        set vname = 'dbkey'
        callfunc('putsmvar',vname,dbrecval)

comment Run the subsurvey
subrun  ask '#SUB='+script+' '
        resp num 1
        route (caff='no') go ctn

comment Pick up and convert data from subsurvey
        set vname = 'fromsub'
        set dcbrand = ''
        callfunc('getdbvar',vname,dcbrand)
        route (dcbrand='null') go ctn
        if (script='tea') {
            if (dcbrand='1') {
                set type = 'camomile'
            }
            if (dcbrand='2') {
                set type = 'red berries'
            }
            if (dcbrand='3') {
                set type = 'orange surprise'
            }
            if (dcbrand='4') {
                set type = 'lemon'
            }
        }
```

```
        if (dcbrand='5') {
            set type = 'strawberry and mint'
        }
    }
    if (script = 'coffee') {
        if (dcbrand='1') {
            set type = 'honey roast'
        }
        if (dcbrand='2') {
            set type = 'columbian'
        }
        if (dcbrand='3') {
            set type = 'mocha'
        }
    }

comment Use subsurvey data in main script
sample   ask 'Would you like a free sample of '+type+' '+tc+ '?'
          resp sp 'yes' / 'no'
ctn      continue
drink    ask 'What other non-alcoholic beverages do you drink?'
          resp mp 'milk' / 'fruit juices' / 'water' / 'chocolate' /
                 'None ^s' other

comment Run a library script
      include 'demog'
thank    display 'Thank you for your co-operation'
end
```

The script for the tea subsurvey is as follows:

```
comment Sample subsurvey script for Quancept Scriptwriter's Manual,
comment Chapter 25
d1       display 'Starting subsurvey'
          pause
hdc      define 'camomile' / 'red berries' /'orange surprise' /
          'lemon' / 'strawberry and mint'
usual    ask 'Do you use loose tea, tea bags or instant tea?'
          resp mp 'loose tea' / 'tea bags' / 'instant tea'
type     ask 'Which types of tea do you drink?'
          resp mp 'assam' / 'earl grey' / 'lapsang souchong' /
          'darjeeling' / 'herbal teas' / 'english breakfast'
how      ask 'How do you drink your tea?'
          resp sp 'black (as it comes)' / 'with milk' / 'with sugar' /
          'with lemon' / 'with milk and sugar' other
```

```

comment Pick up information passed from main script
subcaff  dummyask 'Interested in low caff drinks?'
          resp sp 'are interested in low caffeine drinks' /
          'not interested' /
          'have never thought about low caffeine drinks'
nodata
set vname = 'frommain'
callfunc('getsmvar',vname,txtvar)
if (txtvar = '1') {
    set subcaff = 1
}
if (txtvar = '2') {
    set subcaff = 2
}
if (txtvar = '3') {
    set subcaff = 3
}
route (subcaff = 'not interested') go ctnt2
dcaff   ask 'You said that you '+subcaff+'. Have you tried any of
the low caffeine teas marketed by The Healthy Drinkers Company?'
          resp sp 'yes' / 'no' go ctnt1
hdctry  ask 'Which ones have you tried?'
          resp mp hdc
ctnt1   continue

comment Select one untried flavor at random and pass back to main script
l1       for item = 1 to 5 ran
          set chosen = bit(hdctry/item)
          route (chosen) go nxtone
          set vname = 'fromsub'
          callfunc('putdbvar',vname,item)
          goto ctnt2
nxtone  next
ctnt2   display 'About to leave subsurvey'
          pause
          end

```

The script for the coffee subsurvey is similar:

```

comment Sample subsurvey script for Quancept Scriptwriter's Manual,
comment Chapter 25
d2       display 'Starting subsurvey'
          pause
hdc      define 'honey roast' / 'columbian' / 'mocha'
usual    ask 'Do you use fresh coffee, coffee bags or instant
coffee?'
          resp mp 'fresh coffee' / 'coffee bags' / 'instant coffee'
type     ask 'Which types of coffee do you drink?'
          resp mp 'columbian' / 'kenyan' / 'blue mountain'

```

Quancept and mrInterview Scriptwriter's Manual

```
how      ask 'How do you drink your coffee?'
        resp sp 'black' / 'with milk' / 'with sugar' /
               'with cream' / 'with milk and sugar' other

comment Pick up information passed from main script
subcaff  dummyask 'Interested in low caffeine drinks?'
        resp sp 'are interested in low caffeine drinks' /
               'not interested' /
               'have never thought about low caffeine drinks' nodata
set vname = 'frommain'
callfunc('getsmvar',vname,txtvar)
if (txtvar = '1') {
    set subcaff = 1
}
if (txtvar = '2') {
    set subcaff = 2
}
if (txtvar = '3') {
    set subcaff = 3
}
route (subcaff = 'not interested') go ctnt2
dcaff    ask 'You said that you '+subcaff+'. Have you tried any of
the low caffeine teas marketed by The Healthy Drinkers Company?'
        resp sp 'yes' / 'no' go ctnt1
hdctry   ask 'Which ones have you tried?'
        resp mp hdc
ctnt1    continue

comment Select one untried flavor at random and pass back to main
comment script
12      for item = 1 to 3 ran
        set chosen = bit(hdctry/item)
        route (chosen) go nxtone
        set vname = 'fromsub'
        callfunc('putdbvar',vname,item)
        goto ctnt2
nxtone   next
ctnt2    display 'About to leave subsurvey'
        pause
        end
```

The library script of demographic questions is as follows:

```
gender    ask 'Is the respondent ...'  
          resp sp 'male' / 'female'  
age       ask 'How old are you?'  
          resp num 18 to 99 ref  
region    ask 'Which of the following television areas covers the  
area in which you live?'  
          resp sp 'TVS' / 'Anglia' / 'HTV' / 'TSW' /  
                  'Yorkshire' / 'S4C' / 'Central'  
hhsizze   ask 'How many adults are there in your household?'  
          resp num 1 to 9  
income    ask 'What is the approximate annual income of your household?'  
          resp sp 'Under 10,000' / '10,001 to 20,000' /  
                  '20,001 to 30,000' / '30,000 to 40,000' /  
                  '50,000 and over' ref  
end
```


26 Text manipulation

This chapter introduces some statements for working with text. You will find out how to extract a string of characters from a text, how to convert characters into numbers, how to copy texts into variables, and how to concatenate a number of texts into a single variable. The keywords associated with these features are:

append	append one text to another
nsubstr	extract characters from a text and convert them to a number
setstr	extract characters from a text and copy them into a variable
substr	extract characters from a text

26.1 Appending one text to another

Quick Reference

To append one text to another, type:

callfunc('append', dest_var, source_var)

where *dest_var* is the name of the text variable to be appended to, and *source_var* is the text to be appended.

The texts can be open-ended responses (including answers to specified other) or temporary variables that have had some form of text set into them.

The example below suggests how you might use this facility to merge the name of the brand the respondent tried with the things he/she says that he/she likes about that brand. This can be useful for coding purposes when you want to assign different codes according to which brand the answers refer to, even though all the answers are coded in the same field. For example, you could code liking the smell of brand A differently to liking the smell of brand B, even though both answers are coded in the same column or field.

```
brand    ask 'Which brand did you try?'
        resp sp 'Suds' / 'Washo' / 'Gleam' / 'Sparkle'
comment Question whose answer is to be built with append
dumlike dummyask 'What liked about brand tried'
        resp coded
like     ask 'What did you like about '+brand+'?'
        resp coded nodata
comment Get text of brand tried
        set brandno = nbit(brand)
        set brdtry = elm(brand/brandno)
```

```
comment Set punctuation to join the two texts and append it to
comment brand tried
    set punct = ': '
    callfunc('append',brdtry,punct)
comment Join reasons liked to brand tried and assign to dumlike for
comment storing in tex file for coding
    callfunc('append',brdtry,like)
    set dumlike = brdtry
d1      display 'Brand chosen and reasons liked are: '+dumlike
    pause
```

The points to notice about this example are as follows. First, we have used a dummy question to store the reasons the respondent gives for liking the product. The main question whose texts will be copied into the text file has its answer set at the end of the script when the text will also contain the name of the brand tried. Second, we have defined some punctuation to separate the two responses. Without this, the responses would be concatenated with no intermediate spaces or punctuation, as follows:

```
WashoLeaves clothes looking bright and clean
```

With punctuation, the two items are more clearly identifiable:

```
Washo: Leaves clothes looking bright and clean
```

The example writes the merged texts to the text responses file, but if the client wants a printed list of reasons for liking and disliking each product, you could write the texts out to report files instead.

There are some restrictions associated with using *append*. They are:

- You may append only one text at a time. If you want to append several texts, write a separate *append* statement for each one. In our example, we used *append* twice: the first time for the punctuation, and the second for the reasons liked.
- You should append texts to a question variable only if that variable already has a non-null value. If you want to append to a *coded* question and the script contains routing which may cause that question to be skipped, assign a dummy text value to the question before the *append* is executed:

```
if (qname = null) {set qname = ''}
```

-
- . In mrInterview, you will need to set the variable *noaneqn1* to a non-zero value before this test to force mrInterview to treat unanswered questions as if they had a null value. If you simply want to test whether the question is unanswered, use the expression (*qname=empty*) instead.
 - + See chapter 14.11, ‘Testing for null-response and unanswered questions’ for details of both options.
-

If the question appended to is on the path through the interview (that is, is executed during the interview), additional care should be taken to preserve the integrity of the script.

The same applies to the text to be appended. If the text is the answer to a question and if that question may be unanswered, you must assign a dummy value when the question is unanswered, as shown in the example below.

- We recommend that you do not append texts to questions and variables on the interviewer's path; that is, to questions that are asked or to dummy questions or variables whose values determine which questions are displayed. Doing so may affect the way the script is replayed if the interviewer snaps back to a statement before the *append*. For example:

```

see      ask 'What brands did you see advertised?'
        resp mp 'Suds' / 'Washo' / 'Gleam' / 'Sparkle'
comment If Washo advert seen, ask about it
        set seewash = or(see/'Washo')
        if (seewash) {
visual     ask 'What impression did you get of Washo?'
        resp coded
}
hear      ask 'For which brands did you hear advertising?'
        resp mp 'Suds' / 'Washo' / 'Gleam' / 'Sparkle'
comment If Washo advert heard, ask about it
        set hearwash = or(hear/'Washo')
        if (hearwash) {
audio     ask 'What impression did you get of Washo?'
        resp coded
}
comment Append audio to visual. If either question is
comment unanswered assign a dummy value. For mrInterview,
comment remember to set noaneqnl=1 first or change 'null' to 'empty'.
        if (visual = null) {
            set visual = ''
}
        if (audio = null) {
            set audio = ''
}
set punct = ''
callfunc('append',visual,punct)
callfunc('append',visual,audio)

```

In this script, *visual* is asked only if the respondent saw advertising for Washo. If not, it is skipped; the question is not on the interviewer's path and, therefore, has an unanswered (null) value.

Next, we execute a similar set of statements for radio advertising, after which we merge the two sets of answers to get an overall answer for Washo. However, in doing so, we change the value of *visual*; it is now a *coded* response with either a blank value if advertising was not seen, or a text value if it was. Because *visual* now has a value, it will appear in a listing of all questions and their values if the interviewer types ??, even though the respondent did not mention Washo at the 'see' question. This may be misleading.

The correct way to deal with this situation is to merge the two answers using a dummy variable, as in our previous example.

- *append* writes data into the interviewing program's memory (technically called the heap space); in some circumstances, snapping back and re-executing an *append* causes text to be appended a second time.

26.2 Extracting strings from texts and converting them to numbers



Quick Reference

To extract a string of characters from a longer text and convert the result into an integer, type:

```
set dest_var = nsubstr (text_var, start, num_char)
```

where *dest_var* is the variable that will hold the integer result, *text_var* is the name of an open end or text variable containing the text to be copied, *start* is the position in *text_var* of the first character to copy, and *num_char* is the number of characters to copy (maximum nine).

The *nsubstr* keyword converts characters that appear in an open end or a text variable into an integer and stores them in a numeric variable. You can use this keyword to extract numeric information from the responses given to an open-ended question. You can also use *nsubstr* to read in data from an external file containing numeric values, and convert the texts to integer values before setting single-punched or multipunched questions to these values. For example:

```
comment Set the current time
      set now=time
comment Extract hour at which interview started
      set schar=1
      set nchar=2
      set hours=nsubstr(now,schar,nchar)
comment Extract minutes at which interview started
      set schar=4
      set mins=nsubstr(now,schar,nchar)
comment Calculate end time, assuming that the interview takes 10 minutes
      set length=10
      set mins=mins+length
      route(mins < 60) go okmins
comment mins is now >60 so add 1 to the hour and subtract 60 from mins
addhour  continue
      set mins=mins-60
      set hours=hours+1
      route(mins > 59) go addhour
okmins  continue
comment Ensure minutes is a 2-digit number. This avoids displays like
comment 11:0 or 7:1
      set mindisp=''
      route(mins>9) go showit
      set mindisp='0'
```

```
showit    display 'This survey will only take '+length+' minutes of your
           time, so we should be finished at about '+hours+':'+mindisp+'.'+mins+'.'
           pause
```

In this example, the *time* keyword is used to set the current time into a temporary variable called ‘now’. Quancept returns the time in the format: hh:mm:ss. The *nsubstr* keyword is then used to extract the values in the hour and minute fields so that the script can calculate the time at which the interview should finish.

The hour at which the interview started is extracted by reading the first two characters from the time string. This process is repeated to extract the minutes, except that two characters are read starting at character four.

Now that the hours and minutes are held separately, the script adds the expected duration of the interview to the number of minutes. If this causes the number of minutes to be greater than 60, the script increases the hour by one and subtracts 60 from the count of minutes. For example, if the interview started at 10:55am, adding 10 minutes to the minutes gives 65 minutes. After manipulation the hour variable contains 11 and the mins variable contains 5.

So that the end time is always displayed in the proper format, the script inserts a zero in front of the number of minutes if the value of mins is less than 10. In the example, this produces an end time of 11:05am.

26.3 Extracting characters from a text and copying them into a variable



Quick Reference

To extract characters from a text variable and place them in a second variable, type:

setstr (*dest_var*, *start*, *num_char*, *text_var*)

where *dest_var* is the name of the variable in which the characters are to be placed; *start* is the position in *dest_var* at which to insert the first copied character; *num_char* is the number of characters you want to copy; and *text_var* is the name of a text variable containing the text to be copied.

To place a fixed text in a variable, type:

setstr (*dest_var*, *start*, *num_char*, 'text')

where *destvar*, *start* and *num_char* are as described above, and *text* is the text string to be copied.

The **setstr** keyword extracts characters from a text variable, or takes a given string of characters, and copies them into another variable. The *num_char* and *start* arguments specify the number of characters to be copied and the position in the destination variable to which the first character will be copied. These arguments can be either explicit integer values or integer (num) variables.

The following example illustrates how you might use *setstr* to find the starting point of a word in a string when you don't know exactly what text the string contains. The example also uses the *substr* keyword, which is described in the next section.

```
comment Create a text string for demonstration purposes.
      set fullstr='Eight bells and all''s well'
comment Loop through each character to check if it and the next 5
comment characters make up the word 'bells'.
11      for i = 1 to 25
          set check=substr(fullstr,i,5)
          if (check='bells'){
              setstr(bell,i,5,fullstr)
              goto foundit
          }
      next
d1      display 'Could not find the starting word'
      goto none
foundit  display 'Found the word '+bell
none     continue
```

26.4 Extracting strings from texts

Quick Reference

To extract a string of characters from a longer text, type one of the following:

All: **callfunc ('substr', *text_var*, *start*, *num_char*, *dest_var*)**

CATI: **set *dest_var* = substr(*text_var*, *start*, *num_char*)**

text_var is the name of an open end or text variable containing the text to be copied; *start* is the position in *text_var* of the first character to copy; *num_char* is the number of characters to copy (maximum 200); and *dest_var* is the name of the variable in which the text is to be placed.

The **substr** function and statement both extract a string of characters from a text and place them in a variable. A simple example is:

```
          set txt = 'The cat sat on the mat'
spos    ask 'Which character do you wish to start with?'
        resp num 1 to 22
many   ask 'How many characters to copy?'
        resp num 1 to 22
        set chars = spos + many
        route (chars <= 23) go ctn
err     display 'There are not that many characters to copy.
Please try again.'
        goto spos
ctn    callfunc('substr',txt,spos,many,copied)
```

```

comment Or: set copied=substr(txt,spos,many)
rep      display many+' characters starting at character '+spos+' is:
@'+copied

```

You can use this statement whenever you want to copy part of a text, either an open end or a temporary text variable, into another variable. In section 18.4, Writing to a report file, we showed how to build a file of names and addresses to use as the basis of a mailing list. We stored the address details as one line. If you asked interviewers to separate each address field with, say, a semicolon, you could use *substr* to split the address into lines so that the report file would be ready for use immediately.

To split a text whenever you reach a certain character in the text, you need to step through the text one character at a time, testing whether you have reached the field separator. As you move along the text, you count the number of characters read so far. On reaching the field separator (for example, the semicolon), you write out the number of characters you have counted, skip the separator, and reset the count of characters to 0, ready to read the next field. The following is an example of a script you could write. It assumes that the address contains no more than 100 characters so that it is easy control the number of times the loop is repeated.

```

comment Define output format for writing to report file
      set fmt = '%s\n'
comment CAPI syntax is: substitute fmt '%s\n'
address ask 'Could I have your full postal address please?@'
INTERVIEWER: TYPE ADDRESS ALL ON ONE LINE WITH ; AT THE END OF
EACH ADDRESS FIELD'
      resp coded nodata
      set pos = 1
      set fstart = 1
      set numchar = 0
11     for times = 1 to 100
          set numchar = numchar + 1
comment Read a single character from the address
      callfunc('substr',address,pos,1,copied)
      if (copied <> ';'){
          set pos = pos + 1
          goto nxtchar
      }
comment Read all characters belonging to this part of the address
comment Do not include the ; at the end of the field
      set numchar = numchar - 1
      set copied = substr(address,fstart,numchar)
comment NOTE: rfprint is not available in Quancept Web. You could
comment replace it with statements that write each address line
comment to a new card in the data file
      callfunc('rfprint',3,fmt,copied)
comment CAPI syntax for writing to report file is:
comment      rfprint 3,fmt,copied
      set pos = pos + 1
      set fstart = pos
      set numchar = 0
nxtchar  next

```

Subscripted variables with substr



Using subscripted variables as parameters to the *substr* callfunc fails at the compilation stage. This is a design feature of the function. If you need to use subscripted variables, write the function using unsubscripted variables and then extract the information you need using an assignment statement with a subscripted variable. For example, instead of writing:

```
callfunc('substr',intext,startpos,numchar,sampvar(fieldno))
```

you can write:

```
callfunc('substr',intext,startpos,numchar,outtext)
set sampvar(fieldno)=outtext
```

26.5 Sample scripts

Script A

```
comment Sample script A for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 26, Text manipulation (CATI and CAPI only)

comment *** Timing section is CATI only. Comment out for CAPI ***
comment Set the current time
    set now=time

comment Extract the hour at which the interview was started
comment from first 2 characters of now
    set schar=1
    set nchar=2
    set hours=nsubstr(now,schar,nchar)

comment Extract the minutes by taking 2 characters starting
comment at character 4
    set schar=4
    set mins=nsubstr(now,schar,nchar)

comment Interview takes ten minutes. Add this to the current
comment number of minutes past the hour to get the expected
comment end time. If this value is less than 60, there is
comment no need to adjust the value of hours
    set length=10
    set mins=mins+length
    route(mins < 60) go okmins

comment If greater than 60, take away an hour from the
comment value of mins, i.e. 60 mins, and add 1 to the
comment current value of hours
addhour    continue
    set mins=mins-60
    set hours=hours+1
    route(mins > 59) go addhour
okmins    continue

comment If the minutes field is less than 10, insert a leading zero
    set mindisp=''
    route(mins>9) go showit
    set mindisp='0'

showit    display 'This survey will only take '+length+' minutes
of your time, so we should be finished at about '
+hours+':'+mindisp+'.'+mins+'.'
    pause
comment *** Start CAPI interviews here ***
brand    ask 'Which brand did you try?'
    resp sp 'Suds' / 'Washo' / 'Gleam' / 'Sparkle'
```

```
dumlike  dummyask 'What liked about brand tried'
          resp coded
like      ask 'What did you like about '+brand+'?'
          resp coded nodata
          set brandno = nbit(brand)
          set brdtry = elm(brand/brandno)
          set punct = ': '
          callfunc('append',brdtry,punct)

comment Join reasons liked to brand tried and assign to dumlike for
comment storing in .tex file for coding
          callfunc('append',brdtry,like)
          set dumlike = brdtry
full     display 'Brand chosen and reasons liked are: '+dumlike
          pause

          set newlab = ''
comment CATI syntax is:
          callfunc('rfopen',3)
          set fmt = '%s{\#0a'
          callfunc('rfprint',3,fmt,newlab)
comment CAPI syntax is:
comment  rfopen 3
comment  substitute fmt '%s\n'
comment  rfprint 3,fmt,newlab

name      ask 'To show our appreciation of your help in this study,
we''d like to send you some vouchers for our products. I''ll need
your name and address for this.@@Could I take your name first, please?'
          resp coded nodata
comment CAPI syntax for next statement is: rfprint 3,fmt,name
          callfunc('rfprint',3,fmt,name)

address   ask 'Could I have your full postal address please?@'
INTERVIEWER: TYPE ADDRESS ALL ON ONE LINE WITH ; AT THE END OF
EACH ADDRESS FIELD'
          resp coded nodata
          set pos = 1
          set fstart = 1
          set numchar = 0
11       for char = 1 to 100
          set numchar = numchar + 1

comment Read a single character from the address
          callfunc('substr',address,pos,1,copied)
          if (copied <> ';' ) {
              set pos = pos + 1
              goto nxtchar
          }
```

```
comment Read all characters belonging to this part of the address
      set numchar = numchar - 1
      callfunc('substr',address,fstart,numchar,copied)
comment CAPI syntax for next statement is: rfprint 3,fmt,copied
      callfunc('rfprint',3,fmt,copied)
      set pos = pos + 1
      set fstart = pos
      set numchar = 0
nxtchar   next

comment CAPI syntax for next statement is: rfclose 3
      callfunc('rfclose',3)
end
```

Script B

```
comment Sample script B for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 26, Text manipulation

brand    ask 'Which brand did you try?'
         resp sp 'Suds' / 'Washo' / 'Gleam' / 'Sparkle'
dumlike  dummyask 'What liked about brand tried'
         resp coded
like     ask 'What did you like about '+brand+'?'
         resp coded nodata
         set brandno = nbit(brand)
         set brdtry = elm(brand/brandno)
         set punct = ': '
         callfunc('append',brdtry,punct)

comment Join reasons liked to brand tried and assign to dumlike for
comment storing in .tex file for coding
         callfunc('append',brdtry,like)
         set dumlike = brdtry
full     display 'Brand chosen and reasons liked are: '+dumlike
         pause

comment CATI/CAPI/Web: Write name and address details onto separate
comment cards in the data file. Ignored by mrInterview.
         newcard
name     ask 'To show our appreciation of your help in this study,
         we''d like to send you some vouchers for our products. I''ll need
         your name and address for this.@@Could I take your name first, please?'
         resp coded(0) nodata
fname    fix (72) name
address  ask 'Could I have your full postal address please?@'
INTERVIEWER: TYPE ADDRESS ALL ON ONE LINE WITH ; AT THE END OF
EACH ADDRESS FIELD'
         resp coded(0) nodata
```

```
        set pos = 1
        set fstart = 1
        set numchar = 0
        set linenum = 0

comment Dummy question to hold individual address lines.
comment Assume a maximum of 7 lines. Data is not saved
comment and no columns are reserved because the lines
comment are written out later with 'fix'.
11      for x = 1 to 7
adline    dummyask 'Address line'
            resp coded(0) nodata
next

comment Now extract address lines into separate variables
12      for char = 1 to 50
            set numchar = numchar + 1

comment Read a single character from the address
            callfunc('substr',address,pos,1,copied)
            if (copied <> ';' ) {
                set pos = pos + 1
                goto nxtchar
            }
            set linenum = linenum + 1

comment Read all characters belonging to this part of the address
            set numchar = numchar - 1
            callfunc('substr',address,fstart,numchar,copied)
            set adline(linenum) = copied
            set pos = pos + 1
            set fstart = pos
            set numchar = 0
nxtchar   next

comment Assume a maximum of 7 address lines
13      for oput = 1 to 7
            newcard
faddr     fix (72) adline(oput)
next
end
```

27 Callfuncs

Callfuncs, or callable functions, are external C programs which you can call from inside a script. They are named on a **callfunc** statement which also defines constant or variable data that will be used by the function.

You have already come across a number of these callfuncs. For example, you have seen the *setcols* callfunc in chapter 8, which determines the number of columns used for displaying response lists on the screen, and the *convdata* callfunc in chapter 5, for applying scaling factors.

There are many more callfuncs you can use with widely varying purposes, including date and time manipulation and multilingual scripts. The callfuncs covered in this chapter are:

abendivr	cancel an Interactive Voice Response (IVR) connection
aca	run an automated conjoint analysis
acctinfo	write information to a customized accounting field in the qca file
adddatepart	increment/decrement part of the date/time string by a given value
alphdate	convert a text date/time to operating system format
autostop	save response data periodically during an interview
commitivr	transfer current call to the IVR
datepart	copy part of the date/time string into a variable
datetext	convert the operating system date/time string to a text
datetime	get the current date/time in operating system notation
disable_audio_recording	disable audio recording in a QTS interview
enable_audio_recording	enable audio recording in a QTS interview
getenv	find the value of an environment variable
keepstopped	save original data for stopped interviews
makeappt	make an appointment without returning to SMS
prepareivr	dial IVR
qc_hangup	hang up a call
redial	reconnect a dropped call
setdataber	set the data serial number for this interview
setdatepart	set part of the date/time string to a given value
setivr	specify the initial IVR configuration
setlang	set the language in scripts for multilingual interviewing
setlocale	set the language for alphabetic sorting, punctuation and line breaks
showinf	name a section of the script
subprog	run an external program without leaving the interview
testmode	switch test mode on/off inside the script
timesect	record time spent in current section of script
varlist	write data out to a temporary file

27.1 Date and time manipulation callfuncs



Quancept provides a number of callfuncs for working with dates and times within the script. These enable you to:

- copy part of a date/time string into a script variable, for example, copy the day number into a variable for testing.
- assign a specific value to a section of a date/time string, for example, set the date to the 28th rather than the 27th.
- increment or decrement a section of a date/time string by a given value, for example, increment the number of hours by 2.

You are most likely to need these facilities if you are using *makeappt* to set appointment times within the script.

-
- ❖ Quancept presently obtains the current date and time from the operating system, which stores them as a number of seconds since 00:00 on 1 January 1970. Whenever you want to work with dates, you must always start by copying the system-style date into a variable. Whether you subsequently convert it into a more user-readable form depends on whether or not you want to display it on the screen.
-

Date and time in operating system format



Quick Reference

To copy an operating system date into a variable, type:

`callfunc('datetime', variable)`

There is no need to define the variable as numeric before you use it, but if you use an existing variable, make sure it is an integer variable.

The operating system stores time in seconds. Because we tend to work in minutes, *datetime* sets the variable to an operating system-format time converted into minutes. If you wish, you can use the callfunc *datetext* to convert this value to a text representation of the date.

Converting an operating system date and time into a text



Quick Reference

To convert an operating system date and time into a text, type:

```
callfunc('datetext', opsys_var, text_var)
```

The following statements show you how to use the *datetime* and *datetext* callfuncs together:

```
callfunc('datetime', udate)
callfunc('datetext', udate, tdate)
d1   display 'The date is '+tdate
```

If the operating system date in *udeate* is 15540951 minutes, this translates into 08:55am on Tuesday, 20 July 1999.

Breaking the date and time into separate sections



Quick Reference

To extract part of a time or date from the operating system date/time, type:

```
callfunc('datepart', opsys_var, which_part, date_var)
```

where *opsys_var* holds a date in operating system format and *date_var* is the variable in which the extracted information will be placed. *which_part* is a numeric value or variable defining the part of the date required, and may be one of:

0	last two digits of the year	4	minutes
1	month number	5	day in week; Sunday is day 1
2	day in month	6	day number in the year; January 1 is day 0
3	hour		

Although *datetext* produces a user-readable date and time string, it does not provide control over the order in which the date and time components are displayed. If you want to display the date and time in a different order or print only certain items from the string, you will need to extract those items and place them in separate variables. Note that all information is extracted as numeric values.

The *which_part* parameter is either a number or a numeric variable between 0 and 6 that determines which part of the date should be extracted. The table below shows the value associated with each date part and the value returned if the date is Thu 27 Dec 10:42 1990 (that is, 11038242 minutes):

Item	Part of date	Value returned
0	Last two digits of the year	90
1	Month number	12
2	Day in the month	27
3	Hour	10
4	Minutes	42
5	Day number in week	5
6	Day number in year	361

-
- ☞ *datepart* extracts the year as a numeric value, not a string. This means that the years 2000 to 2009 will be extracted as a single digit: 2000 is therefore returned as 0 rather than 00, 2001 as 1 rather than 01, and so on.
-

If you plan to set appointments within the script, and you want appointments for weekdays only, you could check which day the call is being made and set the delay time for the appointment accordingly. We will deal with the various methods of setting appointment times within the script shortly; the script below just shows how to extract the day number and test for a weekend:

```
callfunc('datetime',udate)
callfunc('datepart',udate,5,daynum)
if (daynum=1 .or. daynum=7) {
    actions for weekends
}
```

Setting dates and times within the script



Quick Reference

To alter part of an operating system date/time string, type:

callfunc('setdatepart', opsys_var, which_part, new_value)

to overwrite the current value with a given value, or:

callfunc('adddatepart', opsys_var, which_part, inc_value)

to increment or decrement the current setting by a given value.

which_part is a numeric value or variable defining the part of the date to be altered. It may be:

0	last two digits of the year	4	minutes
1	month number	5	day in week; Sunday is day 1
2	day in month	6	day number in the year; January 1 is day 0
3	hour		

Using the previous example, let us set the suggested appointment time for a week's time. If we are calling on a Friday, we want to set appointments for the Monday (10 days' time) rather than for the weekend. All appointments must be made for 30 November 2001 at the latest. Here's our script:

```

callfunc('datetime',udate)
comment Extract day and set suggested callback day as appropriate.
comment If today is Friday, delay is 10; otherwise delay is 7.
    callfunc('datepart',udate,5,daynum)
    if (daynum = 6) {
        set delay = 10
    }
    else {
        set delay = 7
    }
    callfunc('adddatepart',udate,2,delay)
comment Test that appointment is not after 30 Nov2001
    callfunc('datepart',udate,1,monthval)
    callfunc('datepart',udate,0,yearval)
    set oct99 = logical (monthval = 10 .and. yearval = 01)
    set nov99 = logical (monthval = 11 .and. yearval = 01)
    route (oct01 .or. nov01) go showap
comment Appointment later than 30 Nov 2001, so reset to 30 Nov 2001
    callfunc('setdatepart',udate,0,01)
    callfunc('setdatepart',udate,1,11)
    callfunc('setdatepart',udate,2,30)

```

```
comment Convert suggested appointment time into a text
showap    callfunc('datetext',udate,tdate)
d1        display 'The suggested appointment time is '+tdate
pause
```

You will notice that we are reading the date into a variable called udate and then changing it later on. This has no effect on the actual date on the computer; it just changes the value of the date as it is stored in the script. Details on how to save the new time and date as an appointment are given below.

Converting a text date and time into operating system format



Quick Reference

To convert a date and time into operating system format, type:

callfunc('alphdate', *text_var*, *opsys_var*)

If the date contains a year specification, the year can be entered as two or four digits, but must come at the beginning of the date. Dates with the year at the end are not acceptable. For example, acceptable formats for 19 March 2001 are:

- 01/3/19 or 2001/3/19 if you are using American date format, or
- 01/19/3 or 2001/19/3 if you are using European date format

-
- ☞ Dates entered in the form aa/bb are assumed to be in American format, namely mm/dd. If interviewers wish to enter dates in the European format (dd/mm), they should set the environment variable QCEUROPEAN to 1. If interviewers have the environment variable QCLOCALE set to a value between 30 and 50 inclusive, QCEUROPEAN is set to 1 automatically.
 - ☞ For further information on environment variables for interviewers, see section 39.2, Environment variables.
-

If the text date is not in a recognized format, the operating system date will be set to 0. Acceptable date formats are shown in the examples below:

14:00
6pm
9/30-6pm
01/9/30-11:31

Making appointments from within the script



Quick Reference

To have the script prompt the interviewer to make an appointment, type:

```
callfunc('makeappt', opsys_date)
```

where *opsys_date* is the variable in which the date of the appointment will be held.

There are three ways of setting appointment times during an interview:

- by returning to the SMS menu with a *gotosms* statement
- automatically using *setdatepart* and *adddatepart*
- by prompting the interviewer with *makeappt*

Each method has its advantages, so you will choose which one to use according to what is to happen once the appointment has been set. With *gotosms*, the interview is terminated as soon as this statement is read, and the interviewer is returned to the SMS menu to select a suitable option.

Using *setdatepart* and *adddatepart* allows you some control over when appointments are made and can be useful for product tests where you know that you will need to call back after a set time. You can, if you wish, write statements which allow the interviewer to check the appointment time and request another one if the first is unsuitable.

Using *makeappt* allows the interviewer to arrange a convenient appointment time with the respondent and to enter it exactly as he/she would at the SMS menu. For example:

```
callfunc('makeappt', appt_time)
```

saves the appointment time in the variable *appt_time*. Once the appointment is set, qtip continues with the next statement in the script.

Unless the project uses a calendar file, interviewers may set appointments up to a year in advance of the current time and date. If the project directory contains a calendar to control appointment setting, the rules in this file will apply to appointments made in this way.

-
- ☞ See section 9.9, Returning to the SMS menu, for information on *gotosms*.
 - See 'Arrange appointment with respondent' in chapter 33, Testing scripts with qtip, to find out about setting appointment times in the SMS menu.
 - See the *Quancept Sample Management and Telephony Systems User's Guide* for information on the calendar.
-

Writing appointment times into the SMS sample record

Each sample record contains a variable called apptime which SMS uses for storing appointment times that it arranges itself (for example, busy or no-answer callbacks) or that the interviewer sets via the SMS menu. You can write appointments made with *makeappt* into this variable using *putsmvar*, as shown in the example below:

```
callfunc('makeappt',uxdate)
set apvar = 'apptime'
callfunc('putsmvar',apvar,uxdate)
```

☞ See section 29.3, Updating a sample record from the script, for more about *putsmvar*.

27.2 Scripts for multilingual CATI interviews

Setting the language in which interviewing will take place



Quick Reference

To set the language for a multilingual script, type:

```
callfunc('setlang', langq)
```

where *langq* is a question whose integer response defines the language to be used.

When a project is aimed at a cross-section of nationalities, it may happen that the respondent does not speak the language that you would normally use for writing the script. If you have multilingual interviewers and you have translations of the question and response texts, you can help the interviewer by displaying the script in the respondent's own language.

When writing scripts for interviews that may be conducted in different languages, you write the script in one language. Translators then use the Qolyglot program to translate the questions, responses and display texts into as many languages as are required. These translations are held in the ptf file.

☞ Text on *set* statements is not translated.

☞ For more information on Qolyglot and the ptf file, see the *Quancept Translation Manual*.

Each language has a number. The default is language 0 (zero) — the base language in which the script is written (although it is possible to select a different base language when you parse the script). The script will normally contain a question asking which language to use for the interview.

If you already know which language the respondent speaks, perhaps from an SMS database or because you are restarting a stopped interview, you can make this question a dummy question whose value is assigned automatically by the script.

-
- ☞ The interviewer can also change the language using the special response *ln.
 - ☞ For details of this and other special interviewer commands, see section 33.30, Miscellaneous commands.
For information on setting the interview language in mrInterview scripts, see section 12.9, Setting the language in multilingual mrInterview scripts.
-

Setting the language for alphabetic sorting, punctuation and line breaks



Quick Reference

To specify the interview language for sorting and other language-specific areas in the script, type:

callfunc('setlocale',*langvar*)

where *langvar* is a temporary variable that defines the language code.

For example:

```
set lang = 'nl_BE'  
callfunc('setlocale',lang)
```

to set the language to Flemish as it is spoken in Belgium.

setlocale and its corresponding environment variable QCLANG control the following aspects of multilingual interviewing:

- Keyboard input — that is, which characters acts as punctuation.
- Screen output — for example, where to break long lines.
- Case conversion when comparing texts — for example, in list-in-list selections.
- Collation sequence for alphabetic sorting.
- Pattern matching when % is used with the *strngchk* callfunc to match any letter.

If neither *setlocale* nor QCLANG is not defined, qtip uses the standard Unix LANG variable. You can override both LANG and QCLANG with *setlocale*.

27.3 Setting the respondent serial number inside the script

Quick Reference

To set the respondent serial number inside the script, type:

```
callfunc('setdataser', sernum_var)
```

Quancept and mrInterview allocate a unique serial number to each new interview started. In Quancept CATI, Quancept CAPI and Quancept Web, a record of the latest number allocated is stored in the serial number file.

Occasionally, you may have manually recorded interviews which you wish to add to a computerized project. If these interviews do not have serial numbers attached to them, you can enter them as if you were conducting an interview, and Quancept or mrInterview will assign unique serial numbers in the normal way. If the interviews already have serial numbers and you want to keep those numbers, you'll need to modify your script slightly so that the interviewer is prompted to enter the serial number.

The way to do this is to write a question with a numeric response which prompts for the serial number, and then to set this as the serial number for this interview with *setdataser*. The given serial number will then be attached to the record in the dat file and also to any other related records in other files (open ends, for instance). For example:

```
dowhat    ask 'Are you entering a manually recorded interview?'
            resp sp 'Yes' / 'No' go ctn nodata
sernum    ask 'What is the interview serial number?'
            resp num 1 to 99999 nodata
snverif   ask 'Please confirm that the serial number for this
            interview is '+sernum
            resp sp 'Yes, correct' go sds / 'No, not correct' nodata
            goto sernum
sds       callfunc('setdataser', sernum)
comment   Start of interview proper
ctn      continue
```

There are two things to notice about this example. First, all the questions associated with the serial number are flagged with *nodata* because we do not want any columns allocated to these questions. If we had wanted to save data for these questions, and we would have added the questions after interviewing had started, we would have needed to use the *force* keyword to force the columns for those questions to appear at the end of the record.

☞ For information about *force*, refer to section 17.1, Adding extra questions to the script.

Second, we've included a question to verify that the serial number entered is correct. This is important because typing mistakes are easy to make and can have far-reaching consequences on the data, especially if the mistake results in a duplicate serial number in the data file.

-
- ☞ It is your responsibility to ensure that duplicate serial numbers do not occur if you use this callfunc.
-

27.4 Finding the value of an environment variable



Quick Reference

To copy the value of an environment into a text variable, type:

callfunc('getenv', varname, text_var)

where *varname* is a text variable containing the name of the environment variable whose value is required, and *text_var* is a variable into which the value is returned.

The **getenv** callfunc allows you to pull the value of an environment variable into the script. If the environment variable does not exist (that is, it is undefined), the callfunc returns a null value.

-
- ☞ All values are returned as texts, so if you want to test the value of a variable, remember to enclose it in single quotes so that Quancept reads it as a text rather than a number.
-

Here is an example of how you could use *getenv* to copy the interviewer's name or code into the data. We will assume that each interviewer has his/her name or code defined in the environment variable QCWHOAMI:

```
set lname = 'QCWHOAMI'
callfunc('getenv', lname, who)
whofix fix (5) who
```

-
- ☞ Another example of using *getenv* is shown in 'Setting the scaling factor' in chapter 5, Keywords in response lists, where we use the value of the interviewer's international dialing code as a means of identifying the respondent's national currency (the example assumes that the interviewer calls respondents only in his/her own country).
-

27.5 Automatic test mode

In Quancept CATI and Quancept CAPI



Quick Reference



To switch out of automatic test mode, type:

callfunc('testmode', 0)

at the point at which you wish to take control of the interview.

To return to automatic test mode, type:

callfunc('testmode', 1)

Quancept CATI and Quancept CAPI provide an option for automatic testing of scripts. After initial testing, you may find that while most of the script works, there is one part that does not, and you will want to run that part manually while leaving the rest of the script to run automatically. Here is an example:

```
gender    ask 'Is the respondent ...'
          resp sp 'male' / 'female'
age       ask 'How old are you?'
          resp num 18 to 99
comment   Test mode off
          callfunc('testmode',0)
employ    ask 'What is your work status?'
          resp sp 'full-time' / 'part-time' / 'self employed' /
                  'student' / 'trainee' / 'pensioner' /
                  'homemaker' / 'unemployed'
          display gender+' '+age+' '+employ
          set fage = logical(gender='female' .and. age<60)
          set mage = logical(gender='male' .and. age<65)
          if (employ='pensioner' .and. (fage .or. mage)) {
whypen     ask 'You said you are a pensioner, yet you have
            not yet reached the age for a State pension. Do you receive
            some other type of pension?'
            resp sp 'yes' / 'no' go ctn
whatpen    ask 'What type of pension is that?'
            resp mp 'disability pension' / 'company pension' other
ctn       continue
}
comment   Test mode on
          callfunc('testmode',1)
region    ask 'In which region do you live?'
          resp sp 'north' / 'south' / 'east' / 'west'
```

In the example above, we have switched test mode off so that we can verify that the test for pensioners is working correctly. The employment question is displayed, and the interviewing program waits for us to enter an answer on the keyboard. The interview continues interactively until it reaches the region question when the interviewing program regains control of the interview.

- ☎ In Quancept CATI, *testmode* is ignored if you are not running the interviewing program in test mode; that is, you are not using *qtip -t*.
- 💻 In Quancept CAPI, *callfunc(testmode,1)* always switches into automatic test mode even if you are not running the interviewing program in that mode. This means that you must remove all calls to *testmode* from the script and recompile it before using it for live interviews.

In mrInterview



Quick Reference

To run an automatic test interview, place the statement:

`callfunc('testmode',1)`

at the start of the questionnaire and then compile and activate the questionnaire in the usual way.

Automatic script testing is a method of testing in which mrInterview selects the responses to each question at random rather than displaying each question on the screen and waiting for you to choose an answer. The responses selected are written to the case data database in the same way as standard responses.

To run a test interview, start the interview in your browser. There will be a pause while the interview runs, and then the closing page of the interview is displayed. To run more test interviews simply repeat the command in your browser.

If a questionnaire contains this statement, automatic testing will take place regardless of whether the project is in a test or active state. If you want the statement to apply only when the project is in test mode, use *testmode* with *smstest* as follows:

```
if (ststest) {  
    callfunc('testmode',1)  
}
```

☞ See section 9.10, Actions for test interviews only (CATI), for further details.

27.6 Running an external program from within the script



Quick Reference

To run an external program, type:

callfunc('subprog', *command_var*)

where *command_var* is a text variable containing the command to be executed.

The maximum length of the command to be executed is 32 characters. Commands that are longer than this are truncated to this length.

The interview pauses while the program runs. When the external program finishes you are returned to the interview at the place you left it.

 If you are using Quancept CAPI, be aware of the following issues:

- If you call a Windows program its window will open on top of the interview window. However, under Windows 3.11 the program window then jumps to the back and must be brought to the top manually.
- You must always include the file extension of the called program and any other files in the *subprog* statement.

27.7 Writing variables to a temporary file



Quick Reference

To write responses to the file qc_<i>id</i>.txt, where *id* is the last two characters of the interviewer's line number, type:

callfunc('varlist', *var1*, *var2*, ...)

varlist is designed to be used with the *subprog* *callfunc* (described in the previous section). It allows you to write selected information out to a file and to process that file using an external program without leaving the interview. The situations when you can use this facility will vary from company to company, but one example is as follows. Suppose you are running screening interviews to find respondents willing to take part in the trial of a new product. You want to send a covering letter with each sample, explaining exactly what the tester is required to do. If you have a template letter and a mailmerge program or script, you could write the respondent's name and address out to a file and then merge them into the template letter for printing.

varlist writes the respondent ID and the values of named variables to a file, and *subprog* processes that file using a program of your choice.

Each interviewer has his/her own file, which qtip calls qc_*id*.out.txt. The *id* part of the filename is the identifier of the line on which the interviewer is working. If Ben is working on line ttyp3, for example, his output file will be qc_p3.out.txt. The file is overwritten for each new interview that the interviewer conducts, so you cannot use this facility for building up files of information about many respondents.

Let us look at how to implement the mailmerge example. Your template letter file is as follows:

Thank you for agreeing to take part in the trial of the Superchoc Plain and Milk Slice.

We're enclosing three bars of this chocolate for you to try. Please make a note of your opinions, trying to be as specific as possible. For example, what did you like or dislike about the flavor and why; have you any opinions on the packaging; and so on.

Yours sincerely,

Benjamin McDonald

The mailmerge script that converts this into a personalized letter is as follows:

```
#  
set line = `tty | sed 's/.*\(\..\\).*/\\1/'`  
set file = qc_${line}out.txt  
echo $file  
ex - $file <<'EOF'  
1d  
1,$s/|/\\  
/g  
1,2j  
$-1j  
s/ .* / /  
s/^/Dear /  
i  
  
.br  
$r choclet  
w ./letter$$  
q  
'EOF'
```

Quancept and mrInterview Scriptwriter's Manual

The script segment that gathers the information and writes it out is:

```
title      ask 'What is your title?'
            resp coded
name       ask 'Please let me have your first initial or your first
            name and then your last name.'
            resp coded
address    ask 'And your address. INTERVIEWER: Do not enter
            postcode here'
            resp coded
PCODE      ask 'And your postcode'
            resp coded
            callfunc('varlist',title,name,address,PCODE,title,name)
            set dothis = 'csh do_let'
            callfunc('subprog',dothis)
```

If Ben's qc_p3out.txt file contains the lines:

```
00123
Ms
Veronica Burr
999 The Warren|Woodview|London
SE2 1ZZ
Ms
Veronica Burr
```

the final letter for printing will be:

```
Ms Veronica Burr
999 The Warren
Woodview
London
SE2 1ZZ
```

Dear Ms Burr

Thank you for agreeing to take part in the trial of the
Superchoc Plain and Milk Slice.

We're enclosing three bars of this chocolate for you to try.
Please make a note of your opinions, trying to be as specific as
possible. For example, what did you like or dislike about the
flavor and why; have you any opinions on the packaging; and so on.

Yours sincerely,

Benjamin McDonald

-
- ❖ If you write scripts such as this, you must ensure that interviewers know exactly how to enter texts which are used in the script. In this example, the script searches for | symbols. If interviewers do not enter the different parts of the address on separate lines, there will be no | symbols in the file, and the script will silently fail at this point.
-

27.8 Storing accounting information



The accounting information produced for an interview tells you, among other things, how the interview ended (for example, completed interview, abandoned without data), how long it took, how many questions were asked, and the name of the last question answered. In some projects, you may wish to obtain more specific information about completed or partially completed interviews. For example, you may wish to set flags according to whether or not the respondent reached a certain part of the script, or to record which route the respondent took through a script with complex routing, or to record how long was spent in each section of the interview.

This section covers the following keywords:

timesect	record time spent in current section of script (for the qca and accounts.sms file)
acctinfo	write customized accounting information to a field in the qca file

Timing sections of the interview



Quick Reference

To time a section of an interview, start the section with:

callfunc('timesect', sectnum[, sectid])

sectnum is a number between 2 and 31 defining the cell in *sect_times* in which the timing will be placed, and *sectid* is an optional numeric identifier that goes in the corresponding cell of *sect_ids* for use with SMS.

Besides setting flags for each section completed, you may also wish to store the amount of time spent in each section. You can use **timesect** to calculate timings and store them in the qca file, in addition to the SMS server log and accounts.sms files. *timesect* records the amount of time spent in different sections of the script and is particularly useful for timing omnibus studies where different sections of the questionnaire may be asked each month. Timings include time accumulated during snapbacks and are not affected by delays due to stopping and restarting the interview.

A record is written to the qca file each time an interview is stopped, so the file may contain one or more records per interview, depending on how many times an interview was stopped for later restart. qtip uses the first 17 fields in each record to store its own timing and accounting information, such as the interviewer name and number, the beginning and end time of the interview session and its length.

You can use *timesect* to write additional timing information to a 32-cell array called sect_times. qtip appends the contents of this array to the end of each record in the qca file (as well as the serverlog and accounts.sms files). We'll refer to each of the cells within this array as sect_times[0], sect_times[1], and so on. By writing the information to the qca file, it becomes accessible from supermen's Interviewer statistics menu option, so you can create reports to analyze this timing information.

The cells sect_times[2] to sect_times[31] are available for you to write your own script-generated timings to using *timesect*. The first two cells, sect_times[0] and sect_times[1], are reserved for use by qtip or SMS. The information that appears in these cells differs depending on whether or not you are using SMS.

If you are using SMS, sect_times[0] contains the length of time spent in the telephone contact section, and sect_times[1] contains the length of time between qtip starting to execute the qoc file and returning the record to SMS.

If you are not using SMS, the value of sect_times[0] is always 0 except for the first interview in each qtip session. For the first interview, the cell contains the number of seconds that the question 'Do you want to restart a stopped interview?' was displayed. sect_times[1] contains a timing for the period between qtip starting to execute the qoc file and reaching the first *timesect* statement or, if there are no *timesect* statements in the script, reaching the end of the script.

The sect_ids array runs parallel to the sect_times array and stores section numbers. For example, if cell 5 of sect_times holds timings for section 8 of the script, cell 5 of the sect_ids array also stores the number 8. You are not required to give numbers to sections in your script, so qtip allows cells in sect_ids to be blank, even when the corresponding cell in sect_times contains a timing. The cells in sect_ids are available to the SMS algorithms via an SMS variable.

Timing of a section begins when a *timesect* statement is encountered on the interview path and ends either when another *timesect* statement is read or at the end of the script.

Here is an example with two *timesect* statements:

```
gender ask 'INTERVIEWER: Is the respondent ...'  
       resp sp 'Male' / 'Female'  
age   ask 'Which age group do you belong to?'  
      resp sp '18 and under' / '19 to 25' / '26 to 35' /  
           '36 to 45' / '46 to 55' / 'over 55'  
  
comment Start timing the next section and write timing information  
comment to the third timing field in the qca record, sect_times[2]  
      callfunc('timesect',2)  
sweets ask 'Do you eat sweets?'  
       resp sp 'Yes' / 'No' go bye
```

```

prefer ask 'Which sort of sweets do you prefer?'
      resp sp 'Boiled' / 'Hard mints' / 'Soft mints' /
             'Toffees' / 'Chocolate' / 'Soft centers' other

comment Start timing the next section and write timing information
comment to the fourth timing field in the qca record, sect_time[3]
callfunc('timesect',3)
tried ask 'Have you tried the new Plain and Milk Slice chocolate bars?'
      resp sp 'Yes' / 'No' go bye
rate  ask 'Could you tell me what you thought of the bar?'
      resp coded
bye   end

```

In this example, the interviewing program automatically generates timing information for the time when it starts to execute the qoc file to the point when the first *timesect* statement is read. So, timing information for the first two questions will be written to *sect_times*[1]. The first *timesect* statement generates a timing from the moment the interviewing program executes the statement to the next *timesect* statement — so the time taken to answer the questions ‘sweets’ and ‘prefer’ will be stored in the third cell, *sect_times*[2]. The second *timesect* statement generates timings from the moment the statement is executed to the end of the script, because there are no more *timesect* statements. The time taken to answer the questions ‘tried’ and ‘rate’ is therefore written to *sect_times*[3].

This is the record generated by qtip in the qca file for one interview:

```
tsecteg:sgray:625:873891128:873891184:'Wed_Sep_10_12:33:04_1997':56:3:2:0:1:0:0
:0:1:BYE:NONE:0:13:9:6:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:
:::::::
```

Each field in the qca file record is delimited by a colon. Note that field 17 contains the word NONE, indicating that this record was generated for an interview that did not use SMS. The next 32 fields are the timing fields — that is, the information from *sect_times*[0] through *sect_times*[31].

We've highlighted the first timing field to help you identify it. In this example, the field contains 0, indicating that the question about restarting a stopped interview was not displayed (that is, this was not the first interview conducted in this session). The next three fields show the timing information from *sect_times*[1] to *sect_times*[3].

Writing customized information to the qca file



Quick Reference

To write customized accounting information to a field in the qca file, type:

callfunc('acctinfo',*field_number*,*info_var*)

where *field_number* is a number in the range 1 to 10 denoting a field, and *info_var* is the name of a variable containing the information to be written. For each field, qtip allocates as much space as is needed for the information being written.

Records in the qca file end with ten fields in which you can store customized accounting information using **acctinfo**. You can choose which information to store in these fields. For example, you could:

- write flags or markers denoting which sections of the questionnaire the respondent answered
- store the respondent's telephone number
- write out answers to key questions, such as the brand currently used or the product preferred
- store the time of day the interview was conducted
- monitor the number of samples that need to be sent out for a brand survey

↖ If you want to use *acctinfo* to write multipunched data to the qca file, you must first convert the data into a numeric format because *acctinfo* is unable to write data from multipunched responses directly to the qca file.

Note that fields in the qca file are delimited by colons, so if the information you write to a field also includes a colon, any software you use to analyze the file's information must take this into account.

Here is a variation of the *timesect* script that adds information to the customized accounting fields:

```
gender ask 'INTERVIEWER: Is the respondent ...'  
       resp sp 'Male' / 'Female'  
age     ask 'Which age group do you belong to?'  
       resp sp '18 and under' / '19 to 25' / '26 to 35' /  
              '36 to 45' / '46 to 55' / 'over 55'  
  
comment Record gender and age in fields 1 and 2 of the acctinfo section  
callfunc('acctinfo',1,gender)  
callfunc('acctinfo',2,age)  
comment Start timing the next section and write timing information  
comment to the third timing field in the qca record, sect_times[2]  
callfunc('timesect',2)  
sweets ask 'Do you eat sweets?'  
       resp sp 'Yes' / 'No' go bye  
prefer ask 'Which sort of sweets do you prefer?'  
       resp sp 'Boiled' / Hard mints' / 'Soft mints' /  
              'Toffees' / 'Chocolate' / 'Soft centers' other  
  
comment Start timing the next section and write timing information  
comment to the fourth timing field in the qca record, sect_time[3]  
callfunc('timesect',3)  
tried ask 'Have you tried the new Plain and Milk Slice chocolate bars?'  
       resp sp 'Yes' / 'No' go bye  
rate  ask 'Could you tell me what you thought of the bar?'  
       resp coded  
bye   end
```

This is the record generated by qtip in the qca file for one interview. The *acctinfo* fields are shown in bold and tell us that the respondent was a woman aged 55 years or over.

27.9 Naming script sections for use by the supervisory program



Quick Reference

To break the script down into named sections for use by the supervisory program, type:

```
set variable = '['sect_name'  
callfunc('showinf', variable)
```

where *variable* is a temporary text variable containing the section's name, *sect_name*. An asterisk at the start of the section name causes it to be highlighted when displayed by the supervisory program.

The supervisory programs which report on interviewer productivity are able to show the name of the section each interviewer is in. Information will appear in these columns only if you break the script down into named sections with *showinf*.

Because the supervisory program allocates eight columns per section, you are advised to use section names of fewer than eight characters to stop the section names from running into the other columns of the report.

- ☞ For further information about the supervisory program, supermen, see the *Quancept Supervisor's Manual*.

A named section starts at a *showinf* statement and continues until the next one or until the end of the script.

In the example shown in section 17.9, you could insert a `showinf` statement immediately after the comment line at the start of each section; for example:

```
comment First section - set accounting flag to 1 when completed
        set section = 'first'
        callfunc('showinf',section)
q1a      (question goes here)
q1b      (question goes here)
        privsig 1
        route (...) go ex
```

```
comment Second section - set accounting flag to 2 when completed
    set section = 'second'
    callfunc('showinf',section)
    (statements go here)
    privsig 2
```

Here is another example, this time using *timesect* and *showinf*:

```
set sectnum = 1
callfunc('timesect',sectnum)
set sectname = 'demog'
callfunc('showinf',sectname)
gender ask 'INTERVIEWER: Is the respondent ...'
resp sp 'male' / 'female'
age ask 'Which age group do you belong to?'
resp sp '18 and under' / '19 to 25' / '26 to 35' /
      '36 to 45' / '46 to 55' / '56 to 65' / '66 and over'
set sectnum = 2
callfunc('timesect',sectnum)
comment The * before sweets causes the entry for the sweets section to
comment be highlighted in the supervisory menu.
set sectname = '*sweets'
callfunc('showinf',sectname)
sweets ask 'Do you eat sweets?'
resp sp 'yes' / 'no' go bye
prefer ask 'Which sort of sweets do you prefer?'
resp sp 'boiled fruit sweets' / 'soft fruit sweets' /
       'hard mints' / 'soft mints' / 'chewing gum' /
       'toffee' / 'chocolate' / 'chocolate snack bars' /
       'chocolate biscuits'
set sectnum = 3
callfunc('timesect',sectnum)
set sectname = 'prodtry'
callfunc('showinf',sectname)
tried ask 'Have you tried the new Plain and Milk Slice
chocolate bars?'
resp sp 'yes' / 'no'
bye end
```

This extract has three sections: demog, sweets, and prodtry numbered 1, 2, and 3, respectively. The timing to be used in the qca and accounts.sms files for the demog section starts at the *timesect* statement for that section and ends at the *timesect* statement for the sweets section. The timing to be used by the supervisory program for the demog section starts at the *showinf* statement for that section and ends at the *showinf* statement for the sweets section.

If you wish, you may write your script so that the supervisory program will highlight the section name and the time spent in that section. Highlighting takes two forms: the section name and time are always displayed in reverse video, and the terminal will issue an audible signal the first time a section name appears on the screen. If the screen is refreshed without the section name changing, there is no audible signal.

To use this feature, make sure that the section name starts with an asterisk. For example:

```
comment Highlight this section name and time
    set sectname = '*demog'
    callfunc('showinf',sectname)
    more statements
comment Do not highlight this section name and time
    set sectname='sweets'
    callfunc('showinf',sectname)
```

27.10 Saving response data during an interview



Quick Reference

To save response data periodically during an interview, type:

callfunc ('autostop', n)

where *n* is the interval at which the interviewing program writes to the file, in seconds.

The callfunc *autostop* enables you to save response data periodically during an interview. This helps safeguard against loss of data in the event of a system crash. qtip stores the autostop file in a project subdirectory called *project_auto*, using the respondent serial number as the filename. The file takes the same format as a stopped data file, so if a system crash occurs it can be moved to the stopped directory and the interview restarted when interviewing recommences. The file will contain data gathered up to the last periodic save.

By including an *autostop* statement at the top of the script, you ensure that data is saved throughout an interview at the specified interval. For example, if the following statement appears at the top of the script:

```
callfunc('autostop',60)
```

response data will be saved every 60 seconds. You can change this interval at any point in the script by including another *autostop* statement that specifies the new interval. While an interview is in progress, qtip writes to the autostop file if more questions have been answered since the file was last written.

-
- ❖ The file generated by *autostop* is deleted automatically by qtip when an interview's data has been successfully written to disk; that is, the interview has been stopped from the script or by the interviewer or it has been completed. The autostop file is also deleted if the interviewer types quit or abandon, or the interview is stopped by quota control.
-

When using *autostop*, remember that writing a file to disk is slower than keeping the information in memory. When deciding on the save interval for interview data, you need to consider the length of the script and the number of interviewers who will be working on it at any one time. For example, if your script is 30 minutes long, you have 30 working interviewers and you have set the *autostop* interval to 30 seconds, each interview will be written to disk at least 60 times before its completion. When all 30 interviewers have completed just one interview, there will have been 1800 automatic saves.

We recommend that when you use *autostop*, you change the save interval during the course of the interview so that data is written more frequently where data is essential and less frequently where it is not.

In the following example, a regular save interval of five minutes is set at the top of the script. This is sufficient for general, non-essential information. The save interval is lowered before the product enjoyment section to reduce the risk of data loss.

```
comment Set general save interval for every 5 minutes
      callfunc('autostop',300)
gender   ask 'INTERVIEWER: Is the respondent ...'
      resp sp 'male' / 'female'
age      ask 'Which age group are you in?'
      resp sp '18-24' / '25-34' / '35-44' / '45-54' / '55+'

comment We don't want to risk losing the product enjoyment information
comment so reduce the save interval to every 30 seconds
      callfunc('autostop',30)

brand    ask 'The product you tried was labeled with either A or B.
      Which product did you try? '
      resp sp 'Product A' / 'Product B'
like     ask 'Could you tell me what you liked about '+brand
      resp coded
dislike   ask 'And what was it that you disliked?'
      resp coded
comment We're less interested in the demographic info, so we'll
comment go back to the standard save time of 5 minutes
      callfunc('autostop',300)
demog    include demog
thank    display 'Thank you for taking time to answer our questions.
      Good-bye.'
      pause
end
```

27.11 Retaining original data for stopped interviews



Quick Reference

To save data for questions which are on the interviewing path in the stopped portion of the interview but which are removed from the interviewing path when the interview is restarted, type:

callfunc('keepstopped', 1)

at the earliest point at which you want all data to be saved.

qtip does not write data to the data files until the end of the interview. It holds the data for the interview in its memory, and any changes that you make overwrite the original data in memory. The data written to the data files represents the respondent's final answers to each question.

When you restart a stopped interview, it is possible that the respondent may want to change answers to questions held in the stopped part of the interview. qtip allows you to do this and overwrites the original responses in the usual way. If changing a response alters the path through the rest of the interview, you may lose responses which you originally had because those questions are no longer on the interviewing path.

If you wish to keep these old answers, even though they are no longer relevant, use the **keepstopped** callfunc. Once qtip reads this statement, it keeps all data gathered before the interview was stopped, even if changes when the interview is restarted would normally cause this data to be deleted. If the interview is stopped before qtip reaches *keepstopped*, then any changes made to the stopped data when the interview is restarted overwrite the original data in the normal way.

Here is an outline script that illustrates how *keepstopped* works:

```

qa      ask 'Which advertisement did you like best?'
        resp sp 'first' / 'second' go sec2
q1a    ask ....
        resp ...
q1b    ask ....
        resp ...
        goto ex
sec2   continue
        callfunc('keepstopped',1)
q2a    ask ....
        resp ...
q2b    ask ....
        resp ...
ex     stop 'ads'
```

If the respondent likes the second advertisement best, he/she goes to the statement sec2. qtip reads the *keepstopped* statement which tells it that all data must be saved even if it is changed after the interview is restarted. When the interview is restarted, the respondent changes his/her mind and says he/she prefers the first advertisement. qtip overwrites the code 2 for qa with a code 1 and then continues with question q1. When the data is written to the data files, it will contain the answers to questions q1a, q1b, q2a, and q2b.

If, instead, the respondent starts by preferring the first advertisement, the *keepstopped* statement is not executed, so the stopped portion of the interview does not contain a flag saying that the data must be retained. When the interview is restarted, the respondent says he/she prefers the second advertisement. This time, qtip deletes the answers to q1a and q1b and stores only the answers to q2a and q2b.

27.12 Automated conjoint analysis



Quick Reference

To call an ACA job from within the script, type:

callfunc('aca', acadir, retval)

where *acadir* is a text variable containing the location of the ACA job and all the associated ACA program files, and *retval* is an integer variable which qtip uses to signal whether or not the ACA job started successfully. If the variable has a value of 1, the job started successfully; if its value is 0, the job has failed.

Automated Conjoint Analysis (ACA) is a program which asks respondents their preferences among various combinations of a product's features to establish a numerical model of how each respondent values these features. Features for a meal, for example, might be hot, tasty, nutritious, nonfattening, and so on. This program is normally run on PCs with the CI-2 interviewing package.

The advantage of this facility, besides the obvious one of providing an interface to ACA jobs in general, is that it allows you to run multiple ACA jobs from a single script.

For best results, the ACA job should be located in a subdirectory of the project directory. If the variable that names the location is blank or is not a text variable, the interviewing program assumes that the ACA project is in the project directory itself. (We recommend that you keep the ACA files in the subdirectory as shown in the example).

-
- » It is possible for an ACA job to start successfully, but for the interviewer to abandon it prematurely. If this happens, the value of *retval* does not reflect that the ACA job terminated unsuccessfully.
-

The following is an example of a simple ACA script, followed by a directory listing showing how the files are stored.

```
qacal    ask 'First question'
        resp sp 'ok'
        set acadir='acal'
        set acalret=0
        callfunc('aca',acadir,acalret)
daca1   display 'First aca returned "'+acalret+'"'
        pause
qaca2   ask 'Second question'
        resp sp 'ok'
        set acadir='aca2'
        set aca2ret=0
        callfunc('aca',acadir,aca2ret)
daca2   display 'Second aca returned "'+aca2ret+'"'
```

A recursive directory listing of the project directory follows. As you can see, all ACA program files are in both ACA subdirectories.

Volume in drive C is MS-DOS_6

Directory of C:\EXAMPLE

[.]	[..]	[ACA1]	[ACA2]	QCSUR1
QCSUR1.ERR	QCSUR1.QC8	QCSUR1.QOC	QCSUR1.FR1	QCSUR1.PSM
QCSUR1.MAQ	QCSUR1.REF	QCSUR1.NAM	QCSUR1.MAP	QCSUR1.SUM
QCSUR1.FR2	QCSUR1.QDF	QCSUR1.SER	QCSUR1.DAT	[QCSUR1.DDR]
QCSUR1.DRX	QCSUR1.DRS			
22 file(s)		2,564 bytes		

Directory of C:\EXAMPLE\ACA1

[.]	[..]	ACA.BAT	ACANUMS.DAT	ACASIM.BAT
ACQ.EXE	AUTOEXEC.BAT	FACA.EXE	MERGC.EXE	MERGE.BAT
RUNINT.BAT	RUNSIM.BAT	SETINT.BAT	SETSIM.BAT	SETUPINT.EXE
SETUPSIM.EXE	SIM.EXE	SIM87.EXE	SIMSTART.EXE	START.EXE
USERID.ACA	ATTRIB.ACA	CONTROL.ACA	CUM.BAT	CUMX2.EXE
DIFFS.EXE	FRAMES.ACA	LABEL.EXE	POINTS.EXE	SETPNUM.EXE
SPECS.ACA	ACA.DAT	UTIL.DAT	REPORT.DAT	NUMSTART
35 file(s)		490,645 bytes		

```
Directory of C:\EXAMPLE\ACA2
[.]          [...]          ACA.BAT        ACANUMS.DAT      ACASIM.BAT
ACQ.EXE      AUTOEXEC.BAT   FACA.EXE       MERGC.EXE       MERGE.BAT
RUNINT.BAT    RUNSIM.BAT     SETINT.BAT     SETSIM.BAT      SETUPINT.EXE
SETUPSIM.EXE   SIM.EXE       SIM87.EXE      SIMSTART.EXE    START.EXE
USERID.ACA     ATTRIB.ACA    CONTROL.ACA    CUM.BAT        CUMX2.EXE
DIFFS.EXE      FRAMES.ACA    LABEL.EXE      POINTS.EXE     SETNUM.EXE
SPECs.ACA      ACA.DAT       UTIL.DAT      REPORT.DAT     NUMSTART

35 file(s)           489,530 bytes
```

```
Directory of C:\EXAMPLE\QCSUR1.DDR
[.]          [...]          [SUB0]
3 file(s)           0 bytes
```

```
Directory of C:\EXAMPLE\QCSUR1.DDR\SUB0
[.]          [...] 1          2
4 file(s)           315 bytes
```

27.13 Switching QTS recording on and off



Quick Reference

To switch recording off during an interview, type:

callfunc ('disable_audio_record')

To switch recording back on during an interview, type:

callfunc ('enable_audio_record')

The default is for audio recording to be enabled when you are using the QTS dialer.

If the project uses the QTS, you can add instructions to your script to disable or enable recording. This will cater for situations where the respondent agrees to be interviewed but does not want to be recorded:

```
qrecok ask 'Some of your responses will be recorded, instead of being
written. Like all your responses the recordings are not attributable. Is
this OK?'
      resp sp 'OK' go q1 / 'Refused to be recorded'
d1    display 'In that case I will write your answers down instead.'
      pause
      callfunc('disabled_audio_record')
```

27.14 Hanging up a QTS call



Quick Reference

To hang up the call from within the script, type:

callfunc ('qc_hangup')

Interviewers are not allowed to replace the handset — that is, hang up the telephone — during a QTS project. The qtip special command **hangup* is provided to cater for situations where the interviewer no longer needs to talk to the respondent, but where the interview is not yet finished.

You can add instructions to your script to perform a scripted hang-up so that the interviewer does not need to use **hangup*. This is particularly useful when the interviewer has finished talking to the respondent but needs to enter the verbatim responses that he or she gave. A *qc_hangup* statement has the same effect as **hangup*, but it allows the script rather than the interviewer to control when the call ends.

☞ See section 33.30, Miscellaneous commands, for information about **hangup*.

In the following example, the call is terminated before the end of the script because the statements at the end of the script are solely concerned with classifying the respondent and extracting the name and telephone number from the sample record and fixing it into the data.

```
comment ** >>> Questionnaire ends here <<< **
dispend display 'Thank you for participating.'
    pause
    callfunc('qc_hangup')
qclassb ask 'CODE FROM LIST'
    resp sp 'A' / 'B' / 'C1' / 'C2' / 'D' / 'E' ref
    set svarname = 'name'
    set sname = ''
    callfunc ('getsmvar',svarname,sname)
fixa   fix (25) sname
    set svarname = 'telno'
    set stelno = ''
    callfunc ('getsmvar',svarname,stelno)
fixb   fix (15) stelno
end
```

27.15 Redialing a dropped QTS call



Quick Reference

To dial a number from within the script, type:

callfunc ('redial')

If the project is using the QTS, the *redial* function lets you control the dialing of the telephone number from within the script. This is particularly useful if you use preview dialing in combination with company or respondent profiles that appear within the script. For example:

```
comment Respondent profile
profile display 'Respondent/company profile goes here'
    pause
check ask 'INTERVIEWER: Have you read the profile and are you ready
to start the interview?'
    resp sp 'yes - continue' / 'no - read profile again' go profile
    callfunc ('redial')
```

27.16 Interactive Voice Response callfuncs



Interactive Voice Response (IVR) is a system that allows telephone interviews to be conducted with little or no interviewer intervention. The IVR system speaks the questions to the respondent and waits for the respondent to enter the answers using the keypad on his/her phone. A very simple analogy is a telephone banking system, where you can manage your account by responding to automated prompts using the telephone handset.

Projects that use a combination of Quancept and IVR have the following advantages:

- You can use the QTS to dial the numbers, and then have the interviewer follow a Quancept script to interest the respondent in the survey. Having screened out unsuitable respondents, you can then use the IVR system to conduct the rest of the interview without interviewer intervention.
- If you have projects that are concerned with very personal subjects or with illegal activities such as drug taking, you may find that you obtain more interviews by using IVR because respondents feel happier talking to a machine rather than to another person.

A Quancept script that *hands off* (transfers) interviews to IVR must perform the following tasks:

- Define initial configuration settings such as the telephone number of the IVR system, DTMF strings and a no-answer timeout. This information is used by other IVR-related functions in the script. You use the **setivr** callable function or the QCIVR environment variable for this.

- Dial the IVR system and pass any relevant data from Quancept to IVR. Typically, you will pass at least a respondent identification code to link the Quancept and IVR data. You use the **prepareivr** callable function for this.
- Connect the respondent to the IVR system ready to start the automated portion of the interview. You use the **commitivr** callable function for this.
- Cancel the call to the IVR system, for example, if the respondent decides not to proceed with the interview after all. You use the **abendivr** callable function for this.

These callable functions are described in the following sections, and there is a full example script at the end of this chapter that illustrates how to use these functions in a script. As with all sample scripts, a runnable version of this script is available in the Quancept installation directory. The script snippets shown for the individual functions are taken from this script but may have been modified to highlight particular aspects of the function being described.

The sections describing the callable functions are followed by some notes explaining how Quancept interfaces with IVR and suggesting ways of setting up a simple IVR questionnaire to test the interface. These notes contain some references to IVR commands, keywords and screens where this is necessary to understand the Quancept side of the interface, but you should not assume that the information provided about IVR is complete in any way. Always refer to your IVR manuals for full details.

Defining the initial IVR configuration



Quick Reference

To specify the IVR system's initial configuration, type:

callfunc('setivr', *properties*, *result*)

where *properties* is a list of configuration settings in the form *keyw1=value1;keyw2=value2;...* and *result* is a variable that will contain an error message if the configuration is not correct.

Quancept needs to know how to communicate with the IVR system. In particular, it needs to know what telephone number to dial. You define this and other information using the following property keywords:

Property	Description
dtmf	<p>The initial DTMF string to send to the IVR system. This may contain one or more of the following:</p> <ul style="list-style-type: none"> • The 16 tone generating symbols 0–9, *, #, A–D • A comma for a one-second pause. • A dot (period) for a five-second pause. <p>The default is the # character.</p>
end_dtmf	The DTMF string to send immediately before hand-off.
grp	The extension group that will use IVR. The default is group 997.
	` This functionality has to be configured as part of your autodialer set-up. Contact your local Support Representative for details.
noansw	The no-answer time-out delay in seconds. The default is 10 seconds.
ph	The IVR telephone number. Enter only the digits that are to be dialed; do not include spaces or other formatting characters.
tgrp	The trunk group that will use IVR.
	` When defining properties, type a continuous string of non-blank characters; do not separate keywords or values with spaces. Also, type all keywords in lower case.

The property string may be up to 2048 characters long.

There are several ways of defining IVR properties. You can:

- assign the complete list of properties to a single variable; for example:

```
set ivrprops = 'ph=02071234567;noansw=5'
callfunc('setivr',ivrprops,result)
```

- use the *append* callfunc to build up a list of properties in a variable that you then use in the *setivr* callfunc statement; for example:

```
set proptxt = 'ph=02071234567'
set ivrna = 'noansw=5'
set punct = ';'
callfunc('append',proptxt,punct)
callfunc('append',proptxt,ivrna)
callfunc('setivr',proptxt,result)
```

- assign them to the QCIVR environment variable; for example:

```
setenv QCIVR noansw=5;ph=020712345678
```

If a property is not defined using *setivr*, Quancept looks for that property in the QCIVR environment variable. If the property is not defined in either place, the default for that property is used.

Although any or a combination of these methods is acceptable, it is advisable to define all properties using the QCIVR environment variable because they are usually fixed for a particular survey or can also be set using *prepareivr*.

If there are any errors in the configuration, the function returns an explanatory error message in the *result* variable. If there are no errors, the function returns a blank text in this variable. If you define configuration parameters using *setivr*, your script should check for errors before continuing:

```
comment Define error message for when things don't work
set errtxt = 'There is a technical problem. The interview cannot continue.'
set ivrpar = 'ph=00496172917650;noansw=5'
callfunc('setivr', ivrpar, result)
if (result <> '') {
    display errtxt
    goto exit
}
```

☞ *setivr* does not dial the IVR system; it only defines parameters to be used for dialing and communication purposes.

Dialing the IVR system



Quick Reference

To dial the IVR system, type:

```
callfunc('prepareivr', dtmf, result)
```

where *dtmf* is an initial string to send to the IVR system and *result* is a variable that will contain an error message if the configuration is not successful.

Once you have verified that the configuration properties are correct you can dial the IVR system using the *prepareivr* callfunc. This establishes a connection between Quancept and IVR and passes any Quancept data that you want to be available in the IVR interview. It does not connect the respondent to the IVR questionnaire (this happens when the *commitivr* callfunc is executed), so if there are errors the Quancept script can determine what to do next.

The DTMF string contains information that you want to pass to the IVR system at this point. It can contain any or all of the following:

- The serial number that Quancept has allocated to this respondent. We recommend that you always pass at least this information as it ensures that you will be able to combine the IVR data with the rest of the Quancept data for the interview later on.
- Questionnaire data from the Quancept portion of the interview that you want to be available in the IVR interview or in the IVR database. This might include some screening information such as age or gender. In theory, you could pass all the Quancept data to IVR so that the whole interview is stored in the IVR system. However, while this approach may be satisfactory for short surveys or surveys that are mainly done using IVR, the time required to transfer the data will usually make it unsuitable for longer surveys.
- Special symbols to force a pause in the data. This can be necessary if the set-up of the audio path is particularly slow, say, on intercontinental connections.

We'll look at examples of how to send these types of information shortly.

When a connection is made with the IVR system, *qtip* sends the DTMF value defined in the configuration properties (in either *setivr* or QCIVR) and, if the DTMF value defined with *prepareivr* is not blank, follows it with this value.

prepareivr does not return a value straight away because it has to wait for the IVR system to answer. However, once the system answers, the interview can continue without waiting for *prepareivr* to send the DTMF strings.

If a connection cannot be made or there are other errors, *prepareivr* places an explanatory message in the *result* variable. You should always check this variable in case the connection failed. If the connection succeeds, you will need to use either *commitivr* to transfer the respondent to the IVR system or *abendivr* to cancel the connection if the respondent does not want to be interviewed after all. If you forget to do this, Quancept will automatically run *abendivr* at the end of the interview.

-
- ❖ Even though Quancept terminates an unused IVR connection at the end of the interview, it is more efficient in terms of resources if you terminate connections as soon as they are no longer required.
-

Passing just a serial number to IVR

To pass the Quancept respondent serial number to IVR, place a reference to the *respondent* keyword in the DTMF string.

```
set dtmfstr = '[+respondent+]'  
callfunc('prepareivr',dtmfstr,result)
```

If the SMS record key is wholly numeric you can pass that value to IVR so that you have a link between the IVR data and the sample record. To do this, extract the SMS record key into variable and then name that variable in the DTMF string.

```
set varname = 'key'  
callfunc('getsmvar', varname, smskey)  
set dtmfstr = '[+smskey+]'  
callfunc('prepareivr', dtmfstr, result)
```

Passing response data to IVR

You can pass responses to Quancept questions to IVR. You might do this if you want to save the answers to some of the Quancept screening questions in the IVR data or if you want the IVR questionnaire to follow different paths dependent on respondents' characteristics.

Because the respondent answers the IVR questions using the keys on the telephone handset, most questions in an IVR questionnaire have to be answered using the digits or codes 0 to 9. This restriction affects which responses you can pass from Quancept, because you have to be able to set up a suitable question in the IVR questionnaire to receive the Quancept data. It means that you can pass numeric (integer) and single-punched responses but not real or open-ended data. To pass multipunched data you would have to convert it to single-punched format first.

The most important thing about passing data from Quancept to IVR is the number of digits in a response or response code. IVR is not particularly concerned with whether the digits are numeric values, punch codes or response positions in a list. Instead, it is concerned with whether a question can be answered by a single digit, by a fixed number of digits, or by a variable number of digits.

Examples of Quancept response lists that result in a single digit being passed to IVR are as follows:

- *resp num 1 to 9*
- *resp sp* with up to nine quoted responses

A Quancept response list that results in a fixed number of digits being passed to IVR is:

- *resp num 18 to 99*

Examples of Quancept response lists that result in a variable number of digits being passed to IVR are as follows:

- *resp num 1 to 99*
- *resp sp* with more than nine quoted responses

Passing single-digit responses

To pass a single-digit numeric response such as the number of children in a household, you place a reference to the question name in the DTMF string. For example:

```
child ask 'How many children do you have?'
resp num 0 to 9
set dtmfstr='[+child+]'
callfunc('prepareivr',dtmfstr,result)
```

The IVR questionnaire must contain a blank Get Response command to receive the data, or a blank Get Value command with the minimum and maximum number of digits set to 1.

To pass a single-punched response, you must pass the response number (that is, its position in the list), so you'll type:

```
gender ask 'INTERVIEWER: Is the respondent ...?'
resp sp 'Male' / 'Female'
set mf = nbit(gender)
set dtmfstr='[+mf+]'
callfunc('prepareivr',dtmfstr,result)
```

To pass lists of single-digit responses, place the variable names one after the other in the DTMF string. For example:

```
child ask 'How many children do you have?'
resp num 0 to 9
gender ask 'INTERVIEWER: Is the respondent ...?'
resp sp 'Male' / 'Female'
set mf = nbit(gender)
set dtmfstr='[+child+][+mf+]'
callfunc('prepareivr',dtmfstr,result)
```

Passing responses with a fixed number of digits

Responses with more than one digit must be passed to Get Value commands in the IVR questionnaire. When all the responses to a numeric question have the same number of digits, you can name them in the DTMF string in the same way as single-digit responses. For example, if Quancept accepts values in the range 18 to 99 for the respondent's age you know that every age will have two digits so you can type:

```
age ask 'How old are you?'
resp num 18 to 99
set dtmfstr='[+age+]'
callfunc('prepareivr',dtmfstr,result)
```

You can pass lists of responses in the same way as single-digit responses:

```
age ask 'How old are you?'
resp num 18 to 99
child ask 'How many children do you have?'
resp num 0 to 9
set dtmfstr='[+age+][+child+]'
callfunc('prepareivr',dtmfstr,result)
```

Here, the first two characters of the DTMF string will be assigned to the first blank question in the IVR questionnaire and the third character will be assigned to the second blank question.

Passing responses with variables numbers of digits

If the number of digits in a response may vary, you must send a # symbol after each response to mark the end of that response. The exception is if the response is at the end of the DTMF string. In this case only, omit the terminating # symbol. (If you place a # at the end of the DTMF string, IVR will continue with the first non-blank question in the IVR questionnaire even though the respondent is not yet connected to the IVR system.) For example, suppose you want to pass the respondent serial number, and the respondent's gender and age to IVR. You would write:

```
age      ask 'How old are you?'
        resp num 18+
gender ask 'INTERVIEWER: Is the respondent ...'
        resp sp 'Male' / 'Female'
        set mf = nbit(gender)
        set dtmfstr = '[+respondent+]#[+mf+][+age+]'
        callfunc('prepareivr',dtmfstr,result)
```

The respondent serial number can be anywhere between one and five digits long so you must send a # symbol to mark the end of the serial number. gender is a single digit going into a Get Response command so it does not need a terminating symbol. age is a numeric value of two or more digits, but because it comes at the end of the string it does not need a terminator.

The same rules apply to single-punched response lists with more than nine responses. The *nbit* values will be between 1 and 9 for the first nine response and will then consist of two or more digits for the other responses. You will therefore have to include a # marker in the DTMF string so that IVR knows where the response code for this question ends. For example:

```
comment Apple and Blackberry, Forest Fruits, Black Cherry and Other
comment have two-digit response numbers .....
flavor ask 'Which flavor do you like best?'
        resp sp 'Peach' / 'Pineapple' / 'Apricot' / 'Raspberry' /
                'Mango' / 'Vanilla' / 'Banana' / 'Blackcurrant' /
                'Rhubarb' / 'Apple and Blackberry' / 'Forest Fruits' /
                'Black Cherry' / 'Other'
        set pref = nbit(flavor)
brand ask 'And which brand do you usually buy?'
        resp sp 'Brand A' / 'Brand B' / 'Brand C' / 'Brand D'
        set brd = nbit(brand)
comment .... so the flavor must be terminated by # in the DTMF string
        set dtmfstr='[+pref+]#[+brd+]'
        callfunc('prepareivr',dtmfstr,result)
```

☞ See 'Creating an IVR script to receive Quancept data' later in this chapter for further information about the Get Value and Get Response commands.

Null, dk, ref and specified other

Be careful if a response list contains any of the special responses *null*, *dk*, *ref* and *other*. *Null*, *dk* and *ref* selected for numeric questions insert non-digits in the DTMF string, causing the IVR questionnaire to fail. It is therefore best to avoid these keywords in numeric responses lists if you can. If you cannot — for example, you want to allow respondents not to give their age — you should write some additional Quancept code to process the response before it is added to the DTMF string. For example:

```
age ask 'How old are you?'
      resp num 18 to 99 ref
      if (age = ref) {
          set agestr = '00'
      }
      else {
          set agestr = '[+age+]'
      }
      set dtmfstr = '[+agestr+']'
```

Later, when collecting the SQL data from IVR, you would need to convert 00 back to *ref*. You could expand this example to cater for other special responses simply by allocating a different code to each one.

In a single-punched response list, *nbit* returns a value of zero for *null*, *dk*, *ref* and *other*. If you want to differentiate between these responses in the IVR database, it may be better to write them as quoted responses.

-
- 『 All unanswered questions and questions whose responses have been unset have a *null* value, even if *null* is not defined as a valid response for the question. You must test your scripts thoroughly to ensure that you do not pass these values to IVR.
-

Passing pause symbols

When testing your Quancept script, you may find that you need to wait a few seconds after dialing the IVR system before passing information to it. You can achieve whatever delay you need by starting the DTMF string for *prepareivr* with one or more of the following:

- , (comma) a one-second pause
- . (dot/period) a five-second pause

The following example shows how to send a one-second pause followed by the respondent's serial number, age and gender:

```
age      ask 'How old are you?'
        resp num 18 to 99
gender ask 'INTERVIEWER: Is the respondent ...?'
        resp sp 'Male' / 'Female'
        set mf = nbit(gender)
        set dtmfstr=',[+respondent+]#[+mf+][+age+]'
callfunc('prepareivr',dtmfstr,result)
```

Transferring the respondent to IVR



Quick Reference

To transfer the call on the current extension to the IVR system, type:

callfunc('commitivr', *dtmp*, *result*)

where *dtmp* is an initial string to send to the IVR system and *result* is a variable that will contain an error message if the transfer is not successful.

If *dtmp* is not empty, the callfunc sends this string to the IVR system. Then it sends the *end_dtmp* string defined with *setivr* or in QCIVR. The default *end_dtmp* string is # which tells IVR to go to the first non-blank question in the script.

Here is an example that shows how to use specify the *commitivr* callfunc when you want to use the default *end_dtmp* string:

```
set dtmfstr = ''
callfunc('commitivr',dtmfstr,result)
```

❖ qtip does nothing while the DTMF string is being sent to the IVR system, and usually the respondent hears nothing. It is therefore advisable not to send long DTMF strings at this point otherwise the respondent may hang up and abandon the interview.

Once a call has been transferred to the IVR system it cannot return to Quancept, so the script should not contain any more IVR-related callfuncs or questions for the current respondent. It can contain postprocessing statements that you want to execute after the interview has been handed off, for example to write accounting or other details to the data files.

Cancelling an IVR call



Quick Reference

To cancel a call made with *prepareivr*, type:

callfunc('abendivr', result)

result is a variable that will contain an error message if the action is not successful.

Cancelling a call terminates the connection and hangs up the call. Even though qtip will cancel an unused connection at the end of the interview, it is better to terminate the connection yourself as soon as it is no longer needed as this immediately releases the resources for other interviews.

Creating an IVR script to receive Quancept data



This section provides some basic background details about IVR and offers suggestions for how to create a simple IVR script to test that data is being passed correctly from your Quancept script to the IVR system. For full details about writing IVR scripts, refer to your IVR documentation.

The DTMF string

In the preceding sections you've seen many examples of how to write the various IVR callfuncs and, in particular, how to pass information from Quancept to IVR. Before we look at how to write a simple IVR script to test the data transfer, it will be useful to see how the DTMF string looks when it arrives on the IVR system. Suppose your Quancept script is as follows:

```
comment Questions to pass to IVR
q1 ask 'Have you ever used breath fresheners on a regular basis?'
      resp sp 'Yes' / 'No'
q2 ask 'And are you using a breath freshener regularly now?'
      resp sp 'Yes' / 'No'
      route (q1='No' .and. q2='No') go exit
q3 ask 'How many times a week do/did you use a breath freshener?'
      resp num 1 to 99
q4 ask 'Which type of breath freshener do you mainly use or have you
      mainly used in the past?'
      resp sp 'Spray' / 'Mouthwash' / 'Tablets' / 'Gum' / 'Other'
comment Set up the DTMF string to pass these responses.
      set q1code = nbit(q1)
      set q2code = nbit(q2)
      set q4code = nbit(q4)
comment Start with a two-second pause
      set dtmfstr = ',,[+respondent+]#[+q1code+][+q2code+][+q3+]#[+q4code+]'
      callfunc('prepareivr',dtmfstr,result)
      ....
      set dtmfstr = ''
comment Connect respondent to IVR interview
      callfunc('commitivr',dtmfstr,result)
```

If respondent number 681 has used breath freshener regularly in the past (1) but does not currently use it (2), used to use it 10 times a week, and mainly used tablets (3), the DTMF string that Quancept passes to the IVR system will be as follows:

, ,681#1210#3#

The two commas are the two-second pause. The # at the end of the string is the default value for the *end_dtmf* property that *commitivr* sends when it connects the respondent to the interview. This tells IVR to proceed to the first non-blank question. The IVR system speaks the question text for this question and waits for the respondent to answer using his/her telephone keypad.

To test this script you need to write an IVR questionnaire that contains at least four blank questions to hold the data you are passing from the Quancept script, plus at least one non-blank question so that the IVR interview can take place.

IVR basics

In the IVR system, the characters * and # are special. * means repeat the last question, and # means that data entry is complete for the current question or field.

The IVR system refers to questions as commands. It has a number of commands, most of which have Quancept equivalents. The ones you will need in your test questionnaire are Get Response and Get Value.

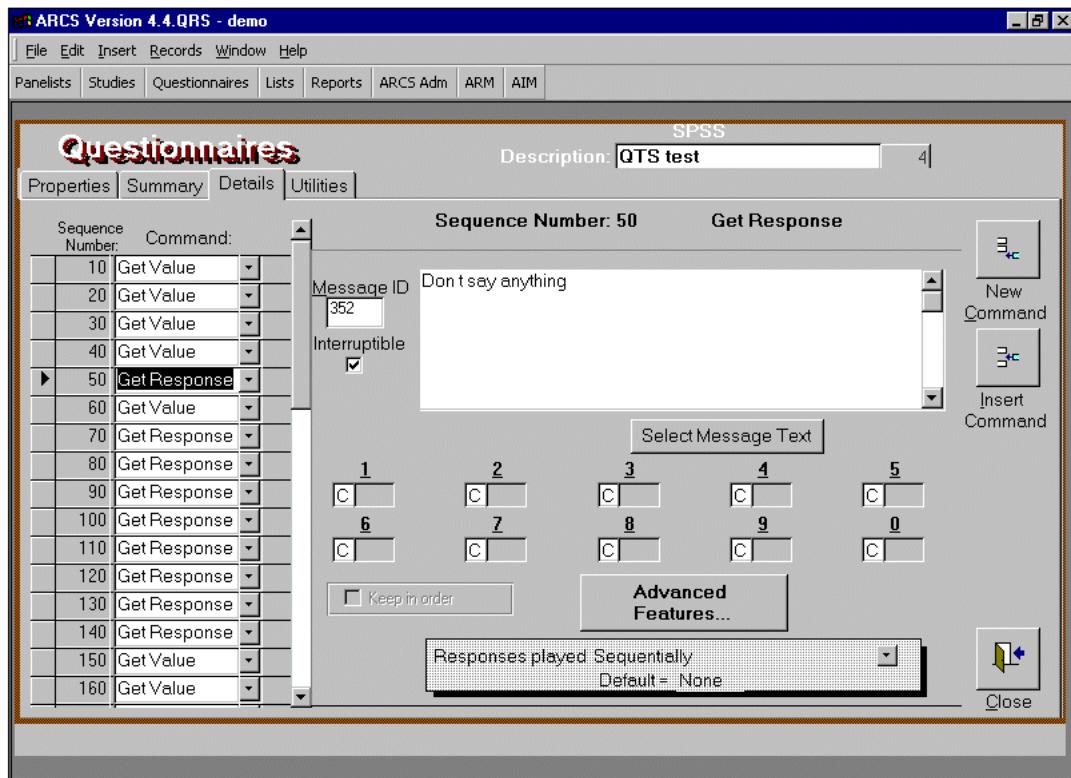
Get Response accepts a single answer in the range 0 to 9. You can use it to receive single-punched data from Quancept, when the response list contains nine or fewer responses, or to receive numeric responses in the range 1 to 9. Each answer may have logic associated with it if appropriate. Once an answer has been received for a Get Response question, the system automatically moves to the next question; there is no need to type # to indicate that data entry is complete.

The following example from a questionnaire recording script shows that the fifth question in an IVR script is receiving a single answer from Quancept. The question accepts any answer in the range 0 to 9:

Questionnaire Recording Script

Seq. #	Message #	Action	Question Text									
			1	2	3	4	5	6	7	8	9	0
50	352	Get Response	Question Don't say anything	C	C	C	C	C	C	C	C	C
			Interruptible	<<<		>>>						

The IVR programming screen for this question looks like this:



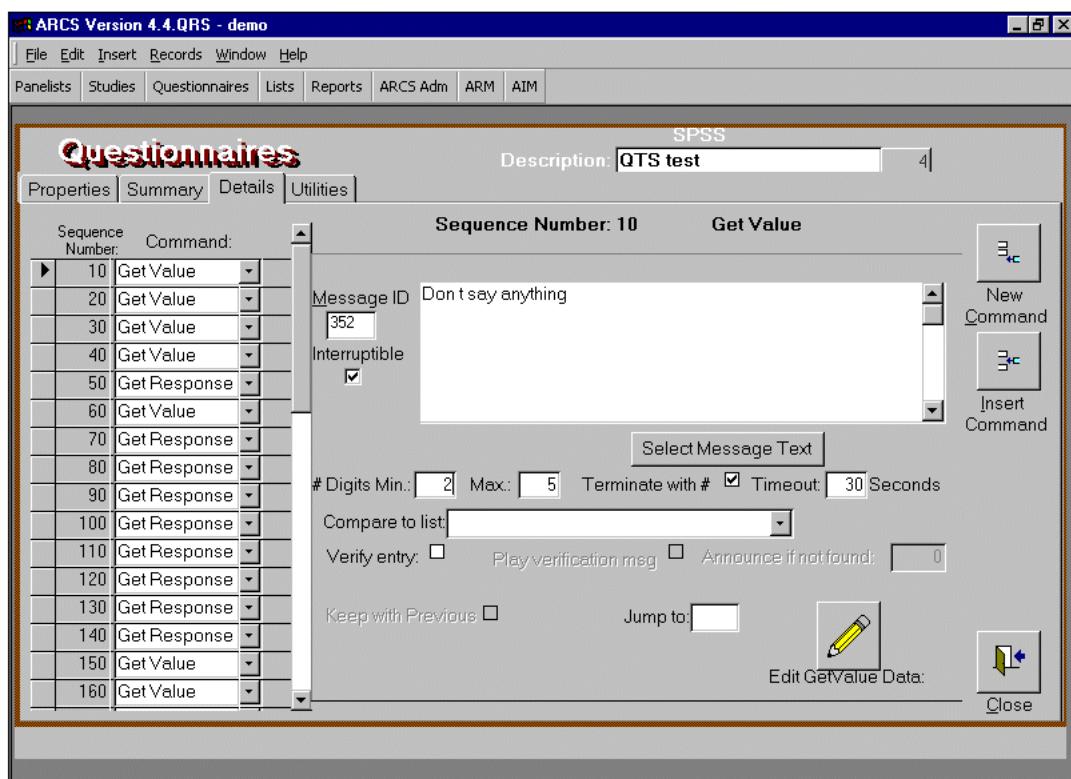
You can pass a series of questions with single-digit responses from qtip into the IVR system as a single string by copying them into a Get Value command (see below).

Get Value accepts a string of digits in the range 0 to 9. Use it when you want to pass integer responses from the Quancept script or to pass single-punched data when the response list contains more than nine responses. The number of digits that a Get Value command accepts is controlled by the *# Digits Min* and *Max* fields. If *# Digits Max* is undefined or you want to accept responses with fewer than the maximum number of digits, the *Terminate With #* check box must be selected so that IVR terminates the data for this command when it receives a # symbol. (If you do not request a termination symbol, IVR will wait until it times out before moving to the next question. At this point, it clears the DTMF string so data for any other blank questions will be lost.)

The following example from a questionnaire recording script shows that the first question in an IVR script is receiving the respondent serial number from Quancept. Up to five digits between 0 and 9 will be accepted, but since not all five values are required the string can be terminated by the # character:

Questionnaire Recording Script												
Seq. #	Message #	Action	Question Text									
			1	2	3	4	5	6	7	8	9	0
10	352	Get Value	<u>Question</u> Don't say anything									
			Interruptible Input 5 digits; terminate with #.									
				<<<			>>>					

The IVR programming screen for this question looks like this:



If IVR receives an invalid response to a Get Response command, it repeats the question. If it receives three invalid responses for the same question it terminates the call. With Get Value commands, if an invalid response is given or there is too long a delay in responding, IVR repeats the question up to three times and then hangs up the call.

Practical suggestions for the test script

Here are some practical suggestions for writing a test script and then testing the interface between Quancept and the IVR:

- Since you'll be using the SQL Analyzer to view data on the IVR system, it is a good idea to use variable names in the Quancept script that match the Question Sequence Numbers in the IVR system. This makes it easy to see which Quancept variables go with which IVR questions.
- A blank Get Response command must define the same number of responses as its Quancept counterpart. When IVR receives data for a Get Response command, it checks that the data is within the range set for that command. If not, it repeats the question. If this happens during hand-off, IVR will take the next character in the DTMF string as the response to the repeated question. This means that you lose synchronization between the DTMF string and the blank questions, and all subsequent questions are offset by one.
- When setting up blank Get Response commands, make sure that you define the responses in the same order as they appear in Quancept. Responses are set according to their position in the list, and if the list orders differ responses will be allocated incorrectly.
- If you are passing a response from Quancept and all responses to the question will have the same number of digits (for example, ages in the range 18 to 99), set the minimum and maximum number of digits in the corresponding Get Value command to the appropriate number and uncheck *Terminate With #*. Do not place # after the variable name in the DTMF string.
- If you are passing a response from Quancept and the number of digits in the response may vary between respondents, you must select *Terminate With #* in the corresponding Get Value command. In the DTMF string the variable must be followed with #, except if it is the last variable; in this case the # is supplied by *committivr*. Any errors in this respect will cause the data to be incorrectly assigned.
- Get Value commands wait 30 seconds to receive either the specified maximum number of digits or the # termination symbol. If neither is forthcoming, the question times out and the dtmf string is cleared. If there is more data in the string it will be lost. If you follow the previous two rules time-outs should never occur. However, to make certain, set the *Timeout* field to a large value — say, 300 seconds.
- Do not send data to IVR unless you need to use it in the IVR questionnaire script. If you don't have to send data, send just the respondent serial number or SMS record key (if it is numeric). If you have to send responses because you want them to appear in IVR's SQL database, but you do not need to use them in the interview, pass the data into a Get Value command as a single string, terminated by #. You can then separate this data afterwards when you have retrieved it from the database.
- To test the IVR questionnaire, print the Questionnaire Recording Script and mark answers to the questions on the script as they would be answered in Quancept. Then dial the IVR system manually and enter these answers using the telephone keypad to test the IVR script before you try passing any data from Quancept.

- Once you are satisfied that the IVR system is receiving data correctly, you can test it with your Quancept script. Mark the answers you will be giving on the Questionnaire Recording Script and then enter these answers in qtip. Your Quancept script should store the string of answers you will be passing to the IVR system in one or more dummy questions so that they can be referenced in the drs file or displayed on the interviewer's screen.

27.17 Sample scripts

Sample script A

 This script requires an SMS server to be running.

```
comment Sample script for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 27, Callfuncs (CATI only)

comment Save data every 300 seconds (5 minutes)
    callfunc('autostop',300)

comment Check whether interviewer is interviewing in the afternoon.
    set pod=partofday
comment Store info in qca record accounting field
    callfunc ('acctinfo',1,pod)

comment Set data serial number if keying in interviews from
comment printed sheets
dowhat    ask 'Are you entering a manually recorded interview?'
            resp sp 'yes' / 'no' go ctn
sernum    ask 'What is the interview serial number?'
            resp num 1 to 99999 nodata
snverif   ask 'Please confirm that the serial number for this
interview is '+sernum
            resp sp 'Yes, correct' go sds /
                    'No, not correct' nodata
            goto sernum
sds       callfunc('setdataser',sernum)

ctn      continue

comment Start timing sections
        set sectnum = 1
        callfunc('timesect',sectnum)
        set sectname = 'demog'
        callfunc('showinf',sectname)
gender    ask 'INTERVIEWER: Is the respondent ...'
            resp sp 'male' / 'female'
age      ask 'Which age group do you belong to?'
            resp sp '18 and under' / '19 to 25' / '26 to 35' /
                    '36 to 45' / '46 to 55' / '56 to 65' /
                    '66 and over'
            set sectnum = 2
            callfunc('timesect',sectnum)

comment The * before sweets causes the entry for the sweets section to
comment be highlighted in the supervisory menu.
        set sectname = '*sweets'
        callfunc('showinf',sectname)
```

```
sweets    ask 'Do you eat sweets?'
          resp sp 'yes' / 'no' go exit
prefer   ask 'Which sort of sweets do you prefer?'
          resp sp 'boiled fruit sweets' / 'soft fruit sweets' /
                  'hard mints' / 'soft mints' / 'chewing gum' /
                  'toffee' / 'chocolate' / 'chocolate snack bars' /
                  'chocolate biscuits'
          set sectnum = 3
          callfunc('timesect',sectnum)
          set sectname = 'prodtry'
          callfunc('showinf',sectname)
tried    ask 'Have you tried the new Plain and Milk Slice
          chocolate bars?'
          resp sp 'yes' / 'no'
q3      ask 'Would you be willing to take part in a test of a new
          peppermint chocolate bar?'
          resp sp 'yes' / 'no' go exit

comment Store info in qca record accounting field
        callfunc('acctinfo',2,q3)

comment Save data every 30 seconds
        callfunc('autostop',30)

        display 'I''ll need your name and address so I can send
you some samples of the peppermint chocolate bar to try. I''ll
also include some information on how the test should be carried
out.'
title   ask 'What is your title?'
        resp coded
name    ask 'Please let me have your first initial or your first
          name and then your last name.'
        resp coded
address ask 'And your address. INTERVIEWER: Do not enter postcode here.'
        resp coded
PCODE   ask 'And your postcode'
        resp coded

comment Revert to saving data every 300 seconds (5 minutes)
        callfunc('autostop',300)

comment Write name and address data to a file and run the
comment command 'csh do_let'
        callfunc('varlist',title,name,address,PCODE,title,name)
        set dothis = 'csh do_let'
        callfunc('subprog',dothis)

comment Calculate callback appointment time.
comment Get current time/date information in Unix format
        callfunc('datetime',udate)
```

Quancept and mrInterview Scriptwriter's Manual

```
comment Extract day and set suggested callback day as appropriate.
comment If today is Friday, delay is 10; otherwise delay is 7.
    callfunc('datepart',udate,5,daynum)
    if (daynum = 6) {
        set delay = 10
    }
    else {
        set delay = 7
    }
    callfunc('adddatepart',udate,2,delay)

comment Test that appointment is not after 30 Nov 2001
comment You may want to change the dates!
    callfunc('datepart',udate,1,monthval)
    callfunc('datepart',udate,0,yearval)
    set oct01 = logical (monthval = 10 .and. yearval = 01)
    set nov01 = logical (monthval = 11 .and. yearval = 01)
    route (oct01 .or. nov01) go showap

comment Appointment later than 30 Nov 2001, so reset to 30 Nov 2001
    callfunc('setdatepart',udate,0,01)
    callfunc('setdatepart',udate,1,11)
    callfunc('setdatepart',udate,2,30)

comment Convert suggested appointment time into a text
showap    callfunc('datetext',udate,tdate)
chkap    ask 'I''ll need to call you back in about a week''s time to
        find out what you thought of the test product. Would '+tdate+'
        be convenient? @@INTERVIEWER: If not convenient, you''ll be
        prompted to arrange a suitable time with the respondent.'
        resp sp 'yes' / 'no'
        route (chkap='yes') go put

comment Interviewer to be prompted for appointment time
mkap      callfunc('makeappt',udate)

comment Write appointment time to apptime field in SMS record
put      set apptime = 'apptime'
        callfunc('putsmvar',apptime,udate)
exit     display 'Thank you for your help and goodbye.'
        pause
        end
```

The template letter, choclet, that is used by the external program, do_let, is:

Thank you for agreeing to take part in the trial of the Superchoc Plain and Milk Slice.

We're enclosing three bars of this chocolate for you to try. Please make a note of your opinions, trying to be as specific as possible. For example, what did you like or dislike about the flavor and why; have you any opinions on the packaging; and so on.

Yours sincerely,

Benjamin McDonald

and the do_let shell script that converts it into a respondent-specific letter is:

```
#  
set line = `tty | sed 's/.*\(\..\\).*/\\1/'`  
set file = qc_${line}out.txt  
echo $file  
ex - $file <<'EOF'  
1d  
1,$s/|/\\  
/g  
1,2j  
$-1j  
s/ .* / /  
s/^/Dear /  
i  
  
.br  
$r choclet  
w ./letter$$  
q  
'EOF'
```

The sed statement in the first line of the script is designed for two-digit line numbers such as /dev/tty05 or /dev/pts/19. You will need to modify this statement if you have single-digit line numbers.

Sample script B

```
comment Sample script B for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 27, Callfuncs (CATI only)

comment Questions to pass to IVR
q1      ask 'Have you ever used breath fresheners on a regular basis?'
        resp sp 'Yes' / 'No'
q2      ask 'And are you using a breath freshener regularly now?'
        resp sp 'Yes' / 'No'
        route (q1='No' .and. q2='No') go exit
q3      ask 'How many times a week do/did you use a breath freshener?'
        resp num 1 to 99
q4      ask 'Which type of breath freshener do you mainly use or have you
mainly used in the past?'
        resp sp 'Spray' / 'Mouthwash' / 'Tablets' / 'Gum' / 'Other'

comment Define error message for when things don't work
set errtxt = 'There is a technical problem. The interview cannot continue.'

comment Define IVR system configuration properties. The recommended procedure
comment is to specify them in the QCIVR environment variable (possibly stored
comment in the QCENV file). For demonstration purposes, the setivr callfunc is
comment used instead. It defines the phone number of the IVR system and the
comment number of seconds to wait before treating the call as unanswered. The
comment defaults are used for all other properties.
set ivrpar = 'ph=02071234567:noansw=5'
callfunc('setivr',ivrpar,result)
if (result <> '') {
    display errtxt
    goto exit
}

comment Set up the DTMF string to pass these Quancept data to IVR.
comment Start the string with a two-second pause.
comment respondent and q3 are followed by # because they do not have a
comment fixed number of digits and we want to ensure that the correct
comment number of digits are assigned to each IVR command.
comment qlcode, q2code and q4code are single characters so IVR moves
comment to the next question as soon as it receives a code.
set qlcode = nbit(q1)
set q2code = nbit(q2)
set q4code = nbit(q4)
set dtmfstr = ',,[+respondent+]#[+qlcode+][+q2code+][+q3+]#[+q4code+]'

comment Explain procedure and ask if respondent is willing to be interviewed
comment in this way. Establishing the connection to the IVR system may take
comment some time, so have the interviewer read the explanatory text while
comment this happens.
display 'The rest of this interview will be conducted as an automated
survey. A machine will ask the questions, and you can answer by typing numbers
on your telephone keypad.'
```

```

callfunc('prepareivr',dtmfstr,result)
if (result <> '') {
    display errtxt
    goto exit
}
ivrchk ask 'Are you willing to be interviewed in this way?'
resp sp 'Yes' / 'No' go abend

comment Build the final DTMF string sent to the IVR system and connect the
comment respondent to the IVR interview. At this place you can add values
comment that were not available earlier in the script.
comment The default property end_dtmf=# takes the respondent to the first
comment non-blank question.
set dtmfstr = ''
handoff ask 'Thank you for your help. You are now being transferred
to the automated interview.'
resp sp 'Continue' / 'Abort' go abend
callfunc('commitivr',dtmfstr,result)
if (result <> '') {
    display errtxt
    goto exit
}

comment Hang up the call to the IVR system.
abend callfunc('abendivr',result)
thank display 'Thank you for your help'
exit end

```

Creating the IVR questionnaire

Follow the instructions in your IVR manuals for creating a questionnaire. The questionnaire needs the following questions, each with a blank question text to receive data from the Quancept script:

Question	Command type	Settings
1	Get Value	Minimum number of digits=1. Maximum number of digits=5. Terminate with #.
2	Get Response	Accept codes 1 and 2 only.
3	Get Response	Accept codes 1 and 2 only.
4	Get Value	Minimum number of digits=1. Maximum number of digits=2. Terminate with #.
5	Get Response	Accept codes 1, 2, 3, 4 and 5 only.

Add at least one non-blank question of your choice. When *commitivr* transfers the respondent to the IVR system, this will be the first question asked.

28 Writing and using local callfuncs

Local callfuncs are callable functions that you write yourself or that are written for you. They provide a means of performing tasks that are not covered by the standard Quancept language. This chapter describes how to write these functions, and is designed mainly for programmers. However, it also contains information explaining how to use local callfuncs which may be of interest to all users.

28.1 Local callfuncs in Quancept CATI

Using local callfuncs



Quick Reference

To call user-defined functions in a script, type:

```
callfunc('local', sec_num, variable_names)
```

where *sec_num* is a number or numeric variable defining the section of the local callfunc definition to execute, and *variable_names* are the variables to be used in that callfunc.

Although Quancept provides you with a wide variety of facilities, it is still possible that there may be something that is not provided or that is not done in exactly the way you require. Quancept therefore allows you to write your own callfuncs and call them in the same way as the standard callfuncs. All local callfuncs must be defined in a file called local.c.

If the function was written by someone other than yourself, he/she will tell you what number to type for each task, which variables are required, and in what order they must be named. Here is a sample statement for address labels:

```
callfunc('local', 1, smskey, name, address)
```

This might produce a label of the form:

1001/256

Barbara Horton
SPSS MR
67 Maygrove Road
London NW6 2EG
ENGLAND

where 1001/256 is the SMS key, Barbara Horton is the name, and the rest of the text is the address (the function itself breaks the address into separate lines).

Naming local callfuncs



>All local callfuncs must be defined in a file called local.c.

The standard callfuncs provided with Quancept are all defined individually with their own unique names. The function name **local** is reserved as a general reference name of any number of user-defined functions that may vary from site to site. Functions within the main local function are defined using a *switch* statement. Scriptwriters wishing to use a specific user-defined function will write a *callfunc* statement naming not only the local function itself but also the switch value, which refers to the function required. We call this the switch reference number. For example:

```
callfunc('local', 4, .... )
```

will execute the following statements in the function defined in the local.c file:

```
switch (i) {
    case 4:
        statements
        statements
```

How the parser sees callfuncs



When the parser encounters a *callfunc* statement, it creates an array based on the information in that statement, starting at the second parameter inside the parentheses. In the example above, the second parameter is the number 4. For each parameter, qparse allocates a pair of elements in the array. Within each pair, the first element indicates the data type of the parameter, and the second indicates its value. The first four elements that appear in the array are as follows:

Parameter	Description
Offset 0	The data type of the switch reference number: QU_SP single punched QU_MP multipunched QU_NUM numeric QU_CODED open ended (<i>resp coded</i>) QU_TEXT text (<i>set variable='text'</i>)
	You always enter the switch reference as a numeric value, so you should expect the data type to be QU_NUM. Test for this at the start of the function and reject any statements in which this is not so.
Offset 1	The value of the switch reference number.

Parameter	Description
Offset 2	The data type of the first Quancept variable named inside the parentheses: 1 – QU_SP 2 – QU_MP 3 – QU_NUM 4 – QU_CODED 5 – QU_TEXT
Offset 3	The contents or value of the first variable (in the case of multipunched variables, this is a pointer to a list of values stored elsewhere rather than the values themselves)
Offset 4, onwards	If the <i>callfunc</i> names more than one variable, offsets 4 and 5 will contain the data type and value of the second variable, offsets 6 and 7 will contain the data type and value of the third variable, and so on

As an interview progresses, Quancept stores answers and other information in a large array called the *heap*. When the *callfunc* statement names a multipunched variable, the Quancept parser stores a reference to the location of the values (answers) of that variable in the heap rather than the values themselves.

Making a special version of qtip



Quick Reference

To create a special version of qtip that contains your local callfuncs, go to the directory containing the file local.c and type:

makeqtip

A long as there are no errors in your local.c file, makeqtip creates a file called qtip in the same directory as the local.c file.

There are two ways you can make this version of qtip available to interviewers. You can either:

- move or copy the file into the directory containing the script file, or
- ensure that the path description in the interviewers' .login files points to the directory containing this special version of qtip before it points to the QCHOME directory.

If you want to use the same qtip for several projects, we recommend using the second method.

↳ makeqtip requires the GNU C or C++ compiler and runtime library in order to run.

28.2 Using your own functions in mrInterview



You can write your own functions in mrInterview to perform tasks that are not covered by keywords in the Quancept language. The functions must be written in VBScript and stored in the project's Sample Management script (.sam) file. Apart from the speed with which a function runs, there are no restrictions on what you can use functions for: anything that you can normally do in VBScript functions is valid in mrInterview-related functions.

All VBScript functions in Sample Management are run serially within the project to which they are defined, and on the server on which the interview is running. If multiple users are working on the same project at the same time, then when one user of the project is running the function on a particular server all other users of that project on that server will have to wait before they run the function. If a function takes a long time to run, users who are waiting may have their interviews timed out. Users running the same function in the same project but on a different server will not be affected, nor will users running the same function on a different project on any server.

The basic steps for using your own functions are as follows:

1. Create a Sample Management script defining the functions you want to call from the questionnaire script. Each function receives information from the questionnaire script as a collection of properties and returns information to the questionnaire script in a designated variable.
2. In the questionnaire script, set up the properties collection using temporary variables. All temporary variables that define properties for a function must start with the same character string; for example, f1brd, f1shop, f1price.
3. Call your function from the questionnaire using the **smscript callfunc**. The *callfunc* statement names the property collection to be used with the function.
4. Check the return code that the function passes to the questionnaire to verify that the function ran correctly.
5. Use any returned or updated information in the rest of the questionnaire.

Here is a very simple example that illustrates how this process works: more detailed information is provided in the rest of this chapter. The questionnaire passes a numeric response to a function. The function adds 10 whatever value it receives and returns the new value as a new property (temporary) variable. The questionnaire script is as follows:

```
comment Define prefix for property variables
      set propprefix = 'prop'
q1      ask 'Choose a number between 1 and 20.'
      resp num 1 to 20

comment Copy the response into a property variable
      set propq1 = q1
```

```

comment Name the function to be called and initialize the property variable
comment in which the function will place the result of 'ql+10'
    set scrfunc = 'MyFunc'
    set propplus10 = 0
comment Call the function and name the property collection to be used
    callfunc('smscript',scrfunc,proprefix)

comment Test whether smscript was successful and display the value of
comment the propplus10 property variable or an error message as appropriate.
comment smscrval is set automatically to the HRESULT of the callfunc.
    if (smscrval < 0) {
dfail        display 'The function failed.'
    }
else {
dok         display 'Original value: '+ql+'. New value: '+propplus10
    }

```

The MyFunc function is defined in the Sample Management script file as follows:

```

Function MyFunc(Properties)
    ' Add 10 to the value of propq1 and save result in propplus10
    Properties.Item("propplus10").Value = Properties.Item("propq1").Value + 10
    MyFunc = SUCCESS
End Function

```

Writing the function



Quick Reference

Define the function in the project's Sample Management script as follows:

Function *Name(Properties)*

statements

End Function

You define the tasks you want to carry out as functions in the project's Sample Management script file. If the project uses Sample Management, you just add the functions to the existing sam file. If the project does not use Sample Management, you will need to set up a dummy sample table and a Sample Management script file containing an AuthenticateSampleRec function that always returns SUCCESS. You can then add your functions to this file.

You write the functions using the VBScript language. Apart from the speed with which the function runs, there are no restrictions on what you can have the functions do: anything that you would normally do in a function outside mrInterview is acceptable within the functions that you write for mrInterview. All VBScript functions in Sample Management are run serially. If multiple users are working at the same time, then when one user is running the function all other users will have to wait before they run the function. If a function takes a long time to run, users who are waiting may have their interviews timed out.

Besides using standard VBScript facilities, you can also access any Interview and Project properties that are maintained by mrInterview.

-
- ☞ Refer to the Sample Management section of the online *mrInterview User's Guide* for information about the methods and properties available in the object models associated with mrInterview. It is also worth looking at the sample scripts that are installed in the `mrInterviewSource\Samples` directory.
-

Most functions will receive information from the questionnaire script, and many will return modified or additional information to the questionnaire, so it is important that you refer to property variables consistently in both the function and the questionnaire script. If you write the function first, you must ensure that you use the same names in the questionnaire script, and vice versa.

-
- ☞ Temporary variable names in the questionnaire may be up to 16 characters long and must start with a letter, so do not use names longer than this in the function if the variables or properties are to be available to the questionnaire. Items that are used only in the function may follow the standard Visual Basic naming conventions.
-

All functions should return one of the standard Sample Management return codes:

SUCCESS	The function ran successfully.
FAILURE	The function did not run successfully.
REJECT	The function rejected the interview.
PARAM_ERROR	There was a problem with the parameters passed to the function.

mrInterview reads the return code and sets the `smscrval`, `smscrtxt` and `smscrres` temporary variables in the questionnaire so that the script can check whether or not the function was successful.

-
- ☞ See 'Checking whether the function ran successfully' for more about these variables.
-

If you want your function to write out or manipulate data received from the questionnaire, you'll need some knowledge of how the function sees the data it receives. The following table summarizes this information.

Script variable type	Variable type in function	Example data
num	<code>vbLong</code>	46
real	<code>vbDouble</code>	16.87
coded	<code>vbString</code>	It was expensive.
sp	<code>vbVariant</code>	{23}

Script variable type	Variable type in function	Example data
mp	vbArray	{23,26,27}

-
- ☞ The LogQuestions project in the mrInterviewSource\Samples directory provides a runnable example of a function that writes out the different data types to the mrInterview log file.
 - ☞ For further information about mrInterview data formats see section 38.4, The mrInterview case data file, and also the SPSS MR Data Model documentation.
-

Defining property variables



Quick Reference

To define the prefix for a property collection, type:

```
set propprefix[(n)] = 'name'
```

Use the array notation — **propprefix(n)** — if you want to define more than one property collection.

To assign values to property variables, type:

```
set propvarname = varname
```

where *propvarname* is the name of the property variable and *varname* is the name of the question or temporary variable whose value is to be copied. You may use an expression that includes the name of a question or temporary variable — for example, *varname+3* — instead of just a variable name if this is appropriate.

To pass information back and forth between the questionnaire and a function you create a **Property Collection**. This is a set of temporary variables that share the same prefix for their variable names. For example, the variables propname, propage and propq1 are one collection while the variables testbrd, testcost and testq2 are another. In order to distinguish temporary variables that are part of a properties collection from temporary variables that are used for other purposes, we will refer to them as **property variables**.

To define the prefix for a set of property variables and the name of the property collection itself, type:

```
set propprefix = 'name'
```

Once this statement has been executed, all subsequent variables that start with *name* are assumed to be property variables within a collection called *name*. Like all temporary variable names, the names of property variables may be up to 16 characters in length and must start with a letter.

If the value of a property variable is changed by a *set* statement during the course of an interview, the new value is copied into the property collection. Similarly, when you read the value of a property variable — for example, if you assign the value of a property variable to a question or temporary variable — mrInterview extracts the variable's value from the property collection and assigns that value to the question or temporary variable. This ensures that the questionnaire always uses the latest value, in case a function has changed the value of the property variable after it was originally set in the questionnaire script.

Creating more than one property collection

If you have more than one function and the functions need different information, you will need to create a separate property collection for each function. You cannot create two property collections that exist simultaneously simply by defining *propprefix* twice, because the second definition overwrites the first one, causing the property variables for the first prefix to become temporary variables. Instead, specify *propprefix* as an array in which each cell refers to a different collection. For example:

```
set propprefix(1) = 'top'  
set propprefix(2) = 'last'
```

defines two property collections. All variables whose names start with 'top' will be part of the first collection and all variables whose names start with 'last' will be part of the second collection.

The example uses unique names for the two property collections and you are advised to do so too. In particular, avoid prefixes whose names overlap. The following example illustrates why.

```
set propprefix(1) = 'pr'  
set propprefix(2) = 'prop'  
comment pr is a property collection, so prtvar1 must be a property  
comment variable within this collection  
    set prtvar1 = tvar1  
comment The next statement is ambiguous. It could be creating a property  
comment variable called prop in the pr property collection, or it could  
comment be an incorrect definition of a variable in the prop collection.  
    set prop = tvar2
```

Assigning values to property variables

Once you have named a property collection, you can define property variables containing information that you want to pass to the function. You define property variables using a *set* statement, in the same way as temporary variables:

```
set prop_varname = value
```

In this statement, *value* can be an explicit value such as 3 or 'Brand B', or a variable name, or an expression. In most cases, it will be the name of a variable whose value you want to use in the function, or an expression such as varname+3 that contains a variable name.

The data type of the property variable is determined by the type of data that you place in it, and once the data type is set it cannot be changed. Any new value that is assigned to the variable is converted to the variable's data type. In this respect, property variables are the same as question and temporary variables.

Storing the same data in more than one property collection

A question or temporary variable can belong to only one property collection. If you assign the same question or temporary variable to property variables in different collections, the latest assignment overrides earlier assignments. If you need to place the same variable in more than one property collection, make a copy of the original variable and then assign the original variable to one property variable and the copy to the other property variable, as shown below.

```
comment How to make q1 available in two property collections
q1    ask ...
      resp ...
      set copyq1 = q1
      set propprefix(1) = 'top'
      set topq1=q1
      set propprefix(2) = 'last'
      set lastq1=copyq1
```

Converting property variables back into temporary variables

When you have finished with a property collection you can either leave the variables as they are or you can convert them into ordinary temporary variables by typing:

```
set propprefix = ""
```

This also deletes the property collection.

- ❖ Although they are temporary variables, property variables work more slowly than ordinary temporary variables and may be more restricted in how they can be used. This is because the sif file converts numeric, real and text temporary variables into Long, Double and String variables respectively and converts categorical variables into an array of variants of Long. Property variables are always created as Variants. Also, you can assign Other specify text and the special responses *null*, *dk* and *ref* to temporary variables, but not to property variables. Because of this, you may wish to convert all property variables back into temporary variables as soon as you have finished with them.

Calling a user-defined function

Quick Reference

To call a function defined in the Sample Management script file, type:

```
callfunc('smscript', funcname, proppref)
```

where *funcname* is a variable containing the function's name and *proppref* is a variable that defines the prefix of the temporary variables that make up the function's property collection.

For example:

```
set propprefix = 'pr'  
set fname = 'Maths'  
callfunc('smscript', fname, propprefix)
```

to run the Maths function using values in the pr property collection.

If the function returns data to the questionnaire, it is a good idea to initialize the associated property variable in the questionnaire before calling the function. This lets you control the data type of the property variable so that the data is returned in the format you require. For example, if the result of a calculation is an integer but you want to store the result as a real number, initializing the property variable with a real value forces any integer results to be converted to reals when the function copies them into the property variable.

Passing information between functions

If two or more functions use the same property collection or you call a function more than once, you can pass information between the functions or calls using property variables. The variables that you use for this purpose need not be used or initialized in the questionnaire, and need not have a prefix. You might do this when you want to pass a file handle between the functions. You can also use these types of property variables for data types such as dates that the Quancept language does not recognize.

Checking whether the function ran successfully

Just before *smscript* returns control of the interview to the questionnaire, it assigns values to some special variables indicating the success or failure of the function. Unlike the property variables, these special variables may be defined as (dummy) questions or temporary variables.

smserval Set to the HRESULT of the *smscript* callfunc. A value less than zero indicates failure.

smsctxt	Set to the text for the HRESULT of the <i>smscript</i> callfunc. If <i>smscript</i> failed then <i>smsctxt</i> contains an error message, otherwise it contains ‘The operation completed successfully’.
smscrres	If <i>smscript</i> was successful, <i>smscrres</i> is set to the function result of the called function. This value is always converted to a string. It can be used to return a value from the function in addition to those in the property collection. If <i>smscript</i> fails this variable is not set.

For example:

```

        set propprefix = 'prop'
q1      ask 'Choose two colors from the list?'
        resp mp 2 ran 'red' ('red') / 'blue' ('blue') / 'green' ('green') /
                'yellow' ('yellow') / 'purple' ('purple') /
                'orange' ('orange')
        set propq1 = q1
        set scrfunc = 'MyFunc'
        set prop1color = ''
        callfunc('smscript',scrfunc,propprefix)

comment If the smscript callfunc failed, display the error return code
comment and message generated internally by the callfunc. If smscript
comment was successful, display the original value of q1, the value of
comment the prop1color property variable, and the additional return
comment value set in MyFunc.
        if (smscrval < 0) {
dfail      display 'MyFunc failed: '+smsctxt+' ('+smscrval+')'
}
        else {
dok       display 'Colors chosen: '+q1+'. First chosen was: '+prop1color.
Additional information returned: '+smscrres
}

```

The MyFunc function is defined as follows:

```

Function MyFunc(Properties)
    'Read data from property variable
    prop1color = Properties.Item("propq1").Value
    'If two responses were chosen, drop the one chosen second
    If UBound(prop1color) > 0 Then
        Redim Preserve prop1color(UBound(prop1color) - 1)
    End If
    'Save the response in the new property variable
    Properties.Item("prop1color").Value = prop1color
    ' The value of MyFunc will be available in the smscrres variable in
    ' the questionnaire
    MyFunc = SUCCESS
End Function

```

Restarting interviews that use functions

- When a respondent stops and restarts an interview that has user-defined functions, mrInterview re-initializes the property collections when the interview is restarted. As the existing part of the interview is replayed, the property collections are rebuilt and the functions are executed again. Depending on the logic in the script, the property collections may or may not contain the same values as in the original interview.

29 Using information in sample records

The Sample Management System (SMS) is the part of Quancept CATI that stores the telephone numbers to be dialed and deals with the presentation of those numbers to interviewers at the appropriate times.

When a project uses SMS rather than printed phone lists, the numbers to be called, which are known as *sample*, are stored in files. Additional information about a respondent is sometimes available in a separate database file associated with SMS. It may be data such as name and address, which was available when the phone list was compiled, or data such as car driven or brand tried, which becomes available during the course of the interview.

Any information in the SMS sample record or database is available for use during the interview. If new or updated information is gathered, it may be written back into the sample record or database so that it can be used the next time that respondent is called.

mrInterview's Sample Management system currently allows for inbound calling only but, like Quancept CATI, still allows for information to be passed from the sample record to the script, and for the sample record to be updated with information gathered during the interview.

Quancept CAPI has a minimal implementation of Sample Management. Although you cannot currently issue predefined sample to interviewers, it is possible to define the structure of a sample record and to allow interviewers to enter data about the respondents they speak to. Records created in this way can be written to and read from by statements in the script. It is also possible to return the sample records to the central CAPI server, but this would require changes to the standard LLQ script.

☞ Further information about the CAPI server and the LLQ script is available in the *Quancept CAPI Management Manual*.

The keywords described in this chapter are:

getdbvar	read information from the database file into the interview
getsmvar	read information from the sample record into the interview
putdbvar	write new or updated data back to the database file
putsmvar	pass new or updated data back to the sample record
querysms	send a query to the SMS server and wait for a reply

29.1 Sample record format



In Quancept CATI, a sample record contains the telephone number to be called and any information associated with that number which is required for sampling purposes. For example, if you need to call so many respondents in a number of regions, region would be included in the sample records. Additional information about a respondent which is not required for sampling purposes may be stored in the SMS database (see below).

Sampling information is stored in variables, and the sample record is simply a collection of variables and values:

```
*key=0017;telnum=071-625-7222;contact=Diane Da'Costa;region=1
```

Every sample record must have a *key* variable. This defines a unique code that SMS uses to identify the record. In many cases, the telephone number will be used as the key, but our example has a separate key.

☞ For further information on sample files, see the *Quancept Sample Management and Telephony Systems User's Guide*.



In mrInterview, where only inbound calling is supported, sample records need not exist in order for interviews to take place. If the survey is open only to people who have been invited to participate, then it is likely that you will have sample records that contain information about each respondent. With open-access surveys, where the questionnaire is freely available on your web site for anyone to use, you will not have sample records. Instead, the Sample Management system creates them as interviews are started.

Sample records are held in sample tables in a sample database that you create using Microsoft SQL Server. You can create a different sample table for each interview or, if you use a panel of respondents, you can reuse the same table for a number of surveys. Every sample record must have a unique record ID by which it can be identified.

☞ For further information on Sample Management in mrInterview, see the online *mrInterview User's Guide*.

29.2 Reading sample information into the script



Quick Reference

To copy information from the sample record into the script, type:

```
callfunc('getsmvar', samp_varname, script_varname)
```

where *samp_varname* is a variable containing the name of the sample variable whose value you want to use, and *script_varname* is the name of the script variable which will store that value.

- ☎ The sample variable named must be an SMS record variable defined in the variables or recvars files, and it must be one whose definition is preceded or followed by an asterisk indicating that it is to be passed from SMS to the interviewing program.

There is no relationship or link within the interviewing program between the data serial number, which the interviewing program assigns, and the sample record key/ID, which is defined in the sample file, so you are advised to use this function to copy the sample record key into the interview data files. Here's an example of how this can be done:

```
comment Copy value of sample key variable into sreckey and fix it
comment into the data
    set svarname = 'key'
    set sreckey = ''
    callfunc('getsmvar', svarname, sreckey)
smskey    fix (4) sreckey
```

Here, the variable svarname stores the name of the sample variable containing the information we require; in this case, the record key/ID. The variable sreckey is defined with a blank value ready to receive the record key/ID when it is retrieved from the sample record. The function gets the record key/ID from the Sample Management system, and the *fix* statement writes it into the data file. If the record key/ID is 0017 as in the earlier example, this will be placed in the variable sreckey and then fixed into the data, giving a permanent connection between the interviewing data and the sample record.

- ▣ mrInterview stores the sample record ID in the DataCollection.InterviewerID system variable in each respondent's case data record so there is no need to copy the sample record ID into the case data manually.

29.3 Updating a sample record from the script



Quick Reference

To pass data back to the sample record, type:

```
callfunc('putsmvar', samp_varname, script_varname)
```

where *samp_varname* is a variable containing the name of the sampling variable you want to update, and *script_varname* is the name of the variable in the script containing the new data.

The **putsmvar** function updates variables in the sample record with new or additional information gathered during the interview. For example, to take the respondent's name from the interviewing program variable *respname* and place it in a variable called *contact* in the sample record, you would write:

```
respname ask 'Please tell me your full name'  
        resp coded(0)  
        set svarname = 'contact'  
        callfunc('putsmvar', svarname, respname)
```

If the record already contains a contact name, the new name will overwrite the old one.

 The sample variable named may be any SMS variable (defined in either the *variables* or *recvars* files) but need not be one which is passed to the interviewing program (that is, it need not be flagged with an asterisk).

If the new data is a text containing semicolons, the semicolons are converted to commas before the text is passed to SMS. This is because semicolons act as a termination flag in SMS and may cause the text to be truncated at the semicolon when it is read into the SMS variable.

If you use *putsmvar*, remember that it is SMS rather than the interviewing program that changes SMS record variables. The function merely sets a flag to inform SMS that the current record needs updating. Records are changed only when control of the record is passed back from the interviewing program to SMS, either at the end of the interview or via a *querysms* statement later in the script .

Also, *putsmvar* does not alter your original sample records. Instead, it updates the copy of the sample records held in SMS. If you want to write the updated sample records to a new sample file you must write an SMS algorithm to do so.

- While an interview is in progress, mrInterview holds a copy of the related sample record in memory. Any changes that you make using *putsmvar* are made to that copy of the sample record immediately, so if your script reads the value of a sample variable, changes it and then later rereads the value of that variable, the script will report the new value of the variable rather than its original value. The sample record in the sample database is updated at the end of the interview when mrInterview passes its copy of the sample record back to the Sample Management system.

Note that any variables that you write to must exist in the sample table. You cannot use *putsmvar* to create new variables.

If your company writes its own Sample Management scripts rather than using those supplied with mrInterview, you may also write to variables that have been created by the *AuthenticateSampleRec* function in the Sample Management script. However, since these variables will not exist in the sample table, so any information that is placed in them will not be added to the sample table at the end of the interview.

The earlier example shows how to pass text back to the Sample Management system. The same procedure may be used for numeric data. For single-punched and multipunched data, however, you must convert it into a number or a text before passing it to the Sample Management system. The following examples show how to do this. The first one shows how to convert a single-punched response into a number, while the second shows how to convert a multipunched response into a series of texts. The two methods are interchangeable, so you can save single-punched data as text and multipunched data as numbers if you prefer.

```

comment Convert a single-punched response into a numeric value
spq      ask 'Would you like to be interviewed again?'
          resp sp 'Yes'/'No'
comment Use nbit to change sp values of spq to integers: Yes=1 and No=2.
comment 'again' is an SMS record variable
          set number = nbit(spq)
          set svarname = 'again'
          callfunc('putsmvar',svarname,number)

comment Convert a multipunched response into a series of texts
mpq      ask 'Which of these TV channels do you watch?'
          resp mp 'Channel 1' / 'Channel 2' / 'Channel 3' / 'Channel 4'
          set chan1 = bit(mpq/1)
          if (chan1){
              set tv1=1
              set svarname = 'Channel 1'
              callfunc('putsmvar',svarname,tv1)
          }
          set chan2 = bit(mpq/2)
          if (chan2){
              set tv2=1
              set svarname = 'Channel 2'
              callfunc('putsmvar',svarname,tv2)
          }
similar statements for Channels 3 and 4

```

29.4 Updating sample records changed by the script



Quick Reference

To update the sample record with information passed back with *putsmvar*, type:

callfunc('querysms', action, retval)

action is a number or numeric variable whose value has been defined in SMS as meaning ‘update the record’, and *retval* is a variable which receives a numeric value from SMS indicating whether or not the action was successful.

When you update the sample record with *putsmvar*, this sets a flag which SMS will read when the record is returned to it at the end of the interview. Only when the server reads this flag will the record itself be updated. If you are making several changes to the record at various places in the script, you may want to force the sample record to be updated while the interview is in progress. To do this, enter a *querysms* statement at the point at which you want the record updated.

The variable named for receiving the numeric value from SMS need not exist in the script prior to the *querysms* statement that uses it. If it exists, it will be forced to be numeric; if not, it will be created as such.

If you use this facility, you (or the algorithm writer) must also define an algorithm called **tipquery** to carry out the updating or any other request from the interviewing program. In this algorithm, you define a series of numeric values representing a number of different tasks, one of which might be ‘update the sample record’. This is the value you should assign to *action* before calling *querysms*. For example, if you use 1 to introduce the statements for ‘update the sample record’ in the algorithm, you must set *action* to 1 in the script.

The algorithm should always set a numeric flag to indicate the success or failure of each action. This value will be passed back from the algorithm to the interviewing program in the *retval* variable.

Here is a short script to place the respondent’s name in the sample record:

```
respname ask 'INTERVIEWER -- Record name of respondent'
        resp coded nodata
        set namevar = 'name'
        callfunc('putsmvar',namevar,respname)
comment Pass info to SMS and update sample record. In tipquery
comment algorithm, 1 means update sample record.
query      set action = 1
            callfunc('querysms',action,retval)
```

The algorithm that goes with this script would read:

```
#alg tipquery
tipquery ()
{
int val_arg2=0;          /* variable for qtip return code */
if (val_arg1 == 1) {      /* if querysms action flag was 1 */
    val_arg2 = 1;          /* set successful return value */
    return(val_arg2);     /* return that code to qtip */
}
if (val_arg1 == 2) {      /* flag for different task */
    .....
}
```

When qtip executes a *querysms* statement, it passes the data serial number of the interview, together with any SMS variables which may have been updated in the script, to the *tipquery* algorithm for processing. There are no statements in this algorithm which specifically say ‘update the sample record’, since SMS does this automatically when it receives a sample record from qtip. Thus, whenever you use *querysms*, the sample record will always be updated if necessary. However, we set the return value to 1 to show that the algorithm received the information.

You'll notice that the variable names in the algorithm are not the same as those in the script. The interviewing program and SMS determine the type of information a variable contains by its position in the *querysms* statement. The first variable is assumed to contain the action code (read into the variable *val_arg1* in the algorithm) and the second is assumed to be for the return code (passed from the variable *val_arg2* in the algorithm).

You can use these facilities for other tasks as well. Again, these are described in some detail in the SMS manuals, but we'll look at them briefly here as well.

If you have a product placement study where you want to find respondents or families who are willing to test a new product, you may wish to have one sample record for each person trying the product. If two or more people in the same household are testers, you'll need to duplicate the original sample record for each additional tester. In the script below, we're asking for one record per child:

```
comment numchild stores number of children in household.
11      for child = 1 to 5
            route (child > numchild) go ctn
name      ask 'What is the name of child number '+child+?
resp coded nodata
comment Define names of SMS variables for each child
        if (child=1) {
            set varname='child1'
        }
        if (child=2) {
            set varname='child2'
        }
more statements
```

```
comment Update sample record with name of current child
    callfunc('putsmvar', varname, name(child))
comment Create a duplicate sample record
    set action = 99
    callfunc('querysms', action, retval)
next
ctn    continue
```

The statements in the *tipquery* algorithm which generate the duplicate records will start with the line:

```
if (val_arg1 == 99) {
```

and will include, before the terminating } character, a statement which sets val_arg2 (that is, retval) to a non-zero value confirming that new records have been created.

☞ See the *Quancept Sample Management and Telephony Systems User's Guide* for information on writing a *querysms* algorithm.

29.5 SMS database format



The SMS database is stored in a subdirectory of the project directory called 'database'. Inside this directory, there is one file for each respondent for whom additional information is available. The files are linked to the respondent's SMS sample record using the sample record key or the database key defined in the sample record. For example, if the sample record is:

```
*key=0017;telnum=071-625-7222;contact=Diane Da'Costa
```

any information associated with this record will be stored in a file called database/0017. If the record is:

```
*key=1871;telnum=081-131-1871;contact=Barbara Horton;dbkey=925
```

a file called database/925 will be created instead of one called database/1871. We'll use this record as the basis for some of our other examples below.

Database files consist of a number of lines each with the format *variable=value*. If two or more lines start with the same variable name (for example, you have updated the original information via a script), the line that is latest in the file is used. Here is an SMS database file which we'll use as the basis for the other examples in this chapter:

```
carmake=Vauxhall
caryear=1987
```

You will refer to these variables and those flagged with an asterisk in the sample record when you want to read information into the script or write out similar data collected during the interview.

-
- ❖ If you want to use SMS database keywords getdbvar and putdbvar to pass information between subsurvey scripts, or between the main survey script and a subsurvey script, you must have the main script, the subsurvey scripts and the database directory all in the same directory.
-

29.6 Reading data from the SMS database file



Quick Reference

To read data from an SMS database file into a variable in the script, type:

```
callfunc('getdbvar', dbname_var, tip_varname)
```

where *dbname_var* is a variable containing the name of the database variable to read, and *tip_varname* is the name of the script variable that is to receive the data from the database.

To retrieve the car details from the database file, we would write:

```
comment Retrieve the car type and year of manufacture from the SMS
comment database for verification during the call.
      set dbcar= 'carmake'
      set cartype = ' '
      callfunc('getdbvar',dbcar,cartype)
      set dbmanuf = 'caryear'
      set manufyr = ' '
      callfunc('getdbvar',dbmanuf,manufyr)
ok      ask 'Our records show that you own a '+manufyr+ ' '+cartype+'.@'
Is this correct?
      resp sp 'Yes' go getmodel / 'No'
```

The first three statements are concerned with getting the make of car from the database variable called carmake and placing it in the temporary interviewing program variable cartype. The next three statements perform the same function for the year of manufacture, copying it from the database variable caryear into the temporary interviewing program variable manufyr.

Using the database record we showed earlier, the interviewer would see the question as:

```
Our records show that you own a 1987 Vauxhall.
Is this correct?
```

29.7 Writing data into the SMS database file



Quick Reference

To insert information in a new variable in the SMS database or to update information in an existing one, type:

```
callfunc('putdbvar', dbname_var, tip_varname)
```

where *dbname_var* is a variable containing the name of the database variable to create or update, and *tip_varname* is the script variable containing the value to be inserted in the database file.

To illustrate this, let's write the getmodel question for the previous example:

```
getmodel ask 'What model do you own?'
        resp coded
comment Copy model name into the database
        set dbmodel = 'model'
        callfunc('putdbvar', dbmodel, getmodel)
```

If the respondent owns an Astra GTE, the database file database/925 will change from:

```
carmake=Vauxhall
caryear=1987
```

to:

```
carmake=Vauxhall
caryear=1987
model=Astra GTE
```

The examples we've just used are designed to show you how each of the functions works, but in a real script you'd probably want to write something a little more complex to account for people who have changed cars since the database was created or who have more than one car. The example below suggests one way of writing such a script. You've already seen the statements that extract the car make and year from the database, so we will omit them and start with the question that checks whether the current data is correct:

```
verify    ask 'Our records show that you own a '+manufyr+' '
+cartype+'.@ Is this correct?'
        resp sp 'Yes own that car only' go model /
        'Yes but also own another make as well' go temp /
        'No I own a different car altogether'

comment Get make and year of manufacture for replacement car and
comment update database
make      ask 'What make of car do you own now?'
        resp coded
```

```
year      ask 'What year was it made?'
resp coded
set dbcar= 'carmake'
callfunc('putdbvar',dbcar,make)
set dbmanuf = 'caryear'
callfunc('putdbvar',dbmanuf,year)
goto model

comment Get model of existing car. If respondent has more than
comment one car, ask about the one currently in the database first.
temp      protect 'Dealing with your '+cartype+' first .....@@'
model     ask 'What model is it?'
resp coded
set dbmodel = 'model'
callfunc('putdbvar',dbmodel,model)
unprotect temp
set howmany = nbit(verify)
route (howmany <> 2) go nomore

comment Respondent has several cars. Get info on them and add
comment to database
extra    ask 'How many other cars do you own?'
resp num 1 to 2
l1       for carnnum = 1 to 2
cmake    ask 'What make is car number '+carnum
resp coded
cyear    ask 'What year was it made?'
resp coded
cmod     ask 'And what model is it?'
resp coded

comment Put the new info into the database
if (carnum=1) {
    set dbcar='carmakel'
    set dbmanuf='caryearl'
    set dbmodel='modell'
}
if (carnum=2) {
    set dbcar='carmake2'
    set dbmanuf='caryear2'
    set dbmodel='model2'
}
callfunc('putdbvar',dbcar,cmake(carnum))
callfunc('putdbvar',dbmanuf,cyear(carnum))
callfunc('putdbvar',dbmodel,cmod(carnum))

comment Have finished with last car so skip out of loop
route (carnum = extra) go nomore
next
nomore  continue
```

The comments in the script describe what each section does. First, we check whether the existing data on car make and year of manufacture is correct. There are three possible answers. (A fourth possibility would be that the respondent no longer has a car, in which case you would set the variables to null or blank and write those values back to the database.)

This script also assumes that there is already information in the database for each respondent. If there were respondents without database records, the variables manufyr and cartype would be null when displayed as part of the verify question text. If you write a script similar to this, you'll probably want to include some statements to deal with this — for example, by routing those respondents to a modified version of verify.

If the respondent still owns the same car, we ask for the model and write that back to the database as a new variable in the respondent's record. If he or she owns a completely different car, we get the make, model and year of manufacture and write all three values back to the database. Although the database file already contains information about the make and year of manufacture, the interviewing program does not overwrite it. Instead, it appends the new information to the end of the file so that it contains two definitions for car make and year of manufacture. However, because the interviewing program always takes the latest definition of each variable in the database files, you can be sure that the next time you use that file, the interviewing program will pick up the newest car details.

If the respondent owns the same car as well as other cars, we get the model of that car and update the database with it before going on to the additional cars. To write the script as briefly as possible, we start by finding out how many other cars the respondent has. We then write a loop that is repeated for each car that gets the make, model and year of manufacture.

This information is all written to the database as new variables. Each variable has a unique name created by using the main variable name with the loop subscript as an additional identifier. Without this identifier, all the variables would have the same name, and when we next wanted to extract data from the database, the interviewing program would look only at the last value for each variable. Notice that we've set each variable name by hand because you can't use substitution when creating variable names, for example, `set dbcar='carmake(carnum)'` creates a variable called carmake(carnum) and not carmake(1).

29.8 Sample script



This script requires a database directory containing files defining values for the variables carmake and caryear. An SMS server must also be running.

```
comment Sample script for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 29, Using information in sample records (CATI only)

comment Extract information from the sample record
    set namevar = 'name'
    callfunc('getsmvar',namevar,rname)
    route (rname <> '') go query
respname ask 'INTERVIEWER -- Record name of respondent'
    resp coded nodata

comment Add new information to the sample record
    callfunc('putsmvar',namevar,respname)

comment Set flag for updating of sample record
query      set action = 1
            callfunc('querysms',action, retval)

comment Retrieve data from an SMS database for use in the script
    set dbcar= 'carmake'
    set cartype = ' '
    callfunc('getdbvar',dbcar,cartype)
    set dbmanuf = 'caryear'
    set manufyr = ' '
    callfunc('getdbvar',dbmanuf,manufyr)
verify     ask 'Our records show that you own a '+manufyr+' '+cartype+'.'
Is this correct?
    resp sp 'Yes own that car only' go model /
            'Yes but also own another make as well' go temp /
            'No I own a different car altogether'
make      ask 'What make of car do you own now?'
            resp coded
year      ask 'What year was it made?'
            resp coded

comment Update SMS database with changed data
    set dbcar= 'carmake'
    callfunc('putdbvar',dbcar,make)
    set dbmanuf = 'caryear'
    callfunc('putdbvar',dbmanuf,year)
    goto model
temp      protect 'Dealing with your '+cartype+' first .....@@'
model     ask 'What model is it?'
            resp coded
            set dbmodel = 'model'
            callfunc('putdbvar',dbmodel,model)
unprotect temp
```

```
        set howmany = nbit	verify)
        route (howmany <> 2) go nomore
extra    ask 'How many other cars do you own?'
        resp num 1 to 2
l1      for carnum = 1 to 2
cmake   ask 'What make is car number '+carnum
        resp coded
cyear   ask 'What year was it made?'
        resp coded
cmod    ask 'And what model is it?'
        resp coded

comment Update the SMS database with new data
        if (carnum=1) {
            set dbcar='carmake1'
            set dbmanuf='caryear1'
            set dbmodel='model1'
        }
        if (carnum=2) {
            set dbcar='carmake2'
            set dbmanuf='caryear2'
            set dbmodel='model2'
        }
        callfunc('putdbvar',dbcar,cmake(carnum))
        callfunc('putdbvar',dbmanuf,cyear(carnum))
        callfunc('putdbvar',dbmodel,cmod(carnum))
        route(carnum = extra) go nomore
next
nomore  continue

comment Extract more information from the sample record
        set svarname = 'key'
        set srecket = ' '
        callfunc('getsmvar',svarname,srecket)
smskey  fix (15) srecket
thank   display 'Thank you for your time. Goodbye'
end
```

Files in the database directory contain two lines of the form:

```
carmake=Vauxhall
caryear=1987
```

30 Quota control



Quotas are a method of controlling the number of people interviewed within a particular group. You define the requirements for each group and the number of respondents to be interviewed. Once the requisite number of respondents has been interviewed within a group, any subsequent interviews with respondents who would be classed in that group will be terminated unless you specify otherwise.

A simple example of grouping respondents is by gender: we might need to interview 100 men and 200 women. We could keep a manual tally of the number of respondents of each gender interviewed, but this becomes impractical when several interviews are being conducted simultaneously.

Quancept CATI, Quancept Web and mrInterview have their own quota control system involving the use of quota files or tables and quota statements. The quota files or tables determine the number of respondents to be interviewed in each group, and the quota statements cause the interviewing program to check and decrement these counts as necessary. If the quotas become exceeded, they determine what should be done with the interview.

This chapter describes the script aspects of quota control and introduces the following statements:

quorat	check quota ratios
quota	check quotas for current respondent
quotacell	test for unfilled quota cells
quotamap	take snapshot to test for unfilled quota cells
quoval	report on current quota status

☞ Information on setting and inspecting quotas is given in chapter 31, Quota maintenance.

30.1 Types of quotas

The interviewing program can process quotas for:

- single-punched questions
- multipunched questions
- numeric questions
- text questions (mrInterview only)
- dummy single-punched, multipunched, numeric, or text (mrInterview only) questions. In Quancept CATI and Quancept Web the dummy questions must have responses assigned to them before the *quota* statement is executed

- independent quotas — one question by itself (for example, a gender quota, an age quota)
 - dependent quotas — two or more single-punched, multipunched or numeric questions which together form a multidimensional quota (for example, a quota for gender and age combined)
-  Questions defining quotas are named on *quota* statements at the relevant places in the script.
 If a statement names two or more questions, this means that each quota category has two or more conditions; we call this a ***dependent*** quota. For example, if the quota statement names the questions age and gender, the category conditions are formed by pairing each of the answers to gender with each of the answers to age — for example, female under 18, female 18 to 24, male under 18, and so on. If, instead, we had two separate quota statements, one for age and one for gender, each quota category would have one condition only. This is an ***independent*** quota.

30.2 How the CATI and Web interviewing programs deal with quotas

Quota targets are defined in quota files which are checked when an appropriate *quota* statement is read in the script.

 See chapter 31, Quota maintenance, for information on how to create quota files.

The interviewing program checks whether the respondent belongs in any of the controlled categories and, if so, decrements the counts for those categories by 1 at the end of the interview. Suppose that we need to interview 50 people in each of the age groups 18–20, 21–24 and 25–30. Whenever a respondent between 18 and 20 years old is interviewed, the count for the cell 18–20 is decremented at the end of that interview. The same is true for the other cells.

Once the quota for 18–20 has been filled, it is possible to terminate all interviews with respondents in that age group automatically or to route these respondents to a different section of the script. Interviews with respondents in other age groups may continue until the respective quotas are filled. Where no quotas are specified, an infinite number of interviews may be conducted.

Decrementing quotas

Quotas are checked whenever a *quota* statement is read, but they are decremented only when the *end* statement in the script is reached or when the interview is stopped by a *stop* statement with the parameter *decrm=quota.identifier*. So, if a respondent satisfies the characteristics of a particular quota-controlled category, the interviewing program will remember which category to decrement until the end of the interview.

 If the interviewer stops the interview by typing *stop* or terminates it prematurely by typing *quit* or *abandon*, quotas are not decremented even if the quota statement has already been executed. Similarly, if the *end* statement is reached but the termination flag has been set by a *signal* statement to something other than ‘successful completion’, the quotas will remain unchanged.

If a script contains two quotas and the respondent passes the first quota but fails the second, the count of respondents rejected by the second quota will be incremented, but the cell for the first quota will remain unchanged. Suppose the script contains separate quotas for age and gender, and the cell for women is full. If the current respondent is a woman under age 21, she will pass the age quota but will fail the gender one. The quota program increments the count of respondents failing the quota for women but leaves the quotas for people under age 21 unchanged.

-
- ☞ For information on stopping interviews, see chapter 16, Ending the interview.
-

Restarting stopped interviews



If an interview is stopped and the quota cells for that interview become full before the interview is restarted, the message ‘Interview now overquota — accept anyway (y/n) ?’ will be displayed when the interview is restarted.

If the interviewer answers ‘y’, the interview will continue from where it was stopped (as usual) and will pass successfully through quota control. The quota cell counts will be decremented as they should be and will reflect the fact that the quota target has been exceeded.

If the interviewer answers ‘n’, flags will be set as appropriate, the interview will terminate, and all data collected before the interview was stopped will be written to the data files. The quota counts will show that the interview has been rejected.

Reviewing completed interviews



If you review completed interviews for scripts with quota control, all interviews will pass quota control, but the cells applicable to those interviews will not be updated. This applies to all interviews, even those which failed quota control when they were originally done. The reason for this is to allow you to review completed interviews while the project is still in progress without affecting the quotas.

When reviewing completed interviews, the *cell* value for multipunched quotas may differ between the original interview and the reviewed interview.

-
- ☞ See ‘Finding a respondent’s quota cell’ for further information about the *cell* keyword.
-

Single-punched quotas

When a quota is based on single-punched responses, the quota program decrements the cell whose number corresponds to the position of the response in the list. If the response list contains the words male and female in that order and the respondent is female, the count in cell 2 will be decremented.

Multipunched quotas



When a quota is based on multipunched responses, the quota program decrements the cells for all responses chosen by the current respondent. When quotas start being filled, the quota program terminates an interview only if the quotas for all responses mentioned have been filled. As long as the respondent chooses at least one response in an unfilled quota cell, the interview will be allowed to continue.

Since the interviewing program increments counts for all responses chosen from a multipunched list, some quotas may exceed their targets. If you do not want to go over quota on any item, you can filter out responses which are in filled quota cells by using a dummy question to quota on only those responses whose quotas are not filled.

☞ For an example of filtering with multipunched quotas, see 'Checking for unfilled quota cells' later in this chapter.

Numeric quotas

Numeric quotas are based on the number of values or ranges in the response list. If the response list contains a single number or range, for example:

```
age      ask 'How old are you?'
        resp num 18 to 99
```

Quancept will treat it as a single response similar to an entry in a single-punched response list. To quota on individual numbers or ranges of numbers, you must enter them as separate items in the response list, as shown below:

```
age      ask 'How old are you?'
        resp num 18 to 24 / 25 to 40 / 41 to 60 / 61 to 99
```

null, dk and ref



You cannot quota on *null*, *dk* or *ref* responses. If you anticipate that such responses may be given to the quota-controlled questions, you should write statements in the script to deal with them before the quota statement is reached, otherwise the interview will terminate as soon as the quota statement is reached. The interviewer will see the message 'Invalid type or value of quota variable near label *xx*', where *xx* is the name of the most recently read label. If the quota statement has a label, *xx* will be the label name of that statement.

Effect of snapbacks on quotas



If the interviewer snaps back to a previous question, the interview is rewound. All quotas are unchecked (that is, they are reset so that it is as if the interview had not taken place) and are then rechecked as the interview progresses. If other interviews are in progress at this time, it is possible that the interview may now fail a quota that it passed the first time.

This is particularly noticeable with multipunched quotas, where the rule is to allow the quota to pass as long as at least one of the respondent's answers belongs in a below-target cell. Even if the respondent still passes a multipunched quota test, you may find that the respondent has been allocated to a different quota cell (the *cell* variable). Again, this would be due to some cells reaching their target since the time that the respondent was first tested for this quota.

Dealing with quota errors during interviews



If you test your script properly, respondents should rarely encounter errors to do with quota control. Any errors should be picked up during testing and corrected. The most common error occurs when you copy the project files from the test directory into the live directory and forget to copy the quo file. Another common mistake that you may come across during testing is forgetting to define some of the quota targets. In either case, Quancept Web will be unable to locate the quota target and will issue the message 'A quota cell is not found'. You can define your own message using the ErrQuotaNotFound variable in the [HTML] section of the project.ini file. For example:

[HTML]

```
ErrQuotaNotFound=There is a problem with the quota file. Please call  
+44171 625 7222 and ask for the Quancept Web administrator.
```

30.3 Checking quotas in Quancept CATI and Quancept Web



Quick Reference



To check a quota, write a **quota** statement naming the questions that define that quota:

quota *qname1* [/ *qname2* / ... *qnameN*]

Independent quotas are quotas based on the responses to a single question, so you'll type the name of that question by itself on the *quota* statement. For example, to write an independent quota statement for marital status, you would include the question name mstat in the *quota* statement:

quota mstat

If the mstat question is:

```
mstat    ask 'What is your marital status?'
        resp sp 'single' / 'married' / 'divorced/separated' /
              'widowed' / 'living with partner'
```

the quota control program will expect to find a quota for mstat containing five cells, one for each response.

A dependent quota is based on the responses to more than one question, so you will type the names of those questions on the *quota* statement separated by slashes. To write a dependent quota statement for married men, you would include the questions names gender and mstat in the *quota* statement:

```
quota gender / mstat
```

Be very careful when listing question names in a *quota* statement. For quota control to be successful, question names must be listed in the order in which the questions (dimensions) were or will be defined using the quota program. If the first question named on the *quota* statement is gender, then you must ensure that gender is the first dimension in the quota file, and vice versa. If dimension 2 is marital status, then the question name mstat must be the second question named on the *quota* statement.

In all cases, the rule is to keep the questions and dimensions in the same order at all times. Let's look at a diagram to clarify this. It shows how Quancept thinks of quotas and how it decides which targets belong to which category of respondent. If you write the script before you set up the quota files, and you write:

```
quota gender / mstat
```

you must ensure that when you set up the quota files, you define gender as dimension 1 and marital status as dimension 2. For each dimension, you will be asked how many responses there are in the list. Quancept then builds a table in which it places the target figures:

	Single	Married	Widowed	Divorced
Male	1	2	3	4
Female	5	6	7	8

The numbers indicate the order in which Quancept deals with targets and dimensions. If you defined dimension 1 as marital status and dimension 2 as gender instead, the table would look like this:

	Male	Female
Single	1	2
Married	3	4
Widowed	5	6
Divorced	7	8

The quota statement for this layout is:

```
quota mstat / gender
```

In the first table, the target for married men is in cell 2; in the second table, it is in cell 3. You can see that unless you keep questions and dimensions in the same order, you will not be using the correct targets to control your quotas.

-
- ❖ Do not place statements in the script that change the response to a quota question after the *quota* statement that tests that question. This ensures that the correct quota cells will be decremented at the end of the interview.
-

Scripts with more than one quota



Quick Reference



If the script contains more than one quota, you must identify the quota files containing the targets for the quota to be tested. Type:

quota question names file='identifier'

where *identifier* is the unique two-character code assigned to the quota being tested.

For example:

```
quota gender / mstat file='a1'  
      statements  
quota region file='a2'
```

This tells us that the quotas for gender/marital status are defined in file whose quota identifier is a1, and the quotas for region are defined in file whose quota identifier is a2. Because there are two quota statements and two files, the interviewing program keeps these quotas completely separate.

-
- ❖ Do not use uppercase letters in quota identifiers because the interviewing program does not differentiate between upper-case and lower-case letters.
-

Controlling how interviews terminate



Quick Reference



To intercept the termination command that the quota program sends to the interviewing program when the respondent belongs in a full quota category, type:

quota question names [file='identifier'] pass=variable

variable will be set to false if the respondent fails the quota or true if he/she passes the quota.

When you interview a respondent who falls into a full quota cell, the interview is normally terminated immediately and the interviewer sees a brief message informing him/her that the interview was terminated by quota control. The **pass** parameter is an optional means of intercepting this message so that you can choose what to do next.

pass names a variable which the interviewing program sets to true if the respondent does not exceed the quotas or false if he/she does. Your script will then include statements which check the value of this variable and route to various points in the script according to the requirements of the survey. Using this method, it is still possible to continue the interview even though a quota is exceeded. For example, if the *pass* variable is true, you may want the respondent to continue with the next question, whereas if it is false, you may want to route the respondent to the demographic questions at the end of the script before flagging it as terminated by quota control. If the *quota* statement is skipped, the *pass* variable will be null.

Let's look at an example.

```
age      ask 'How old are you?'
        resp sp 'under 21' / '21 to 35' / 'over 35'
gender   ask 'Is the respondent ...'
        resp sp 'male' / 'female'
        quota age / gender file='aa' pass=ok
        route (ok) go ctn1
comment Respondents whose quota cell is full
thank    display 'Thank you for your help, but we have already
completed our quota of interviews for '+gender+'s aged '+age+'.'
        pause
        signal 5
        goto exit
comment Respondents in unfilled cells come here
ctn     continue
```

This quota has six cells, one for each combination of age and gender. Our *quota* statement indicates that the quota files for the survey are suffixed aa. It also defines a variable (called *ok*) which is to be set to true or false depending upon whether the respondent fails the quota test. Notice how we have dealt with filled quotas. Having routed everyone in unfilled quota cells to the next stage in the script, we then display a message that the interviewer reads to the respondent, explaining how the interview is being terminated. Since we do not want such interviews to be flagged as successfully completed (and thus muddled with successfully completed interviews where quotas were not exceeded), we use the *signal* command to force the interview to be flagged as terminated by quota control.

☞ For further information about the *signal* statement, see section 16.4, Forced termination of an interview.

-
- ❖ There is one anomaly in the way a *quota* statement with a *pass* variable behaves. When a respondent fails the quota test but the interview is allowed to continue, if the interviewer uses any snapping commands or types any special response (**time*, for instance), the interview will be flagged as a successfully completed interview.

This behavior is not what would you would expect and is subject to change.

Combining responses in quotas



Sometimes you may need to deal with quotas which do not exactly match the responses in the corresponding response lists. For example, the script may define six age categories which you need to collapse into three categories for quota control purposes. As long as there is some correlation between the categories in the script and the quota categories, this causes no problems, since you simply define a dummy question for the quota and assign its responses according to the actual response given. The script for such a task is shown below. It shows how to collapse six age groups into three for a three-cell quota:

```

age      ask 'In which age group do you belong?'
        resp sp 'Under 18' / '18-30' / '31-40' / '41-50' /
               '51-65' / 'Over 65'
        set agroup = nbit(age)
dumage   dummyask 'Dummy question for quota control'
        resp sp 'Under 31' / '31-50' / 'Over 50'

comment Collapse age groups into dumage
qst      if (agroup < 3) {
           set dumage = 1
       }
       if (agroup = 3 .or. agroup = 4) {
           set dumage = 2
       }
       if (agroup > 4) {
           set dumage = 3
       }
quota dumage file='a1' pass=ok

```

30.4 Additional facilities for Quancept CATI

Finding a respondent's quota cell



Quick Reference

To find which quota cell a respondent belongs in, type:

set variable = cell

If the script contains several *quota* statements, the cell value will refer to the categories on the most recently executed *quota* statement. Using the diagrams above as an example, if the respondent is a single man and this is cell 1, the value of *set x=cell* will be x=1.

When a *quota* statement fails, the respondent's *cell* value is set to zero.

Checking for unfilled quota cells



Quick Reference

To check for unfilled quota cells, take a snapshot of the current values in a set of quota cells by typing:

callfunc ('quotamap', identifier, num_cells)

where *identifier* and *numcells* are the names of variables containing the identifier of the quota files and the number of cells in the table of quota targets.

Then check the status of each cell in turn by typing:

callfunc('quotacell', cell_num, result)

where *cell_num* is a variable representing the cell number you want to check, and *result* is a variable which will be set to true if the cell is full or false if it is not full.

You can save time by checking whether a quota cell is full before the interviewer asks to speak to a respondent with those characteristics. For example, interviewers may have to check whether there are any women under age 21, between 22 and 30, or over 30 before saying that there are no eligible respondents in the household. You can save the interviewer time by checking whether each of these cells is full before prompting him/her to ask for women in each category. Thus, if the cell for women over 30 is full, the interviewer need only ask to interview women under 21 or between 21 and 30.

To do this, use *quotamap* to take a snapshot of the current values in the quota cells, and then use *quotacell* to test whether each cell is full.

-
- ❖ If you use this facility, make sure that the quota statement associated with these checks contains both the *file* and *pass* parameters.
-

We will look at two examples of using *quotamap* and *quotacell*. The first is designed to save the interviewer time by allowing only respondents in unfilled quota categories through to the main part of the interview. Notice how the use of a defined list and dummy questions have achieved this.

```

comment Take a snapshot of the quota table
      set sufxt = 'a1'
      callfunc ('quotamap',sufxt,6)
cats   define 'men under 21' / 'men 21 to 35' / 'men over 35' /
        'women under 21' / 'women 21 to 35' /
        'women over 35'
qcats  dummyask 'Dummy to store unfilled quota categories'
resp mp cats
qdum   dummyask 'Dummy to store age/gender of respondent for quota-ing'
resp sp cats
unset qcats
unset qdum
comment Check each quota cell in turn. full is true if cell is full
11     for cnum = 1 to 6
          callfunc('quotacell',cnum,full)
          route (full) go back
          set qcats = cnum
back   next
      if (qcats=null) {
          display 'INTERVIEWER: All quota cells are full.
Please speak to your supervisor'
          pause
          goto term
      }
agegend ask 'INTERVIEWER: Which respondent are you interviewing?'
resp sp qcats in cats
quota agegend file='a1' pass=ok
route (.not. ok) go term
.....
term   signal 5
exit   end

```

The first thing the script does is take a snapshot of the quota cells; the rest of it deals with checking the individual quota cells and informing the interviewer of those which are still incomplete.

qdum is a dummy question defining the characteristics of each quota cell. Notice that it is multipunched. This means that once inside the quota-checking loop, we can build up a cumulative array of those cells which are still incomplete. For example, if the cells for male over 35 and female over 35 are incomplete, the third iteration of the loop will give us *qdum*=‘male over 35’ and the sixth iteration will result in *qdum*=‘male over 35’, ‘female over 35’.

The *quotacell* statement in the loop checks quota cells 1 to 6 in turn and sets the variable full to true if the quota has been filled or false if it has not. If all quotas are full, a message to this effect is displayed for the interviewer and the interview is terminated as if by quota control.

If some quotas are unfilled, their names are displayed on the screen so that the interviewer need only ask to interview people in those categories. The next question prompts the interviewer to indicate which category the respondent belongs in or to select the last response if there is no one in any of those categories available. If there are no eligible respondents, the interview is terminated as if by quota control.

The second example below uses *quotamap* and *quotacell* with multipunched quotas. It shows you how to ensure that where a respondent chooses some items in full cells and some in unfilled cells, only the quotas for the unfilled cells are decremented.

```
brands    define  'alpine' / 'helen bradley''s' / 'dairy fresh' /
           'kentish farm' / 'finborough' / 'alsace'
ybrand   ask 'Which brands of yogurt do you buy?'
resp mp brands
yog      dummyask 'Dummy to store responses in unfilled cells'
resp mp brands
unset yog

comment See which cells are full
chk      set ncells = 6
          set suf = 'a2'
          callfunc('quotamap',suf,ncells)
comment Set flag for unfilled cells
          set emptyq = 0
11       for respnum = 1 to 6
          set given = bit(ybrand/respnum)
          callfunc('quotacell',respnum,full)
          set notfull = logical(.not. full)

comment An unfilled cell - store the response text and set
comment the unfilled cell flag
          if (given .and. notfull) {
              set yog = respnum
              set emptyq = 1
          }
next
d1       display 'Responses in unfilled quotas cells are: '+yog
pause
comment Reject respondents with all answers in full cells
route (emptyq = 0) go term
```

```

comment Quota on responses in unfilled cells
    quota yog file='a2' pass=oky
    route (oky) go ctnl
close    display 'We already have enough interviews for those
brands, but thank you for your help anyway.'
    pause
term     signal 5
ctnl     continue

```

The example starts with the question on whose responses the quotas are based. It is followed by a dummy question of identical format which we will use to store only those answers that are in unfilled quota cells with *quotamap*.

Having found out which flavors the respondent likes, we take a snapshot of the current status of the quota cells so that we can check which ones are full. The variable *emptyq* at the start of the loop is a flag which we set to 1 if a respondent chooses an answer in an unfilled cell. If this variable is still 0 at the end of the loop, we know that the respondent chose answers only in cells which were full. We can therefore bypass the quota statement and go straight to the end of the interview where we terminate it as if by quota control (with a *signal 5* statement).

The loop itself looks at each quota cell in turn. The variable *full* will be true if the cell is full. We're interested in unfilled cells, so we set a variable *notfull* to true if *full* is false. Notice the use of *logical* to reverse the value of a logical variable. We then check whether the respondent chose the answer associated with this cell and, if the cell is unfilled, we set this answer into the dummy question. Since we have found a response in an unfilled cell, we also set the *emptyq* flag to 1.

When all responses and cells have been checked, we finish by passing respondents in unfilled cells through the quota statement. We use the dummy variable and decrement cells which we know in advance will be unfilled. In this way, we are able to prevent respondents who would cause the quotas to exceed their targets from being interviewed.

Checking quota ratios



Quick Reference

To find the quota cell with the lowest ratio of completes to the target, type:

`callfunc('quorat', identifier, cell_num)`

where *identifier* is a variable storing the two-character quota file identifier and *cell_num* is a variable which will store the cell number which *quorat* returns. If two or more cells have the same ratio, *qtip* will report the number of the first cell that it finds.

Under normal circumstances, *qtip* controls the order in which quota cells are decremented, and it is therefore possible for one cell to be filled far in advance of the others. For studies with very rigid quota requirements, you may need to take control of this yourself. To help you with this, the Quancept language provides a facility for obtaining the number of the quota cell with the lowest

ratio of interviews to the target, so that you can then decide whether or not to terminate interviews which do not fall into this cell.

Note that the ratio is a percentage figure calculated by dividing the number of interviews in a cell so far by the target for that cell. It is NOT solely the number of interviews in the cell. Thus, if cell 1 has a target of 50 interviews with 12 done so far, and cell 2 has a target of 30 interviews with 10 done so far, cell 1 has a ratio of 24% whereas cell 2 has a ratio of 33%. Cell 1 is therefore the cell that will be returned by *quorat*.

Here is an example of how to use *quorat* in a script. In this questionnaire, we need to interview a fixed number of respondents for each brand of ice cream. At all times, we need to ensure that the ratio of the current quotas to the cell targets is roughly even. We do this as follows.

If the respondent uses only one brand of ice cream, he/she is allocated to the appropriate cell as normal. If he/she uses more than one brand, we check to see which is the cell with the lowest quota ratio and whether one of the brands used corresponds to that cell. If so, the respondent is allocated to that cell. If the respondent does not use the brand with the low quota ratio, we then select one of his/her chosen brands at random and allocate him/her to that quota.

```

ibrand    ask 'Which brands of ice cream do you buy?'
          resp mp brands
ice      dummyask 'Dummy to store responses in unfilled cells'
          resp mp brands
          unset ice
          set nuse = numb(ibrand)

comment If more than one brand chosen, find cell with lowest ratio
comment and check if this brand was chosen. If so, quota on this brand.
        if (nuse = 1) {
          set ice = ibrand
        }
        else {
          set qsuf='a3'
          callfunc ('quorat',qsuf,cellno)
          set chosen = bit(ibrand/cellno)
          if (chosen) {
            set ice = cellno
            goto iquot
          }
          else {
comment Did not choose brand with lowest quota ratio, so choose
comment one at random from those chosen, and quota on that
11           for ipos = 1 to 6 ran
              set ranchs = bit(ibrand/ipos)
              if (ranchs) {
                set ice = ipos
                goto iquot
              }
            next
          }
}

```

```

        }
iquote quota ice file='a3' pass=oki
        route (.not. oki) go term
        ....
term    signal 5

```

Although this script does not show it, it would also be possible to check that the randomly selected brand did not belong to a cell that was already full. If the cell was full, we could try to select another brand before terminating the interview.

Requesting quota status information in the script



Quick Reference

To request quota status information in the script, type:

`callfunc('quoval', identifier, what_info, cell_num, result)`

where *identifier* is a variable containing the identifier of the quota file to read, *cell_num* is the cell number you wish to inspect, and *result* is the variable into which the cell value will be written. *what_info* defines the type of information required and is one of targets, refer, current, overquota, whenfull, and tempcells.

The *quotamap* and *quotacell* functions which we looked at earlier in this chapter report only on which cells are full and which still have room. **quoval** provides a wider view of the quota status, including:

- The cell target
- The individual targets which make up the target for a combined cell
- The number of completed interviews in a cell
- The number of respondents rejected because the cell was full
- The time (in seconds since 1 January 1970) when this cell became full
- The number of completed interviews in a cell, plus the number of interviews still in progress

The keywords associated with the various types of information are:

targets quota targets

refer cell number to which this quota cell is referred

current number of completes

overquota number of respondents rejected because the cell was full

whenfull time when the cell became full

tempcells number of completes plus interviews still in progress

As an example, assume you have a quota with the identifier a3, and you want to know how many completed interviews you have in each of the cells it contains:

```
set suffix = 'a3'
set info = 'current'
l1  for whichcel = 1 to 6
    set icetxt = elm(ibrand/whichcel)
    callfunc('quoval',suffix,info,whichcel,complete)
d1      display 'There are '+complete+' completes for people
who use buy '+icetxt+' ice cream. This is cell '+whichcel
    next
```

-
- ❖ *quoval* supersedes the callfuncs *quotacnt* and *quotaval*, which together performed this task in versions prior to v7. These callfuncs are still supported by Quancept, so scripts that use them will work as they always have. However, you are advised to use *quoval*, because the delay between taking the snapshot with *quotacnt* and inspecting the values with *quotaval* could mean that the values you see are already out of date.
-

Creating a quota log file for debugging purposes



Quick Reference

Set the QCQUOTALOG environment variable to 1 or 2, depending on how you want the logging information to be reported.

If you think that your quota control system is not working as you expected, and you cannot find the problem by placing additional debugging statements in the script, you may find the quota log file useful. This file reports the actions that qtip takes when it executes a *quota* statement. qtip creates it in the project directory and calls it *script.qdbxx* where *xx* is the quota identifier.

There are two actions for each *quota* statement. These are usually testing the quota cell that the respondent belongs to and then updating it so that the respondent is counted towards the target. Other possible actions are to reject the respondent if the cell is full, or to decrement the cell counts if the interview is stopped by a *stop* statement with the parameter *decrm=quota.identifier*, or if the interviewer snaps back to before the *quota* statement, or the interview is terminated with *quit* or *abandon* or because the respondent fails another quota in the script.

In order that you can track exactly what qtip is doing, it reports the action it is about to take and then confirms the action it has taken. At each stage the log file shows the cell number affected, the target for that cell, the number of completed interviews in that cell, the number of rejected (over quota) interviews in that cell, and the number of completed and in-progress interviews for that cell.

You can create the file in either of two formats. In the first format, which you request by setting QCQUOTALOG to 1, each action is reported over four lines. The first two lines report the action to be taken and the cell's status at that point. The second two lines report the action that has been taken and the cell's status after that action. Here is a typical example for an unfilled quota cell:

```
PID 24681-      TESTING cell 2
                  target 75 current 51 overquota 0 temp 51
PID 24681-      TESTED cell 2
                  target 75 current 51 overquota 0 temp 52
PID 24681      UPDATING cell 2
                  target 75 current 51 overquota 0 temp 52
PID 24681      UPDATED cell 2
                  target 75 current 52 overquota 0 temp 52
```

and here is the report for the same cell when it is full:

```
PID 24681-      TESTING cell 2
                  target 75 current 75 overquota 0 temp 75
PID 24681-      TESTED cell 2
                  target 75 current 75 overquota 0 temp 76
PID 24681      REJECTING cell 2
                  target 75 current 75 overquota 0 temp 76
PID 24681      REJECTED cell 2
                  target 75 current 75 overquota 1 temp 76
```

In both cases there was only one interview running for cell 2. The Unix process ID of the qtip that was running the interview was 24681.

☞ See ‘Defining quota targets’ in chapter 31, Quota maintenance, for further information about the cell values reported.

Although this file is easy to read, it is not suitable for loading into other programs and it does not give any indication of which interviewer conducted the interview. If you set QCQUOTALOG to 2, you can create a file containing one record for each event, as follows (the example splits records over two lines and indents the continuation lines):

```
01/02 18:59:24 (@)type=TESTING;pid=28026;line=AM;inum=2;user=ben;serial=10;
  cell=8;target=2;current=0;overq=0;temp=0
01/02 18:59:24 (@)type=TESTED;pid=28026;line=AM;inum=2;user=ben;serial=10;
  cell=8;target=2;current=0;overq=0;temp=1
01/02 19:00:03 (@)type=UPDATING;pid=28026;line=AM;inum=2;user=ben;serial=10;
  cell=8;target=2;current=0;overq=0;temp=1
01/02 19:00:03 (@)type=UPDATED;pid=28026;line=AM;inum=2;user=ben;serial=10;
  cell=8;target=2;current=1;overq=0;temp=1
```

Each record starts with the date and time at which the event took place and then contains the following fields:

Field name	Value
type	The action type.
pid	The qtip process ID.
line	The interviewer's line number.
inum	The interviewer's user number.
user	The interviewer's name.
serial	The interview serial number.
cell	The quota cell affected by the event.
target	The quota target for the cell.
current	The number of completed interviews for the cell.
overq	The number of over quota (rejected) interviews for the cell.
temp	The total number of completed and in-progress interviews for the cell.

A quota cell can be referred to another quota cell. This happens when you want to combine two cells to form a single quota.

☞ See 'Combining quota cells' in chapter 31, Quota maintenance, for details.

The quota logs take account of this by displaying the information for the original cell and the cell to which it has been referred. When QCQUOTALOG is set to 1, the report will show, for example:

```
PID 24681 -      TESTING ORIGINAL cell 7 REFERRED to cell 6
                  ORIGINAL target 20 current 0 overquota 0 temp 0
                  REFERRED target 30 current 0 overquota 0 temp 0
```

When QCQUOTALOG is set to 2, the report shows:

```
01/02 18:58:33 (@)type=TESTING;pid=28010;line=AM;inum=2;user=ben;serial=9;
cell=7;target=2;current=2;overq=0;temp=2;rcell=6;rtarget=4;rcurrent=4;
roverq=1;rtemp=4
```

30.5 How the mrInterview quota system works

- ■ The mrInterview quota system is based on a set of SQL tables that contain the quota definitions, targets, current cell counts, and other information, and quota statements in the script. Each quota has a unique name which you use in the script when you want to check that quota specifically. If you do not name the quota then the interviewing program checks all quotas for which data exists at the current point in the interview. This allows you maximum flexibility in the way you set up and test quotas.

Keeping track of quotas

While interviewing is in progress, mrInterview maintains three counts for each cell. The main one is the count of ***completed*** interviews in that cell — for example, the number of completed interviews with women or the number of completed interviews with people aged 18 to 24. If the number of completed interviews matches the target for that cell, the interview fails the quota and will be terminated.

The second count, known as the ***pending*** count, is the number of interviews currently in progress in a cell. When the interviewing program reaches a quota statement in the script, it checks which quota cells the respondent belongs to and, if the quotas have not been met, increments the pending counts for those cells by 1. If the interview later completes successfully (that is, is not stopped or does not time out), mrInterview decrements the pending counts by one and increments the corresponding completed counts by one.

If the interview belongs to cells whose targets have all been met, the interview fails quota control and is terminated.

If you are testing more than one quota at a time (that is, you have not named a quota on the quota statement), and a respondent passes one quota and fails another, the pending counts for the unfilled cells will be temporarily incremented by 1, and all counts for the full cells will remain untouched. Interviews of this type are usually terminated by quota control, although you can choose to flag them as completed interviews by routing to the end of the script or using *signal 1* or *signal 6* if you wish. If you terminate the interview by quota control, the relevant pending counts are decremented by 1, and the corresponding rollback counts are incremented. The corresponding completed counts remain unchanged. If you treat the interview as a successful completion, the pending counts are decremented and the completed counts are incremented for the quotas that did not fail. The counts for the quotas that were full remain unchanged.

Pending counts are incremented once per respondent, regardless of the number of times the quota statement is executed. If a respondent snaps back during an interview causing a quota statement to re-executed, the pending counts are not incremented when the statement is executed the second time.

The third count is the **rollback** count. This counts the number of times the pending counts have been rolled back to exclude interviews that have timed out, or that have been stopped, or that have been terminated because the interview failed another quota that appears later in the script. While the interview is running, the pending counts for the unfilled cells that the respondent belongs to are incremented by 1. When the interview terminates, those pending counts are decremented by 1 and the corresponding rollback counts are incremented by 1. The interviews are not completed so the completed counts remain unchanged.

The following example illustrates how the pending and rollback counts are maintained. The script is:

```

region ask 'Where do you live?'
resp sp 'North' ('North') / 'South' ('South') /
       'East' ('East') / 'West' ('West')
quota matrix='region' pass=ok1
if (.not. ok1) {
d1      display 'We have enough people who live in the '+region
        signal 5
}
gender ask 'Are you ...?'
resp sp 'Male' ('Male') / 'Female' ('Female')
quota matrix='gender' pass=ok2

```

After a period of interviewing the pending, completed and rollback counts are as follows. Notice that the target for women has been met so any subsequent interviews with women will be terminated.

Quota cell	Target	Pending	Completed	Rollback
North	25	0	22	0
South	25	0	23	0
East	25	0	19	0
West	25	0	23	0
Male	50	0	37	0
Female	50	0	50	0

The next interview is for a woman in the East. After the respondent has answered the region question, the cell counts are as follows:

Quota cell	Target	Pending	Completed	Rollback
North	25	0	22	0
South	25	0	23	0
East	25	1	19	0
West	25	0	23	0
Male	50	0	37	0
Female	50	0	50	0

Another interview starts, this time for a man in the East. When he has answered the region question, but before answering the gender question, the cell counts are as follows:

Quota cell	Target	Pending	Completed	Rollback
North	25	0	22	0
South	25	0	23	0
East	25	2	19	0
West	25	0	23	0
Male	50	0	37	0
Female	50	0	50	0

When the female respondent fails the gender quota, nothing happens to the gender counts, but the pending count for the East quota is decremented by 1 and the rollback counter for that cell is incremented by 1. The table of cell counts at the end of the interview is as follows:

Quota cell	Target	Pending	Completed	Rollback
North	25	0	22	0
South	25	0	23	0
East	25	1	19	1
West	25	0	23	0
Male	50	0	37	0
Female	50	0	50	0

When the male respondent finishes his interview, the pending count for East is decremented by 1 and the completed count is incremented by 1. The completed count for Male is also incremented:

Quota cell	Target	Pending	Completed	Rollback
North	25	0	22	0
South	25	0	23	0
East	25	0	20	1
West	25	0	23	0
Male	50	0	38	0
Female	50	0	50	0

Quotas for categorical responses

When a quota is based on single-punched responses, mrInterview increments the pending, completed and/or rollback counters for that response as described above. When a cell's completed count matches its target, all subsequent interviews in that cell will be terminated.

When a quota is based on multipunched responses, mrInterview increments the counters for all responses chosen by the respondent. When quotas start being filled, mrInterview terminates an interview if the respondent chooses responses in any of the full quota cells. The interview can continue only if all the multipunched responses chosen are in unfilled cells.

-
- ☞ Quotas based on questions with large numbers of multipunched responses makes quota testing very slow, particularly when a number of interviews are running concurrently.
-

Quotas for numeric and text responses

Expression Quotas is the term used for quotas based on numeric or text responses. (You can also use them for quotas based on specific combinations of characteristics, such as teenagers who have a weekend job and who visit the cinema at least twice a month.) You can define any number of Expression Quotas which may or may not be related to one another. For example, if you want to define quotas for a numeric question, you would define a set of expressions that group the possible responses into ranges so that you can set targets for each range. If you want to quota on a text question, you might define expressions that specify a word or words that must appear in the response text.

When mrInterview reads a quota statement for an Expression Quota, it increments the counters for each expression that matches the respondent's answer. With numeric response ranges, the ranges are usually the equivalent of single categorical responses, so only the counters for one range will be affected. With quotas based on text responses (or combinations of responses) it is possible that more than one set of counters may be incremented if the respondent satisfies more than one expression.

Quotas for null, dk, ref and specified other

For quota purposes, *null*, *dk*, *ref* and specified other are treated as ordinary categorical responses. When you set up quotas for a question that allows these responses, you may specify targets for these responses so that all interviews in a category will be terminated once the target has been met. If you do not want to quota on these responses, you can either set a very large target for each one, or set no quota at all, or you can accept the default target and flag these quota cells as being counter quotas only.

-
- ☞ See ‘Overquota and counter quotas’ for details.
-

Quotas for stopped or timed out interviews

When an interview times out or is stopped, any pending counts for that interview are rolled back and the corresponding rollback counts are incremented by 1. From a quota perspective it is therefore as if the interview never took place.

When an interview is restarted, the interviewing program replays the earlier part of the interview, filling in the original responses that the respondent chose. When the interviewing program encounters a quota statement, it tests the respondent’s answers against the current values in the quota table, increments the appropriate pending counters and either passes or fails the interview in the usual way. If the respondent belongs in cells whose targets have been met since the original portion of the interview took place, the interview will be terminated by quota control and any remaining answers from the original portion of the interview will be ignored.

-
- ☞ When an interview is restarted and the pending counts are re-incremented, the rollback counts that were incremented when the interview was stopped or timed out are not decremented. If the restarted interview becomes a completed interview, the rollback counts will still report that an interview was rolled back.
-

Overquota and counter quotas

Overquota and counter quotas are options that you can choose when setting up the quotas using the Quota Setup program rather than scripting features. However, they may affect the way the script behaves in certain instances.

When you define quotas, each cell is set up as a Normal quota. This means that as soon as the sum of pending and completed interviews in a cell matches the target, the next interview in that cell will fail quota control and will be terminated even if the other interviews are still running. For example, if the cell target is 50 and you have 47 completed interviews, you can have up to three interviews running that belong in that cell. If a fourth interview starts in that cell, it will be terminated when the quota is tested. If one of the pending interviews fails a later quota test, the pending counts are rolled back and you will be left with an unfilled cell where you could have reached the target.

Depending on how strict your quota requirements are, you may be able to get round this problem by allowing cells to go slightly *over quota*. This allows you to have any number of pending interviews in a cell, even if this means that the sum of pending and completed interviews exceeds the target. Once the target has been met, any interviews that are already pending for that cell will be allowed to complete, but any new interviews in that cell will be terminated. For example, if the cell target is 50 and there are 49 completed interviews and two pending, and one of the pending interviews completes, no new interviews will be allowed in that cell but the other pending interview will be allowed to complete, giving you a total of 51 completed interviews.

Counter quotas never fail; they simply count the number of interviews achieved in a cell. You might want to use this facility for *null*, *dk*, *ref*, and specified other.

Quotas in projects that use Sample Management

When a project uses Sample Management, quota control can be based on sample data, questionnaire data, or a combination of the two. In fact, you can define dependent quotas where some of the characteristics come from the sample and others come from the questionnaire. For example, if the sample records contain personal data such as age and gender and the questionnaire asks which make of car the respondent owns, you can set up quotas for men who own Porsches or young people who own sports cars.

-
- ☞ Refer to the following sections of the online *mrInterview User's Guide* for further information about Sample Management scripts. See 'How to Write A Quota Control System' to find out how to write Sample Management scripts that implement quota control. See the notes for the `basic_with_quotas`, `basic_quota_pend`, `qmster`, and `qmastersmq` example projects in 'Standard Scripts' for information on the sample scripts supplied with mrInterview.
-

Quotas based solely on sample data

When a quota is based solely on sample data — in the example, only on age or gender — and the quota is to be tested before respondents start the questionnaire, you should do the following:

- Define the quota groups and targets using Quota Setup.
- Write a Sample Management script (sam file) that tests the quotas and either starts the questionnaire or terminates the interview depending on whether the quota target has been met.

There is no need to use a *quota* statement in the script because quota control takes place completely outside the script.

-
- ❖ The Sample Management script must deal with everything that the *quota* statement does for script-based quotas. This includes not only deciding whether to terminate the interview when the sample-based quotas are exceeded, but also specifying what must happen when an interview is stopped or times out. You must also deal with incrementing or decrementing the associated completed, pending and rollback counts at the appropriate points, either by placing the necessary statements in the ReturnSampleRec function in the Sample Management script, or by setting the QuotaAutoCommit project variable to 1 using the Project Editor.
 - ☞ Refer to ‘Processing Records Returned After Interviewing’ and ‘Adding Project Properties’ in the online *mrInterview User’s Guide* for details. You will also find examples of how to write these sorts of scripts in the Sample Management section of this guide.
-

Quotas based solely on questionnaire data

When a quota is based solely on questionnaire data — in the example, only on car ownership — all the quota control takes place once the questionnaire has started, so you should do the following:

- Define the quota groups and targets using Quota Setup
- Place one or more *quota* statements in the script at the points at which you want to test quotas.
- If the project has a Sample Management script, increment or decrement the associated completed, pending and rollback counts at the appropriate points. To do this, either place the necessary statements in the ReturnSampleRec function in the Sample Management script, or set the QuotaAutoCommit project variable to 1 using the Project Editor.

-
- ☞ See ‘Processing Records Returned After Interviewing’ and ‘Adding Project Properties’ in the online *mrInterview User’s Guide* for details. (If the project does not use Sample Management, then you omit this step.)
-

Quotas based on a combination of sample and questionnaire data

When quota control is based on a combination of sample and questionnaire data — in the example, men who drive Porsches or young people who drive sports cars — you should do the following:

- Import the sample data that is to be used with the questionnaire data into the script using *getsmvar*. This makes the sample data available in script variables.
- Define the quota groups and targets using Quota Setup. Use the script variables that you used with *getsmvar* to access the sample data in Quota Setup. (You will not be able to use the original sample variables in the same quota as questionnaire variables.)
- Place one or more *quota* statements in the script at the point you want to test quotas.

- Write a Sample Management script that increments the complete or rollback quota counts and decrements the pending counts once the sample record has been returned from the interview. You can either place the necessary statements in the ReturnSampleRec function in the Sample Management script, or set the QuotaAutoCommit project variable to 1 using the Project Editor.

☞ See ‘Processing Records Returned After Interviewing’ and ‘Adding Project Properties’ in the online *mrInterview User’s Guide* for details.

Using more than one quota type in the same script

You can use more than one of these methods in any project. For example, you can use sample-based quota control to restrict the number of people taking the survey, and then use script-based quota control to restrict the number of respondents giving certain answers to questions in the script. In this scenario, you must follow the rules for both types of quotas. In other words, you must write a Sample Management script to process the sample-based quotas and include *quota* statements in the script to test the script-based quotas. You must also use Quota Setup to define all the quota groups and targets.

☞ Refer to the Sample Management section of the online *mrInterview User’s Guide* and to the example Sample Management scripts for information about the Sample Management system.

Additional functionality with Sample Management-based quota control

As noted earlier, the Sample Management script controls everything to do with how sample-based quota control works. This gives you flexibility and options that are not available for script-based quotas. For example, suppose you have a project where you want to obtain 100 interviews for each of five different soft drinks. You want to make the best use of your sample, so do not want to reject a respondent just because you already have 100 interviews for one of the chosen soft drinks. You want to be able to ignore that drink and allocate the respondent to one of the cells for a chosen drink in one of the unfilled cells, or perhaps even in the cell that is least full. If the respondent drinks fizzy orange, lemonade and mineral water, for example, and the mineral water quota is full, you can allocate the respondent to the fizzy orange or lemonade categories, whichever is the least full.

Quota prioritization for multiple response categorical questions

When a quotas are defined for multiple response categorical questions, the quotas for all responses chosen by the respondent are pended. If the interview is completed, then the corresponding complete counts are incremented. When a respondent chooses more than one response, this results in more than one quota cell being incremented. Quota prioritization lets you decide how many cells to pend, and allows you to choose the method by which those cells are chosen. There are three possibilities.

- **Priority Pend.** Pend the cell with the highest priority, if this response is chosen. If this response is not chosen, try the cell with the next highest priority, and so on. Cells with no priority defined have the lowest priority of all.
- **Random Pend.** Pend a cell at random from the cells for the responses that have been chosen.
- **LeastFull Pend.** Pend the cell that is least full from the cells for the responses that have been chosen. If an interview is restarted after timing out, the LeastFull cell is recalculated using the latest quota information. This may mean that a different cell is pended in the restarted interview to the one that was pended when the interview was originally started. If the questionnaire has routing based on the pended cell, the path through the interview may change once the completed portion of the interview has been replayed.

You specify the method you want to use by editing the QUOTA_Matrix table in the project's quota database.

☞ See ‘The QUOTA_Matrix Table’ in the online *mrInterview User’s Guide* for information about this table and technical details about how prioritization works.

The following steps summarize how to set up a project with quota prioritization.

1. Parse and compile the questionnaire. Do not activate yet.
2. Open QCompile and click Quota to start Quota Setup using the project’s mdd file.
3. Define the quotas. Save your changes and close the program.
4. Activate the project, choosing the option to create a new quota database.
5. Start SQL Enterprise Manager and expand the tree in the left-hand frame to show *machine_name*, Databases, *project_name*, Tables.
6. Right-click the QUOTA_Matrix table and select Return all rows.

The frame on the right will display one line for each quota matrix you defined using Quota Setup.

7. For each matrix that requires prioritization, enter one of the following values in the PendMode column: 1 for Priority Pend, 2 for Random Pend, or 3 for LeastFull Pend.
8. For each matrix that has prioritization set, enter the maximum number of cells that can be pended for that matrix. If the number of cells is unset or is set to zero, all cells will be pended.
9. Close the QUOTA_Matrix table.
10. If you requested Priority Pending, right-click on the QUOTA_Quota table and select Return all rows.

You will see one row for every quota cell in the project displayed in the right-hand frame.

11. For each cell that is to be Priority Pended, in the MetaData column enter the cell's priority relative to the other cells in the same matrix.
12. Close the QUOTA_Quota table.

Sharing quota files between projects

Sometimes you may have a set of projects that share the same quotas. This typically happens when you have sample-based quotas and the Sample Management script allocates respondents to projects based on information in the sample records or gathered on the authentication page or by a screening script. For example, suppose you have two questionnaires, one about winter holidays and the other about summer holidays. A screening script gathers information about age, gender, and the time of year when the respondent takes his/her main holiday. You want to obtain interviews with 500 women in two age groups and 500 men in two age groups across the two projects.

You can vary this approach by having additional script-based quotas within the individual questionnaire, or by having script-based quotas only so that any number of respondents may take each survey.

To implement any form of quota control where a Sample Management script determines which survey a respondent will take, do the following:

1. Create a master project with a dummy questionnaire script and a Sample Management script. The dummy questionnaire must contain copies of all the quota-controlled questions in the projects to which respondents may be routed. We call these projects *slave* projects. If there are no script-based quotas in any of the slave projects, just create a master script containing a dummy question.
2. Use Quota Setup to define all the quotas for the master and the slave projects. This includes the master project's Sample Management quotas as well as any script-based quotas required for any of the slave projects. Save the quotas using the name of the master project.
3. Create the questionnaires for the slave projects in subdirectories within the master project's source directory. To test quotas, write *quota* statements that use to the matrix names defined in the master project's mqd file.
4. Activate the master project and create a quota database, as for a standard project.
5. Activate the slave projects and choose the master project's quota database as the source of quota information for each project.

When interviewing starts, the Sample Management script will allocate the respondent to a project. The questionnaire script for that project will run, but all quotas will be controlled by the quota tables in the master project's database.

-
- ☞ See ‘Quota tab’ in chapter 32, Parsing and compiling scripts, for further information about choosing quota databases using the activation wizard.
- Refer to the example projects in c:\mrInterviewSource\Samples\Quota for runnable examples of shared quotas.
-

30.6 Checking quotas in mrInterview



Quick Reference

To check quotas, write a **quota** statement that optionally names the quota to be checked. For Table Quotas, type:

```
quota [matrix='quota_name']
```

For Expression Quotas, type:

```
quota [matrix='Expressions[:quota_name]']
```

If you do not name a quota, all quotas for which data currently exists will be checked.

When you define quota groups and targets using Quota Setup, you can define quotas as tables or expressions. Normally, you'll use tables for quotas based on categorical (single-punched or multipunched) responses and expressions for quotas based on numeric, text or boolean variables.

Each table has a unique name which you can specify on the *quota* statement to test just the quotas defined in that table. For example, a table that defines dependent quotas for age and gender will have cells for male respondents aged 21 to 24, female respondents aged 21 to 24, and so on. To test just the quota defined in this table you would type:

```
quota matrix='agegender'
```

Quotas defined in other tables or in expressions are ignored.

With Expression Quotas, you can define any number of expressions for any number of variables. For example, you might define a series of expressions that group the various ages into bands such as 18–24, 25–34, and so on, and define targets for each band (expression). You might also define quotas for respondents who mention the word ‘expensive’ when giving their opinions about a product.

To test Expression Quotas, type:

```
quota [matrix='Expressions[:quota_name]']
```

For example:

```
quota matrix='Expressions:prodexpens'
```

to test only the quota for people who said the product was expensive, or:

```
quota matrix='Expressions'
```

to check the age and product quotas but not any quotas that are defined as tables.

-
- ☞ If you are testing a named Expression Quota, do not type spaces before or after the name as these will be included as part of the name and the test will fail.
-

If you want to check all quotas for which data currently exists, type **quota** by itself. If a quota exists for a question that has not yet been asked, or the respondent skips a question that has quotas associated with it, there is no data available for that question so the quotas for that question are not tested.

-
- ☞ See section 31.4, The mrInterview quota program, for more detailed information about Quota Setup, quota tables and quota expressions.
-

If you have a *quota* statement that tests a named matrix followed by one that tests all matrices, the pending counts for the cells that the respondent belongs to are only incremented once, even though one quota will be tested twice. For example:

```
age      ask ...
       resp sp ...
       quota matrix='age'
gender   ask ...
       resp sp ...
       quota
```

Here, the first *quota* statement increments the pending count for the respondent's age category. The second quota statement tests quotas for all data that exists, namely age and gender. However, because age already has a pending count for this respondent, it is ignored and only the pending count for gender is incremented.

-
- ☞ If the script has many quotas, and particularly if some of those quotas are based on multipunched variables, using *quota* with no matrix name can make quota testing very slow. If possible, always name the quotas you want to test.
-

Controlling how interviews terminate



Quick Reference

To intercept the termination command that the quota system sends to the interviewing program when the respondent belongs in a full quota category, type:

quota [matrix='quota_name'] pass=variable

variable will be set to false if the respondent fails the quota or true if he/she passes the quota. If the statement names a non-existent quota, *variable* will be null.

When a respondent falls into a full quota cell, the interview is normally terminated immediately and the respondent sees a message informing him/her that the interview was terminated by quota control. The **pass** parameter is an optional means of intercepting this message so that you can choose what to do next.

pass names a variable which the interviewing program sets to true if the respondent does not exceed the quotas or false if he/she exceeds the quotas or the named quota does not exist. Your script will then include statements which check the value of this variable and route to various points in the script according to the requirements of the survey. Using this method, it is still possible to continue the interview even though a quota is exceeded. For example, if the *pass* variable is true, you may want the respondent to continue with the next question, whereas if it is false, you may want to route the respondent to the demographic questions at the end of the script before flagging it as terminated by quota control. If the quota statement is skipped, the *pass* variable will be null.

Let's look at an example.

```

age      ask 'How old are you?'
        resp sp 'under 21' ('u21') / '21 to 35' ('e2135') /
                  'over 35' ('o35')
gender   ask 'Are you ...'
        resp sp 'male' ('male') / 'female' ('female')
        quota matrix='agegender' pass=ok
        if (pass=null) {
qerr     display 'Quota failed due to error.'
        goto exit
        }
        if (.not. ok) {
comment Respondents whose quota cell is full
thank    display 'Thank you for your help, but we have already
completed our quota of interviews for '+gender+'s aged '+age+'.'
        signal 5
        goto exit
        }
comment Respondents in unfilled cells come here
ctn     continue

```

The quota targets for this example have been set up as a two-dimensional table of age by gender in which each cell represents one combination of age and gender. The quota table or *matrix* is called agegender and is based on the responses to the age and gender questions in the script.

The quota statement defines a variable (called ok) which is to normally be set to true or false depending on whether the respondent fails the quota test. If the quota is full, the script displays a message explaining how the interview is being terminated. Since we do not want such interviews to be flagged as successfully completed (and thus muddled with successfully completed interviews where quotas were not exceeded), we use the *signal* command to force the interview to be flagged as terminated by quota control.

-
- ☞ For further information about the *signal* statement, see section 16.4, Forced termination of an interview.
-

If the quota check fails for a reason other than the fact that the quota cell is full, the ok variable will be set to null. In this case, check that the name on the *quota* statement matches the name in the mqd file. If this is not the problem, check the ISE log file for other messages.

Checking for missing quota specifications



Quick Reference

To check that the quota you are testing exists in the quota database, place the following statements after each *quota* statement that identifies a quota by name:

```
if (quotrval < 0) {  
label    display ['text'+]quotrtxt  
}
```

When you name a quota on the *quota* statement, the quota control system looks for this quota and no others. If a quota with that name exists, the quota control system compares the respondent's answer against the targets and, as long as the targets have not been met, makes the appropriate adjustments to the pending counts.

If you name a quota that does not exist (perhaps you mistype the name), the error message 'Unable to pend quota for Group *project:quota_name*. group may not exist: Invalid index' appears in the ISE log file, but the interview continues as if the respondent had passed the quota test.

This type of error is easy to miss, but if you add the following statements to the script after each quota statement that tests a named quota, you will see the messages displayed on the screen:

```
if (quotrval < 0) {  
label    display ['text'+]quotrtxt  
}
```

quotrval is a variable that mrInterview sets equal to the value returned from the quota control system. If there are problems, this value is always negative. *quotrtxt* is a description of the error and matches the text at the end of the error message in the ISE file.

In the following example, the quota has been defined as *agegender*, but the script refers to it as *genderage*:

```

age      ask ...
gender ask ...
    quota matrix='genderage' pass=ok
comment Check for missing quota specifications.
    if (quotrval < 0) {
qerr      display 'Error at genderage check: '+quotrtxt
    }
    if (.not. ok) {
qterm     display 'Thank you for your help, but we already have enough '
+age+ year old '+gender+s.'
        signal 5
    }

```

The error text generated for a missing quota specification is ‘Invalid index’, and this will be substituted in the more descriptive message defined in the script.

-
- ❖ This facility does not check that a quota cell exists for the respondent within the named quota. For example, if you have defined quotas for a numeric age variable, and do not have a quota for people aged over 60, anyone who is aged over 60 will always pass the quota test. There is no requirement to define quotas for all responses that a question may have, although for completeness you may prefer to use counter quotas for groups that have no quota requirements.
-

The table below summarizes the values that *pass=*, *quotrval* and *quotrtxt* can return:

pass=	quotrval	quotrtxt	Meaning
1	0	The operation completed successfully.	The quota exists and is unfilled.
0	1	Incorrect function.	The quota exists but is full.
null	negative	Invalid index.	The quota does not exist.

Working with quota controlled projects in DimensionNet

When you work on a project that uses Quota Control in DimensionNet, you still need to set up the quotas using the Quota Setup program on your desktop. Quota Setup uses the project’s mdd file to obtain information about the variables in the questionnaire, and creates an mqd file containing information that the activation procedure needs for creating the quota database.

Because you will be using a combination of browser based and desktop applications, it is important that you have the correct files available in the folders in which the browser based and desktop applications expect to find them. This requires you to copy files between folders at specific points in the project's development cycle. If you follow the steps below you should always have the right files in the right place at the right time.

1. Create the project in DimensionNet.

This creates a project folder in your Users folder. This is where all DimensionNet applications look for and create files.

2. Create the Quancept script using DimensionNet applications. You can either use Build to create an mdd file (placing the *quota* statement in a QCScript item) and then the Quancept activity to generate a Quancept script from that file, or you can type the script in the Quancept language directly into the Quancept activity, or you can upload an existing qqc file into the Quancept activity.
3. If necessary, upload any other files that the script uses, such as include files or presentation templates, using the Quancept activity or Files.
4. Use the Quancept activity to parse and compile the questionnaire. If you typed the script manually or you uploaded an existing qqc file, the parser will create a matching mdd file.
5. In DimensionNet, use Files to download the project's mdd file into a folder on the machine on which you will be running Quota Setup. (It is a good idea to create a project folder in c:\mrInterviewSource as this is where desktop applications normally look for files.)
6. Use Quota Setup with the mdd file you have just downloaded to define the quota categories and the targets for each category.
7. In DimensionNet, use Files to upload the mqd file that Quota Setup created. If your quotas are based on Sample Management data, you must also upload the mdd file that Quota Setup created for these variables. The upload process will ask whether you want to rename this file so that it matches the project name. **Do not** do this as this will cause the activation process to fail. You must keep the filename exactly as Quota Setup created it, so click Cancel at this point and then close Files.
8. If the project uses sample, use UploadSample in DimensionNet to copy the sample records into a sample database.
9. In DimensionNet, use Launch to activate the project. The first time you activate the project, the activation process will create a quota database for the project using the information in the mqd file. If you change the mqd file and you want to update the quota database, you must re-activate and select the option for synchronizing the quota database with the mqd file.

30.7 Sample scripts

The scripts for Quancept CATI and Quancept Web need three sets of quota files. The first, a1, is for age and gender and requires seven cells (the cell for no suitable respondent available should be set to 0); the second, a2, is for yogurt; and the third, a3, is for ice cream. The quotas for yogurt and ice cream have six cells each and are based on the same brand list.

Script A

```

 comment Sample script A for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 30, Quota control (CATI)

comment Take a snapshot of the quota table
      set sufxt = 'a1'
      callfunc ('quotamap',sufxt,6)
cats    define 'men under 21' / 'men 21 to 35' / 'men over 35' /
          'women under 21' / 'women 21 to 35' /
          'women over 35'
qcats   dummyask 'Dummy to store unfilled quota categories'
resp mp cats
qdum    dummyask 'Dummy to store age/gender of respondent for quota-ing'
resp sp cats
unset qcats
unset qdum
comment Check each quota cell in turn. full is true if cell is full
11      for cnum = 1 to 6
          callfunc('quotacell',cnum,full)
          route (full) go back
          set qcats = cnum
back    next
        if (qcats=null) {
          display 'INTERVIEWER: All quota cells are full.
Please speak to your supervisor'
          pause
          goto term
        }
agegend ask 'INTERVIEWER: Which respondent are you interviewing?'
resp sp qcats in cats
privsig 1
quota agegend file='a1' pass=ok
route (.not. ok) go term

comment Choose whether to ask about ice cream or yogurt
12      for iy = 1 to 2 ran
          route(iy=1) go yogctn
          route(iy=2) go icectn
next

```

Quancept and mrInterview Scriptwriter's Manual

```
comment Yogurt section
yogctn    continue
brands     define  'alpine' / 'helen bradley''s' / 'dairy fresh' /
            'kentish farm' / 'finborough' / 'alsace'
ybrand    ask 'Which brands of yogurt do you buy?'
resp mp brands
yog       dummyask 'Dummy to store responses in unfilled cells'
resp mp brands
unset yog

comment See which cells are full
chk       set ncells = 6
          set suf = 'a2'
          callfunc('quotamap',suf,ncells)
          set emptyq = 0
13        for respnum = 1 to 6
          set given = bit(ybrand/respnum)
          callfunc('quotacell',respnum,full)
          set notfull = logical(.not. full)
          if (given .and. notfull) {
              set yog = respnum
              set emptyq = 1
          }
next
duf      display 'Responses in unfilled quotas cells are: '+yog
pause
privsig 2

comment Reject respondents with all answers in full cells and
comment quota on responses in unfilled cells
if (emptyq = 0) {
close      display 'We already have enough interviews for those
brands, but thank you for your help anyway.'
pause
goto exit
}
quota yog file='a2' pass=oky
route (.not. oky) go term

yloop    for ibrd = ybrand in brands
yflav    ask 'Which flavors do you usually buy in '+ibrd+'?'
          resp mp 'apricot and nectarine' / 'banana' /
                  'black cherry' / 'strawberry' / 'raspberry' /
                  'melon and passion fruit' / 'pineapple' /
                  'forest fruits' / 'apricot' / 'orange' other
next
goto exit
```

```

comment Ice cream section
icectn  continue
ibrand   ask 'Which brands of ice cream do you buy?'
         resp mp brands
ice      dummyask 'Dummy to store responses in unfilled cells'
         resp mp brands
         unset ice
         set nuse = numb(ibrand)

comment If more than one brand chosen, find cell with lowest ratio
comment and check if this brand was chosen. If so, quota on this
comment brand.
        if (nuse = 1) {
            unset ice
            set ice = ibrand
            display 'ice is '+ice
            pause
        }
        else {
            set qsuf='a3'
            callfunc ('quorat',qsuf,cellno)
            set chosen = bit(ibrand/cellno)
            if (chosen) {
                set ice = cellno
                goto iquot
            }
            else {
comment Did not choose brand with lowest quota ratio, so choose
c4mment one at random from those chosen, and quota on that
15           for ipos = 1 to 6 ran
                set ranchs = bit(ibrand/ipos)
                if (ranchs) {
                    set ice = ipos
                    goto iquot
                }
                next
            }
        }
iquot   quota ice file='a3' pass=oki
        route (.not. oki) go term
        if (.not. oki) {
            display 'Our quotas for ice cream are full. Thank
you for your help.'
            goto term
        }

iflav   ask 'Which is your favorite flavor in the '+ice+
brand of ice cream?
        resp sp 'chocolate' / 'mint' / 'toffee crunch' /
               'raspberry ripple' / 'strawberry' / 'vanilla' other
        set suffix = 'a3'

```

```

        set info = 'current'
15      for whichcel = 1 to 6
            set icetxt = elm(ibrand/whichcel)
            callfunc('quoval',suffix,info,whichcel,complete)
dcomp      display 'There are '+complete+' completes for people
who use buy '+icetxt+' ice cream. This is cell '+whichcel
            next
            goto exit

term      signal 5
exit      end

```

Script B



```

comment Sample script B for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 30, Quota control (Web)

comment Take a snapshot of the quota table
        set sufxt = 'a1'
agegend  ask 'Which category do you belong to?'
        resp sp 'man under 21' / 'man 21 to 35' / 'man over 35' /
                'woman under 21' / 'woman 21 to 35' / 'woman over 35'
        quota agegend file='a1' pass=ok
        route (.not. ok) go term

comment Choose whether to ask about ice cream or yogurt
11       for iy = 1 to 2 ran
            route(iy=1) go yogctn
            route(iy=2) go icectn
        next

comment Yogurt section
yogctn   continue
brands    define 'alpine' / 'helen bradley''s' / 'dairy fresh' /
                'kentish farm' / 'finborough' / 'alsace'
ybrand   ask 'Which brands of yogurt do you buy?'
        resp mp brands
        quota ybrand file='a2' pass=oky
        route (.not. oky) go term

yloop    for ibrd = ybrand in brands
yflav    ask 'Which flavors do you usually buy in '+ybrand+'?'
        resp mp 'apricot and nectarine' / 'banana' /
                'black cherry' / 'strawberry' / 'raspberry' /
                'melon and passion fruit' / 'pineapple' /
                'forest fruits' / 'apricot' / 'orange' other
        next
        goto exit

```

```
comment Ice cream section
icectn  continue
ibrand   ask 'Which brands of ice cream do you buy?'
         resp mp brands
iquot    quota ibrand file='a3' pass=oki
         route (.not. oki) go term

iflav    ask 'Which is your favorite flavor in the '+ibrand+
brand of ice cream?
         resp sp 'chocolate' / 'mint' / 'toffee crunch' /
               'raspberry ripple' / 'strawberry' / 'vanilla' other
         goto exit

term    display 'Interview terminated by quota control.'
        signal 5
exit    end
```

Script C

```
comment Sample script C for Quancept & mrInterview Scriptwriter's Manual
comment Chapter 30, Quota control (mrInterview)

agegend  ask 'Which category do you belong to?'
         resp sp 'man under 21' ('mu21') / 'man 21 to 35' ('m2135') /
               'man over 35' ('mo35') / 'woman under 21' ('wu21')/
               'woman 21 to 35' ('w2135') / 'woman over 35' ('wo35')
         quota matrix='agegend' pass=ok
         route (.not. ok) go term

comment Choose whether to ask about ice cream or yogurt
11       for iy = 1 to 2 ran
         route(iy=1) go yogctn
         route(iy=2) go icectn
         next

comment Yogurt section
yogctn  continue
brands   define 'alpine' ('alpine') / 'helen bradley''s' ('helenb') /
         'dairy fresh' ('dfresh') / 'kentish farm' ('kentish') /
         'finborough' ('finb') / 'alsace' ('alsace')
ybrand   ask 'Which brands of yogurt do you buy?'
         resp mp brands
         quota matrix='ybrand' pass=oky
         route (.not. oky) go term
```

Quancept and mrInterview Scriptwriter's Manual

```
yloop    for ibrd = ybrand in brands
yflav      ask 'Which flavors do you usually buy in '+ybrand+'?'
            resp mp 'apricot and nectarine' ('ap-nect') /
                  'banana' ('banana') / 'black cherry' ('bcherry') /
                  'strawberry' ('strawberry') / 'raspberry' ('raspberry') /
                  'melon and passion fruit' ('mel-pas') /
                  'pineapple' ('pineapple') / 'forest fruits' ('forest') /
                  'apricot' ('apricot') / 'orange' ('orange') other
            next
            goto exit

comment Ice cream section
icectn   continue
ibrand    ask 'Which brands of ice cream do you buy?'
            resp mp brands
iquot     quota matrix='ibrand' pass=oki
            route (.not. oki) go term

iflav     ask 'Which is your favorite flavor in the '+ibrand+
          ' brand of ice cream?'
            resp sp 'chocolate' ('chocolate') / 'mint' ('mint') /
                  'toffee crunch' ('toffee') / 'raspberry ripple' ('ripple') /
                  'strawberry' ('strawberry') / 'vanilla' ('vanilla') other
            goto exit

term      display 'Interview terminated by quota control.'
            signal 5
exit       end
```

31 Quota maintenance

This chapter introduces the Quancept CATI, Quancept Web and mrInterview quota programs which you use for defining and maintaining quotas. All programs allow you to enter quota targets and descriptions, and to change targets as required. The Quancept CATI and Quancept Web programs also allow you to inspect quota cell status and change cell counts, and the Quancept CATI quota program allows you to merge cell targets.

31.1 Quota files



Quota files are the link between the *quota* statement in the script and the internal workings of the quota control program.

In Quancept CATI there are two files for each quota: one which stores the quotas themselves, and another which stores descriptions of the quota cells (for example, response texts). The file that stores the quotas has the extension *quotxx*, where *xx* is a two-character identifier that links the file to a specific quota statement in the script. The file that stores the quota descriptions is suffixed *qtxx*. For example, the quota files for the *a1* quota in the *cars* survey are named *cars.qota1* and *cars.qtta1*.

Quancept Web has a single file that contains quota counts and quota texts. The file has the extension *quo* and refers to all quotas in the script.

-
- » When you create these files, remember that the interviewing program expects there to be some correlation between the information in the quota files and the information on the quota statement.
-

The links between the quota-controlled responses and the quotas in the quota files are:

- the order of questions on the quota statement
- the order of responses in a response list
- the order of quotas in the quota files

If these orders differ, the interviewing program will not be able to check your quotas correctly. If you are setting up quotas before writing the script, you must make sure that you write the quota statements in the script so that they match the order of information in the quota files. However, if you have already written the script, you must take great care that you set up the quotas to match the script.

The interviewing program matches the answer the respondent gives with the quota counts in the file by taking the position of the response in the list and reading the value in the corresponding position in the quota file. For example, if female is the second response in the gender response list, the interviewing program assumes that the female quota is the second category in the file. The quotas for the question:

```
gender ask 'Is the respondent ...'  
resp sp 'male' / 'female'
```

must be defined with male as the first category and female as the second. If you define the responses the other way around — that is, female as the first category and male as the second — the interviewing program will decrement the female quota whenever a man is interviewed and the male quota whenever a woman is interviewed.

With dependent quotas involving two or more questions, the quota dimensions must be defined in the same order as the questions are named on the quota statement. If the quota statement names the first question as gender and the second as mstat, you must define the categories for gender before those for marital status. Within each category, the responses must be entered in the same order as they appear in the response lists. You need not concern yourself with the intersection of two or more individual categories; quota deals with this internally and will prompt you accordingly.

☞ For information on quotas in scripts see chapter 30, Quota control.

31.2 The Quancept CATI quota program



The Quancept CATI quota program is called quota.

-
- ☞ Certain Unix systems provide a disk space control program called quota. If the Quancept CATI quota program does not respond as described in this chapter, it is possible that your path is set up in such a way that you are picking up the Unix disk space control program rather than the Quancept quota program. To check whether this is the case, type, for example:

```
$QCHOME/bin/quota cars a1
```

If this starts the Quancept quota program, then the problem lies in the order of directories in your path. To avoid this problem, define a system alias so that typing quota picks up the Quancept program automatically:

```
alias quota $QCHOME/bin/quota
```

Setting up quotas for a new project



Quick Reference

To set up quotas on a new project or to work on existing quotas, use the Quancept quota program:

```
quota study_name identifier
```

where *identifier* is the two-character code that links the quotas you are about to define with the relevant quota statement in the script.

You can use any characters other than slash and backslash (/ and \) as the identifier, but note that the interviewing program does not differentiate between upper and lowercase letters (that is, files suffixed AA will be matched by quota statements with 'AA' or 'aa' identifiers).

If there are no qot/qtt files with the given identifier, a message to this effect is displayed, and you are asked whether you wish to create them. If you answer 'yes', Quancept asks for the number of axes and the number of elements for each axis. By this it means how many questions are there in this quota, and how many responses are there for each of those questions. For example, a quota for the questions:

```
gender    ask 'Is the respondent ...'  
          resp sp 'male' / 'female'  
mstat     ask 'What is your marital status?'  
          resp sp 'single' / 'married' / 'widowed' / 'divorced'
```

has two axes, gender and mstat. The gender axis has two elements, and the mstat axis has four elements.

If one of the questions is numeric, the number of elements depends on the number of numbers or ranges in the response list. For example:

```
hhsizs   ask 'How many people are there in your household?'  
         resp num 1 to 10
```

is treated as one element because there is one range, whereas:

```
hhsizs   ask 'How many people are there in your household?'  
         resp num 1 / 2 / 3 to 4 / 5 to 10
```

is treated as four elements because there are four sets of values in the response list. It does not matter that some of them are single numbers and others are ranges.

The quota program then displays a menu listing the options available:

- 1 - Current Status Report
- 2 - Cells Sorted Starting with LEAST Full
- 3 - Enter New Targets
- 4 - Enter New Combinations

- 5 - Enter New Texts
- 6 - Examine/change Individual Cell
- 7 - Cells Sorted Starting with MOST Full
- 8 - Change Current Values
- 9 - Quit
- 10 - Remove a Hung Temp Entry

The options you will use for setting up new quotas are 3, Enter New Targets, and 5, Enter New Texts. You may use these in any order, but because the quota program displays the texts for each quota when prompting for targets, we recommend that you select option 5 first. If you start with option 3, there will be no texts displayed, and it may not be clear for which quotas you are entering targets.

Defining quota texts

To define quota cell texts, select option 5 from the menu. The quota program asks whether you wish to change texts for the current dimension (on a new survey, the dimension name will be printed as asterisks as it has yet to be defined). Answer *y* and then enter the dimension title (question name) and the row titles (responses) when prompted to do so. Here is an example of setting up a new quota for gender:

```
Do you wish to change texts for dimension ***** ?  
Y  
Enter new texts or <rtn> to leave unchanged  
Title -  
New Title - gender  
Row 1 -  
New Row 1 - male  
Row 2 -  
New Row 2 - female
```

If you have already written the script, make sure that you enter row texts in the order that the responses appear in the response list. As long as the order is the same, it does not matter whether the texts match the responses in the script exactly. In our example, male is the first response in the response list for the question called gender, and female is the second, but we could have typed men and women as the row titles if we wanted.

-
- ¶ Do not enter quotas for null, dk, or ref responses. The quota program will allow you to define quotas for them (it just sees them as texts and numbers), but the interviewing program will not work properly when it reaches the quota statement in the script.
-

If you have a multidimensional quota, you will be prompted to deal with the second and subsequent dimensions as soon as you have finished with the first.

When all texts have been defined, the program checks whether to store the texts. Answer '*y*' if you want the texts to be stored or '*n*' if you do not.

Defining quota targets

Targets for an independent quota (one question only) are defined by entering the target value for a particular category when prompted to do so (for example, by typing 50 at the prompt for the target for male respondents). With a dependent quota (more than one question), the targets refer to the intersection of one category from each of the quota dimensions. In a dependent quota of gender and marital status, for example, you will be asked to enter quotas for single men, single women, married men, and so on.

To define quota targets, select option 3, Enter New Targets, from the menu. The program displays the cell number and description for the first cell and prompts you to enter the target:

```
Cell number 1 as at Thu May 29 10:35:00 1986
gender - male
marital status - single
target - 0    completed - 0    rejected - 0    temp - 0
Enter new target, <rtn> for Unchanged, q for no more cells - 50
```

Type the number of completed interviews you want in this category and press Return. Quota displays the prompt for the next cell and waits for you to enter the target. It continues doing this until you have entered a target for each cell. If you do not want to set a target for a category, do not leave it blank; instead, enter a large number which is more than the number of interviews you expect to conduct.

When you have dealt with all the quota cells, you are asked whether to store the figures. Answer 'y' to save the targets or 'n' to ignore them.

As the previous example shows, the quota program reports four values for each cell. These are:

target	the number of interviews required in this cell
completed	the number of completed interviews obtained in this cell so far
rejected	the number of interviews rejected (that is, over quota) from this cell so far
temp	the number of completed interviews and interviews in progress in this cell

These figures are updated automatically as each interview is terminated.

Changing cell texts and targets

If you make a mistake while entering new texts or targets, just select option 3 or option 5 from the menu and repeat the steps you took to enter the texts and targets in the first place. If you reach a cell you do not need to change, press Return to move to the next one.

Inspecting quotas

There are three options for inspecting quotas:

- 1 – Current Status Report (displays cells in order of definition)
- 2 – Cells Sorted Starting with LEAST Full
- 7 – Cells Sorted Starting with MOST Full

Select the type of report you want and then choose whether to display the report on your screen or write it to a *script.rpt* file. If you decide to display the report on your screen, quota will display one cell at a time. Press Return to see the next cell or q to quit.

Here is a typical screen report:

```
Cell number 1 as at Thu Sep 24 16:05:00 1992
gender - male
marital status - single      21% full      ( 23% with pending)
target -100      completed -21      rejected -0      temp -23
```

The difference between the completed and temp counts tells you that there are 21 completed interviews for single men, with two more in progress, which have passed the quota statement.

A fullness percentage is displayed based on the number of completes relative to the target. If the temp count exceeds the complete count, an additional pending fullness percentage is displayed.

If the quota has been met, a message shows the time at which the quota was filled.

Changing individual quotas

You can use option 3, Enter New Targets, to change quota targets. This takes you through each cell in turn, which is unnecessarily laborious when you want to change only one or two values. To change individual cells, use option 6, Examine/change Individual Cell. This allows you to select a specific cell by its conditions, so if you want to change the target for married men only, you could select it by choosing the elements male and married from the gender and marital status dimensions.

To help you with choosing the conditions, the quota program displays each dimension in turn, with the elements numbered sequentially from 1, and prompts you to select one. The status of the selected cell is then displayed with notes on possible actions. To leave the existing target intact, just press Return; otherwise, enter the new target value.

The example below shows how to change the quota for single men from 50 to 100.

```
Enter Value of **gender**
1 - male
2 - female
Value is - 1
Enter Value of **marital status**
1 - single
2 - married
3 - widowed
4 - divorced
Value is - 1

Cell number 1 as at Thu May 29 11:55:00 1986

gender - male
marital status - single
Target - 50    completed - 0    rejected - 0    temp - 0

Hit <rtn> to leave unchanged
Hit r to refer this cell to another cell
Enter a new number to change the target - 100
```

The quota program then asks whether you wish to select another cell; if so, the selection process is repeated. When no more changes are necessary, the program asks whether to write the new quotas back to the disk. If you fail to do this, your changes are ignored.

Combining quota cells

Quota cells may be combined — for example, you may combine the cell for single men with that for married men to produce one quota for single and married men. This does not delete the two original cells; it just merges the quotas (consequently, there is no need to change the script). You may still inspect the texts associated with the individual cells, but the targets on the first cell (single men) will be set to 0 and a note will be displayed telling you the cell with which this cell has been merged (here, married men). The quota for married men will be enlarged to incorporate the counts for single men. Quancept still maintains separate counts for each cell internally and if you unrefer the cells at a later date you will see the counts for each cell just as if they had never been referred.

-
- » You can combine any number of cells within a quota, not just two. If necessary, you can also combine cells hierarchically such that cell A is combined with cell B, and cell B is combined with cell C. The result is that cells A and B are both combined with cell C.
-

The mechanics of selecting the first cell are as described above for changing an individual cell. Type 'r' and then repeat the selection process to identify the second cell. Quotas are not combined until you answer 'y' to the question 'Ok to write info to disk'. Here is the screen dialog for combining the quotas for single and married men:

```
Enter Value of **gender**
1 - male
2 - female
Value is - 1
Enter Value of **marital status**
1 - single
2 - married
3 - widowed
4 - divorced
Value is - 1
Cell number 1 as at Thu May 29 11:55:00 1986
gender - male
marital status - single
Target - 50 completed - 0 rejected - 0
Hit <rtn> to leave unchanged
Hit r to refer this cell to another cell
Enter a new number to change the target - r
Enter Value of **gender**
1 - Male
2 - Female
Value is - 1
Enter Value of **marital status**
1 - single
2 - married
3 - widowed
4 - divorced
Value is - 2
Any more cells to be accessed? - n
Ok to write info to disk? - y
```

When qtip reaches a *quota* statement for a quota with referred cells, it treats all the cells that have been referred as if they were one big quota cell. However, it still maintains the counts for each cell individually even though you cannot see these counts.

When quota cells are combined and you inspect the quota values for those cells, the values for cells that are referred to other cells are shown as zero and the value for the cell to which they are referred is the sum of the quota for all the combined cells together. Similarly, quota reports in SMS (reports 407, 408 and 410) show 0 for the referred cells and the sum of all combined cells in the referred-to cell. The same is true for quota reports in supermen and in the quota program itself.

Referring quota cells is not a one-way process and you can return cells to their original unreferred state if you wish. This will allow you to see the individual cell counts that were hidden while the cells were referred. However, bear in mind the following points when considering whether to unrefer cells:

- Undoing a referral on a cell with a large target will often result in one cell being below target and another being over target.
- The timestamps indicating when cells went over quota will be incorrect because the standard by which cells are judged to be over quota has now changed.

- With multipunched quotas, the decision about which cell to use may have been different had the cells not been merged at the time that the interview took place.

Changing the completed count

Option 10, Change Current Values, allows you to change the number of completed interviews in a cell. A table is displayed for each cell in turn, showing the target and the number of completed, rejected, and temp interviews for that cell, as shown above. You are prompted to enter the new number of completed interviews or to press Return to skip to the next cell. When all cells have been dealt with, the program checks whether the new counts should be saved. Answer 'y' or 'n', as appropriate.

This is useful if you have quotas with many cells that you have used for testing, and you now want to use the quotas for live interviewing. Instead of removing the files and redefining the quotas, you can simply reset the cells in the current files to 0.

Removing hung temp entries

As the project progresses, the quota program maintains two internal counts of the number of interviews in each quota cell. 'complete' is the number of interviews that have passed the quota statement and have left the interviewing program classified either as completed interviews (that is, without the interviewer typing stop, abandon, or quit) or as interviews that have left the interviewing program as a result of a *signal 1* or *signal 6* statement in the script. 'temp' is the number of completed interviews *plus* the number of interviews in progress that have already executed the quota statement.

If an interview leaves the interviewing program classified as a completed interview, the quota program increments the completed count to bring it in line with the temp count that was incremented at the time the quota statement was executed for this interview. If an interview leaves the interviewing program as anything other than a completed interview, the complete count will not be incremented; instead, the temp count will be decremented to bring it in line with its previous count.

If an interviewer does not leave the interviewing program properly, the interview is not classified as a completed interview. However, the quota program has not had an opportunity to decrement the temp count, so it appears that there are more completed interviews than there really are. Option 10, Remove a Hung Temp Entry, allows you to remove these entries, one at a time. It is best to remove them when no interviewers are working so you can be certain that you do not remove temp entries that relate to interviews currently in progress.

31.3 The Quancept Web quota program



The Quancept Web quota program is called quotedit.

When you parse a script that contains quota statements, QCompile creates a file called *project.quiz*. This file contains information about the questions and responses that define each quota, and the identifier associated with each quota. When you are ready to define the quota targets, all you have to do is rename or copy the *quiz* file to *project.quo* and then run quotedit to define the targets.

Once interviewing is under way, the interviewing supervisor can run quotedit to check how quotas are being filled and, if necessary, may manually alter the count of interviews in a particular quota cell.

Having QCompile create a *quiz* rather than a *quo* file caters for situations where you need to recompile the script after quotas have been defined, but do not want to alter the targets in any way. However, if the changes to the script invalidate the original *quo* file, it is entirely your responsibility to recreate the *quo* file to match the new script.

To start quotedit and open the project's quo file:

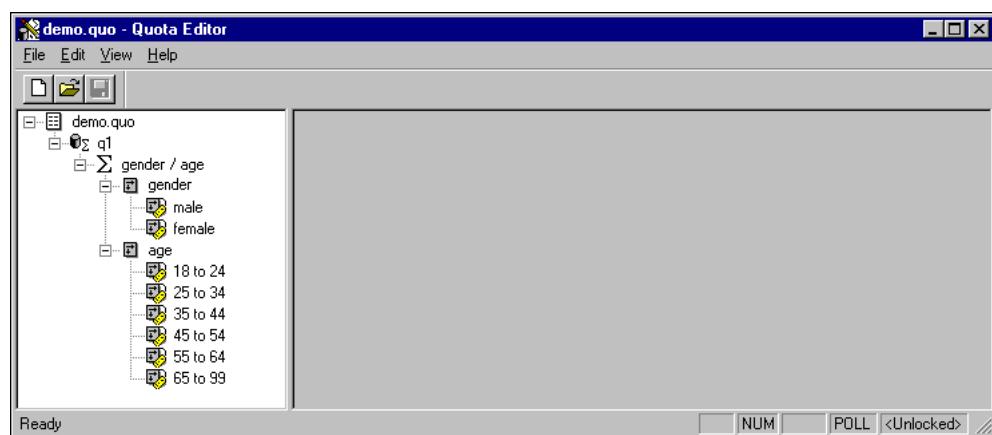
1. Rename or make a copy of the *quiz* file so that you have a file called *project.quo*.
2. Double click on the quotedit icon in the Quancept Web v1.1 program group:



The quotedit window opens.

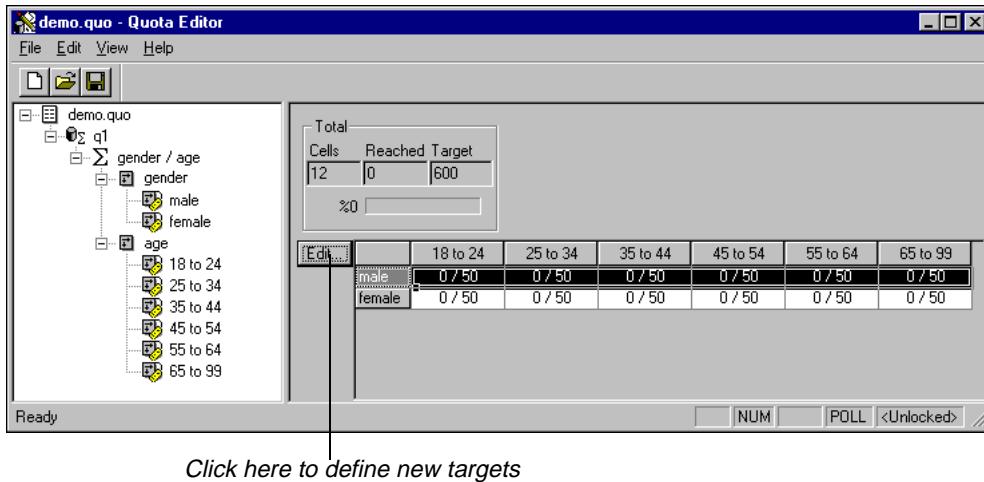
3. From the File menu select Open, and select the name of project's quo file.

The program displays the quota structures for this project in the left-hand pane of the window:



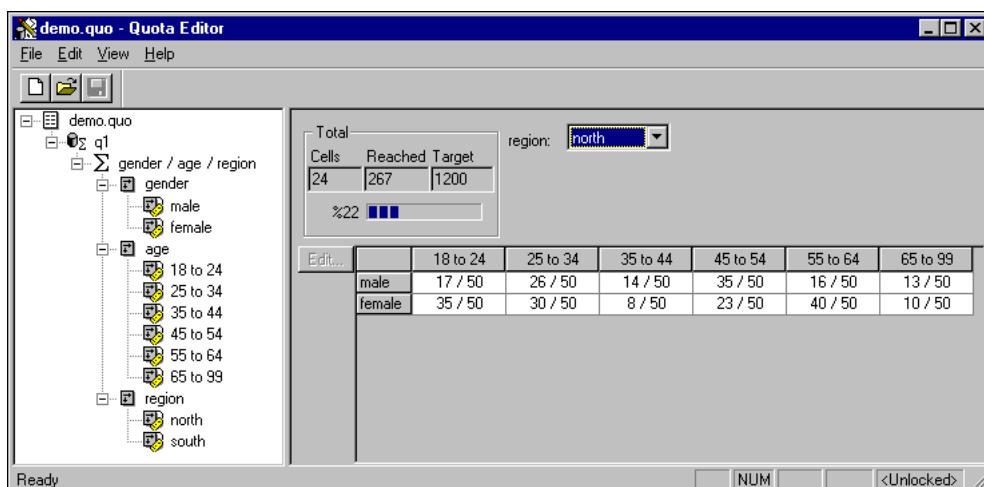
Defining targets

You define targets using the target grid that quotedit displays in the right-hand pane. Here is a completed grid. Each cell contains two numbers separated by a slash. The number on the left is the number of completed interviews so far and the number on the right is the target. In this example, each cell has a target of 50 interviews but no completed interviews so far.



The total box reports the overall totals for this quota and the percentage of completed interviews achieved.

The illustration here is for a two-dimensional quota. With three or more dimensions you have at least one additional selection box for defining the higher dimensions to which the targets refer:



There is never any need to define targets for each cell individually if you do not want to. Instead, you may select a complete row or column, or the entire grid, and enter a single target value that will be inserted in each cell of the selected row or column.

To define targets:

1. In the left-hand pane, double click on any of the lines in the quota you want to edit.

This displays the target grid in the right-hand pane.

2. If your quota has three or more questions (dimensions), you'll notice that there is a list box above the grid for each of the additional questions. Choose a response in each box. These responses are added to the row and column responses for each cell of the grid to define the full characteristic for each cell.

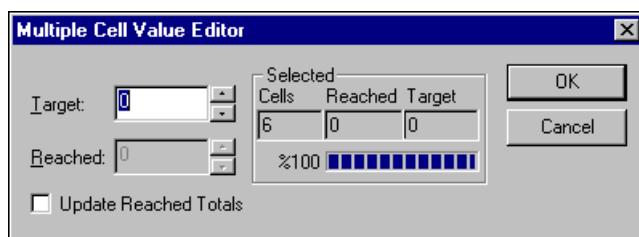
For example, if your *quota* statement uses the questions gender, age and region, each cell of the grid will refer to one combination of gender by age. By selecting, say, north from the region list box you make each cell include north as one of its characteristics.

3. Select the cells, rows or columns you want to work with.

To select	Do the following
A single cell	Click once in that cell.
A block of adjacent cells	Click in the first cell and, holding down the mouse button, drag the highlight over the cells you want to select.
A set of nonadjacent cells	Hold down the CTRL key as you click in each cell.
A whole row or column	Click in the row or column title cell.

4. Click the Edit button to the left of the grid.

A dialog box appears prompting for a target (if interviewing has begun and interviews have been achieved in this cell, this dialog box will also report the number of interviews achieved):



5. Enter the target and click OK to add it to the grid.

If you have selected more than one cell, the figure you enter is the target for each of those cells, not the overall target for all of those cells together. For example, if you select the male row in a gender by age quota and type 50, quotedit sets a target of 50 interviews for each male age group.

The grid is updated with the target and the number reached so far in the form reached/target.

6. Repeat steps 3 to 5 for each cell or set of cells.
7. With multidimensional quotas, select a new combination of responses for the higher dimensions and repeat steps 3 to 6.

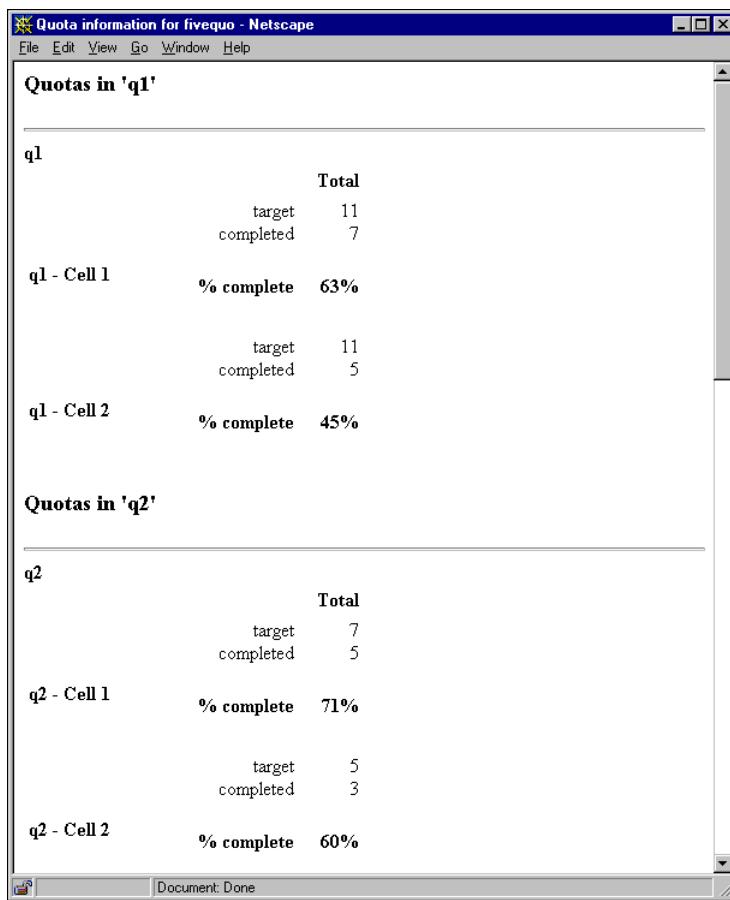
☞ You can use this procedure for changing targets that have been set previously. However, although quotedit allows you to do so, do not change targets while people may be working.

Checking and monitoring quotas

You can check quotas using quotedit or using your browser.

Using a browser

If you use your browser, you'll see a simple table of quota information similar to the one shown below:



The screenshot shows a Netscape browser window titled "Quota information for fivequo - Netscape". The menu bar includes File, Edit, View, Go, Window, and Help. The main content area displays quota information in two sections: "Quotas in 'q1'" and "Quotas in 'q2'".

Quotas in 'q1'

q1		
Total		
target	11	
completed	7	
q1 - Cell 1	% complete	63%
	target	11
	completed	5
q1 - Cell 2	% complete	45%

Quotas in 'q2'

q2		
Total		
target	7	
completed	5	
q2 - Cell 1	% complete	71%
	target	5
	completed	3
q2 - Cell 2	% complete	60%

☞ This facility has been implemented for two-dimensional quotas only: quotas with three or more dimensions do not display correctly.

To check quotas using a browser:

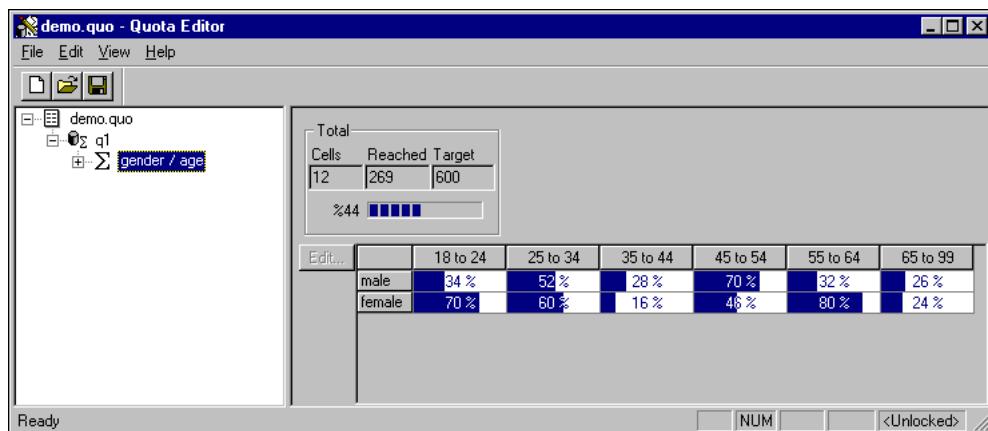
- Type the following URL:

http://server_name/scripts/qcweb.dll?quotainfo&server_name&project_name

where *scripts* is the name of the directory in which the qcweb.dll file is installed (usually scripts), *server_name* is the name of your Quancept Web server and *project_name* is the name of the project.

Using quotedit

If you use quotedit, you can view actual counts of targets and interviews reached (as shown above), or percentages that report what percentage of the target has been reached. Here is a report using percentages. Notice how the amount of highlighting in each cell represents the percentage reached:



When displaying raw counts, quotedit also gives you an immediate visual overview of reached or exceeded targets. Cells for targets that have been reached have white figures displayed on a blue background; cells for targets that have been exceeded have white figures displayed on a red background. If you see red in the target grid it is generally a sign that something is wrong: perhaps you have omitted the pass= parameter from the quota statement, or you have included it but have forgotten to check its result. (Red is also used in percentage mode to highlight over-quota cells.)

To check quotas using quotedit:

1. In the left-hand pane, double click on any of the lines in the quota you want to check.

This displays the target grid in the right-hand pane.

2. If the grid shows actual counts and you would prefer to see percentages, select Percents from the View menu. If the grid shows percentages and you want to view actual counts, select Raw Counts from the View menu.

Up-to-the-minute cell counts

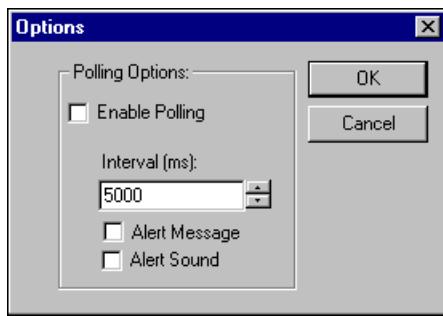
quotedit obtains its figures from the quo file, which Quancept Web updates every time an interview passes the quota statement. Normally, quotedit reads the quo file once when you open the project and does not update the figures it displays as the file is subsequently updated. If you want the grid display to be updated more frequently, you need to make quotedit reread the file. This rereading process is called **polling**.

If you switch polling on, it rereads the quo file once every five seconds (shown as 5000 milliseconds), but you can change this to suit your requirements. You'll know if polling is switched on because you'll see the word POLL in the text bar at the foot of the quotedit window.

To switch on and configure polling:

1. From the Edit menu select Options...

The Options dialog box is displayed:



2. Select Enable Polling.
3. To change the frequency with which polling occurs, type or select a new value in the Interval box. Note that this value is in milli-seconds, and using the arrow buttons alters the value shown by 1000 milliseconds (one second) at a time.
4. Click OK to close the dialog box.

Changing the count of interviews reached

Occasionally, the count of interviews reached in a cell may be incorrect. This is usually due to a problem with the interview, and supervisors will know or be told what to do in these situations. Sometimes the solution will involve manual editing of the count of reached interviews for a particular cell. This is very easy, but it must be done with extreme care and caution. As a security precaution, quotedit is prevented from changing the reached counts all the time that the server has control of a quo file.

quotedit keeps track of when reached counts can be changed and displays the words <Server locked> or <Unlocked> in the text bar at the foot of its window. If you wish, you can have the program issue visual and audible warnings when this status changes.

To change the count of interviews reached:

1. In the left-hand pane, double click on any of the lines in the quota you want to check.

This displays the target grid in the right-hand pane.

2. If necessary, select the higher dimensions for the cell you need to change.

3. Double click on the cell you need to change.

quotedit displays the cell editor dialog box.

4. Select the Update Reached Total check box.

This activates the count of reached interviews.

5. Type a new value for the count of reached interviews, or use the arrow keys at the side if the box to increase or decrease the value shown.

6. Click OK to return to the main screen.

To request a warning when the reached-lock status changes:

1. From the Edit menu select Options...

The Options dialog box is displayed.

2. Select one or both of the Alert Message and Alert Sound check boxes.

3. Click OK to close the dialog box.

Deleting targets and counts

You can delete targets and counts of reached interviews. This removes the entries that contain these figures from the quo file but it does not delete the dimensions themselves. Instead, Quancept Web will then allow unlimited numbers of interviews with respondents who belong in those cells.

To delete targets and counts:

1. Select the cell or cells you want to delete, as described above for defining targets.

2. From the Edit menu select Delete, or click the Delete button.

quotedit asks you to confirm that these cells may be deleted.

3. Click Yes to remove the targets and counts for those cells from the grid.

31.4 The mrInterview quota program

 In mrInterview, you use the Quota Setup program to define the quota categories and targets that you want to use for a project. Quota Setup provides the following features:

- **Questionnaire and sample variables.** Quotas can be based on categorical, numeric, and text variables from the questionnaire script, as well as on variables present in the sample data.
- **Independent quotas.** Quotas based on a single variable such as gender, age, region or preferred brand. Independent quotas are also known as one-dimensional quotas.
- **Dependent quotas.** Quotas based on a combination of between two and six variables. For example, a dependent quota for gender and age will define targets for gender and age combinations such as male aged 18 to 24, male aged 25 to 34, female aged 18 to 24, and so on. Dependent quotas are also known as multidimensional quotas.
- **Over quota cells.** When a target has almost been met and there are several interviews running that belong in that cell, you can allow those interviews to continue even though the target will be exceeded if all those interviews complete successfully. Interviews that start after the target has been met will be terminated in the usual way.
- **Counter quotas.** Interviews may be counted for a cell without the cell being used for quota control. You can use this type of quota for *null*, *dk*, *ref* and specified other categories where quota control is not required.
- **Table or Expression Quotas.** Quotas may be defined in tabular form or in text form using an expression builder.
- **Sample Management quotas.** You can define quotas based on variables present in the sample files. Sample data and questionnaire data can be combined in the same quota if required.
- **Block setting of targets.** Groups of quota cells with identical targets and other settings can be pre-selected, so that the specifications for those cells can be entered once and then applied automatically to each selected cell.

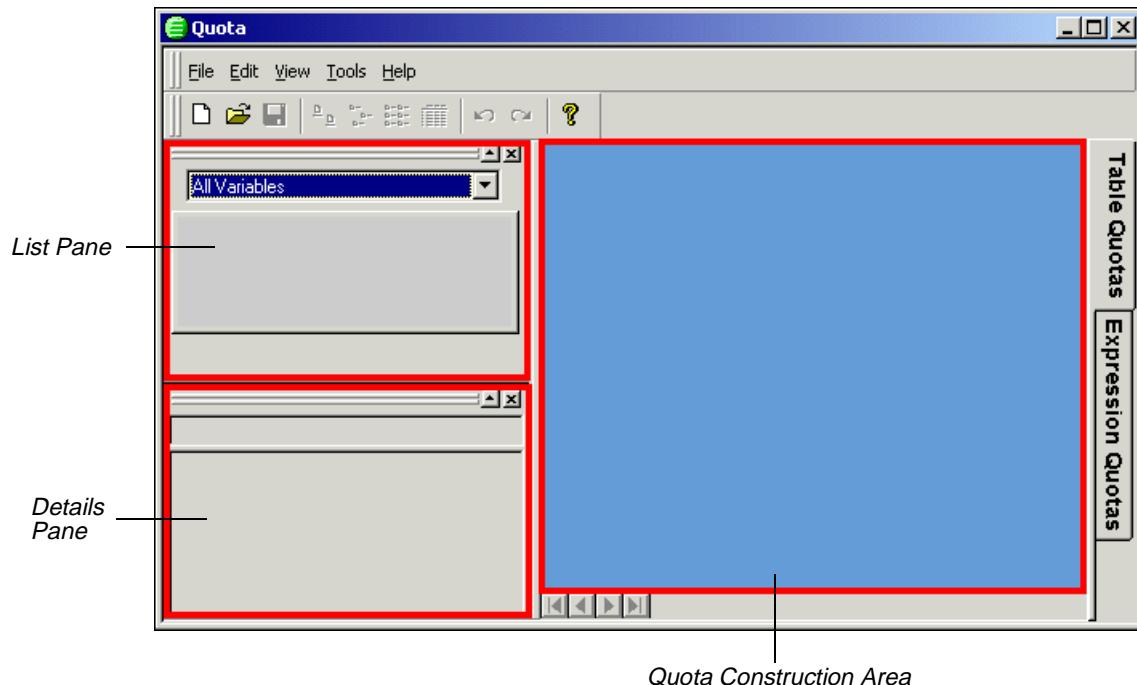
 For further information about the mrInterview quota control system, see chapter 30, Quota control. For information about writing quota control functions in Sample Management scripts, refer to the Sample Management section of the online *mrInterview User's Guide* and the sample scripts that are installed in the *mrInterviewSource\Samples* folder.

Starting Quota Setup

- From the Start menu choose Programs, SPSS MR, mrInterview 2.1, Quota.

The Quota Setup window

The Quota Setup window is divided into three panes:



The List Pane lists the variables that are present in the project's mdd file. For each variable, you see its name, variable number, data type, and description (question text). You can change the layout of the List Pane so that only icons and names are displayed if this is what you prefer.

The List Pane normally shows only categorical variables as these are the ones most commonly used for quotas, but you can display numeric, text and boolean variables as well. Quota Setup also hides system variables as these are rarely useful for quota-ing, but you can choose to display system variables if you wish.

The Details Pane lists the categories (responses) in the variable that is currently selected in the List Pane.

The Quota Construction Area is where you define the quotas by dragging in variables from the List Pane and then adding targets.

The List and Details Panes are ***undockable***. This means that you can convert them into windows, resize them, and move them anywhere on the screen, even outside the Quota window.

Options in the View menu also allow you to hide components of the Quota window, which can be useful when you have finished defining quotas on a large grid, and you want to see the whole grid on the screen.

Docking/undocking the List and Details Panes

To undock a pane:

- Double-click the double bars at the top of the pane.

To dock an undocked pane:

- Double-click the pane's title bar.

Changing the layout of the List Pane

- Click one of the following buttons in the toolbar:



Display a small icon followed by the variable name.



Display a list of icons and variable names.



Display a list of icons and variable names.



Return to the default, four-column layout.

Displaying all variable types in the List Pane

It is less common to define quotas based on numeric, text and boolean variables, so the List Pane does not display them unless requested.

To display all variable types in the List Pane:

- Choose All Variables from the drop-down list at the top of the View Pane.

Displaying system variables in the List Pane

System variables are created by mrInterview when you parse the questionnaire, and contain general information such as the interview serial number, the date and time at which the interview started, and the name of the last question that the respondent answered. As such, they are not normally useful for quota control and do not appear in the standard list of variables.

To display system variables in the List Pane:

- From the View menu choose View System Variables.

Displaying sample variables in the List Pane

When your sample records contain personal data such as age, gender, or brand tried, you can apply quotas based on this data either in the Sample Management authentication script or in the questionnaire script. For example, you could set up Sample Management quotas that restrict the survey to 500 men and 500 women, and then apply quotas in the script based on age or brand tried once respondents have started the interview.

You can create quotas based on variables in the sample records. Once you have told Quota Setup to make the sample variables available (they appear on a separate tab in the List pane), the procedures for using them to build quota definitions are the same as for questionnaire variables.

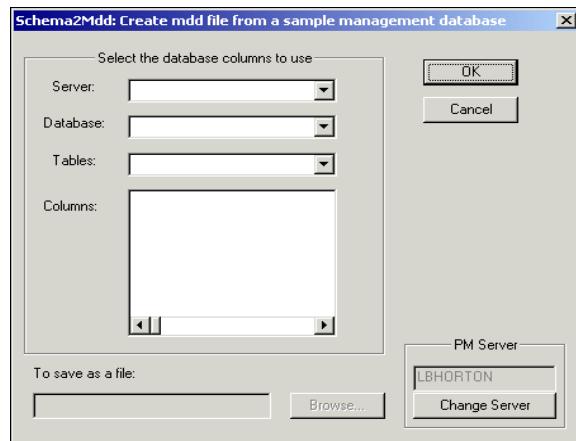
Quota Setup cannot use the sample information directly from the sample files so it creates a separate mdd file for them in the project's source directory. The default name for this file is *tablename_sm.mdd*, where *tablename* is the name of the sample table containing the sample records. You can choose a different name if you wish.

-
- Sample variables are not categorical variables so you will not see them unless you choose to show all variables in the List pane. This also means that all quotas that use sample variables must be defined as Expression Quotas rather than Table Quotas.
-

To display sample variables in the List Pane:

- From the File menu choose Open Sample Management.

The Schema2Mdd dialog box appears:



- If the server that holds the sample database is in a different cluster to the one on which you are working, click Change Server.

The Change Server dialog box appears.

3. Type the name of the cluster's Project Management server in the box provided and click OK.

The dialog box closes and after a brief pause the PM Server box is updated with the new name.

4. In Server, choose the name of the server that holds the sample database. If the server you want to use does not appear in the list, you may type its name or IP address in the box.
5. In Database, choose the name of the sample database.
6. In Table, choose that name of the table that holds the project's sample records.

The names of the columns in the table appear in the Columns box.

7. In Columns, choose the variables that you want to use for quotas. The default is to make all sample variables available.
8. If you do not want to accept the default name for the Sample Management mdd file, click Browse and enter or choose the name you want to use.

9. Click OK.

Hiding components of the Quota window

- Choose one or more of the following from the View menu:

List Pane Show/hide the List Pane. Clicking the X button in the top right-hand corner of the pane also hides the pane.

Details Pane Show/hide the Details pane. Clicking the X button in the top right-hand corner of the pane also hides the pane.

Default Layout Return the window to its default appearance.

Main Toolbar Show/hide the main toolbar.

Status Bar Hide/show the status bar at the foot of the window.

Creating and opening quota files

Quota Setup obtains its information about a project's variables from the mdd file and saves the quota definitions you create in a file with an mqd extension.

-
- ☞ If quotas exist for a project, you should always work on the mqd file rather than the mdd file. If you open an mdd file when an mqd file exists, Quota Setup will overwrite the original contents of the mqd file rather than adding the new quotas you have just created.
-

To create a new quota definition file:

1. From the File menu choose New.

The Open dialog box appears.

2. Navigate to the project's source folder and choose the project's mdd file.

3. Click Open.

The List Pane displays the names of the project's categorical variables, and the Quota Construction Area is prepared for defining quotas.

To open an existing quota definition file:

1. From the File menu choose Open.

The Open dialog box appears.

2. Navigate to the project's source folder and choose the project's mqd file.

3. Click Open.

The List Pane displays the names of the project's categorical variables, and the Quota Construction Area shows details of the existing quotas.

Undoing and redoing actions

Quota Setup keeps track of the drag and drop actions that you take during a session and the order in which you take them, and allows you to roll back or re-instate the changes one at a time.

When using Undo and Redo, bear in mind the following points:

- Once you add targets to a quota table, Quota Setup forgets about the individual changes you made up to that point and treats all actions on that table up to and including the addition of the targets as a single step. For example, if you add a new table with two dimensions and then add targets, clicking Undo removes the second variable and the targets, and clicking Undo a second time removes the first variable; clicking Redo re-instates the first variable and then the second variable and the targets. You cannot undo the targets by themselves or undo the addition of the second variable.
- Saving changes to a table does not prevent you from undoing them.
- You cannot undo the addition of a new table: delete the table instead.
- Undo and Redo are disabled for Expression Quotas.

To undo actions:

1. From the Edit menu choose Undo.
2. Repeat this step to undo the previous change, and so on. When there are no more changes to undo, Quota Setup makes the Undo feature unavailable.

To re-instate undone actions:

1. From the Edit menu choose Redo.
2. Repeat this step to re-instate other actions that you undid in the order in which the actions were originally taken. When there are no more actions to redo, Quota Setup makes the Redo feature unavailable.

Choosing your working language

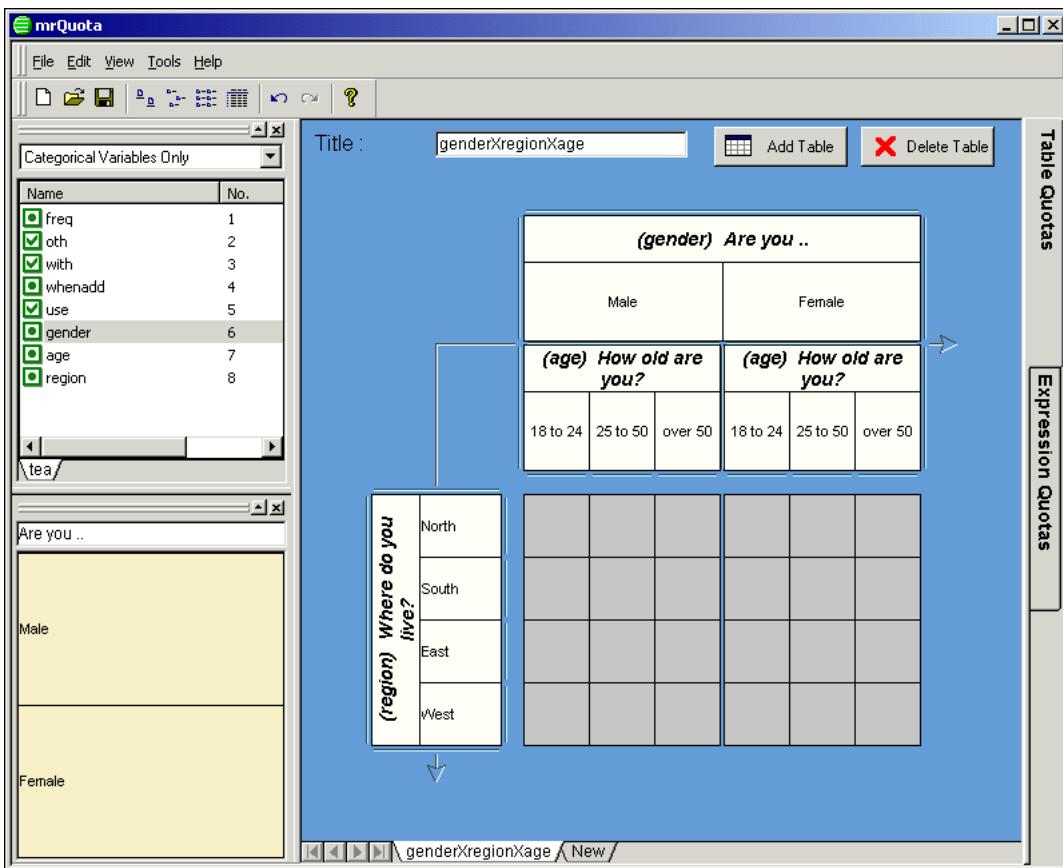
When you work on a multilingual project, Quota Setup displays question and response texts in the project's base language. If you would prefer to work in one of the project's other languages, you may do so.

To choose your working language:

1. From the Tools menu choose Language.
The Select Language dialog box appears.
2. In Language, choose a language from the dropdown list.
3. Click OK.

Table Quotas

The easiest way of defining quotas for categorical variables is to use the Table Quota tab. Using this method, you define the quota dimensions and structure by dragging variables from the List Pane into the Quota Construction Area. The position of the variables in the Quota Construction Area determines the characteristics of each quota cell. The following illustration shows a three-dimensional quota of gender by region by age in which each cell defines a target for a different combination of those three characteristics:



As you build the quota table, Quota Setup prepares a grid in which you can enter the targets and other settings for the various cells.

-
- ❖ Table Quotas are available for script variables only. If you are defining quotas for Sample Management variables, you must define them as Expression Quotas.
-

To define Table Quotas:

1. In the List Pane, choose one of the variables that forms part of the quota you are defining, and drag it onto one of the drop lines in the Quota Construction Area. You will know when you have the variable in the right place because the drop line will turn pink.

The variable's name and categories appear on the drop line, and an empty grid appears in which to define the targets.

2. If you are defining a two-dimensional quota, drag the second variable from the List Pane into the Quota Construction Area, but this time drop it on the other line.

The variable's name and categories appear on the drop line and additional cells are added to the target grid.

3. If you are defining a quota with more than two dimensions, drag the additional variables to the appropriate positions relative to the other variables in the grid. You can place the new variables above, below, and to the left or right of existing variables, but in the context of defining quotas, the most logical positions are above or below the top variable and to the left or right of the side variable.
4. When the structure of the quota grid is complete, you can start defining targets. Select one or more cells in the target grid, as follows:

To Select	Do the following
A single cell	Click in the cell. (Double click to select the cell and open the Quota information dialog box all in one go.)
A number of non-adjacent cells	Hold down the Ctrl key and click on each cell in turn.
All cells in a row	Position the cursor over a cell in that row, right-click and choose Select Row.
All cells in a column	Position the cursor over a cell in that column, right-click and choose Select Column.
All cells in the grid	Position the cursor over any cell, right-click and choose Select Table.

The selected cells turn blue and are the ones that will be affected by your next actions.

5. Right-click and choose Edit from the pop-up menu.

The Quota information dialog box appears.

6. In Quota target, type the target for the selected cell or cells.
7. Optionally, click Allow Over Quota if you want to allow interviewing to continue all the time that the sum of pending and completed interviews in this cell is less than the target. Once the target is met, any interviews that were already in progress will be allowed to continue and complete, so it is possible that the final number of completed interviews will slightly exceed the target. Once the target has been met, all new interviews in the cell will fail the quota test and will be dealt with as specified by the scriptwriter (usually the interview will be terminated).
8. Optionally, click Counter Quota to count the number of interviews that take place for these cells without imposing quota control.
9. Click OK to close the dialog box.

The quota table is updated to show the targets for each cell. The symbol in the top right-hand corner of each cell shows the quota type.

10. Repeat these steps until you have defined a target for each cell.

11. In Title, replace the default title with a something that describes the quota.

The scriptwriter may use the title that you define to identify the quota in the script. It is therefore a good idea to replace the default title that Quota Setup provides with a title that reflects the content of the quota because this will make the script easier to follow.

12. From the File menu choose Save to save the definition.

-
- ☞ You must give the mqd file the same name as the project and save it in the project's source directory otherwise the activation process will fail when the scriptwriter tries to activate the project.
-

Creating more than one Table Quota

You can create as many quota tables as you need, based on different questions or combinations of questions in the questionnaire. In a soft drinks survey, for example, you may want to have one quota based on age and gender and then, further on in the questionnaire, a quota based on the type of soft drink that respondents usually drink.

To create another table:

1. Either click Add Table or choose the New tab at the foot of the Quota Construction Area.
2. Select the quota variables and define the targets as described above.

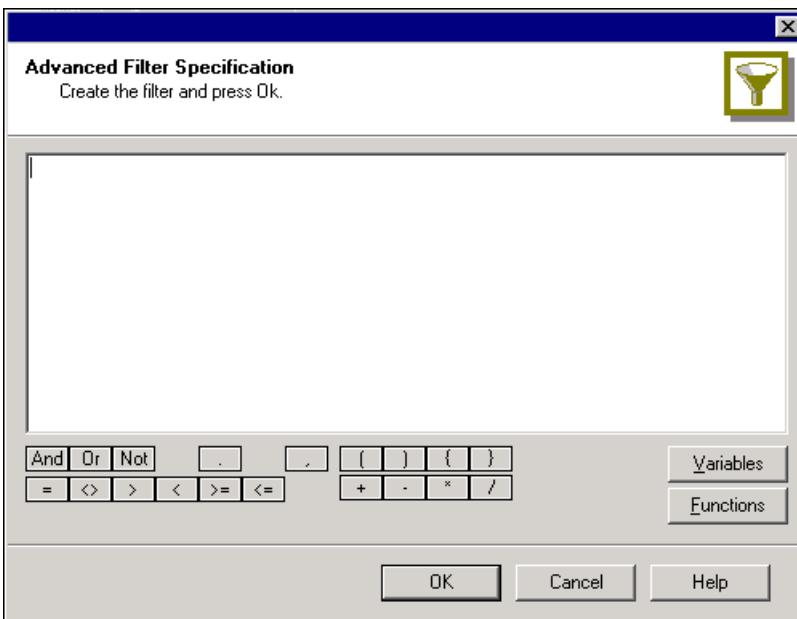
Deleting Table Quotas

- Choose the tab for the quota you want to delete.
- Click Delete Table.
- Confirm your request when prompted to do so.

-
- ☞ Deleting quota tables from Quota Setup does not delete the tables from the Quota database. Instead, the tables are flagged as Inactive, so that they are ignored by the quota control system.
-

Expression Quotas

The Expression Builder lets you define quotas by creating expressions that describe the characteristics of each quota. You can create Expression Quotas using any types of variables, but they are particularly useful for quotas based on numeric, text or boolean variables, because these types of quotas cannot be defined in quota tables.



To understand more about how Expression Quotas work, consider a simple Table Quota for gender. The expression that describes the cell for men is that the gender variable should contain 'Male'; the expression for the cell for women is that gender should contain 'Female'. If you wanted, you could replace the Table Quota for gender with two Expression Quotas using the expressions noted here.

With numeric variables, you can define quotas based on single values or ranges of values. For example, if age is a numeric variable, you can define quotas for people aged 21, people aged 18 to 21, or people who are older than 21.

With text variables, you can define quotas based on words that are present or not present in a variable — for example, people who mention the word 'expensive' when saying why they would not buy a product.

-
- ❖ Expression Quotas exist only for the characteristics you define. If your expressions do not cover all possible values for a variable, there will be no quotas for the missing values. This is not the same as Table Quotas, where choosing a variable automatically creates quotas for all categories in the variable. For example, if you define a single expression for people aged 35 years and below, there will be no quota for respondents aged over 35 and respondents in this age group will not be checked. This is not wrong, but you may prefer to create a counter quota for these respondents instead.
-

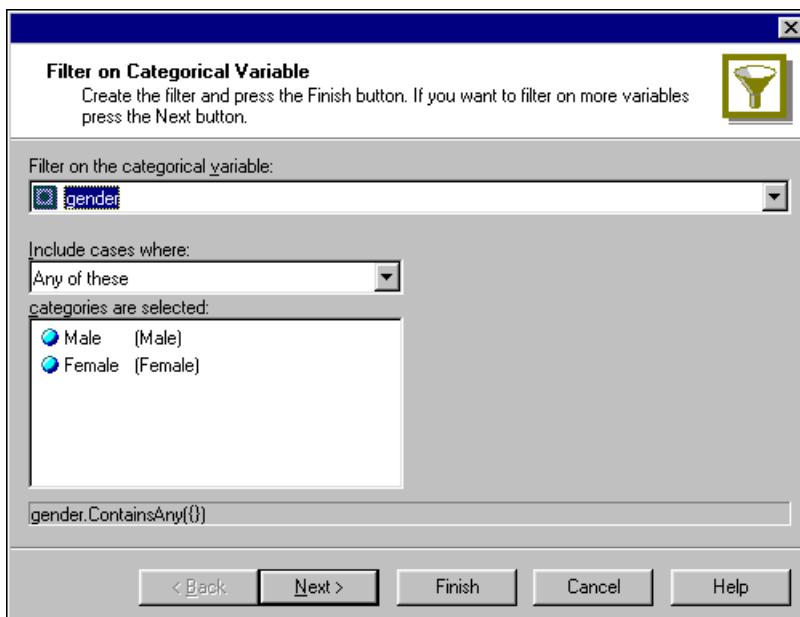
When a quota has more than one characteristic, the expression that defines the quota will consist of a number of sub-expressions, one for each characteristic. So, a quota for young men will consist of two sub-expressions, one for male respondents and the other for young respondents. The two sub-expressions will be linked by an operator that tells the quota system that respondents should be included in the quota only if they have both characteristics. All women and men who are not young will be excluded from the quota.

-
- ❖ Although each expression has a unique name, they are normally stored as a single group with the title ‘Expressions’. Expression Quotas based on Sample Management variables are stored separately from Expression Quotas based on variables that are present in the script and have the title *tablename_sm:Expressions*, where *tablename* is the name of the sample table in the sample database.
-

When you choose a variable for quotas, Quota Setup displays a Filter On dialog box that prompts you for the information required to create a quota for that type of variable. The content of the dialog box varies according to the variable type.

Filter on categorical variable dialog box

You can use expressions to define quotas based on categories from single response and multiple response variables, and grid subvariables. For example, you can set up an expression that includes male respondents, or includes respondents who usually drink assam or darjeeling tea.

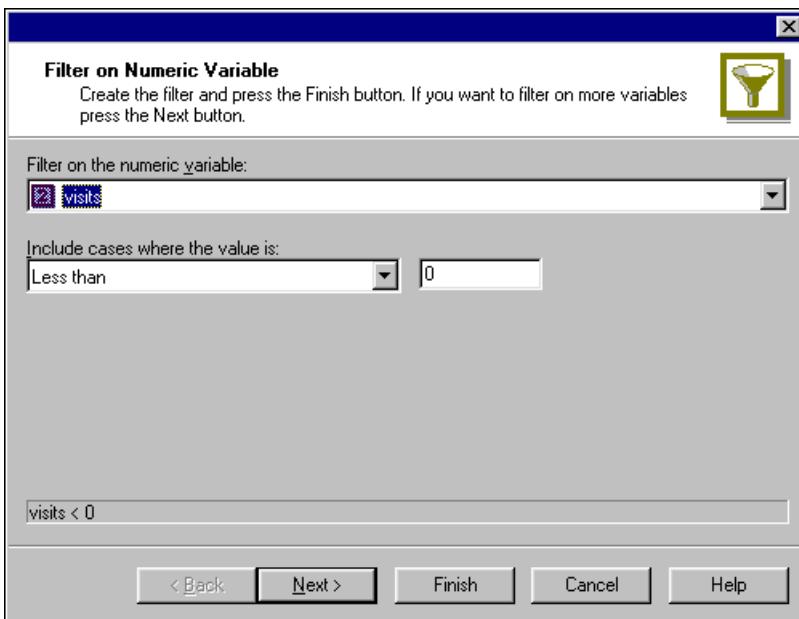


Field/Button	Description
Filter on the categorical variable	Displays the variable that you dragged onto the Expression Quotas tab. To use a different variable, click the arrow and choose another from the drop-down list.

Field/Button	Description
Include cases where	<p>Specifies how to determine whether the categories that you are about to select will be part of the current quota expression. Options are:</p> <p>Any of these. The quota is for respondents who choose at least one of the selected categories.</p> <p>None of these. The quota is for respondents who do not choose any of the selected categories.</p> <p>All of these. The quota is for respondents who choose all of the selected categories. Not available for single response variables.</p> <p>Exactly these. The quota is for respondents who choose all of the selected categories and no others. Not available for single response variables.</p> <p>At least. The quota is for respondents who choose at least the specified number of the selected categories. Specify the number in the box that appears to the right. Not available for single response variables.</p> <p>At most. The quota is for respondents who choose no more than the specified number of categories. Specify the number in the box that appears to the right. Not available for single response variables.</p> <p>Between. The quota is for respondents who choose a certain number of selected categories. Specify the minimum and maximum number of categories in the boxes that appear to the right. Not available for single response variables.</p>
Categories are selected	Select the categories of the variable that define the characteristics of this quota.
Next	Click to add another variable to the expression.
Finish	Click to close the dialog box.
Cancel	Click to close the dialog box without creating the quota expression.

Filter on numeric variable dialog box

You can use expressions to define quotas based on values in numeric variables. For example, you can create a quota for people who are under 24 years of age, or you can create a series of expressions for respondents in different age ranges.

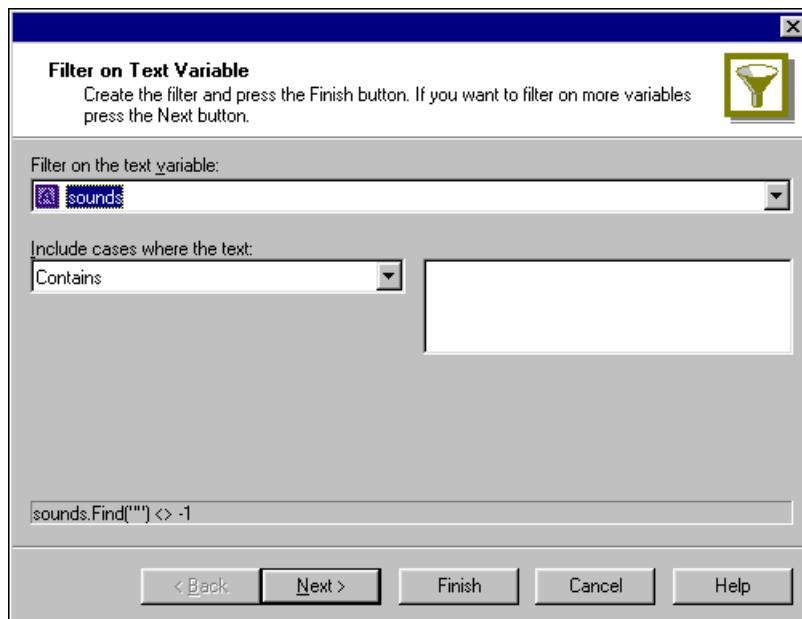


Field/Button	Description
Filter on the numeric variable	Displays the variable that you dragged onto the Expression Quotas tab. To use a different variable, click the arrow and choose another from the drop-down list.
Include cases where the value is	Specifies how to determine which respondents to include in this quota, and the numeric value to which the rule applies. Options are: <ul style="list-style-type: none"> Less than. The quota is for respondents who give an answer less than the specified value. Greater than. The quota is for respondents who give an answer greater than the specified value. Equal to. The quota is for respondents who give an answer that is the same as the specified value. Not equal to. The quota is for respondents who give an answer that is not the same as the specified value. Less than or equal to. The quota is for respondents who give an answer less than or equal to the specified value. Greater than or equal to. The quota is for respondents who give an answer greater than or equal to the specified value.
Next	Click to add another variable to the expression.
Finish	Click to close the dialog box.
Cancel	Click to close the dialog box and cancel the expression.

Filter on text variable dialog box

You can define expressions that are based on text strings in text variables. For example, you can define an expression that includes respondents who mention the word ‘noisy’ in their answer.

-
- ❖ All text expressions are case insensitive. In other words, it does not matter whether you type the text in upper or lower case, because the expression matches the characters in the text and not the case in which they are entered.
-

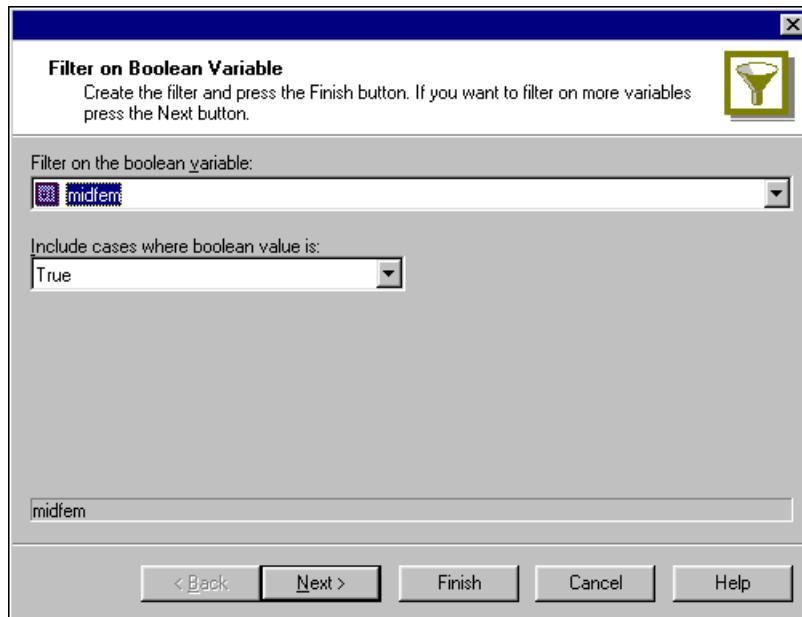


Field/Button	Description
Filter on the text variable	Displays the variable that you dragged into the Expression Quotas tab. To use a different variable, click the arrow and choose another from the drop-down list.
Include cases where the text	<p>Specifies how to determine which respondents will be included in the quota. Type the text that defines the quota characteristic in the box on the right.</p> <p>Contains. The quota is for respondents who mention the specified text.</p> <p>Is exactly. The quota is for respondents whose answers consist of the specified text and nothing else.</p>

Field/Button	Description
Include cases where the text	Begins with. The quota is for respondents whose answers begin with the specified text. Ends with. The quota is for respondents whose answers end with the specified text. Is blank/empty. The quota is for respondents whose answers are blank or empty. This includes null and silent null responses. Is not blank/empty. The quota is for respondents whose answers contain any text at all.
Next	Click to add the values or categories of another variable to the expression.
Finish	Click to close the dialog box.
Cancel	Click to close the dialog box and cancel the expression.

Filter on boolean variable dialog box

You can define expressions that are based on the values in boolean variables. A boolean variable can contain only one of two values: true or false.

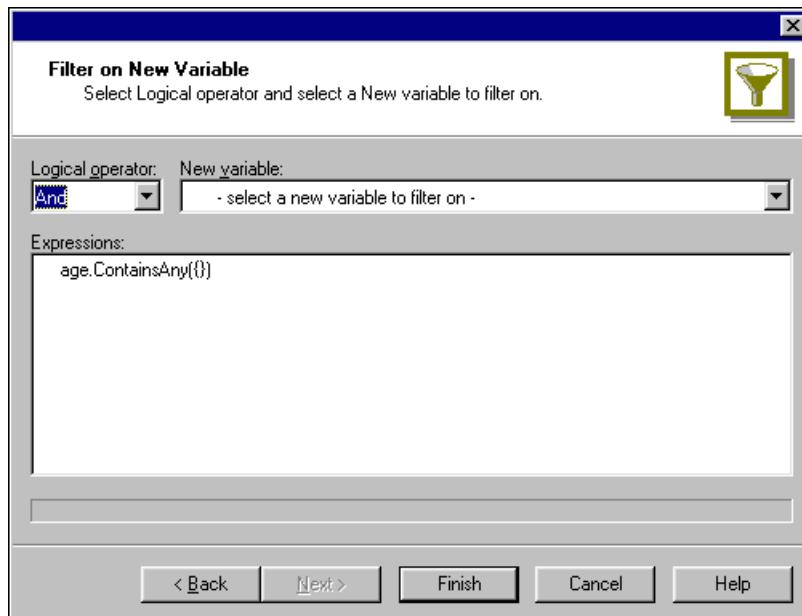


Field/Button	Description
Filter on the boolean variable	Displays the variable that you dragged onto the Expression Quotas tab. To use a different variable, click the arrow and choose another from the drop-down list.

Field/Button	Description
Include cases where the Boolean value is	Choose whether you want the quota to count true or false values.
Next	Click to add another variable to the expression.
Finish	Click to close the dialog box.
Cancel	Click to close the dialog box and cancel the expression.

Filter on new variable dialog box

You use the Filter on New Variable dialog box to combine characteristics from multiple variables into a single expression. This dialog box appears when you define a characteristic using a variable and then click the Next button.



When you create an expression using multiple variables, you can combine the characteristics using either the *And* operator or the *Or* operator. For example, you could create a quota for respondents who drink assam tea and who normally make tea using tea bags.

The *And* operator specifies that respondents must have all the listed characteristics. The *Or* operator specifies that respondents must have at least one of the characteristics to be counted towards the quota.

Field	Description
Logical operator	Choose the operator that you want to use to combine the quota characteristics.

Field	Description
New variable	Choose the variable that you want to add to the expression from the drop-down list.
Expressions	Displays the options that you have chosen for the expression.

To define Expression Quotas:

1. Choose the Expression Quotas tab.
2. Drag a variable from the List Pane onto the Expression Quotas tab.

The Filter On dialog box appears. The content of the dialog box varies according to the variable type.

3. Define the quota characteristics and selection criteria as follows:

Categorical variables

- a) From the Include Cases Where drop-down list, choose the inclusion criteria for the categories that define the quota characteristics.
- b) In Categories are Selected, choose the characteristics that respondents must have in order to be included in the quota. You can select multiple categories by holding down Ctrl while you click.

Numeric variables

- a) From the Include Cases Where the Value Is drop-down list, choose the selection criteria for this quota.
- b) In the number box on the right, type the value to which the criteria apply.

Text variables

- a) From the Include Cases Where the Text drop-down list, choose the option that defines when respondents will be included in this quota.
- b) In the box on the right, type the text that defines the quota characteristic.

Boolean variables

- From the Include Cases Where the Boolean Value Is drop-down list, choose whether you want to include respondents whose values are true or false.
3. If the expression is based on two or more variables or values, do the following, otherwise click Finish to close the dialog box.
 - a) Click Next. The Filter on New Variable dialog box appears.

- b) In Logical operator, choose how the sub-expressions are to be combined. Choose *And* to create a quota expression that requires both sub-expressions to be True; choose *Or* to create a quota expression that requires at least one of the sub-expressions to be True.
- c) In New variable, choose a variable from the drop-down list. This list displays all variable types and includes system variables even if you have chosen not to view these variables in the List Pane.
- d) Define the expression for this variable.
- e) Click Next to add another sub-expression or Finish to end the expression.

When you click Finish, the quota appears on the Expression Quotas tab with a unique name and a default target of 100.

6. In Target, type the target you want to set.
7. In Type, choose one of the following from the drop-down list:

Normal This is an ordinary quota, so the sum of pending and completed interviews for this quota must never exceed the target. Once the sum of pending and completed interviews for this quota matches the target, all subsequent interviews for this quota will fail. If any of the pending interviews do not complete, further interviews will be required to meet the target.

Allow over Allow the sum of pending and completed interviews for this quota to exceed the target. New interviews may be started for this quota until the number of completed interviews matches the target. If all the pending interviews complete, the target may be exceeded. Once the target has been met, any new interviews for this quota will fail the quota test and will be dealt with as specified by the scriptwriter (usually the interview terminates).

Counter There is no target for this quota, but the number of respondents belonging to the quota will be counted.

8. In Description, type a short description of the expression.
9. To save the definition choose File, Save.

☞ You must give the mqd file the same name as the project and save it in the project's source directory otherwise the activation process will fail when the scriptwriter tries to activate the project.

Advanced expressions

You can define quotas whose characteristics are of almost unlimited complexity using the Expression Definition language. This is based on VBScript and supports all of the VBScript functions. It also includes additional functions that have been designed to meet the needs of the market research industry. For detailed information about the many functions that are available, see the documentation that comes with the SPSS MR Data Model (from the Start menu, choose Programs, SPSS MR, Accessories, SPSS MR Data Model Help).

When you create a simple expression, Quota Setup converts the expression into the Expression Definition language automatically. When you create an advanced expression, you write the expression yourself in the Advanced Filter Specification window, which is designed to make creating expressions easy. The Advanced Filter Specification window has buttons for the operators and functions that you are most likely to use, as well as lists of all of the variables and functions that are available. You can build an expression by choosing variables and clicking operator buttons, by entering the expression directly, or by using a combination of the two approaches.

Expression syntax

At its most basic, an expression is:

Variable Operator Value

where *Variable* is the name of a variable. *Operator* is a comparison operator, such as > (greater than) or = (equal to), or one of the functions for selecting respondents on the basis of the answers given to a particular question (variable). *Value* is one or more answers held in the variable. You must enclose text values in double quotes — for example, “noisy”. You specify values of a categorical variable by defining a comma-separated list of category texts enclosed within braces and parentheses — for example:

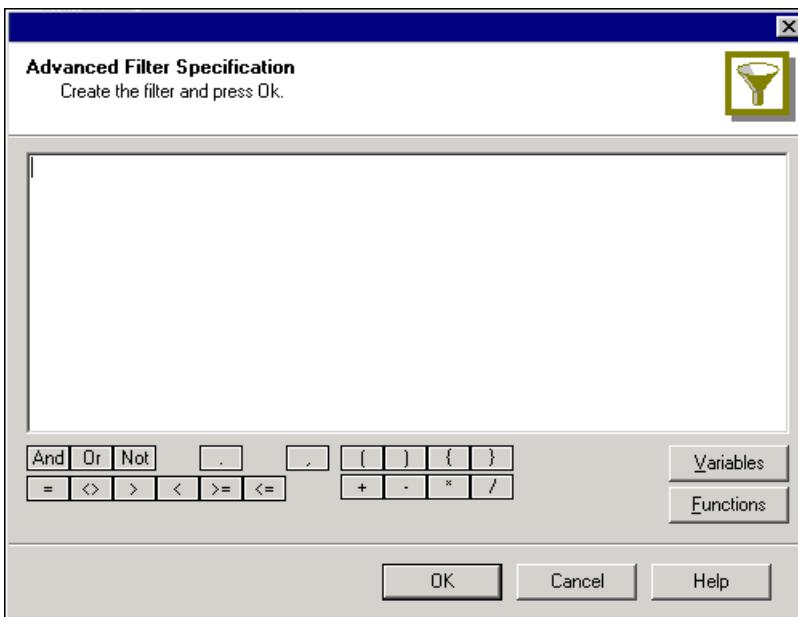
({Assam, Darjeeling, China})

You can specify values from numeric variables as numeric values.

When a quota statement for an Expression Quota is executed during an interview, the expression is evaluated and returns a value of true for a respondent who has the quota characteristics and false for a respondent who does not have the required characteristics.

Specifying expressions using the advanced method

You can use the Advanced Filter Specification window to create new Expression Quotas and to modify existing ones. If you are editing an existing expression, it is displayed in the quota expression box when you open the dialog box. Otherwise, the quota expression box is blank:



You can build Expression Quotas by adding variables and clicking the operator and function buttons. Alternatively, you can enter expressions directly into the quota expression box. In practice, you will probably use a combination of both methods.

For example, if the questionnaire script contains a categorical question called *gender*, with the categories Male and Female, you can create the following expression to select male respondents:

```
gender.ContainsAny({MALE})
```

To create this expression, you would

- Click the Variables button and choose *gender* from the list.
- Click the *.* button.
- Click the Functions button and choose *ContainsAny* from the list that appears.
- Click the *{* button.
- Choose MALE from the list that appears.
- Click the *}* button.
- Click the *)* button.

Comparison operators in Expression Quotas

When you build an expression, you can use the comparison operators to compare the value in a variable with a specified value of the same type. For example, you can compare the value in a numeric variable with a numeric value and the value in a text variable with a text string. The operators are also valid with other variable types. See the SPSS MR Data Model documentation for more details (from the Start menu, choose Programs, SPSS MR, Accessories, SPSS MR Data Model Help). The following table lists the comparison operators.

Operator	Meaning	Used with
=	Equal to	Numeric and text variables
<>	Not equal to	Numeric and text variables
>	Greater than	Numeric variables
<	Less than	Numeric variables
>=	Greater than or equal to	Numeric variables
<=	Less than or equal to	Numeric variables

For example:

```
visits > 5
```

selects respondents for whom the visits numeric variable holds a value greater than 5.

```
visits <= 5
```

selects respondents for whom the visits numeric variable holds a value less than or equal to 5.

Functions

There are many functions that you can use to define Expression Quotas based on values in categorical variables. The functions you are most likely to use for defining quotas are *ContainsAny* and *ContainsAll* for quotas based on subsets of responses from a multiple response list, and *AnswerCount* for quotas based on the number of responses chosen from a list.

For information about the many other functions that are available, see the documentation that comes with the SPSS MR Data Model (from the Start menu, choose Programs, SPSS MR, Accessories, SPSS MR Data Model Help).

ContainsAny

Use the *ContainsAny* function to define a quota for respondents who have chosen any of a number of listed categories. The categories can belong to a single response variable, a multiple response variable, or a grid subvariable. The syntax is:

```
variable.ContainsAny({Category1_ID, Category2_ID, ...})
```

« Note that the syntax uses the unique ID of the category rather than its label.

For example, you might define the quota expression for young people as:

```
age.ContainsAny({E1720_years, E2124_years})
```

This quota expression counts all respondents in the 17–20 years or 21–24 years categories.

You can use the *ContainsAny* function in combination with the logical operator *Not* to define a quota for respondents who have not chosen any of a number of listed categories. The categories can belong to a single response variable, a multiple response variable, or a grid subvariable. The syntax is:

Not *variable*.**ContainsAny**(*{Category1_ID, Category2_ID, ...}*)

For example, you might define the quota expression for all respondents who are older than 21 years of age as:

```
Not age.ContainsAny({E1116_YEARS, E1720_YEARS})
```

This expression selects all respondents except those in the 11–16 years or 17–20 categories.

ContainsAll

Use the *ContainsAll* function to create a quota for respondents who choose all of a number of listed answers for a multiple response question. The syntax is:

variable.**ContainsAll**(*{Category1_ID, Category2_ID, ...}*)

¶ Note that the syntax uses the unique ID of the category rather than its label.

For example, you can define a quota for respondents who regularly drink tea and coffee as follows:

```
drink.ContainsAll({Tea, Coffee})
```

Respondents who drink only tea, only coffee, or neither beverage are excluded from the quota.

¶ You can use *ContainsAll* with a single response variable, but you must be careful not to specify more than one category. If you do, the expression will never select any respondents.

All the listed answers and no others

You can use the *ContainsAll* function with the Exactly flag set to true to define a quota for respondents who choose all of a number of listed categories from a multiple response question and no others.

For example, you would define a quota for respondents who drink water and nothing else as:

```
drink.ContainsAll({Water}, True)
```

AnswerCount

Use the *AnswerCount* function to define a quota based on the number of answers chosen from a multiple response list. The syntax is:

variable.AnswerCount() numeric condition

For example, to define a quota for respondents who can name more than five brands of fizzy drinks:

```
fizzy.AnswerCount( ) > 5
```

Logical operators in Expression Quotas

When a quota definition is based on data from two or more variables, you define the overall expression by combining two or more sub-expressions using logical operators. Each sub-expression defines a single characteristic within the quota. For example, a quota for young men is a combination of a gender characteristic and an age characteristic.

Logical operators that join quota expressions are *And* and *Or*. The *Not* logical operator negates expressions.

And

Combine two sub-expressions with the *And* operator when you want to define a quota for respondents who satisfy both sub-expressions. The syntax is:

Expression1 And Expression2

For example, you would define a quota for young male respondents as follows:

```
gender.ContainsAny({MALE}) And age.ContainsAny({E1720_YEARS, E2124_YEARS})
```

This expression selects respondents whose gender is male and whose age is either 17–20 years or 21–24 years.

Or

Combine expressions with the *Or* operator when you want to define a quota for respondents who satisfy at least one of the sub-expressions. The syntax is:

Expression1 Or Expression2

For example, to define a quota for men of all ages and all women between the ages of 17 and 24, you would specify:

```
gender.ContainsAny({MALE}) Or age.ContainsAny({E1720_YEARS, E2124_YEARS})
```

The gender expression selects all men and rejects all women. The age expression selects anyone aged 17 to 24 and rejects anyone outside that age range. A man passes the gender expression regardless of his age. A woman fails the gender expression but passes the age expression if she is in the specified age group.

Not

Use the **Not** operator when you want to define a quota for respondents who do not have a particular characteristic. The syntax is:

Not *Expression1*

For example, to define a quota for male respondents who are not in the 17–20 or 21–24 age groups (that is, they are younger than 17 or older than 24):

```
gender.ContainsAny({MALE}) And Not age.ContainsAny({E1720_YEARS, E2124_YEARS})
```

Arithmetic operators in Expression Quotas

You can define quotas in which the quota characteristics are based on the results of arithmetic calculations. The arithmetic calculations consist of numeric variables, numeric values, and numeric expressions all joined by arithmetic operators.

Operator	Operation	Description
+	Addition	Use to add numeric values, numeric variables or numeric expressions.
-	Subtraction	Use to subtract one numeric value, numeric variable or numeric expression from another such value, variable or expression.
/	Division	Use to divide one numeric value, numeric variable or numeric expression by another such value, variable or expression.
*	Multiplication	Use to multiply two numeric values, numeric variables or numeric expressions.

For example, the following expression defines the quota characteristics for respondents who visited the museum more than five times prior to the year the survey was conducted:

```
visits - visits12 > 5
```

visits and *visits12* are both numeric variables. *visits* records the total number of previous visits each respondent made to the museum and *visits12* records the number of visits each respondent made in the previous 12 months. This expression subtracts the number of visits made in the previous 12 months from the total number of visits and selects respondents for whom the result is greater than 5.

Operator precedence in Expression Quotas

When there is more than one operator in an expression, they are evaluated in a set order, known as the order of precedence. The order of precedence, in evaluation order, is as follows:

- *Not*
- Multiplication and division
- Addition and subtraction
- Comparison operators
- *And* and *Or*

When operators are at the same level of precedence, they are simply evaluated from left to right.

You can override the order of precedence by using parentheses. Operators that are within parentheses are evaluated before operators that are not contained within parentheses. However, when more than one operator is contained within parentheses, they are evaluated according to the normal order of precedence.

Editing Expression Quotas

1. On the Expression Quotas tab click on the expression you want to change.

An arrow button appears to the right of the expression.

2. Click the arrow button.

The expression is opened for editing. If the expression is a simple expression based on one variable, the Filter On dialog box for that variable type appears. If the expression contains sub-expressions, the Filter On dialog box for the last sub-expression appears. To access other sub-expressions you will need to click the Advanced button to view the whole expression in the Advanced Filter Specification dialog box.

Deleting Expression Quotas

1. On the Expression Quotas tab click the red cross at the end of the line for the Expression Quota you want to delete.
2. Confirm your request when prompted to do so.

31.5 Reusing a quota definition file in another project



If you have a number of projects that have quotas based on the same combinations of variables, you can save yourself time by copying an existing mqd file into the new project rather than recreating the file from scratch. A typical example would be when you have a number of projects that all require quotas on the same demographic variables. As long as the demographic variables are the

same in all projects, you can create the mqd for one project and then copy it to all the other projects' source directories. If the targets vary between projects, or you want to add extra quotas for some projects, you can edit the copied mqd file to make these adjustments as you would on the original file.

-
- ❖ The mqd file contains a number of references to the project name. If you copy mqd files between projects, your first step after copying must be to edit the new file and change these references to point to the new project name. For example, if you copy cars.mqd into the travel source directory, you must change its name to travel.mqd and then edit it to replace all references to cars with references to travel.
-

The mqd file is a text file containing XML code so you can edit it using any text editor. Lines in the file are very long, so it is usually best to make the replacements using a global search and replace command.

31.6 Checking and changing quotas in mrInterview

- ❑ You can check quotas, revise targets or change the way quotas behave in mrInterview by running the ReviewQuotas program in DimensionNet.

 - ❖ Any changes that you make using ReviewQuotas are replicated in the QUOTA tables in the database but not in the project's mqd file. If you later re-activate the project, these changes will be overwritten with the information from the mqd file.
 - ☞ Refer to the online *mrInterview User's Guide* for details.

32 Parsing and compiling scripts

Once you have written your script, the next step is to check it for errors and, if it is error-free, convert it into a form that Quancept can use for conducting interviews. This checking and conversion process is known as **parsing**.

If you are using Quancept CATI, parsing is done with the program qparse. The Quancept-readable form of the script is created in a file called *project.qoc* (the filename's extension stands for Quancept own code — pronounced ‘qwok’). If you are using Quancept CAPI, Quancept Web or mrInterview, you parse the script and compile it to form a project-specific interviewing program. This program is called *project.exe* in Quancept CAPI and *project.sif* in Quancept Web and mrInterview.

This chapter explains how to parse and compile scripts and explains the possible causes of the error messages you might see if the parser finds a mistake in your script.

-
- » The parser finds syntax errors in a script, but it cannot find statements where the logic of the script is unsound. Errors of that type are found by the interviewing program when you test the script.
-

32.1 Parsing scripts using qparse



Quick Reference

To parse a script, go to the project directory and type one of the following:

```
qparse [-t] [-f] [-m] [-ptfo ptf_filename] [-ptfi ptf_filename] script  
qparse -u [-ptfo ptf_filename] [-ptfi ptf_filename] map_filename script
```

Parameters on the command line are:

- t** Display the script on the screen as it is parsed.
 - f** Write the listing to the 1st file.
 - m** Generate a data map in which columns are allocated to questions or variables in the order values are assigned to them rather than the order in which they appear in the script.
 - ptfo** Names the output ptf file.
 - ptfi** Read texts from an existing ptf file.
 - u** Use a predefined map file.
-

You can use most of these options together. The exceptions are *-t* and *-f*, which cannot be used together. If you do use both, qparse ignores *-t*.

When you parse the script, qparse always creates a ptf (project text) file. If a ptf file already exists for the project in that directory, qparse overwrites it. The *-ptf* flags let you name a different output ptf file so that the existing one is not overwritten, and read texts from an existing ptf file into the new one that qparse creates. You may need to read texts from an existing ptf file if translation work has already begun and the ptf file contains text in both the base and the translation languages.

There are two other, less commonly used, qparse options: *-d* and *-y*, which allow you to use the parser and yacc debugging facilities to solve problems with your script. Contact your support representative if you think you need to use these options.

If you enter an option that qparse does not recognize, it is ignored and a message is displayed listing valid options and their functions.

qparse is a Unix script that runs the following programs:

conv8	Concatenates all included files into a single file, and converts any 8-bit characters into a form that Quancept can read. This intermediate version of the script is written to the qc8 file in the project directory. If the project directory contains a qqc file, conv8 will use that instead. (The qqc file is created by qqcept from a ses file, and by the questionnaire design program, Quanquest.)
qcparse	Checks the syntax of your script and creates the qoc file that is used for interviewing.
chekpars	Sets punch codes for assignments of the form <i>set sp_mp_q='text'</i> , where <i>text</i> is a response text from the response list for the question called <i>sp_mp_q</i> .

qcparse and chekpars read each statement in the script and check that they are written in the correct format. If errors are found during the checking procedure, they are written to an errors file. If you did not use the *-t* option to display the parser listing on the screen, errors are also printed in the script listing file beneath or very near to the lines containing the errors.

If no errors are found, a number of files are created ready for interviewing. These include the qoc file, which is the computer-readable version of your script, and the data map file, which describes the layout of the data file. A list of all the files qparse creates is given in section 38.1, Files created by the parser.

Once you have an error-free script and the qoc file has been created, you may start running test interviews using qtip.

↖ If you need to edit and reparse a script for a project on which interviewing has already begun, make a copy of the script in a separate directory and do your editing and reparsing there. When you are sure that your changes work, you can replace the original qoc file with your new one.

Using predefined data map files



The `-u` flag on the qparse command line lets you parse a script using a predefined data map file. This allows you to modify the standard data map, including the columns that hold the serial number and card number, and then to reparse, forcing qparse to use the modified data map.

-
- ❖ Moving the serial number and card number columns may be necessary when you want to export data to other software. However, if you want to retain data compatibility with existing Quancept utilities, you must not move the columns containing the serial number or card number.
-

To create a default data map and then reparse using this data map:

1. Parse your script in the usual way to create a standard data map file.
2. Rename the map file.
3. Edit the map file using any text editor and change the card and/or column numbers of any questions you wish. You do not have to move lines in the file so that the column numbers are still in sequential order. For example, you could move a set of dummy questions from the start to the end of a record simply by changing their column numbers.

-
- ❖ Make sure that the new columns you choose are not already being used by other questions.
-

4. Reparse the script using the `-u` option to name the edited map file:

```
qparse -u map_filename script
```

This creates a new map file based on the edited map file and leaves the edited map file intact.

32.2 Parsing and compiling CAPI and Web scripts with QCompile



Like qparse, QCompile is a dynamic interface to a number of programs. These programs are:

wconv8.exe

Converts scripts written in a language that uses the 8-bit character set into a form that Quancept can read. This intermediate version of the script is written to the qc8 file in the project directory. If the project directory contains a qqc file created by the Quanquest questionnaire design program, you can run QCompile on this file.

wqcparse.exe

Checks the syntax of your script.

The MSV C++ compiler

Compiles the script into an executable program.

qditum.dll

Creates Quantum tables and axes specifications

When you parse and compile a script and wqcparse finds errors, QCompile displays a screen telling you what the errors are and where they occur. If your script contains errors, you need to edit it using a text editor before reparsing it with QCompile. There is no need to close the file in the editor before reparsing. Just save your changes and then switch to the QCompile window and reparse. If there are still errors, you can switch back to the editor window to correct them.

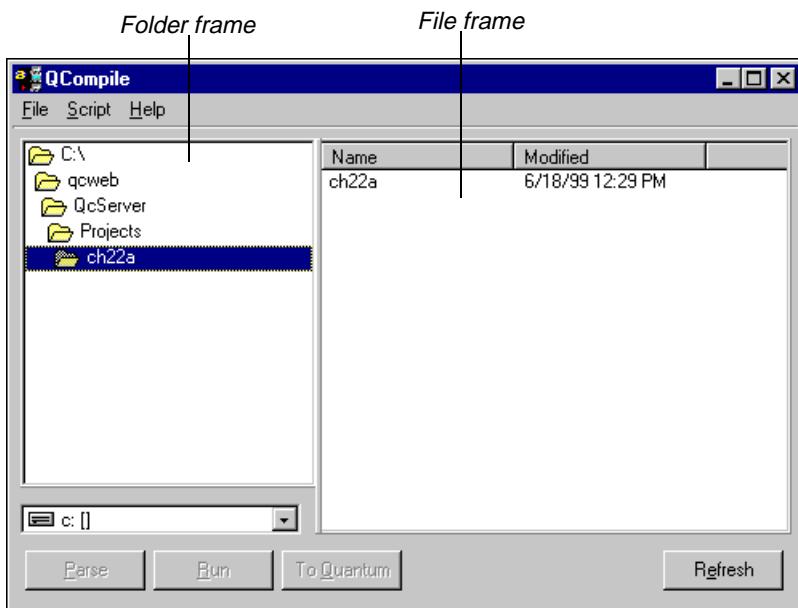
When the script parses without errors, QCompile generates an executable version of it. This is known as the interviewing program. In Quancept CAPI, the interviewing program is stored in a file called *project.exe*, whereas in Quancept Web it is called *project.sif*.

Starting QCompile

To run QCompile, double-click on the QCompile icon on your desktop:



You'll see the main QCompile window. This is where you select the project on which you want to work and the task you want to perform.



When you start QCompile, only the Refresh button is available. When you select a script, the Parse button becomes available and, if you have parsed the script before, the Run (Quancept CAPI only) and To Quantum buttons.

The main menu contains three items:

- File — leave QCompile.

- Script — parsing and running a script and access to QCompile's options. By setting options you can control such things as the way QCompile parses and compiles scripts.
- Help — version information about QCompile.

Selecting a project

Whatever you want to do with QCompile, your first step is to select a project.

To select a project:

1. In the Folder frame, select the folder that contains the script.

As you select folders, the File frame lists the files in the selected folder. QCompile lists only those files that have no extension or a .qqc extension (script files created by Quanquest).

2. In the File frame, select the script by clicking on it once.

As you select a file, the Parse button becomes available. If the script has already been compiled and there is a *script.exe* or *script.sif* file in the folder, the Run (CAPI only) and To Quantum buttons also become available.

Parsing and compiling a script

You use QCompile to *parse* and *compile* a script into an executable interviewing program. Parsing a script means checking it for syntax errors, such as incorrect syntax or a missing parenthesis ')'. A script that contains serious errors cannot be compiled, although certain errors will only generate a warning in which case you can choose to carry on to the compilation stage. The parser also creates certain files that the compiler uses to create the interviewing program.

-
- ☞ For information on setting options for parsing and compiling scripts, see 'QCompile options' below.
For information on the files created by QCompile, see section 38.1, Files created by the parser.
-

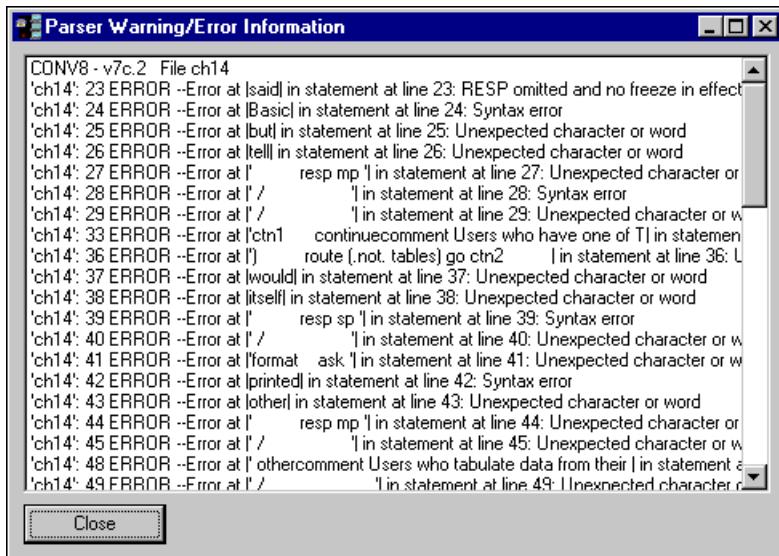
To parse and compile a script:

1. Select the script.
2. Click the Parse button.

A message box appears saying 'Parsing...'. Then if the script parses perfectly the message changes to 'Compiling...'. If there are no errors, QCompile creates the interviewing program file.

If the parser finds inconsistencies or other minor problems in your script that generate warning messages, QCompile offers you the choice of continuing and compiling the script or of canceling the compilation.

If the parser finds errors in your script, QCompile cancels the run and displays a list of error messages. For example:



This message box gives you details of errors found by the parser and the line number at which it became aware of the error. This will either refer to the line that contains the error or one or two lines after the error occurred.

Do not be alarmed if you see a lot of error messages listed in this message box. Sometimes an error in one line invalidates a number of the following lines as well. If you parse the script again after correcting some mistakes, you may find that you have reduced the overall number of errors dramatically.

-
- ☞ When correcting errors, it is good practice to work from the top to the bottom of the script, reparsing frequently to see the results of your corrections.
-

If you want to print this listing or refer to it later as you correct the script, you can find it in the file *project.err*.

Continue correcting any errors and parsing the script until it parses without any errors.

-
- ☞ For information on error messages, see section 32.5, Parser error messages.
-

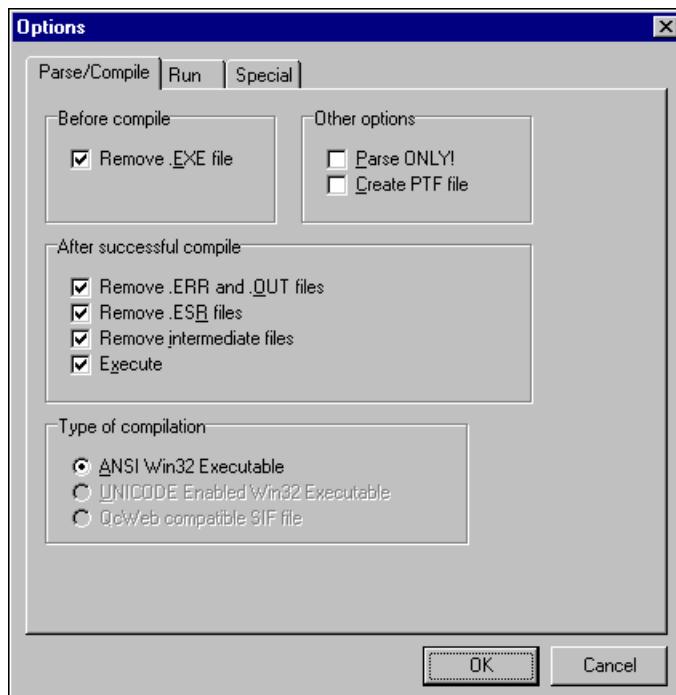
QCompile options

Before you parse and compile the script you have selected, you may want to set some of QCompile's options. These allow you to control such things as the way the program parses a script and, in Quancept CAPI, the type of data you use when you test the script and the appearance of the interviewing program's screen.

To open the Options dialog box:

- From the Script menu, select Options...

The Options dialog box appears (options on the Run and Special tabs are not applicable to Quancept Web).



The options you see selected in this picture are the QCompile defaults for the Parse button in Quancept Web. You can change them by selecting or cancelling the check boxes as necessary.

The Parse/Compile tab

The Parse/Compile tab allows you to control the way QCompile parses and compiles a script to form an interviewing program.

Type of compilation

QCompile can parse and compile scripts that use a range of character sets. By default, it expects to find only characters from the ANSI character set, that is, standard Latin characters, such as a, à or á. This is the character set used by Windows 3.1 and Windows 95.

-
- ❖ If your script contains only standard English or European characters but was written using a DOS editor, such as Edit, you need to select QCompile's DOS character set option as described in 'The Run tab' later in this chapter.
-

 For Quancept CAPI, you must always select 'ANSI Win32 Executable'.

 For Quancept Web you must always select 'QcWeb compatible SIF file'.

If you have both Quancept CAPI and Quancept Web installed, it is advisable to check the compilation setting whenever you swap between CAPI and Web to avoid compiling for the wrong variant.

To select a Parse/Compile option:

1. Make your selections on the Parse/Compile tab:

Select	To
Remove .EXE file	Delete any existing exe file before compiling. If you choose to remove the file and parsing and compilation fails, you will not be able to go back and run the old version of your script. Applicable to Quancept CAPI only.
Remove .ERR and .OUT files	Delete the err and out files from the project directory after compiling. These contain parser and compilation messages.
Remove .ESR files	Delete the esr files from the project directory after compiling. These contain C++ code generated by the parser.
Remove intermediate files	Delete intermediate files created by the parser. (QCompile creates an acl file for use by the flowcharting program, Quanflow. Do not select this option if you are going to use allCLEAR.)
Execute	Execute the script automatically after parsing. Applicable to Quancept CAPI only.
Parse ONLY!	Parse the script without compiling it.
Create .PTF file	Create a ptf file for use with Qolyglot. If you are not planning to translate your script, you can ignore this option as selecting it causes the parser to run more slowly. Applicable to Quancept CAPI only.
ANSI Win32 Executable	Create a CAPI interviewing program for use on an ordinary PC.
UNICODE Enabled Win32 Executable	Create a CAPI interviewing program that works on PCs that use Unicode characters. Not implemented.
QcWeb compatible SIF File	Create a program file for Web interviewing. You must always select this option for Quancept Web scripts.

2. Click OK.

The Run tab



The compiled interviewing program, project.exe, has many command-line parameters that determine precisely how it is to run. For example:

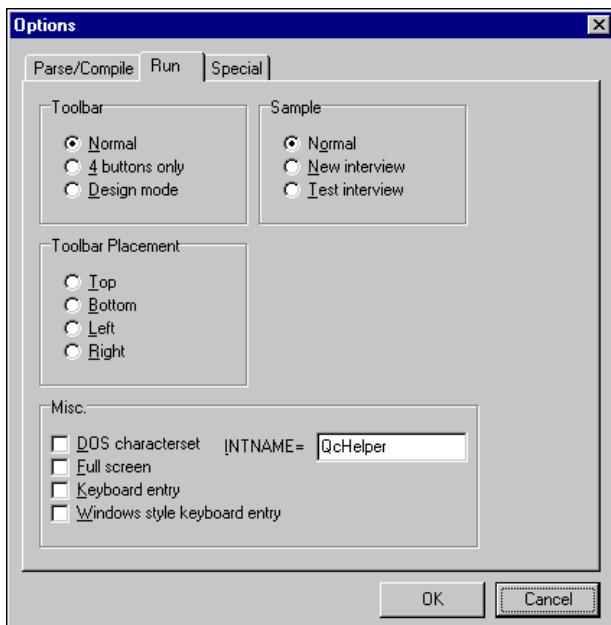
```
intprog.exe /design
```

runs the interviewing program and displays buttons for designing the layout of the screen, whereas:

```
intprog.exe /design /keyboard
```

runs the interviewing program, displays design mode buttons and activates the keyboard interface so that the interviewer can type responses on the keyboard rather than selecting them with the mouse.

Not all command-line parameters are appropriate for scriptwriters testing scripts, so the Run tab provides you with an easy way of running test interviews without having to remember the syntax of each command-line parameter:



If you want interviewers to use the same settings as those used for testing the script, these settings must be defined in the project's ini file so that the CAPI management system, qcms, can pass them to interviewers working on the project.

The DOS character set

If you have written a script using a DOS editor such as edit, you should select the 'DOS character set' option on this tab. This instructs the interviewing program to convert 8-bit DOS characters into 8-bit ANSI characters when it displays question and display text on the interview screen.

Entering responses using the keyboard

The standard method of conducting CAPI interviews is to use a mouse but you can use your keyboard if you prefer. The keyboard method is also a useful back-up if there are problems with the mouse.

The Run window shows two keyboard options. ‘Keyboard entry’ displays the answer frame at the foot of the interview screen and requires you to select responses by typing response numbers and texts in that frame. ‘Windows style keyboard entry’ allows you to select responses by tabbing to the response you want to select and then pressing the space bar to select it.

-
- ❖ The keyboard options work best if you set ShowAnswerList to 0 in the QCW section of your quantime.ini file.
-

To select a run option:

1. Make your selections on the Run tab:

Select	To
Normal	Display the standard six-button toolbar.
4 buttons only	Display the toolbar with the Stop, Previous Question, Next Question and Onresp buttons only.
Design mode	Add design mode buttons to the toolbar so that you can design the script’s layout on the interviewing screen.
Normal	Run the script and, on conclusion, have the option of starting another interview, restarting a stopped interview or stopping interviewing.
New interview	Execute one interview only and return to the Qcompile window. This option suppresses the opening dialog box with the Start Interviewing and Stop Interviewing buttons and always starts a new interview. At the end of the interview you are returned to the QCompile window. If you stop an interview you are still asked whether or not you want to restart it.
Test interview	Run the script in test mode and discard the test data.
Top	Display the toolbar at the top of the screen.
Bottom	Display the toolbar at the bottom of the screen.
Left	Display the toolbar on the left of the screen.
Right	Display the toolbar on the right of the screen.
DOS characterset	Convert the IBM character set to the ANSI character set.
Full screen	Run the interviewing program on a full screen.

Select	To
Keyboard entry	Select responses by typing numbers on the keyboard.
Windows style keyboard entry	Select responses by tabbing to them and pressing the space bar.
INTNAME=	Define the interviewer's name if you have referred to it in the script.

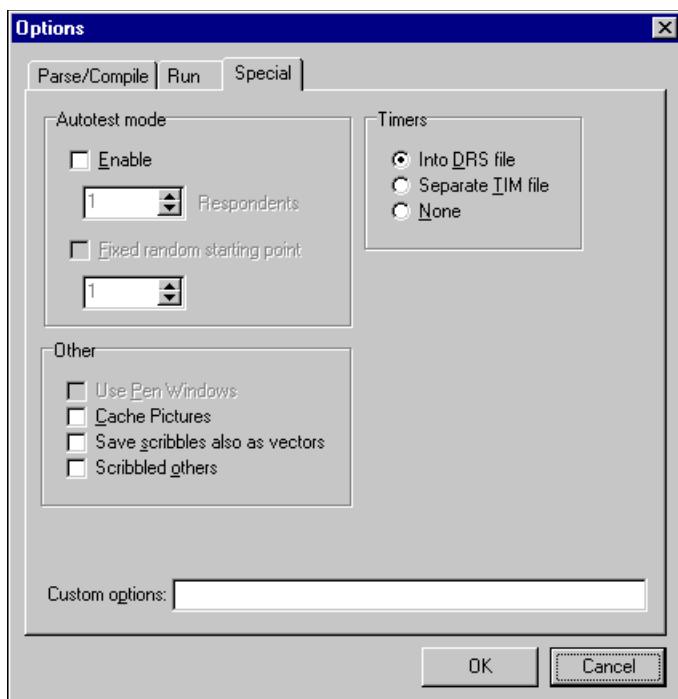
2. Click OK.

The Special tab — additional runtime options



The options on the Special tab allow you to control:

- The appearance of the interview screen
- The generation of interview data during testing
- Where and how the interviewing program finds scaled pictures
- The way in which scribbled responses are processed
- Response timings



Options for Pen Windows

The Use Pen Windows option is always available even though it is grayed out in the dialog box. It provides the following options if you have a light pen attached to your computer:

- Response fields for pictures are displayed without scroll bars, so the interviewing program displays all pictures at the same size.
- A finer pen line for pictures.

Generating response data automatically

If you have a large script with a lot of routing, you may have to run the interview a number of times selecting different responses to check for all potential errors. To help you with this, you can instruct QCompile to generate interview data automatically. So, for example, you can generate data for 500 interviews and then check the resulting data to see whether each question was answered by as many respondents as you would have expected.

When the interviewing program runs in automatic test mode, it chooses responses at random according to internal algorithms. This ensures that the responses reflect as closely as possible the responses you might expect if these were live interviews. If you want to run a set of test interviews and generate identical data for each one, you need the interviews to use the same randomization pattern for selecting responses. You do this by selecting the 'Fixed random starting point' check box and then choosing a number yourself. You would normally do this when you want to test two different versions of Quancept CAPI to ensure that the data produced by the two versions is identical.

Storing scaled pictures

When you use a picture in a script, you may decide to scale it to a smaller size than the original. If you do this, you can tell the interviewing program to store a project's scaled pictures beneath a directory called 'cache'. When the program comes across a scaled picture in an interview, it checks the cache directory for the picture. If it cannot find a scaled copy, it creates one and uses this copy in subsequent interviews. You can then delete the original copy of the picture and so save on the amount of disk space needed to store the project.

The full pathname for a scaled picture is *project\cache\size.unit\picname.bmp*. The size.unit directory means that if you scale a picture called logo.bmp to 500 pixels, the path for the file will be: *project\cache\500.0\logo.bmp*. If you have scaled the picture to 40mm, the pathname will be: *project\cache\40.1\logo.bmp*.

Recording response timings

By default, QCompile records the amount of time spent answering a question in the drs file. You can choose not to record this information or store it separately in a tim file. If you choose the tim file option, Quancept creates the timing file in the project's Results directory.

-
- ☞ For information on the format of response timing records, see 'The tim file' in chapter 38, 'File formats'.
-

Custom options

You use the Custom options field to type command-line arguments for the interviewing program. Most of these arguments correspond to check boxes on QCompile's Run and Special tabs and are not normally needed for testing scripts. Arguments for which there are no check boxes are as follows:

Argument	Description
/autojump	Tells the interviewing program to display the next question as soon as a response is chosen from a single-punched response list. This does not apply to single-punched questions displayed with <i>multitask</i> .
/L <i>number</i>	Instructs the interviewing program to display the script's text in a specific language from the ptf file. Enter <i>n</i> as 1 for the base language, or 2, 3, and so on, for the first, second, and subsequent target language in the ptf file.
/noptf	Ignore the ptf file and use texts from the qti file instead. This is the default.

-
- ☞ If you want interviewers to use the same parameters, they must be defined in the project's initialization file that is passed to the interviewing PCs by qcms.
 - ☞ For a full list of arguments see chapter 6 of the *Quancept CAPI Management Manual*.
-

To select a Special option:

1. Make one or more selections on the Special tab:

Select	To
Into DRS file	Store response timings in the drs file.
Separate TIM file	Store response timings in the tim file.
None	Discard response timings.

Select	To
Enable	Generate interview data automatically during testing. QCompile asks you to specify the number of interviews.
Use Pen Windows	Use Pen Control (if you have a light pen attached to your PC).
Fixed random starting point	Use the same random starting point for all test interviews. Choose the starting point in the box provided.
Cache Pictures	Store scaled pictures in a cache directory.
Save scribbles also as vectors	Save scribbled responses in vector format. A vector file is a binary file that provides better scalability and image compression than a bitmap file.
Scribbled others	Allow 'other' responses to be recorded as bitmap pictures.

2. Click OK.

Generating Quantum axes and table specs



If you intend to use Quantum to analyze the data you collect during interviewing, you can save yourself or a specwriter time by instructing QCompile to create a basic set of axes and table definitions from the questions in the script.

When you do this, QCompile creates files with the following extensions:

qax	axes definitions
tab	table definitions
run	run definitions

☞ You can also create Quantum axes and table definitions using Qwincode.

If your questionnaire contains questions with open-ended or other specify responses, it is a good idea to wait until coding has finished, or at least until the code frames have been built, before creating Quantum specs. In this way your specs will contain coding information from the code frames you created for those questions.

☞ For more information on Quantum spec files, see the *Quancept Utilities Manual*.
For information on creating Quantum specs using Qwincode, see the *Quancept Coding Open Ends Manual*.

You need to parse a script successfully before you can produce Quantum axes and table definitions.

To generate Quantum axes and table definitions:

- With a script selected, click the To Quantum button.

32.3 Reparsing a CAPI or Web script once coding has started



When you parse a script QCompile generates a qdi file which creates a data map allocating cards and columns for questionnaire responses. You use Qwincode to allocate codes to the open-ended responses, in the form of a code frame which is written into the qdi file. If you change the script once coding has begun, either by adding, removing or altering a question, you may change the number of columns required by the data map. If you have used Qwincode to define your code frames then the coding you have already done based on the old qdi may no longer be valid for the new qdi, and you will need to merge the code frames from the old qdi into the new one.

If you have edited the qdi file with Qwincode

If you have updated the existing qdi file by creating code frames for the project with Qwincode and you want to retain these code frames, you need to transfer information from the existing qdi file to the new version. See ‘Copying code frames between qdi files’ below for a description of how to do this. During this procedure, you will be asked to edit the qdi file and keep only those sections of the file that contain code frames that have been created in Qwincode. To help you identify the sections you need to keep, we will first look at the structure of the qdi file.

The structure of the qdi file

The qdi file is divided into a number of sections, with the beginning of each new section indicated by a section type keyword. Section types that appear in the file are:

- File — only one section of this type exists; it describes the qdi file version
- Data — several data sections appear in the file, one for each ‘data item’ that appears in the script

The qdi file regards anything for which data can potentially be stored as a ‘data item’ and describes each item in detail within its own data section. Each section begins with the keyword DATA and its contents are enclosed by square brackets. Data items that are defined in this way could be, for example, a question, the iterations of a question that appears in a loop, a *fix* statement or a *nodata* statement. Each qdi file contains a number of DATA sections, therefore. In the procedure that follows, you will be asked to copy the existing qdi file and edit it down to those sections that contain code frames created in Qwincode. To help you identify these, we will look at the structure of the DATA section in greater detail.

Let's start by looking at an open-ended question as it is defined in the script and its data item description in the qdi file. The question's definition in the script looks like this:

```
la4      ask 'In your own words, could you tell me what it is you like about
Geoffrey Chaucer?'
        resp coded (10)
```

The qdi file creates the following description for the question:

```
DATA=[  
    VARNAME=la4;  
    VARITER=[1]/[1];  
    VARTYPE=CODED;  
    TEXT="In your own words, could you tell me what it is you like about Geoffrey  
Chaucer?";  
    RESPONSES=[  
        [ 1,"",[1,37]|1 ]  
        [ 2,"",[1,37]|2 ]  
        [ 3,"",[1,37]|3 ]  
        [ 4,"",[1,37]|4 ]  
        [ 5,"",[1,37]|5 ]  
        [ 6,"",[1,37]|6 ]  
        [ 7,"",[1,37]|7 ]  
        [ 8,"",[1,37]|8 ]  
        [ 9,"",[1,37]|9 ]  
        [ 10,"",[1,38]|0 ]  
    ];  
]
```

If you have started to create a code frame for the question in Qwincode, some of the response lines will contain text strings. For example:

```
DATA=[  
    VARNAME=la4;  
    VARITER=[1]/[1];  
    VARTYPE=CODED;  
    TEXT="In your own words, could you tell me what it is you like  
about Geoffrey Chaucer?";  
    RESPONSES=[  
        [ 1,"bawdy humour",[1,37]|1 ]  
        [ 2,"topics",[1,37]|2 ]  
        [ 3,"language",[1,37]|3 ]  
        [ 4,"historic content",[1,37]|4 ]  
        [ 5,"social commentary",[1,37]|5 ]  
        [ 6,"",[1,37]|6 ]  
        [ 7,"",[1,37]|7 ]  
        [ 8,"",[1,37]|8 ]  
        [ 9,"",[1,37]|9 ]  
        [ 10,"",[1,38]|0 ]  
    ];  
]
```

Looking at the DATA section, we can see that it is divided into a number of statements. For the purposes of editing the qdi file, we are interested in the following statements:

- VARNAME — the name of the data item; that is, its label.
- VARITER — the iteration number of the data item. The first number tells us which iteration of the question this DATA section describes; the second number tells us the number of possible iterations. In the example, there is just one iteration of the question.
- VARTYPE — tells us the data item type; in this case, a coded question.
- TEXT — the question's text.
- RESPONSES — describes a response list or code frame. In this example:
 - 10 possible responses were allocated by *resp coded (10)* in the script
 - the double quotes contain the texts the coder supplies using Qwincode
 - the responses are allocated to card 1, column 37, punch numbers 1 to 9, or to card 1, column 38, punch number 0

The VARNAME and VARITER statements in combination uniquely identify a data item. You need to remember this when you are asked to edit the qdi file. For example, if a project includes an open-ended question in a loop, you will need to copy each iteration of that question for which you have created code frames in Qwincode. In the case of questions, the VARTYPE statement indicates the question type. Where VARTYPE is CODED, this indicates that a code frame may exist for the item.

There are two types of question for which you may have created a code frame in Qwincode and the way in which each question type is represented in the qdi file is slightly different. In the case of an open-ended question with a code frame, the code frame is defined within the DATA section identified by VARNAME/VARITER. In the case of a single or multipunched question with an ‘other specify’ response, the code frame is defined within its own DATA section. We’ll start by looking at open-ended questions. The following extract from a qdi file shows the code frame for an open-ended question:

```
DATA=[  
  VARNAME=la4;  
  VARITER=[1]/[1];  
  VARTYPE=CODED;  
  TEXT="In your own words, could you tell me what it is you  
like about Geoffrey Chaucer?";  
  RESPONSES=[  
    [ 1,"humour",[1,25]|1 ]  
    [ 2,"language",[1,25]|2 ]  
    [ 3,"timelessness",[1,25]|3 ]  
    [ 4,"subject matter",[1,25]|4 ]  
    [ 5,"",[1,25]|5 ]  
    [ 6,"",[1,25]|6 ]  
    [ 7,"",[1,25]|7 ]
```

```
[ 8,"",[1,25]|8 ]
[ 9,"",[1,25]|9 ]
[ 10,"",[1,26]|0 ]
];
]
```

In this example, the VARNAME and VARITER statements indicate that this is the code frame for the only iteration of a question called 'la4'. When copying code frames for this type of question, this is the only DATA section you need to copy. If the question is in a loop and you have started to create code frames for more than one iteration, you will need to copy each of the DATA sections in which you have started to create code frames. You can identify which iterations you need to copy by looking at the VARNAME statement in conjunction with VARITER.

For questions with an 'other specify' response option, the question is described within a DATA section and the code frame for the 'other specify' response is described separately in its own DATA section. The VARNAME for the 'other specify' DATA section is derived from the label of the question to which it belongs with the addition of [other] suffix.

For example, the following question:

```
read1    ask 'And which of these authors have you read in the last
           six months?'
           resp mp 'Iris Murdoch' / 'Agatha Christie' / 'Paul Theroux' /
           'Iain Banks' other (10)
```

has two DATA sections in the qdi file — one that describes the question:

```
DATA=[  
  VARNAME=read1;  
  VARITER=[1]/[1];  
  VARTYPE=MP;  
  TEXT="And which of these authors have you read in the last six  
months?";  
  OPTIONS=[OTHER];  
  RESPONSES=[  
    [ 1,"Iris Murdoch",[1,8]|1 ]  
    [ 2,"Agatha Christie",[1,8]|2 ]  
    [ 3,"Paul Theroux",[1,8]|3 ]  
    [ 4,"Iain Banks",[1,8]|4 ]  
    [ OS,"Other (specify)",[1,8]|5 ]  
  ];  
]
```

and one that describes the 'other specify' response associated with it:

```
DATA=[  
  VARNAME=read1[other];  
  VARITER=[1]/[1];  
  VARTYPE=CODED;  
  RESPONSES=[  
    [ 1,"Barbara Cartland",[1,9]|1 ]
```

```

[ 2,"Armistead Maupin",[1,9]|2 ]
[ 3,"Rudyard Kipling",[1,9]|3 ]
[ 4,"Susan Hill",[1,9]|4 ]
[ 5,"",[1,9]|5 ]
[ 6,"",[1,9]|6 ]
[ 7,"",[1,9]|7 ]
[ 8,"",[1,9]|8 ]
[ 9,"",[1,9]|9 ]
[ 10,"",[1,10]|0 ]
];
]

```

For this type of question, you need to keep both of these DATA sections.

Copying code frames between qdi files

The following steps tell you how to copy code frames from an existing to a newly created qdi file. Before you begin:

- Familiarize yourself with the structure of the qdi file by reading the preceding section, ‘The structure of the qdi file’ .
- Make a list of labels for the questions that have code frames created in Qwincode and, where relevant, their iteration numbers. This will help you identify which DATA sections you need to keep in the existing qdi file.
- Make a backup copy of the project for safekeeping; you can delete this copy once you have finished copying code frames.

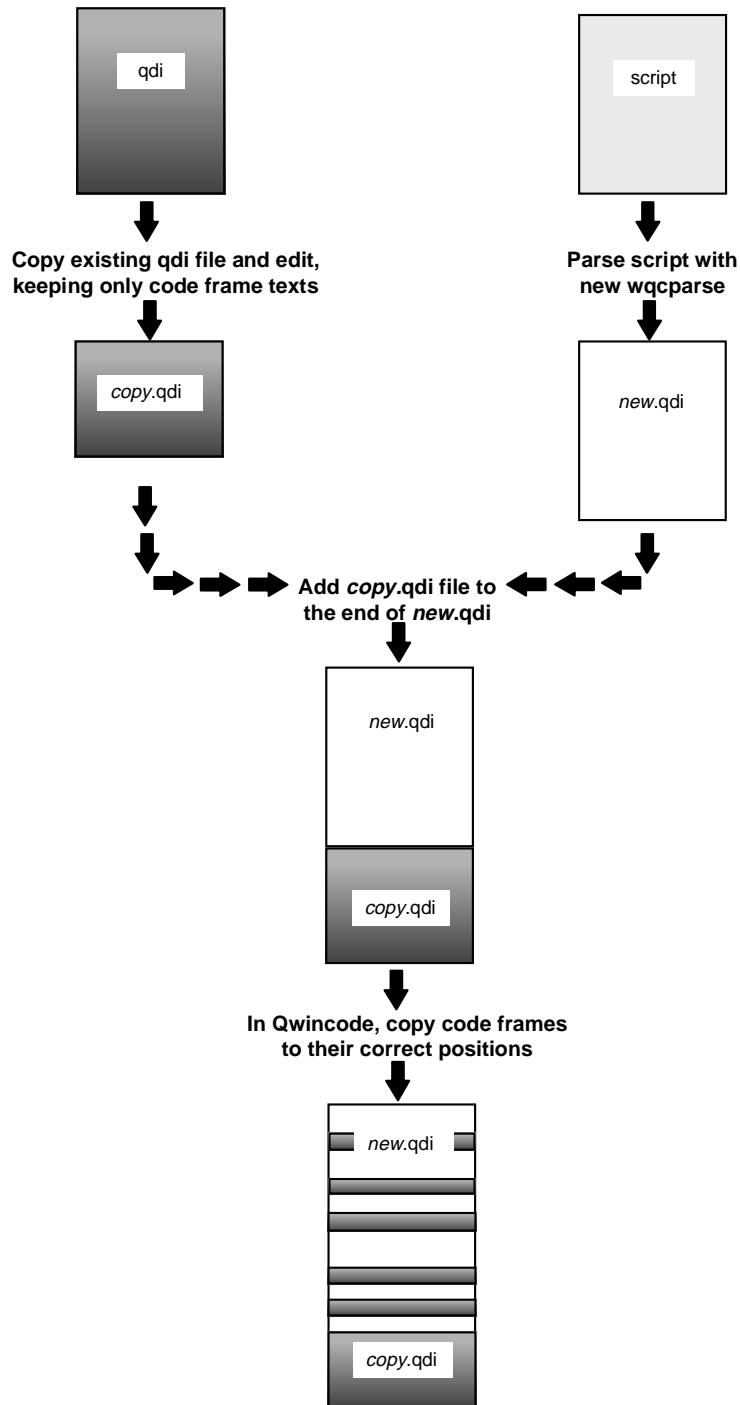
The diagram overleaf provides an overview of the steps you need to perform.

Make a copy of the existing qdi file

- Copy the existing qdi file to another directory.

From now on, we will call this version of the file *copy.qdi*. We suggest you copy the file to a directory whose pathname is *project\fix*. We will assume that you are using this directory in the following steps.

Copying code frames between qdi files



Edit copy.qdi

1. Edit *copy.qdi* by deleting all of the DATA sections except those that contain code frames that have been created in Qwincode.

Use the list of questions with Qwincode-created code frames, which you made before starting this process. Remember that:

- for questions with an ‘other specify’ response, you need to include the DATA section that describes the question itself and the section that describes the code frame you have created for its ‘other specify’ response.
 - for coded questions that are in a loop, you need to retain each DATA section that describes each iteration of that question.
 - for each DATA section you retain, make sure you do not delete its closing square bracket.
2. Rename each DATA section by editing its VARNAME statement; that is, the statement that holds the question’s label. Note that the new label you use must be unique within the qdi file and must not clash with any labels that already exist in the script or that may be used in the script if new questions are added to the script at a later date. You may find it helpful to use a new VARNAME that includes part of the original label, to indicate each DATA section’s origin. For example, you could change:

```
VARNAME=read1;
```

to

```
VARNAME=read1a;
```

and so on. Remember that this change must also be made to those DATA sections that describe an ‘other specify’. Following the previous example:

```
VARNAME=read1[other];
```

would become

```
VARNAME=read1a[other];
```

and so on.

Parse the script

- Open QCompile and parse the script.

This step creates a qdi file that contains the correct data mapping information for the script’s questions, but no code frames. We’ll call this file *new.qdi*.

Append the contents of copy.qdi to new.qdi

- Append the contents of *copy.qdi* (stored in *project\fix*) to the end of *new.qdi*.

The *new.qdi* file now contains the code frames, but at the end of the file. The next step is to move the code frames to their correct position in the file.

Copy the code frames to the correct questions

1. In Qwincode, select the project.
2. At the opening screen, select Build Code Frames.
3. In ‘Code frame manager’, copy any code frames that you added to the end of *new.qdi* to the questions to which they correspond.

For example, if ‘read1a’ contains the code frame for ‘read1’, copy the code frame for ‘read1a’ to ‘read1’. If there are several iterations of a question, copy the code frame for each iteration to its original.

The *new.qdi* file now contains accurate data mapping information for each question’s responses and correctly placed code frames from the original qdi file. These code frames also appear at the end of the file since you have simply copied them to the correct position, rather than moving them. If you wish, you can leave these duplicate versions of the code frames in the file. They will not affect any Quantum data file you produce for the project using QCWinfmt. However, if you intend to update the script in the future by adding new questions, you need to be aware of the label (VARNAME) of each extra code frame section in the qdi file. These labels must remain unique. To ensure that you do not duplicate a label by accident, you can remove the extra code frames from the end of the qdi file at a convenient time.

32.4 Parsing and compiling mrInterview scripts with QCompile

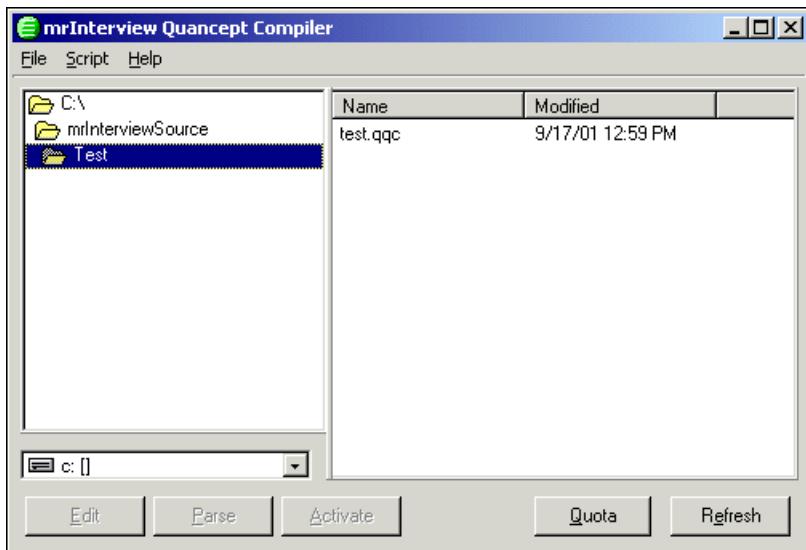
 The version of QCompile that comes with mrInterview is similar in appearance to the QCompile for Quancept CAPI and Quancept Web, and some of its functionality is the same. You use QCompile to:

- Parse a script and compile it into an interviewing program.
- Edit a script to correct errors or add new functionality.
- Activate a project to prepare it for interviewing.
- Check multilingual information in the questionnaire definition (.mdd) file.

Starting QCompile

To run QCompile, select Programs, SPSS MR, mrInterview, QCompile from the Start menu.

The program starts and displays its main window:



Use this window to select the project on which you want to work and the task you want to perform.

When you start QCompile, only the Refresh and Quota buttons are available. When you select a script, the Edit and Parse buttons become available and, if you have parsed the script successfully before, the Activate button is available too.

-
- ❖ If QCompile fails with the message ‘Miscellaneous Elan-related error’ this means that the user account that was used to install mrInterview does not have Power User or Administrative rights.
-

Selecting, parsing and compiling a project

The procedures for selecting, parsing and compiling a project are the same in the mrInterview version of QCompile as in the CAPI and Quancept Web versions. The only differences are as follows:

- You can edit the script to correct errors by clicking the Edit button.
- After a successful compilation a file called *projname_Activate.zip* is created containing the project’s sif, mdd, mqd, htm and xml files. If the project has Sample Management quotas, the zip file also contains the mdd file that Quota Setup creates to hold the Sample Management variables. This file is not used by any of the mrInterview applications, but you may find it useful if you want to give someone a copy of all the files that are used for interviewing on a project. It is also useful as a back-up in case you accidentally delete a file.

- ❖ If you have a very large or complex script, the compilation process may time out before the script has been fully compiled. Refer to 'Increasing Time-outs for the Quancept Compiler' in the Installation Instructions for information on how to resolve this problem.
 - ❖ See 'The mdd file' in chapter 38, 'File formats', for further information on the questionnaire definition file.
-

Activation

Once you have compiled your script into a sif file and have a questionnaire definition (mdd) file, you are ready to prepare the project for interviewing. This involves creating a separate project directory in which the files associated with interviewing can be created, associating sample with the project, and defining which web pages to use for standard things such as the start and end of an interview.

-
- ❖ If you use Quancept Compiler to parse and compile a questionnaire, it is recommended that you also activate the questionnaire using Quancept Compiler. Similarly, if you create a questionnaire using DimensionNet activities, it is recommended that you parse, compile and activate the questionnaire in DimensionNet. This ensures that all the files that the project needs will have the correct names and be in the correct locations.
-

Normally, the Activate dialog box is displayed as soon as the compilation stage has finished, but you can run it at any time simply by selecting a project and clicking the Activate button on the main Quancept Compiler window.

-
- ❖ If you change the script, you must always reparse and reactivate. If you change a project's template file, you need only reactivate. If you change a cascading stylesheet, there is no need to reparse or reactivate; simply copy the new stylesheet into the appropriate location, and stop and restart your browser.
-

In order to activate projects you must be logged in to the Activate component that displays the Activate dialog box and runs the activation process. Quancept Compiler attempts to log you in to the Activate component automatically, but if this fails it displays the following login dialog box:



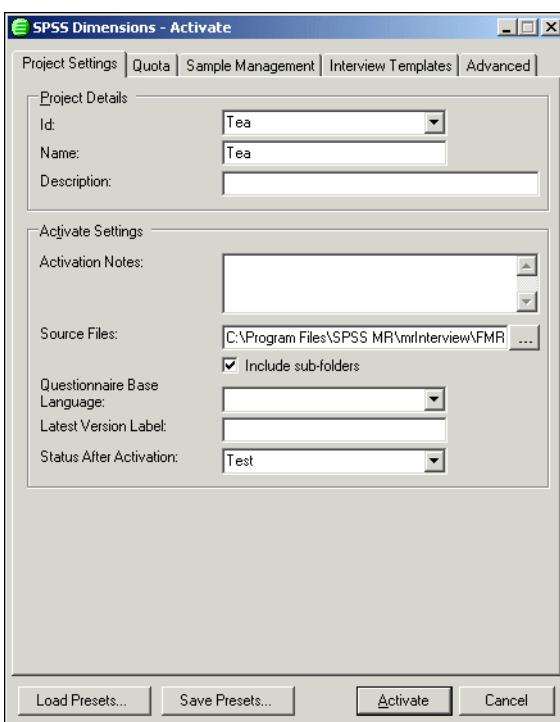
You can log in using your DimensionNet user name and password or your standard Windows account. Make the necessary selections and click Login.

A message reports whether or not you are successfully logged in.

The Activate dialog box opens on the Project Settings tab. You can activate the project at any time by clicking Activate or cancel it by clicking Cancel.

Project Settings tab

The Project Settings tab is where you enter general information about the project and where and how it is to be activated:



The Project Details frame shows details about the project currently selected in Quancept Compiler: you should not need to change these fields. (You can choose a different project here, but it is easier to close the dialog box and choose a new project on the main Quancept Compiler window, as this will automatically update the other fields on the Project Settings tab with the correct information.)

The Activate Settings frame shows the following information:

Activate Notes. Background information about the project that you want to save in the project database for reference by other users. You can leave this box blank.

Source Files. The location of the source files for this project. The *Include sub-folders* check box is not applicable to mrInterview 2.3 and its setting is ignored.

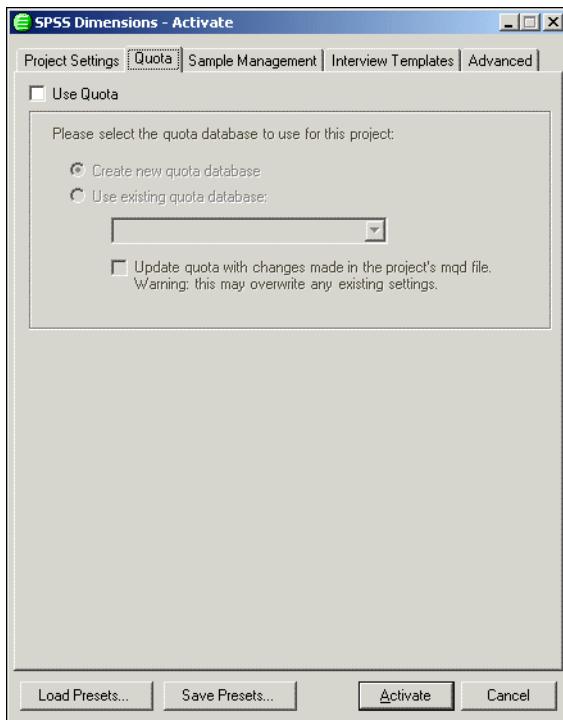
Questionnaire Base Language. The base language for the questionnaire. With multilingual questionnaires, the language in which you write the questionnaire automatically becomes the base language for that script. If the questionnaire does not specify the language in which it is to run, and the information cannot be obtained from the participant record, the interview will run in this language. Once you start translating a script, other languages are added to the questionnaire definition file and you may want to select one of those languages as the base language for the questionnaire. The language list contains only languages that are present in the questionnaire definition file. If the computer's default language does not appear in the questionnaire definition file, the language list defaults to US English.

Latest Version Label. The version label to assign to the version of the questionnaire that will be created during activation.

Status after activation. The status that you want the project to have once it has been activated. Choose *Test* if the project is available for testing (any data collected will be flagged as test data), *Active* if the project is available for live interviewing, or *Inactive* if the project cannot be used for test or live interviewing.

Quota tab

Select the Quota tab to specify Quota Control parameters for this project.



Use Quota. Select this option if the project uses Quota Control. (Quota control requires a quota definition (mqd) file to exist in the project folder. If no such file exists, a warning message to this effect is displayed on the tab.)

Create new quota database. Select this option if your are activating the project for the first time, or if you are re-activating but this is the first time that you have had quota information available. The exception is when your project shares quotas with another project. In this case, if the shared quota database already exists, select the quota from the list instead.

When a project uses quota control and you activate it for the first time, the activation process creates a new quota database for the project using the information in the project's quota definition (mqd) file. The quota database is a set of tables whose names start with QUOTA and which the activation process creates inside the project database. They contain definitions of the quota groups and their targets and, once interviewing starts, counts of completed, pending, and rolled back interviews for each group. The quota definition (mqd) file is the file that the Quota Setup program creates when you save the quota definitions and targets. The activation process uses it to determine the structure and content of the quota database it is to create. The mqd file is not used during interviewing.

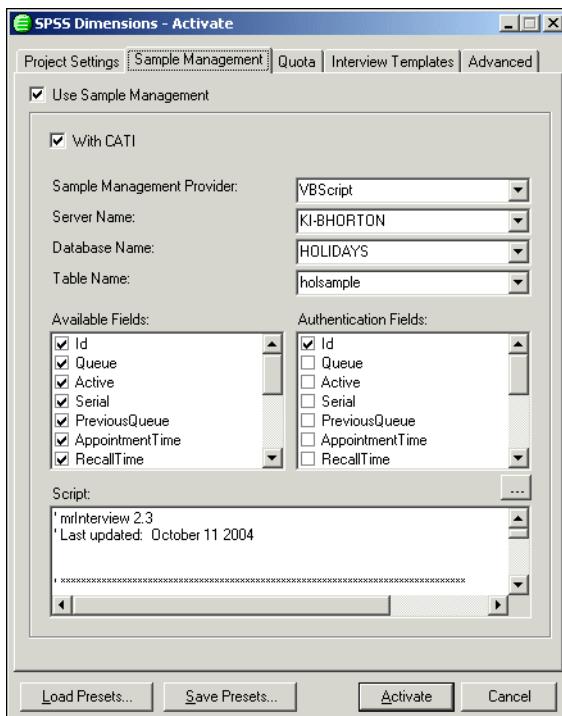
Use existing quota database. If the project has been activated with quota before, or it shares an existing quota database with another project, select this option and then choose the quota database from the drop-down list.

Update quota with changes made in the project's mqd file. Once a quota database exists, this check box is always activated but is unchecked. If you have made changes to the mqd file, select this check box if you want these changes to be implemented in the quota database.

-
- ❖ If you have changed quotas using the Quotas activity in DimensionNet these changes will have been written to the quota database but will not appear in the project's mqd file. If you choose to activate using the mqd file, the changes you made with the Quotas activity will be lost. If you want to keep these changes, you will need to make them in the mqd file using Quota Setup before re-activating.
-

Sample Management tab

Select the Sample Management tab to specify Sample Management parameters for this project.



Use Sample Management. Select this check box if the project uses Sample Management.

With CATI. Select this check box if the project will use CATI for outbound calling.

Sample Management Provider. The tool that will be used for implementing Sample Management in this project.

Server Name. The name of the server on which the participant database is located. The drop-down list contains only those servers that are present in your current domain. If you want to use a server in another domain, you must enter its name manually.

Database Name. The name of the participant database. The drop-down list displays the names of databases that you have permission to use and that exist on the chosen server.

Table Name. The table in this database that contains the participant records for this project. The drop-down list displays the names of tables in the chosen database.

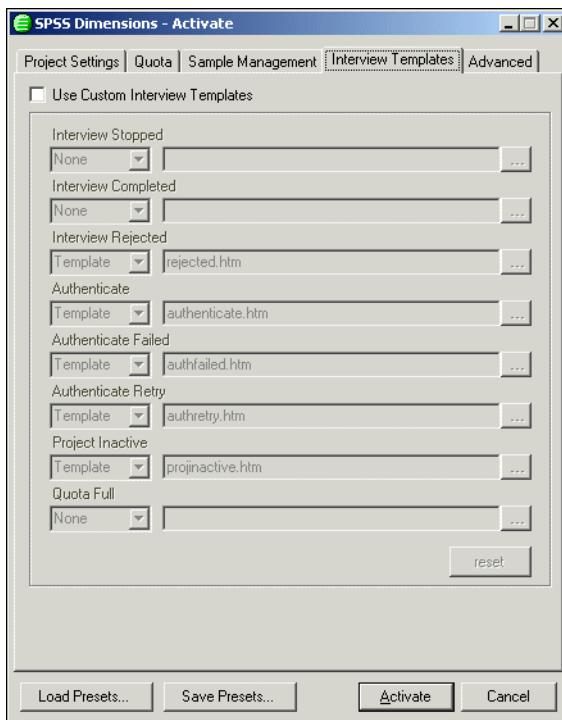
Available Fields. The fields of participant information that need to be made available to the Sample Management script. All Sample Management scripts require fields labeled Id, queue, active and serial to function properly. Select the check boxes of the fields you want to make available to the Sample Management script.

Authentication Fields. The fields that are to be used for authenticating the participants making inbound calls. If you need to be able to select specific participant records, you should select the Id field because this is a key to the database and is guaranteed to contain a unique value for each record. If you authenticate on a field that may contain non-unique values, the Sample Management system will select the first record whose value in that column matches the values specified in the Sample Management script. Select the check boxes of the fields you want to use for authentication.

Script. The Sample Management script for the project. Either type the script directly into the box or click the button with the three dots on it to load the file containing the script. You may choose either a sam file or an mrs file.

Interview Templates tab

mrInterview comes with a number of customizable template files that you can use for standard events such as authentication, stopping interviews and end of interview. You can find the default files in the c:\Program Files\SPSS MR\mrInterview\Projects folder. Select the Interview Templates tab if you want to use your own templates in place of some or all of the defaults.



Use Custom Interview Templates. Select this check box if you want to use your own templates for any of the listed events.

You can specify all pages except Authenticate and Authenticate Retry as templates or URLs. For these two pages we strongly recommend using only templates, as URLs can result in an interview having two connection IDs.

If you specify a page as a template, the Browse button (button with three dots on it) is enabled so that you can select the file rather than having to type its name into the dialog box. Template files must be present in the project's source directory, or in c:\Program Files\SPSS MR\mrInterview\Projects, or in a shared location on your web server. In all cases, you specify the file using a simple filename not a path name. When the project is activated, templates that exist in the project's source directory will be copied to the project's directory in c:\Program Files\SPSS MR\mrInterview\Projects. When interviews take place, the interviewing program will look for the templates first in the project-specific directory and then in the main Projects directory.

Interview Stopped. The page to display when the participant stops an interview or the interview is stopped by a *stop* or *signal* statement in the script. There is no default template but mrInterview itself displays 'End of interview. Thank you for your participation'.

Interview Completed. The page to display at the end of the interview (that is, when the participant has answered all relevant questions in the questionnaire). There is no default page, but mrInterview itself displays 'End of interview. Thank you for your participation'. Note that if the interview ends with a *display* statement, this text is displayed as the last page of the interview instead.

Interview Rejected. The page to display when a participant fails authentication and no retry prompt is required, for example, when the participant fails quota control. The default is a template called rejected.htm that displays the message ‘Thank you for your interest in participating in this survey’.

Authenticate. The page to display when a project uses Sample Management and you need to verify that the person taking the interview is a member of the participant group. The default is a template called authenticate.htm that displays the message ‘Please enter your authentication information’.

Authenticate Failed. The page to display when authentication fails. The default is a template called authfailed.htm that displays the message ‘Your authentication information is incorrect’.

Authenticate Retry. The page to display when authentication of a prospective participant against the participant database fails, and you want the participant to re-enter the authentication details. The default page is a template called authretry.htm that displays the message ‘The authentication information you have entered is incorrect. Please try again’.

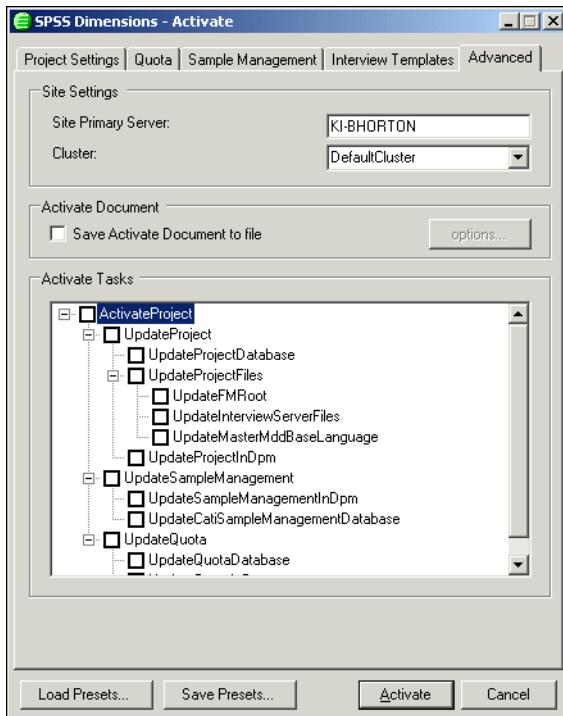
Project InActive. The page to display when the participant attempts to start an interview for an inactive project. The default is a template called projinactive.htm that displays the message ‘Please come back later’.

Quota Full. The page to display when the participant belongs in a quota cell whose target has already been met. There is no default page.

If you make a lot of changes and you want to return to the default settings, click Reset.

Advanced tab

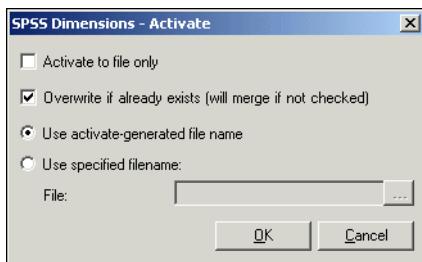
The Advanced tab provides access to a number of additional options that provide you with a more detailed level of control over the activation process.



Site Primary Server. The name of the primary DPM server. This is the server that controls all aspects of project publication, and it defaults to the server named in the Quancept Compiler Options dialog box.

Cluster. The cluster on which to activate the project. If you are activating a project that has already been activated — for example, if you have changed and recompiled the questionnaire — you must activate it on the same cluster as you used before.

Save Activate Document to file. Select this option if you want to save the activation specification to a file. This is a quick and easy way of creating an Activate Document that can be modified and used for activation from the command line. Click the Options button to open a secondary dialog box on which you can specify how the file should be created:



Select *Activate to file only* to create the Activate Document from the settings in the dialog box without running the activation process. Uncheck *Overwrite if already exists* if you do not want the activation process to overwrite any existing project files.

Activate normally creates Activation Document files with names of the form project_yyyymmddhhmmss.xml. The files are created in the project's subfolder inside your User folder. To create the file with a different name and/or location, choose *Use specified filename* and then either type the filename or pathname in the File box or use the Browse button to the right of the text box to specify the file's location and name.

Activate Tasks. Use this frame to specify which tasks the activation process should perform. Once a project has been activated for the first time, this is an ideal way of reducing the time required for subsequent activations when only certain parts of the activation process need to be run. For example, if the project does not use Sample Management or Quota Control you can skip these tasks.

Tasks are displayed in a hierarchical structure which can be expanded by clicking the + symbols at the start of each line. Select the check boxes of the tasks you want the activation process to perform.

ActivateProject. Run the complete activation process. This is the default and only option the first time you activate a project, even if that project does not use Sample Management or Quota Control.

UpdateProject. Run all aspects of the Update Project part of the activation process.

UpdateProjectDatabase. Update the information held for this project in the project database.

UpdateProjectFiles. Update the project files (project_name.xxx) for this project.

UpdateFMRoot. Copy files from FMRoot\Shared into FMRoot\Master.

UpdateInterviewServerFiles. Copy files from FMRoot\Master to the Projects folder on all Interviewing servers.

UpdateMasterMddBaseLanguage. Update the master base language for this project to be the one specified in Questionnaire Base Language on the Project Settings tab.

UpdateProjectInDpm. Update the basic project information held for this project in DPM. If you are activating the project just to change its status, selecting this task and no others makes the change in the shortest possible time.

UpdateSampleManagement. Run all aspects of the Sample Management part of the activation process.

UpdateSampleManagementInDpm. Update the Sample Management information held for this project in DPM.

UpdateCatiSampleManagementDatabase. Update the CATI Sample Management information held for this project in DPM.

UpdateQuota. Run all aspects of the Quota Control part of the activation process.

UpdateQuotaDatabase. Update the project's quota database with information about the project's Quota Control requirements.

UpdateQuotaInDpm. Update the Quota Control information held for this project in DPM.

Saving and loading presets

Presets are collections of predefined settings for a project that you can load to populate fields in the dialog box automatically. They are handy time-savers if you regularly use the same settings for a number of projects.

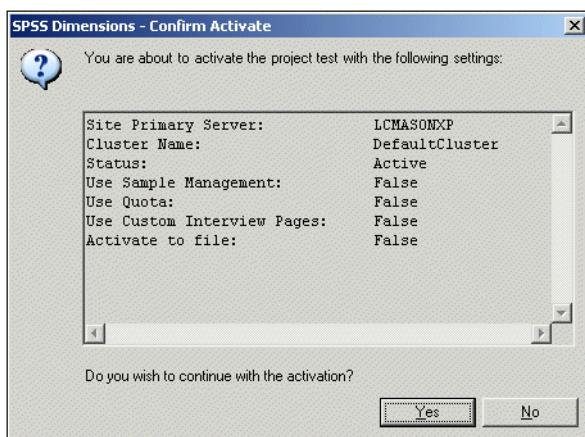
To save presets, make your selections on the various tabs of the dialog box and then click Save Presets. When prompted, enter a name that describes the settings you have chosen and click OK. The saved preset will store the current setting of every option on the dialog box, *including the project name, description, and source file location*.

To load presets into the dialog box, click Load Presets. A dialog box is displayed on which you can choose the name of the preset group you want to load from a drop-down list. Click OK to populate the Activate dialog box with these settings. Any selections that you have already made in the dialog box are overwritten.

-
- ❖ It is important to choose descriptive names for presets as they are saved internally to the Activate procedure rather than in files, and there is no way of checking what settings belong to each preset without loading them.
-

Activation process

When you click Activate, a summary of the activation requirements is displayed and you are asked to confirm them:



Click Yes to start the activation process. A message box with a progress bar is displayed while the following tasks are carried out.

First, the activation process copies the project's htm, mdd, mqd and sif files, as well as any image files, from your machine into subdirectories of the FMRoot\Shared and FMRoot\Master directories on the File Management server. If an mdd file already exists on the server and the project's status is Active, the program tries to merge the newer version of the file into the existing file. If the merge fails or the project's status is Test or Inactive, the newer file overwrites the older one. A progress box is displayed while this happens.

-
- » The activation process does not copy updated files into your DimensionNet user folder.
-

If the project cannot be activated, the message box reports the reason and suggests a solution.

Then, the activation process contacts each Interviewing server and tells it to update its project files. Each Interviewing server responds by copying the new or updated files from FMRoot\Master into its local copy of the Projects directory. Normally, the files from the File Management server overwrite the local copies, but if interviews are in progress this may not be possible, so the old files will be renamed instead. They will be deleted automatically when they are no longer in use.

-
- » The mqd file is not copied from the Master directory to the project directories on the Interviewing servers as it is not required for interviewing.
-

Starting Quota Setup from QCompile

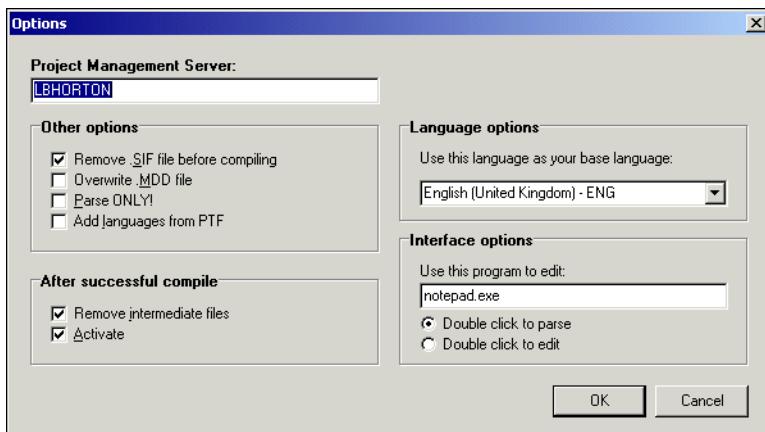
Once you have compiled a script and created a questionnaire definition file, you can use the Quota Setup program to define the quota groups and targets for the project. You can either do this outside QCompile by stepping down through the Start menu and choosing Quota, or you can start it directly from QCompile.

When you click the Quota button in QCompile, Quota Setup starts and automatically opens the file for the current project. If the project already has a quota definition (mqd) file, Quota Setup opens this file, otherwise it opens the project's questionnaire definition (mdd) file.

When you close Quota Setup, you are returned to QCompile ready to activate the project in the normal way.

QCompile options

There are a number of options that you can set to control how QCompile works. You set these options in the Options dialog box. The default settings are shown below:



To set options:

1. From the Script menu, choose Options.

The Options dialog box opens.

2. In Project Management Server, type the name of the computer on which projects will be published and on which interviewing will take place.
3. Select the options you want to use and cancel those you do not:

Select	To
Remove .SIF file before compiling	Delete an existing <i>project.sif</i> file from the project's source directory before compiling.
Overwrite .MDD file	Overwrite an existing mdd file so that the new file contains only the current version of the questionnaire, rather than appending a new version of the questionnaire to the existing file. You will probably want to select this option while you are building and testing a questionnaire, and then cancel it when you compile the version of the questionnaire that will be used for live interviewing.
Parse ONLY!	Parse the script without compiling it.
Add languages from PTF	Add translations from a Quancept .ptf file to the mrInterview questionnaire definition file (only for projects that were originally written for Quancept interviewing).

Select	To
Remove intermediate files	Delete all intermediate files created by the parser so that only the sif, questionnaire definition, quota definition and template files remain in the source directory.
Activate	Run the activate wizard automatically after a successful compilation.
Use this program as your base language	Select the language that will be used as the base language in a multilingual questionnaire. This is the language in which the interview will be displayed if the script does not set a language itself. The default selection is the language in which your computer operates. The language list contains only languages that use code page 1252. If your questionnaire contains text in a language that does not use code page 1252, you can make that language appear in the list by naming the code page using the registry key HKEY_LOCAL_MACHINE\SOFTWARE\SPSS\mrInterview\1.0\QCompile\Options\SupportedCodePages. Type the numbers of the code pages you want to support separated by semicolons. If you compile a questionnaire whose base language differs from the base language that is set in the Options dialog box, the compiler warns that it will overwrite the texts in the questionnaire definition file's base language with the texts in the script, and asks whether you want to continue. Choose No if you want to change the compiler's base language before compiling. For example, if you have a script whose base language is English and that has been translated into German, the questionnaire definition file will contain texts in both languages. If you recompile the script on a machine whose operating language is German and you do not set the reset the questionnaire's base language to English on the Options dialog box, the compiler will ask whether you want to overwrite the English texts in the questionnaire definition file with German texts. If you answer Yes, the base language will still be set to English but instead of English texts you will have German texts. This situation is most likely to happen if you receive a script from abroad or from someone whose normal working language is different to your own.
Use this program to edit	Associate a program with QCompile's Edit button. Type the program's name in the box.

Select	To
Double click to parse	Be able to parse a script by double clicking on it in the right-hand frame.
Double click to edit	Be able to edit a script by double clicking on it in the right-hand frame.
4. Click OK.	

Checking language codes in the questionnaire definition file

When a questionnaire is translated using mrTranslate, the original texts in the base language and the translations are flagged with a three-character code representing their language. For example, an English translation may be flagged as ENU (while a Danish translation will be flagged as DAN). If you have not been responsible for the translations, you may want to check that the language codes you have used in the script are the same as those that have been used in the questionnaire definition file. If the codes are different, you can change your script so that the language codes match the questionnaire definition file and then recompile it.

The Languages option displays the names and three-character codes of the languages in the questionnaire file:



To check language codes in the questionnaire definition file:

- From the File menu select Languages...

32.5 Parser error messages

When the parser finds mistakes in your script it generates error messages. If the error can be pinned down to a specific keyword in the script, a message of the form:

— Error at *keyword* in statement at line *n*

is displayed, where *keyword* is the word at which the error was found. It may be that the keyword was simply misspelled — for example, you typed *rsp* instead of *resp* — or that there was something missing immediately before the keyword, such as a missing single quote at the end of the question text.

If the error is less specific, such as an unclosed loop, then one of the following messages is printed. (Messages are listed in alphabetical order by the first word.)

Arithmetic set to wrong question type — *var_name*

You have a statement that assigns an arithmetic value (either a number or a variable with a numeric value) to a variable which is not an integer or real variable. If you are assigning the value to a variable which already exists, check back in your script to see what type you gave it when it was defined.

Column statement within loop

You have used a *col* or *column* statement inside a loop.

Callfunc has an incorrect number of arguments

There are too many or too few items inside the parentheses.

Column number *n* not allowable

You have specified a column (with the keywords *col* or *column*) less than 8 or greater than 80.

Cannot open EXIMPORT file *desc_filename*

You have specified a GNU database definition file that doesn't exist. Perhaps you have spelled the name of the definition file incorrectly on the *eximport* statement.

Else before If

You have either forgotten to start your condition with an *if* statement or there was an error in that statement causing it to be ignored.

For-next grouping already in effect

The loop whose responses you are trying to group is part of a larger loop which is already grouped. If the outermost loop of a nested set of loops is grouped, the data for all the inner loops is grouped automatically.

from_value* TO *to_value*, *from_value* is greater than *to_value

Ranges must be entered with the minimum value before the maximum value. You have entered the values the other way round.

grouped must be on outermost for of nested for loops

When loops are nested, the columns allocated to the questions in those loops must be all grouped or all ungrouped: a combination of the two is not allowed. Thus, if grouping is required for one loop, all the others in the nested set must also be grouped, and the grouped keyword must appear on the outermost *for* statement. You have omitted grouped from the outermost *for* statement and have placed it on the *for* statement of an inner loop.

If – else block not closed

The statements following an *if* or *else* are not terminated by a } character.

Illegal syntax in a test

You have mistyped a logical expression. Check that you have the correct punctuation between items in the expression and the correct number of parentheses.

Illegal response list type

Response types are *sp*, *mp*, *num*, *real*, *coded*, *dbase* and *list*. You have either misspelled one of these or have used something else altogether.

Illegal use of label

You have used a label incorrectly. Labels may appear at the beginning of a statement or after *go* or *goto*. If the label is a question name, it may be used in logical expressions where you want to test the answer to that question, and in assignment statements if you need to overwrite the current response.

Include file *file.name* not readable

You have either called the include file by a different name, or forgotten to create it, or have created it so that you do not have permission to read it.

Incorrect data base type *type* specified

You have specified an invalid GNU database type on your *eximport* statement. The type may be *gdbm* or *gdbmfix*.

Incorrect line *fieldname field_number 1 field_type in EXIMPORT file desc_filename*

Incorrect line *fieldname field_start_column field_length field_type in EXIMPORT file desc_filename*

You have incorrectly defined the GNU database description file named *desc_filename*. The error message shows you the whole offending line from the description file. The error can be in any of the following places:

- On the second field, *field_number* in a type *gdbm* database or *field_start_column* in a type *gdbmfix* database. The entry must be numeric and must be 1 or greater.
- On the *field_type* field, the field type can only be one of the following: **t** (text), **r** (real numbers) or **n** (numeric)

Item can only be a defined list

You are using a statement that expects to read a list which has been defined with a *define* statement. The variable you are using is not a list.

Keyword used inappropriately

You have used a keyword in a place it is not supposed to be used.

Label *label* used but not defined

You are testing a variable which you have not previously defined (for example, q1 when there is no statement assigning a value to q1) or you are trying to skip to a statement that does not exist (for example, goto zz when there is no statement named zz).

Label was already defined

You are using the same label name twice for different purposes.

mapothzero must be first statement in script

If you already have *mapothzero* as the first statement in the script, check whether it is preceded by a blank line. If so, delete the blank line and reparse.

Missing question or defined list

You have referred to a question or list that is not defined. Check that you have no misspellings.

Missing end statement

You have forgotten to end your qscript with *end* or previous errors in your script have caused it to be ignored.

Missing or bad argument

You have omitted an argument from a callfunc or have typed something that Quancept cannot recognize as a valid argument.

Missing or extra '/' in a list

You have too many or too few / characters in your response list. Check that responses are separated by exactly one / and that the last response is not followed by a /.

Missing or extra '+' in text

You have forgotten that variable names must be enclosed in + signs if they are to be part of a text. When displayed at the end of a text, the variable name is preceded by a +. When displayed at the start of a text, the variable name is followed by a +.

Missing resp

All question texts must be followed by a response list starting with the word *resp* unless a frozen response list is in effect. Perhaps you have a *comment* statement between the *ask* and *resp* lines.

Missing text

The parser expected to read a text, but there either isn't one or what there is cannot be interpreted as a text.

More than one destination in MP question

You have a multipunched response list in which a quoted response has routing defined with *go*. Do not use *go* in multipunched response lists. Instead, write a logical expression that uses <*text*, *text*> or <*text*>, or use one of the statements *and*, *or*, *xor*, or *bit*.

MP question checked for exact match, not any answer

You have a logical expression of the form *qname*=’*text*’, where *qname* is the name of a question with a multipunched response list. This is not correct. To test multipunched responses using response texts, use <*text*, *text*> or <*text*>.

Next before for

You have either forgotten to start your loop with *for*, or there was an error in that statement causing it to be ignored, or you have typed too many *nexts* by mistake.

No freeze in effect

You have forgotten to start the response line with *resp* or you have misspelled it.

Non-subscripted variable *var_name* used with a subscript

You have referred to an ordinary variable as if it was part of a loop or an array. Ordinary variables must not be referred to as *var_name*(*subs*), even if you enter the subscript as 1.

Number expected

The parser expected to read a number, but there either isn’t one or what there is cannot be interpreted as a number.

Number in parentheses expected

To allocate extra columns to a defined list, you type **define** (*n*) where *n* is the number of codes you wish to use. qparse has found a *define* statement with parentheses, but there is no number inside the parentheses.

Number of elements in defined list exceeds maximum specified

You have a defined list which includes a value in parentheses after *define*. This value defines the maximum number of items that list may contain. However, the list contains more than this number of responses. Either increase the value in parentheses or delete the excess items from the list.

Number out of range

The parser expected to read a number within a specific range. The number in the script is outside this range.

Out of order line *fieldname field_number 1 field_type in EXIMPORT file desc_filename*

Out of order line *fieldname field_start_column field_length field_type in EXIMPORT file desc_filename*

You have incorrectly defined the GNU database description file named *desc_filename*. The error message shows you the whole offending line from the description file. The error can be:

- If the database is type gdbm — you defined a *field_number* which is lower than a field number of a line higher up the file.
- If the database is type gdbmfix — you defined a *field_start_column* start number which is lower than a previous start column in the file.

Predefined mapping file (*name*) not found

You have tried to run qparse -u with a map file that doesn't exist. Perhaps you misspelled the file name.

Question *label* – cannot specify column within loop

You cannot assign specific columns (with the keywords *col* or *column*) inside a loop because that would cause multiple questions to be assigned to the same column.

Question *var_name* has ^o but no OTHER

If a response list contains responses with the *^o* flag, it must also include specified other. If you do not want interviewers to select specified other, add some statements to your script which test for specified other and issue a message if it is selected.

Rotend/Ranend before corresponding Rot/Ranstart

You have either forgotten to include a *rotstart/ranstart* statement, or you have misspelled it.

Rotstart/Ranstart groups cannot be nested

You have started a second set of rotated or randomized questions before you have closed the first set. Look back in your script and insert a *rotend* or *ranend* statement at the end of the previous group.

Special Responses with coded(0) in question *label*

You have put *null*, *ref* or *dk* (which require a code) on a question that has been designated as requiring no code. You must either remove *null*, *ref* and/or *dk* from the question, or change the question from being coded(0).

Subscripted variable *var_name* used without a subscript

You have referred to a variable that is part of a loop or an array, but have not indicated which cell of the variable you mean. All variables that are part of a loop or an array must be referred to as *var_name (subs)* where *subs* is the loop iteration number or the position of the cell in the array.

Syntax error

There is an error in the line but there is no specific error message for it.

Temporary prohibition of next directly following next in grouped loops

Quancept does not currently allow two consecutive 'next' statements when those statements terminate loops whose data is grouped. To fix the script so that it will parse, insert an empty *fix* statement between the two *nexts*, as shown below:

```

for fruit = 'Apples' / 'Oranges' / 'Grapes' grouped
q1      ask 'Did you buy '+fruit+ today?'
        resp sp 'Yes' / 'No' go nxt1
        for shop = 'Supermarket' / 'Greengrocers' /
                      'Street Market'
q2      ask 'Did you buy the '+fruit+' in a '+shop+'?'
        resp sp 'Yes' / 'No'
        next
comment Dummy fix here to separate the two nexts
f1      fix(0) '

```

nxt1 next

The quota file suffix should be no more than 2 characters long

You have a *quota* statement with a suffix that is more than two characters. Check that you have not mistyped it.

Too many columns to fix

The value you are trying to place in the data file is longer than 72 characters. If you mean to do this, you will have to split the value into two parts.

Too many for-next levels

Up to ten levels of nesting are allowed — that is, ten *for* statements before the first *next*. You have more than this.

Too many nested 'if...' statements

Up to ten levels of nesting are allowed, that is, ten *if* statements before the first *else*. You have more than this.

Too many responses requested

You have a multipunched response list that specifies the maximum number of responses that may be chosen, but there are not this many responses in the list. Either reduce the maximum value or add more responses to the list.

Unable to place data for question label in column *n*

You have allocated two or more questions to the same columns with the keywords *col* or *column* or with *qparsue -u* or a combination of them. If you want to combine the answers from multiple questions into one or more columns, you should use the keyword *set* to assign the answers into the columns allocated to a dummy question. See chapter 12, Assignment.

Unclosed fornext loop

You have either forgotten *next* at the end of the loop or there was an earlier error which caused this statement to be ignored.

Unclosed parentheses

You have more (’s than)’s in your script, when you should have the same number of each.

Unexpected character or word

The statement contains a keyword which is not valid in this statement. For example, you may have a *resp num* statement containing specified other.

Unrecognized callfunc

The name given with the *callfunc* statement is not the name of a valid function. Possibly you have mistyped the name.

Using list name that has duplicates may lose data

This is a warning only. You have defined a response list with duplicate entries. For example:

```
resp sp '1 terrible' '' '' '' '' 5 excellent'
```

In this example, the three blank entries will be coded (from left to right) as 2, 3 and 4. If you use a response list of this type as a sublist within a master list, any responses from the sublist coded as 3 and 4 will only match code 2 in the master list. This means the data coded 3 and 4 will be combined with code 2, invalidating the data collected for all three responses. See chapter 7, 'Defined response lists'.

Variable *var_name* assigned or compared to impossible value

You have an assignment statement or logical expression in which the value is the wrong type for the variable (for example, a text value with a numeric).

Variable *var_name* is a temporary variable never defined

You will only see this message if your script contains the keyword *warntemp* or *notemp*. It marks the use of a variable which is not the name of a question. This is not wrong, but these keywords are designed to highlight the use of such variables. It is probably a good idea to check your script to make sure that you have used the correct variable in each place.

Variable *var_name* may be used before set

You are testing or otherwise referring to a variable to which no value has previously been assigned. Check back in your script to see whether this statement contains the wrong variable name, or whether you have just forgotten to assign a value to the variable.

Variable *var_name* referenced without being set or declared

You are trying to check the value of a variable which has not previously been defined. For example, checking the value of *cost* before you have given it a value. Alternatively, you have simply misspelled the variable's name.

Variable *name* set although defined as unchangeable

You have tried to write data into a read-only field in a GNU database. The field has been defined as read-only by being flagged with an asterisk in the database definitions file.

Additional QCompile parser error messages



If QCompile finds a mistake in your script, it displays an error message. Many of these are the same as the error messages generated by qparse when you parse a Quancept CATI script. However, for keywords that are implemented only in Quancept CAPI, Quancept Web or mrInterview or that are not implemented in these variants, QCompile displays the following error messages.

Cannot allocate *number* columns for question *q_name*

You have allocated too many codes to an open-ended response. The maximum number of codes you can allocate to a response is 729.

Do not have multiple rotations

You have a *rotstart* or *ranstart* block in a Quancept CAPI, Quancept Web or mrInterview script but have forgotten to include *newrot* or *newran* statements to mark the statement groupings within the block. These statements are required even if each group contains one question only.

Keyword *keyword_name* has been discontinued

The script includes a keyword that is unavailable in Quancept CAPI, Quancept Web or mrInterview.

Keyword *keyword_name* has not been implemented

The script includes a keyword that is not implemented in Quancept CAPI, Quancept Web or mrInterview.

Numeric loops with COLGRID or ROWGRID not supported

You have a *colgrid* or *rowgrid* keyword loop whose value list is a numeric range. The value list for this type of loop must contain quoted texts separated by slashes.

Question *q_name* cannot be in COLGRID or ROWGRID with other questions

You have included two or more single-punched or multipunched questions in a loop that uses the *colgrid* or *rowgrid* keywords. Unless the question is numeric, loops of this type can contain only one question.

Question *q_name* cannot have multiple ranges with the slider option

The question uses multiple numeric ranges in its response list. Response lists that use *slider* can have only one range.

Question *q_name* is in COLGRID or ROWGRID but is not SP or MP

The script includes a *colgrid* or *rowgrid* loop that contains a question that is not single-punched or multipunched. You can only include single or multipunched questions in these loops.

Slider option not allowed for non-numeric question *q_name*

You have used the *slider* keyword on a non-numeric question. This keyword can only be applied to a question that has a numeric response list.

Text string missing after COLGRID or ROWGRID

The *colgrid* or *rowgrid* keywords must be followed by a text string. This is not displayed by the interviewing program, but without it QCompile is unable to parse the script. (Two consecutive single quotes are sufficient.)

Too few texts for slider question *q_name*

You have not supplied texts for a paired slider on a numeric response list. This slider type requires exactly two texts — the first to be displayed on the left hand side of the slider and the second on the right hand side.

Too many texts for slider question *q_name*

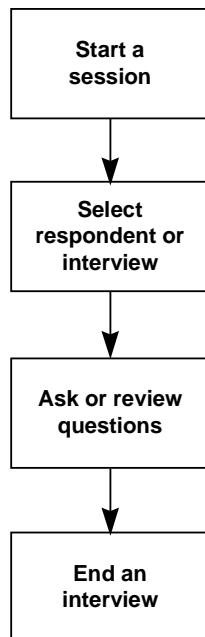
You have supplied texts to be displayed with a default slider on a numeric response list. You cannot display any texts with this type of slider.

33 Testing scripts with qtip

This chapter explains how to test your script using the interviewing program qtip. It begins by giving an overview of the steps in an interview and the things you'll need to do before you start. It then goes on to tell you how to start an interview, answer questions with different types of response lists and end an interview. The chapter also covers some special keywords that you can use with qtip during an interview and some additional facilities for testing scripts.

33.1 Steps in an interview

Each interview uses four different types of screens:



The first screen type prompts you to enter your interviewer name and then passes this information to qtip so that you are logged as a working interviewer.

The second screen type varies, depending on whether or not the project uses SMS. If the project uses SMS, the screen provides you with a number to dial and a menu of call result codes for recording the outcome of the call. If the project does not use SMS, the screen checks whether you want to start a new interview, restart a stopped interview, or, optionally, review a completed interview.

The third screen type displays the questions in the script and gathers the answers you enter as they are given by the respondent.

The last screen type deals with checking and saving the data for the interview. It provides a link back to the third screen type for checking and/or changing the data before it is saved.

We will discuss these screens in the following sections.

33.2 Before you start

Before you can use qtip successfully, you must:

- parse the script to create a qoc file
- define the necessary environment variables

You may also need to:

- place databases in shared memory if the script uses databases
- start the SMS server if the script uses the sample management facilities
- use the SMS supervisory program to load sample records into memory for calling

☞ See chapter 32, Parsing and compiling scripts.

See section 39.2, Environment variables, for information on environment variables.

See section 19.7, Making the response list database available to interviewers, to find out how to place databases in shared memory.

See the *Quancept Sample Management and Telephony Systems User's Guide* for information on the SMS supervisory program.

33.3 Starting a session

Quick Reference

To start an interviewing session, go to the project directory and type:

qtip [-ptf *ptf_filename*] *project_name*

where *ptf_filename* names the ptf file to be used as the source of language information. The default is *project.ptf*.

If you forget to enter the project name, qtip will prompt for it.

If you have the environment variable QCSHARE set to 1, qtip looks for the file *project_name.qsh* to see if the qoc file is already available in shared memory. If so, it uses that copy. If you do not have this environment variable set, qtip places a copy of the qoc file in memory exclusively for you.

☞ See section 34.2, Sharing a .qoc file, for more information.

Next, qtip will obtain your interviewer name, either from the QCWHOAMI environment variable or by prompting for it on the screen. If qtip does not recognize the name, it issues an error message and prompts for another name. You may cancel the session by typing ‘quit’ at this prompt.

If the project does not use SMS, you may be asked two questions before starting an interview. If you have the environment variable QCCOMP set to 1 or 2, you will be asked whether you want to review a completed interview. If you answer ‘y’, you will be asked for the serial number of the interview you want to review.

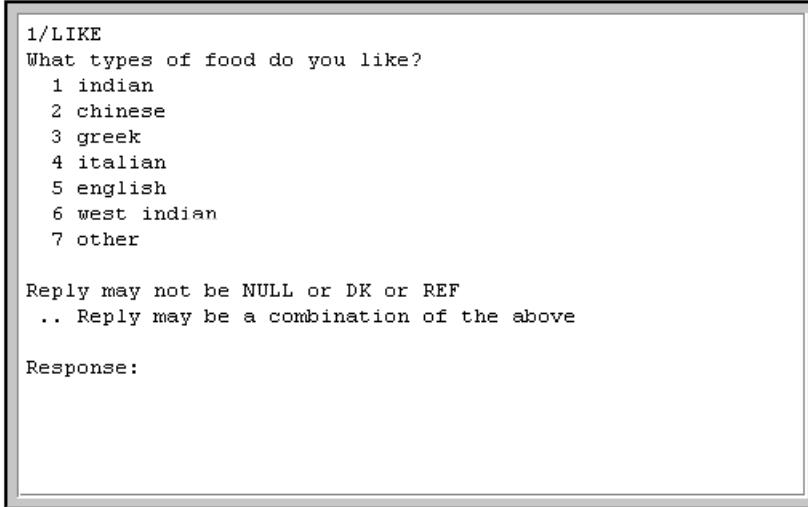
If you answer ‘n’ and you do not have the environment variable QCNORESTART set to 1, qtip will then ask whether you want to restart stopped interviews. If you answer ‘y’ at this prompt, you will be asked to enter the serial number of the interview you want to restart.

If you want to start a new interview, answer ‘n’ to both questions.

☞ See also section 34.3, Reviewing completed interviews.

33.4 The script

The screen clears and the first question is displayed, followed by its response list and a prompt for a response. For example:



```
1/LIKE
What types of food do you like?
  1 indian
  2 chinese
  3 greek
  4 italian
  5 english
  6 west indian
  7 other

Reply may not be NULL or DK or REF
.. Reply may be a combination of the above

Response:
```

The cursor waits next to the response prompt, ready for you to type the answer given. (If you are restarting stopped interviews, the question displayed is the next one to ask rather than the first one in the script; if you are reviewing completed interviews, the question displayed will be the last one the respondent answered).

Enter the response as described in the sections below and press Return to send the answer to the computer. If the answer is correct, for example, it is a code from the list or a number within the specified range, it will be accepted and the next question and response list will appear. If the answer is not correct, you will see an error message explaining what is wrong and qtip will wait for you to enter a different answer.

33.5 Single-punched answers

Quick Reference

To select a response from a single-punched list, type the response number as it is shown in the response list and press Return.

When qtip reaches a question with a single-punched response list, it displays the responses in the list and follows them with the words ‘Reply may be one of the above’.

Each answer in the list has a unique numeric value. Normally, the responses in a list will be numbered sequentially from 1. However, if the response list in the script contains the *atoz* keyword, and you have the environment variable QCRETAIN set to 1, it will be displayed in alphabetical order with the response numbers representing the positions of the responses in the original list rather than being numbered sequentially from 1.

33.6 Multipunched response lists

Quick Reference

To select responses from a multipunched list, type the response numbers separated by spaces.

To cancel an incorrect selection without backspacing, type:

!code

Multipunched response lists look the same as single-punched response lists, except that they are normally followed by the words ‘Reply may be a combination of the above’.

As you press the space bar after each response, the corresponding line in the response list is highlighted. For example, if you type code 1 and a space, the answer whose code is 1 will be highlighted. Do not press Return until you have finished choosing answers, even if the list reaches the end of the line on the screen, otherwise, qtip will think that you have finished with this question. Instead, just keep typing and qtip will wrap the list round onto the next line.

If you defined the response list so that it restricts the number of responses that may be chosen from the list, qtip will reject any set of choices that exceeds the permitted maximum with the message ‘Too many responses (maximum *max*) given *n*’.

If you flagged some answers in the response list definition with the characters ^s or sp, qtip will not let you choose those answers in combination with any others. The response list on the screen will not look any different from a standard multiple-choice list, but if you select a single-choice response with any other response, qtip will display the message ‘Response *x* must be single coded’ and will wait for you to enter an acceptable combination of answers.

If you mistype a response code, you can correct it either by backspacing and retyping or by typing the unwanted code preceded by an exclamation mark. For example:

```
25 6 9 10 17 !6 18 30 21
```

to cancel response code 6 while leaving codes 25, 9, 10 and 17 intact. qtip removes the highlighting from the canceled response.

-
- ❖ The availability of this facility with snapbacks is controlled by the environment variable QCNOCORR. If you have this variable set to 1 you will be able to cancel responses using ! when you first enter them, but if you later snap back to the question you will have to retype the full list of responses chosen. If you want to be able to cancel incorrect selections with ! at all times QCNOCORR must be unset.
-

33.7 Numeric answers

Quick Reference

To enter a numeric response, type the number and press Return.

Questions allowing integer or real responses report the numbers or ranges of numbers that will be accepted and prompt for a response with the words ‘Reply may be a numeric value’.

When entering integers, do not include any form of punctuation to delimit thousands from the rest of the number. When entering reals, the only punctuation that is acceptable is a dot for the decimal point.

33.8 Open-ended responses

Quick Reference

To enter an open-ended response directly into qtip, type the respondent's exact words on your keyboard. Press Return twice when you have finished — that is, once at the end of the text and then again on a blank line.

To record open ends manually, type:

@ or ; or ser

and press Return twice to request a serial number. Write the response out on paper with the serial number and then press any key to continue.

Open ends (*resp coded*) are introduced by the words 'Reply may be open-ended'. You may either type the open end at this point in the interview, or you may request a serial number for manual recording of the response.

If you type the answer now, just keep typing until you have finished the answer. There is no need to press Return at the end of each line, because qtip will split lines automatically between words where necessary. (Pressing Return inserts a | symbol at that point in the text when it is written to the data files.) If you notice typing mistakes, you may backspace to them and correct them.

The amount of text you will be able to type in depends on the amount of free space on the screen. The maximum number of characters that can be displayed on the screen is 1920 — that is, 24 lines of 80 characters each — but this reduces according to the size of the question text. Do not attempt to type more text than will fit in the space available as this may cause qtip to fail.

If you are not a fast or accurate typist, you may prefer to write down the response and type it in the end of the interview. Once you have requested a serial number, qtip waits for you to press any key to show that you are ready to continue the interview.

-
- ☞ If you have the environment variable QCNOAPPEND set you must enter open ends when the questions are displayed as you will not be able to enter them at the end of the interview.
-

33.9 Other and specified other

Quick Reference

For an ordinary other response, enter the response code as for a single-punched or multipunched response.

For specified other, enter the response code and then type the text at the automatic ‘Please specify’ question. Alternatively, type the response code for specified other, then a space, and then the text enclosed in parentheses. Specified other responses may be recorded manually as described above for open ends.

qtip differentiates between other, which was defined as a standard quoted response, and specified other. The former appears in the response list just like a single-punched or multipunched response, whereas the latter is always shown last and in upper case.

When a response list contains specified other, you may either wait for qtip to prompt you for the open-ended text or enter the open end at the same time as the codes for the other responses. If you want qtip to prompt you for the open end, just type the response code for OTHER. When you press Return to send the chosen responses to the computer, Quancept issues the subsidiary ‘Please specify’ question in the lower half of the screen and waits for you to enter an open-ended response.

A quicker method is to enter the open-ended text at the same time as you enter the response code for OTHER. Type the response code first and follow this with a space and then the open-ended response text enclosed in parentheses. You can choose specified other as many times as you wish and at any position in the response list.

33.10 Selecting answers from a response list database

Quick Reference

To scroll through a database, type + to scroll forwards or – to scroll backwards.

To select a response from a database, type the record number of that response as it is shown and press Return. Alternatively, to select a response using its record key (the value before the first space or semicolon), type:

=key

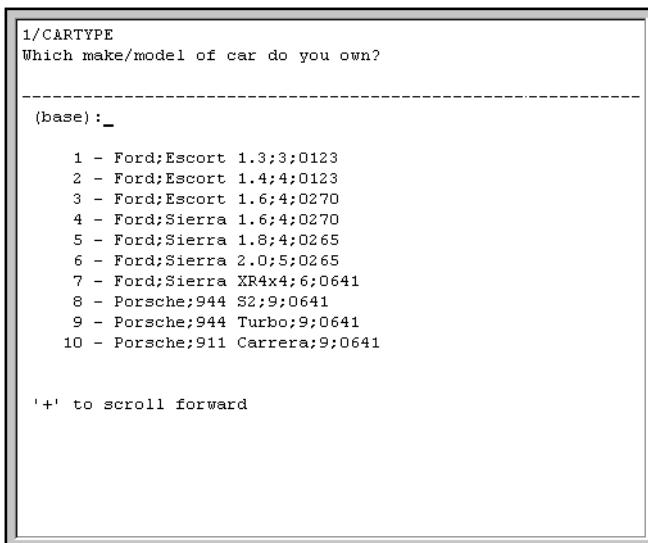
To filter a database so that only certain records are displayed, type

selection text !

To cancel a filter, type: !

To enter a null response, type: \$

Where a question has a database as its responses list, qtip displays the text of the question as usual and follows it with the first ten records in the database. Each record is preceded by its line number in the database:



The screenshot shows a terminal window with a light gray background and a dark gray border. Inside, the text is white. At the top, it says "1/CARTYPE" and "Which make/model of car do you own?". Below this is a horizontal dashed line. Then comes the instruction "(base) :_". Underneath, a list of 10 items is displayed, each starting with a number from 1 to 10 followed by a dash and a car specification (make, model, and year). At the bottom, there is a note "'+' to scroll forward".

```
1/CARTYPE
Which make/model of car do you own?

-----
(base) :_
1 - Ford;Escort 1.3;3;0123
2 - Ford;Escort 1.4;4;0123
3 - Ford;Escort 1.6;4;0270
4 - Ford;Sierra 1.6;4;0270
5 - Ford;Sierra 1.8;4;0265
6 - Ford;Sierra 2.0;5;0265
7 - Ford;Sierra XR4x4;6;0641
8 - Porsche;944 S2;9;0641
9 - Porsche;944 Turbo;9;0641
10 - Porsche;911 Carrera;9;0641

'+' to scroll forward
```

If there are more than ten records in the response list database, a message tells you that you may press + to see the next ten responses or (later) – to see the previous ten.

The '(base):' line above the response list has two functions. First, it is where you enter the line number of the chosen response. For example, if the respondent owns a Porsche 944 Turbo you would type 9 and press Return to continue with the next question. Second, it is used for filtering.

Filtering allows you to display only those records which contain a given text — for example, only those records containing the word Porsche. To produce this list, type in the text followed by an exclamation mark:

```
Porsche!
```

The screen is redrawn to contain only records which satisfy the filter, and the word 'base' is replaced with a 1, indicating that one filter has been applied. As subsequent filters are applied, this number is incremented by one each time. Additionally, the text of the most recent filter is displayed at the end of the line.

When you enter more than one filter, each filter applies only to the records in the current list. Records already rejected by the higher filters are ignored.

-
- ❖ If you are testing a script but have forgotten to share the database, the question text is displayed with just the (base): prompt. To move to the next question, type \$ and press Return.
-

33.11 Echoed responses

If the question has a numeric response list or a long multipunched response list, you may have included the keyword *echo* in the script so that qtip will echo the responses chosen and check that they are correct before going on to the next question.

If the responses are correct, type ‘y’ and the next question will appear. If not, answer ‘n’ and the screen will be redrawn with just the question and response list ready for you to re-enter the response codes.

33.12 The end of the script

When qtip reaches the end of the script, it is time to write out the data and add records to the accounting files. Scripts which end because they have reached the *end* statement with all the relevant questions answered are called *successfully completed*.

Before the data is written out, qtip offers you the choice of changing or inspecting the answers you have entered and of entering any open ends that you may have written down. The first prompt is ‘Do you wish to change or inspect anything (y/n) ?’

If you answer ‘y’ at this prompt, the last question asked is re-displayed with its current response, and you are asked whether or not this is correct. By saying that the response is not correct, you will obtain a blank response prompt, from which you can inspect responses with :*qname* or snap back with <*qname*>.

-
- ☞ See section 33.24, Checking answers to previous questions, and section 33.27, Snapping backwards and forwards in an interview, for more information.
-

Each time you enter an answer to the last question or accept its current value, qtip repeats the prompt. When you are sure that the answers are correct, type ‘n’ in response to this prompt to continue.

The second question is ‘Do you want to append/check verbatims now (y/n) ?’. This is suppressed if you have the environment variable QCNOAPPEND set or if there were no open ends or specified other responses entered during the interview.

If you typed in the responses as the respondent gave them and there is nothing to add, answer ‘n’ to this prompt. Otherwise, answer ‘y’ to call up the serial number for the first open end for this respondent.

If there is already some text associated with this serial number, perhaps because you typed in this response but none of the others, you will see that text displayed at the top of the screen. You may then enter the additional text and it will be appended to the existing text in the tex file. If you want the new text to overwrite the old text, type = at the start of the new text. You may enter up to 1043 characters (about 15 lines) in this way.

-
- ❖ If you are appending to an existing text, the limit of 1043 characters is reduced by the number of characters that are already stored for the response. For example, if 1100 characters are already stored and you try to append 500 characters, qtip silently truncates the text at 1043 characters. This means that not only have you lost the text you were adding, you have also lost some of the text that was originally stored for the response.
-

An alternative, if you have the environment variable QCEDITOR defined, is to type *e and then press Return twice. This copies the open end into a temporary file and opens that file with the chosen editor. You may then make whatever changes or additions are necessary. When you have finished, save your changes and leave the editor in the normal way.

If there are several open ends, each one is displayed in turn. To skip a response without entering anything, just press Return twice. To stop entering open ends altogether, type **quit** and press Return twice.

-
- ❖ qtip does not display open-ended responses to dummy questions at this point even though they have been saved. If you really want interviewers to check the answers assigned to dummy coded questions, place an *editopen* statement in the script to force editing of that variable at that point in the script.
 - ☞ *editopen* is described in section 14.10, Editing open ends and specified other responses.
-

Finally, qtip writes the data to the tex, dat, drs and ddr files as appropriate and confirms this by displaying the respondent serial number allocated to the data. It then asks whether to start another interview.

33.13 Interviews terminated by the script

Termination by quota control

If the project has quotas, qtip keeps a count of how many respondents are interviewed who satisfy the quota requirements. When the appropriate number of interviews has been conducted, all interviews with respondents who fall into these groups are terminated automatically. A message is displayed informing you that the interview has been terminated by quota control and you are asked whether you wish to change or inspect the data collected so far, as for a normal termination.

The data for the interview is written out and you are then ready to start on the next interview.

Stopped interviews

The script may contain a *stop* statement, which stops the interview part way through. A message to this effect is displayed on the screen. qtip allows you to inspect and change responses gathered so far, as described above, before asking whether to write the data collected so far to the data files. If you answer 'n', the data is saved in a stopped data file until the interview is restarted and completed,

when it is written to the main data files as for any other interview. If the script contains a *stopdata* or *setdata* statement and the interview is stopped after this statement has been executed, this question will be suppressed. If the value given with *setdata* or *stopdata* is 0, data is not written; if the value is 1, then data is written.

If the script uses SMS, qtip will prompt you for a callback time and a comment which will be passed back to the SMS server for storage with the record ready for the next time it is called.

Data for interviews stopped in this way is written to a directory called *project dirname*, where *dirname* is the name defined on the statement which stopped the interview.

-
- ☞ For further information on stopped data files, see ‘The stopped interview directory’ in chapter 38, File formats
-

33.14 No answer, don't know and refused

Quick Reference

If the response list does not contain the response texts No Answer, Don't Know and Refused, use the special responses:

null **dk** **ref**

if qtip indicates that they are acceptable.

If the response list includes the keywords *null*, *dk* or *ref*, qtip will translate this into a message of the form ‘Reply may be NULL or DK or REF’ beneath the list of acceptable responses, indicating which of those responses is valid. To choose one of these responses, just type it at the response prompt. If you have the environment variable QCNULL set to 1, you will be able to press Return twice instead of typing ‘null’.

If there are no suitable responses in the list and *null*, *dk* and *ref* are not available, there may still be a number of options open to you.

First, you may be able to force qtip to accept one of these responses by typing ***null** or ***dk**.

Second, if the script contains an *onresp* statement, you may use the response defined by this statement. What happens next is determined by the script.

33.15 Forcing an open-ended response

Quick Reference

To force qtip to accept an open-ended response to a single-punched, multipunched or numeric question, type:

***note**

at the response prompt and enter an open-ended response when prompted to do so.

- » The open end is treated as a secondary response to the question. After you have completed the open end, you must choose a response from the list or enter a value within the specified range.
-

When you use ***note**, the screen clears and the words ‘Enter note’ appear as a prompt for you to record the respondent’s exact words. You must type the response on the keyboard because qtip does not allow the use of @, ser or ; at this point. The response will be written to the *project.com* file, with a message showing when and by whom the note was written.

The question screen is then redrawn with the current question and qtip waits for you to answer the question in the usual way. You cannot bypass a question simply by forcing an open end.

33.16 Scrolled response lists

Quick Reference

If a response list runs over more than one screen, type + to scroll forward to the next screen or – to scroll backward to the previous screen.

When qtip encounters a long response list during an interview, it displays the response texts in three columns in an attempt to fit the whole list on the screen at once (you may have increased or decreased the number of columns with a *setcols* statement in the script). If the response list is very long, or if it is preceded by a large amount of question text, there may still be insufficient room to display the whole response list. If this happens, qtip displays the first screen of responses and provides facilities for scrolling through the other screens one by one.

If the question allows multiple answers, you may select the responses while they are on the present screen. The responses will be highlighted as normal. If you have selected all desired responses from the present screen but still have others to select from previous/subsequent screens, press + or – after the responses you have entered. If any responses from the new screen have been previously chosen, they will be highlighted.

33.17 Changing the layout of long response lists

Quick Reference

To display the responses in a long single-punched or multipunched response list across the screen, type:

across

To revert to the default of displaying responses down the screen, type:

down

qtip normally displays each response in a single-punched and multipunched list on a separate line. If there are more responses than will fit in one column, it starts a second column. Sometimes, switching to displaying responses across the screen may allow you to fit more responses on the page. Once you have changed the display orientation, the change applies until the end of your current interviewing session.

-
- ❖ If qtip can fit all responses into a single column it always displays them down the screen even if you have asked for them to be displayed across the screen.
-

33.18 Early completion of an interview

Quick Reference

If early completion is allowed at this point in the script, type:

complete

to terminate the interview and flag it as successfully completed.

If the script contains an *allow* statement, you may terminate the interview at any point after that statement by typing ‘complete’.

When you do this, qtip asks you to confirm that early completion is all right. If so, you are asked whether you wish to change any of the responses before the data is written to the data file. The respondent serial number is then displayed, followed by the query as to whether another interview is required.

If you have typed *complete* by mistake, you can say that early completion is incorrect. The next question is then displayed.

33.19 Premature termination with data

Quick Reference

To terminate the interview and save the data gathered so far, type:

quit

If the respondent refuses to continue an interview, you may terminate it by typing *quit*. This terminates the interview immediately, clears the screen and displays the message ‘Interview terminated prematurely, is this ok?’. The data up to that point may be inspected or amended as documented above.

Normally, data from interviews terminated in this way is saved, but *quitdata* or *setdata* statements in the script may affect this. For example, if the script contains a *quitdata* statement with a value of 0 and the interview is terminated after this statement has been executed, data for these interviews will not be saved.

33.20 Terminating an interview without data

Quick Reference

To abandon the interview and ignore any data gathered so far, type:

abandon

The response *abandon* is exactly the same as *quit* in that it terminates an interview prematurely. However, no data is saved. It is just as if that interview had never taken place. This difference may be overridden by *quitdata* or *setdata* statements in the script. For example, if the script contains a *quitdata* statement with a value of 1 and the interview is abandoned after that statement has been executed, data for these interviews will be saved.

33.21 Stopping interviews temporarily

Quick Reference

To stop an interview so that it may be restarted at the same point later on, type:

stop

If the respondent does not wish to finish the interview now but is willing to continue it at a later date, you may type **stop**. qtip allows you to inspect and change responses gathered so far, as for other forms of termination, before asking whether to write the data collected so far to the data file. If you answer 'n', the data is saved until the interview is restarted and completed when it is written, as for any other interview. If the script contains a *stopdata* or *setdata* statement and the interview is stopped after this statement has been executed, this question will be suppressed. If the value given with the statement is 0, data is not written; if the value is 1, then data is written.

If the script uses SMS, qtip will prompt you for a callback time and a comment which will be passed back to the SMS server for storage with the record ready for the next time it is called.

The first time an interview is stopped in this way, qtip creates a subdirectory called *project_stop*, in which it saves the stopped portion of each interview. When the interview is restarted, this information is transferred into a *USED* subdirectory within each of the stop directories.

The script may contain a *stop* statement which stops the interview part way through. A message to this effect is displayed on the screen and you are prompted to arrange a suitable time to call back.

33.22 Messages to the supervisor

Quick Reference

To send a message to your supervisor, type:

***msg**

Type the message when prompted to do so and press Return twice to end it.

The ***msg** response lets you send a message to the interviewing supervisor. The screen clears and the prompt 'Enter Message' is displayed. The message is written to the *project.com* file and the screen is then refreshed with the current question and response list.

33.23 Redrawing the screen

Quick Reference

To redraw the screen, type:

regn

and press Return.

If you are dialing in to the computer using telephone lines, funny characters will sometimes appear on the screen. This is line noise which you can fix by typing *regn* and pressing Return. The screen clears, the question and response list reappear and the interview can be continued.

33.24 Checking answers to previous questions

Quick Reference

To check the answer to a previous question without altering your position in the interview, type:

:question_name[(iteration_number)]

The answers given to that question are listed at the foot of the screen and you are asked to press any key to continue. When you press a key, the answer to the earlier question is removed and the current question remains ready for you to type in the response.

33.25 Listing question names

Quick Reference

To see the names of questions on the interview path, type **/**.

The names will be listed across the screen in the order in which they appear in the script and with the current question flagged with an asterisk.

33.26 Finding where to snap

Quick Reference

To see a list of questions you can snap to, type:

?

To see the same list with the current answer to each question, type:

??

It is not always permissible to snap back or forward to just any question. As you conduct the interview, qtip keeps track of which questions have been asked and which have been bypassed due to routing. The questions which have been asked are said to be ‘on the interviewing path’ and may be snapped to. If a question has not been presented, it is not on the interviewing path and you may not snap to it. You may obtain a list of the on-path questions by typing ? at the response prompt for the current question. The screen clears and question names that may be reached with any of the snap facilities are listed across the screen in the order in which they occur in the script. The name of the current question is preceded by an asterisk.

33.27 Snapping backwards and forwards in an interview

Quick Reference

To snap back one question, type:

<

To snap forward one question, type:

>

To snap to a named question, type one of the following:

```
<question_name[(iteration_number)]  
>question_name[(iteration_number)]
```

To snap back to the first question answered in an interview, type:

<<

To snap forward to the furthest point reached in an interview, type:

>>

When you snap back during an interview the screen clears and the question to which you snapped is displayed with its list of permissible responses, followed by the response that was given to that question.

qtip checks whether the original answer is still correct. If it is, type 'y' and the next question will be displayed. If this has already been answered, you will see the current set of answers and will be asked to confirm that they are still valid.

After backtracking without changes, qtip normally checks that the answers to all intermediate questions are still correct. If there are many such questions which you know you do not need to change, you can return quickly to your original place in the script by saying that the answer to the current question is incorrect and then, instead of typing in a new answer, typing >> to skip as far forward as possible.

-
- ¶ The forward snapping commands are valid only if you have previously snapped back to an earlier question. Also, you may snap no further forward than the question you had reached before snapping back.
-

33.28 Altering responses to questions you have snapped to

If you need to alter the current response in any way, even if it is only to enter an additional code, you must say that the response is incorrect. qtip then redraws the screen with just the question and response list and waits for you to enter the new set of response codes.

When typing in the revised response codes, it is important to enter all the responses, not just the additional ones. When you backtrack to a question and enter a new response, qtip overwrites the original answer with whatever you type in this time.

If you snap back and change the answer to a question which has routing associated with it, the next question displayed may or may not be the same as before. Occasionally, changing an answer may result in quite drastic changes in the interviewing path, with some questions which were previously on the path being removed from it.

33.29 Snapping back and making no changes

Quick Reference

If you snap back and say that the answer to the question is incorrect by mistake, you can still retain the original answer by typing a double quote:

"

If you have snapped back and said, by mistake, that the answer to that question is incorrect, you may retype the original set of response codes to continue. Alternatively, if many responses were chosen, you can type " to retain the original answer to that question and go onto the next one, or `>qname` to snap forward to the named question. In both cases, no new list of responses has been entered, so there is no way that the original responses can be forgotten.

33.30 Miscellaneous commands

The commands in this section have no bearing on the data collected or on the progress of the interview. They are simply there to provide you with additional information. Like all other qtip facilities, these commands are entered at the prompt for a response. qtip displays the requested information and then asks you to press Return to continue.

Command	Action
*comment	<p>displays any comments associated with this record. For example:</p> <pre>COMMENTS FOR KEY 1234 *** Tue Oct 16 21:42:26 1990 barbara Result(12) "Stopped interview" Respondent had to go out</pre> <p>and asks 'Append more comments (y/n)?'. If you enter a comment in this way, the report will show your interviewer number and the name of the question at which the comment was entered instead of the call result description shown above.</p> <p>This command is valid only if you are using SMS.</p> <p>You can limit the number of comments by <i>*comment</i> by setting the environment variable QCCOMMENTS to the number of comments you want to see. For example:</p> <pre>setenv QCCOMMENTS 3</pre> <p>shows only the last three comments entered for each record.</p> <p>Comments are normally displayed in the order they are entered, so the most recent comment is shown last. To reverse the order so that the most recent comment is shown first, set QCCOMMENTS to a negative number. For example:</p> <pre>setenv QCCOMMENTS -3</pre> <p>shows the last three comments for each record in reverse date order, most recent first.</p>
*date	<p>displays the current date and time. For example:</p> <pre>Date is Tue Aug 3 14:23:11 1999</pre>
*hangup	<p>When the project is using the QTS, hang-ups up the call without terminating the interview. This is useful if the script ends with some classification statements or other statements that do not require the respondent's input.</p>
*ln	<p>prompts you to enter a number defining the language in which to conduct the interview. This facility is available only in scripts written to support multiple languages (see section 27.2, Scripts for multilingual CATI interviews). The number must be between 0 and 9 and must be one of those defined in the script. If you enter a two-digit number, qtip takes the first digit as the language number. If you enter an invalid number, qtip will repeat the prompt until you enter an acceptable value.</p>
*redial	<p>When the project is using the QTS, reconnects a call that has been accidentally disconnected.</p>
*time	<p>displays the time local to the interviewer and, in parentheses, the time at which the interview started. For example:</p> <pre>Time is/started: 14:23:11 (14:12:54)</pre>

Command	Action
*vers	tells you the date on which the qtip program you are using was created, the qoc file version used for checking compatibility between qtip and the qoc file, and the amount of heap space currently used. For example:

```
Quancept interviewing program
Created on 2002-12-05
Compiler version 19.0           Heaptop 10
```

33.31 Using SMS

If an SMS server is running, qtip requests a record from the server and displays it with a menu of possible call outcomes and, sometimes, some details about the respondent or previous calls made to that record. If the record has been called before, you may see comments entered by the interviewer who made that call. Exactly what you see and the contents of the menu are defined by the SMS algorithm writer and may vary between projects, but a typical screen will look something like this:

```
Hello, I'm barbara, calling from ----- in -----.
Do you have a couple of minutes to answer a few questions.
```

```
key: 0171 625 7222
```

- 1 Proceed with interview
- 2 No answer/ Answering machine
- 3 Busy
- 4 Resp not available
- 5 Business number
- 6 Number disconnected
- 7 Refused
- 8 Abandoned interview
- 9 Appointment

```
Please enter choice, or r to redial:
```

Dial the number and then enter the numeric code representing the outcome of the call. In the example, you would type 1 if the respondent answers the phone and agrees to be interviewed, or 2 if the line is busy. Sometimes you will be presented with the next number to call straightaway, at other times you will be prompted for a callback time, a comment or some other type of information, as described below. (If you have an autodialer, numbers will be dialed for you and only records where contact has been made with the respondent will be passed to you.)

The default prompt for a call outcome code is ‘Please enter choice’, as shown in this example. If you have the environment variable QCANOTH set to 1 you will be able to pause between calls by typing a call outcome code followed immediately by the letter p. Then, instead of receiving the next number from SMS you will see the qtip message ‘Another interview (y/n)?’

If you are using a modem to dial numbers you may be able to pass the record to the modem for dialing by typing the letter r. You'll know if you have this option because the call outcome prompt will include the words 'r to redial'.

☞ For further details on QCANOTH see section 39.2, Environment variables.

Busy and no answer

Records which are busy or unanswered are returned to the SMS server for automatic rescheduling. If a record has been busy or unanswered more than a certain number of times, it may be removed from the list of records available for calling.

Arrange appointment with respondent

If the respondent agrees to be interviewed at a later date when it is more convenient, you will be prompted to enter the appointment time and perhaps also a comment.

The screen clears and the interviewing calendar is displayed showing the dates and times during which interviewing will be taking place. This is followed by a prompt for an appointment time.

There are various ways of entering appointment times. If the appointment is for later on the same day, you need only enter the time. Examples of valid times are:

10:00 10am 14:30 2:30pm

If the time is for tomorrow, you may precede the time with the keyword **tomorrow**:

tomorrow 11:30am

requests an appointment at 11:30 tomorrow morning. Similarly, if the appointment is for the next occurrence of a particular day, you may enter the day name, or as much of it as is necessary for it to be unique, before the time. For example:

wed 16:00

for an appointment at 4 p.m. on the next Wednesday.

You may also enter specific dates or parts of dates, but be sure that what you type is what you mean. When only a month and day number are given, qtip and SMS assume that the month number will come before the day number in the American style, so 04/12 is read as 12 April rather than as 4 December. If you want to enter dates in the European format, you must ensure that you have set the environment variable QCEUROPEAN to 1 or have QCLOCALE set to a value between 30 and 50 inclusive.

When the appointment consists of a date and time, it must be entered with the date first and separated from the time by a hyphen or a space. Here is an appointment for 3:15 p.m. on 15 November:

15/11-15:15 QCEUROPEAN set to 1
11/15-3:15pm QCEUROPEAN not set (the default)

If you are uncertain whether you have QCEUROPEAN set, you can make quite sure of the date by entering it in the form yy/mm/dd, as follows:

99/11/15 3:15pm

Sometimes the respondent may say something like 'call me back in 2 hours' or 'try the same time tomorrow'. In these cases, you might prefer to enter the appointment time as relative to the time now rather than as an absolute time. Relative times may start with the word **now**. This is followed by a simple arithmetic expression indicating the number of minutes, hours, days and/or weeks to elapse until the callback. Here are some examples:

+ 1d	same time tomorrow
now + 4h	in four hours' time
+ 30	half an hour later
now + 1h + 30	an hour and a half from now
now + 1w	same time and day next week

From these examples, you will see that a number by itself is taken as a number of minutes.

If the respondent is in a different time zone than you are, enter the time as the respondent gives it and the computer will take care of the time difference.

You can request a summary of the rules for appointment setting by typing **?** at the prompt for an appointment.

Depending on the way the call result has been defined, you may be allowed to enter the appointment time as a range; for example, between 2 p.m. and 4 p.m. tomorrow. If you select a result code which allows ranges, enter the earliest time at the first prompt, as you would for any other appointment. qtip will prompt for a second time. Enter the latest possible call time here. If the respondent gives a specific time, just enter the same time at both prompts.

qtip reads the appointment you enter and displays it in full. If it is within the interviewing period for the project, you are asked to confirm that it is acceptable. If not, qtip tells you so and asks whether you really want to make this appointment. Appointments will be accepted up to a year in advance of the current date and time. Appointments later than this will be rejected.

Once you have confirmed the appointment, you may be prompted for a comment which can be displayed the next time the record is presented for calling. Comments may be of any length. If you reach the end of a line there is no need to press Return to start a new one, as qtip does this for you. Press Return twice to end the comment. qtip copies it to a file named comments/grpx/rec_key in the project directory (for example, comments/grp15/1234 for comments associated with the record whose key is 1234). If there is no comment, just press Return twice to continue.

If the record already has a comment, yours is appended to it so that the next interviewer will see all comments made so far.

Result codes that prompt for information

Result codes can have scripts associated with them which prompt you for information to be passed back to SMS. You might come across this if you select a result code signaling a language barrier. SMS will need to know what language the respondent speaks so that it can ensure that the record is passed to an interviewer who speaks that language next time. You may therefore see the prompt ‘What language does the respondent speak?’. The script will name an SMS variable in which your response will be stored. When the callback is due, the server will pass that record only to an interviewer whose qualifications indicate that he/she speaks that language.

Another instance of when script files may be used is when the phone numbers are taken from printed lists. If each result code has a script which prompts for a phone number if the phone number variable in the record is blank, it is possible to transfer the numbers into the SMS records as they are used. The prompt in this case might be ‘Enter the phone number dialed:’.

Returning to the SMS menu

If qtip executes a *gotosms* statement, it displays the SMS menu on your screen with a prompt to enter a result code.

Having selected an appropriate code, you will then be asked whether to start another interview. For example, if there is no suitable respondent in the household, you may be asked to select an option which terminates the call, whereas if the respondent is simply not available, you may be directed to select one of the callback options instead.

33.32 qtip error messages

The Quancept parser checks the syntax of your script and tell you about incorrectly specified statements. However, it cannot point out statements where the logic of the script is unsound. Errors of this type are found by the interviewing program when you test the script and cause the test to terminate with an *errorstop* message. This section lists those messages and explains the likely cause of them.

Args to substring functions MUST be type Numeric

Your *substr* statement contains non-numeric values or refers to variables whose values are not numeric.

Argument to NBIT has no SP value

The variable you have used with *nbit* is not single punched.

Bad arguments passed to quoval

The quota suffix and the type of information required must be passed to *quoval* in text variables, and the cell number must be a number or an integer variable. One or more of the variables you are using is of the wrong type.

Bad arguments to map_quota

The quota suffix must be defined in a text variable. The variable you are using is not a text variable — perhaps you have forgotten to type the suffix in single quotes.

Bad arguments to quorat

The quota suffix must be defined in a text variable. The variable you are using is not a text variable — perhaps you have forgotten to type the suffix in single quotes.

Bit must be used with SP/MP questions

You have a *bit* statement which refers to a question whose response list is not single punched or multipunched, or which uses a temporary variable instead of a question.

Callfunc trying to change type of a question

Your callfunc statement names a variable for the callfunc to write to, whose type does not match the type of data that the callfunc wants to write to it. For example, the callfunc names a text variable when the callfunc wants to write out a numeric value.

Cannot create data sub directory

qtip is unable to create the ddr subdirectory structure for individual data files. Check that you have write permission for the project directory.

Cannot create stop file directory

qtip is unable to create the stopped data subdirectory. Check that you have write permission for the project directory.

Cannot load PTF *project.ptf*

qtip cannot locate its default language file. If you press Return qtip will read texts from the tx0 file. This message usually means that you are not running qtip in the project directory.

Cannot load quota file

The script contains quotas, but there are no quota files in the project directory. Perhaps you have forgotten to create them or you have created them with the wrong names.

Cannot make comments directory

qtip is unable to create the comment subdirectory for comments entered at the SMS stage of the call. Check that you have write permission for the project directory.

Cannot write to stop file directory

qtip is unable to write data for this interview to the stopped data directory. Check that this directory exists and that you have write permission to it. If it exists, check that the permissions show the directory as a directory and not as a file (In Unix, changing a directory's permissions from, say, 777 to 666 mean that the directory is seen as an ordinary file).

Cannot find NEXT statement

You have a loop without a *next* statement.

Cannot open look-up file

qtip cannot find the file named on the *readfile* statement, or you do not have read permission for the file or the directory containing the file.

Cannot UNSET a DEFINE

You have an *unset* statement which names a defined list. You can unset only question and temporary variables.

Elm must be used with SP or MP questions

You have an *elm* statement that refers to a question whose response list is not single punched or multipunched or that uses a temporary variable instead of a question.

Excessive looping in walkqoc

On walking back through the script after a snapback, qtip has encountered a *strngchk* statement where the text does not match the pattern. After trying 100 times, qtip has returned control of the interview to you.

FORLOOPS must be temp vars

The pointer variable on a *for* statement must be a temporary variable. You have used a question name, a list name or a statement name.

Impossible arithmetic operator

Arithmetic operators are +, -, * and /. You have an arithmetic statement containing something other than these operators.

Invalid SET function

You have a *set* statement with a keyword that qtip cannot recognize.

List1 in List2 must be a list or a question

You have a *list1 in list2* statement in which *list1* is not the name of a sublist or a question whose responses are based on *list2* or one of its sublists.

Named stop directory is not a directory

You have a *stop* statement that names a stopped data directory that the computer does not see as a directory. Check that its permissions are set correctly. (Changing a directory's permissions from, say, 777 to 666 mean that the directory is seen as an ordinary file).

NBIT/NUMV/NUMB must refer to a SP/MP question

You have an *nbit*, *numv* or *numb* statement that refers to a question whose response list is not single punched or multipunched or that uses a temporary variable instead of a question.

Out of range subscript

You have a reference of the form *variable(subscript)* in which the value of the subscript is greater than the number of cells in the named variable. For example, if q1 is repeated six times, you have q1(1) to q1(6). If you have a subscripted reference to q1 which causes qtip to ask for, say, q1(8), qtip will issue this message.

Quorat: quota suffix not found

There are no quota files in the project directory with the suffix you have defined. Check that you have given the correct suffix.

Quota variable has incorrect type

Quotas can be based on single punched, multipunched and numeric questions. You have a *quota* statement naming a question of a different type.

Quota variable is not a question

You can only quota on questions. You have a *quota* statement that refers to a temporary variable.

Quotamap: quota suffix not found

There are no quota files in the project directory with the suffix you have defined. Check that you have given the correct suffix.

Quoval: quota suffix not found

There are no quota files in the project directory with the suffix you have defined. Check that you have given the correct suffix.

Rot/Ran end before start

You have a *rotend* or *ranend* statement with no previous start keyword.

Stop-decrm with unknown quota

qtip cannot find any quota files with the suffix named with *decrm=* on the *stop* statement. Check that you have not mistyped the suffix and that the quota files exist.

Subscript type must not be CODED or MP

Subscripts must be integers; that is, a whole number or a variable whose value is a whole number.

Too many quota cells

The number of cells generated by the responses to the questions named on the *quota* statement does not match the number of cells defined in the quota files for this quota. Check that you are using the right quota files for these questions and that the quota files have been set up correctly.

Unknown forloop type

The variable named with *pass=* on your *quota* statement must be a temporary variable.

Unknown quota

qtip cannot find any quota files with the suffix you have named. Check that you have not mistyped the suffix and that the quota files exist.

Variable subscripts must be numeric or sp

Subscripts must be integers; that is, a whole number or a variable whose value is a whole number.

Wrong question type for OR/AND/XOR function

The assignment keywords *and*, *or* and *xor* expect to read single-punched or multipunched responses. You have a statement in which the question has some other type of response list.

Xrandom: trying to randomize 0 items

You have a *rotstart/end* or *ranstart/end* block containing no questions.

Zero length qoc file please reparse

A qoc file exists for this project but it is empty.

34 Miscellaneous CATI facilities

This chapter describes the following topics:

- running test interviews in automatic mode
- sharing the qoc file
- reviewing completed interviews
- recording the keystrokes entered during interviews

34.1 Automatic testing

Quick Reference

To test your script in automatic mode, type:

qtip -t *number* *project_name*

where *number* is the number of interviews you want to run.

It is possible to test a script without having to type in answers yourself. When you run qtip in test mode, it carries out the given number of interviews, each time selecting a random set of answers. With single-punched, numeric and list response lists, qtip selects one response from the list. With multipunched response lists, qtip selects between one and four responses at random. With open-ended response lists or specified other, qtip assigns the text OPEN RESPONSE *n*, where *n* is a number generated by qtip. These texts are always accepted by *strngchk* statements.

In order for the test to simulate live interviewing as closely as possible you may use the QCSLEEP environment variable to determine the number of seconds (if any) to pause after each answer is given. If you define a negative value for QCSLEEP, qtip converts the number to a positive number and selects a value at random between 0 and that number (for example, -10 results in a delay of between 0 and 10 seconds).

-
- » In studies that use SMS, it is also possible to simulate call outcomes by adding extra information to the entries in the tipresps file. For further details see the *Quancept Sample Management and Telephony Systems User's Guide*.
-

34.2 Sharing a .qoc file

Quick Reference

To share a qoc file, type:

shareqoc *script_name*

Each interviewer who is to work with a shared qoc file must have the environment variable QCSHARE set to 1.

When you start work on a project, qtip places a copy of the qoc file for that project in memory. If several interviewers are working on the same project, there will be one copy of the qoc file in memory for each of those interviewers. On systems with particularly heavy memory requirements, this can cause slow response times and other similar problems.

Sharing a qoc file places the file in an area of memory that is accessible by more than one user. A reference to that area of memory is held in the file *script.qsh*. This means that however many interviewers you have working on a project there will always be just the one copy of the qoc file in memory which is being used by all interviewers.

If you need to reshare a qoc file that has already been shared, shareqoc does not delete the old segment for the file before creating a new one. To avoid a proliferation of unwanted segments, it is best to wait until no interviewers are using the shared segment and then delete it using rmshm. You can then run shareqoc to share the qoc file in the usual way.

Sharing a qoc file does not start an interview.

The environment variable QCUPDATE, which allows interviewers to pick up a new version of the qoc file without leaving qtip, does not work with a shared qoc file.

- ☞ If the computer is rebooted during the course of the study (even if interviewing is not taking place at the time), you will need to reload the qoc file into memory before interviewing starts.
-

34.3 Reviewing completed interviews

From time to time you may wish to review completed interviews, perhaps to check that the interview has been carried out correctly. To do this, you must have the environment variable QCCOMP set to 1 or 2.

At the start of the qtip session, you will be prompted for the serial number of the interview to be reviewed, in the same way that qtip prompts for restarting stopped interviews. If QCCOMP is set to 1, qtip will ask whether you wish to change or inspect anything about the interview as it normally does at the end of any interview. Type 'y' at this point to have the last question and its current answer displayed on the screen. If QCCOMP is set to 2, qtip immediately displays the first question. You may then use the standard interviewing commands to snap to any point in the script to check the answers given.

Take care if the script contains *strngchk* statements which check the values of open-ended questions. If the review process encounters an invalid text, it will return control of the interview to you and will wait for you to enter an acceptable answer. If the answer you enter does not affect the path through the script, qtip will revert to review mode for the rest of the interview. If the change affects the routing, you will be prompted to enter answers to each question from that point onwards.

Suppose you are reviewing a script which contains the following statements:

```

PCODE      ask 'What is your postcode.'
            resp coded (0)
            set pcfmt1 = '%%# %%'
            callfunc('strngchk', pcfmt1, PCODE, ISOK)
            route (ISOK) go Q3
BADPC      ask 'We were expecting your postcode to start with two
            letters and a digit. Yours does not. May I just check that'
            +PCODE+ is correct?
            resp sp 'Yes, correct' / 'No, incorrect' go PCODE
Q3         ask '.....'
            resp .....
```

There are three possibilities here:

- The interviewer enters a valid postcode. When you review this interview, qtip passes directly from PCODE to Q3. If you replace the valid postcode with an invalid one, the review process stops at the BADPC question and waits for you to confirm that the postcode is acceptable.
- The interviewer enters an invalid postcode but says that it is correct. When you review this interview, qtip passes directly from PCODE to Q3 because the interviewer originally accepted the invalid postcode. If you change the original postcode, it will always be accepted without question because the answer to BADPC indicates that it may be accepted.
- The interviewer enters a valid postcode but then overwrites it with an invalid one at the prompt to append verbatims. In this case, qtip was unable to check the validity of the second postcode, but the review process checks it and finds that it is invalid. This changes the route through the script so qtip displays the BADPC question and waits for you to answer it.

Other points to note when reviewing interviews are:

- The *querysms* callable function returns null,
- The *redial* and *qc_hangup* callable functions do nothing,
- The *setivr*, *prepare_ivr*, *commitivr* and *abendivr* callable functions do nothing,
- '#audio/record' plays back the recorded file.
- If a *quota* statement uses a multipunched variable as an axis, it is passed with the cell variable set to the first cell mentioned, irrespective of which cell caused the quota to pass in the original survey.
- If the script contains quotas, the review process will leave quota information unchanged, and the interview will always pass the quota checks successfully.

At the end of the interview qtip will ask 'Do you want to write data now?'. Unless you have changed the data in some way you should answer 'n' to this question.

Reviewed interviews are copied to a subdirectory *script.ddr/USED*; once an interview has been reviewed it cannot be reviewed again while there is a copy in USED. This is a safety precaution, and if you try to review an interview a second time, a message flashes on the screen reminding you to rename or remove the file in this directory.

You can use the review facility for testing scripts which have failed with run-time errors. If you record the keystrokes of an interview (see below) in the state which causes it to fail and then change and recompile the script, you can test your changes by reviewing the saved interview. If your change has been successful, qtip will stop when it runs out of recorded information and will wait for you to continue the interview by typing answers on your keyboard.

34.4 Recording keystrokes of interviews

Quick Reference

To record the keystrokes used in an interview, type:

touch recordng.qct

To replay a keystroke recording file, type:

qtip *script < recipid*

where *pid* is the interviewer's qtip process ID number.

Quancept provides a facility for recording the keystrokes used during an interview or series of interviews carried out by one or more interviewers. A separate recording file is created for each interviewer called *recpid*, where *pid* is the interviewer's qtip process ID number. For example, keystrokes for the qtip process numbered 19797 would be stored in rec19797. These files are created in a subdirectory of the project directory called recordng.dir.

Recording keystrokes is not an automatic process: it only happens if the project directory contains a file called *recordng.qct* (notice that there is no letter 'i' in the file name). To stop recording, simply delete this file.

Here is a short sample file to act as a reference for explaining what goes into a keystroke recording file:

```
lastwk    ask 'How did you travel to work last week?'
           resp mp 'bus' / 'train' / 'car' / 'other'
           set oth = bit(lastwk/6)
           if (oth) {
qoth      ask 'What other transport did you use?'
           resp coded(9)
}
thiswk   ask 'How have you traveled to work so far this week?'
           resp mp 'bus' / 'train' / 'car' / '[+qoth+]' / 'other'
d1       display 'lastwk was '+lastwk+ and thiswk was '+thiswk
           pause
           end
```

We recorded the keystrokes for two interviews (we have added line numbers for reference purposes later on in this section):

```
1: #19797 Fri Nov 30 18:43:04 1990
2: barbara
3: n1 2 3
4: 2 3 5
5:
6: n**** 4 *** Fri Nov 30 18:43:21 1990
7: y1 2 3
8: 1 2 3
9:
10: yn<<
11: n1 4
12: horse and cart
13:
14: 1 2 3 4 5
15:
16: nn**** 5 *** Fri Nov 30 18:41:11 1990
17: n
```

The file starts with a comment naming the qtip process ID and giving the date and time recording was started. When prompted for an interviewer name we gave it as barbara.

The n at the start of line 3 was the answer to the question about restarting stopped interviews. This is followed by the answers to the first question. Three codes were entered representing the answers bus, train and car. The reason there is no space or new line between the n and the responses is because we did not press Return after typing 'n'. Pressing Return results in a new line in the file, and pressing Return twice results in a new line followed by a blank line.

Line 4 contains the responses to the next question: in this interview it was thiswk because the respondent did not choose 'other' at the first question.

The blank line is the result of pressing Return at the prompt 'Press any key to continue' issued by *pause*.

Line 6 starts with the response to the question about inspecting or changing the interview (we didn't change anything). This is followed by the serial number of the interview and the date and time at which it was written to the data file.

Line 7 starts with 'y' indicating that we said we wanted to start another interview. The answers to the first question follow immediately after it.

Lines 8 and 9 are similar to lines 4 and 5.

Line 10 shows that we said we did want to change the interview ('y') and that the last answer was incorrect ('n'); we then typed << to return to the first question.

Having said the current answer was wrong, we entered a new set of response codes. This time we chose specified other and said that we traveled by horse and cart. We pressed Return twice to terminate the text, hence the new line followed by a blank line.

Line 14 is a new set of responses for the last question.

The next two lines mark the continuation after *pause* and the writing out of the data.

The last line is the answer to the question 'Another interview?'.

↖ Due to the way recording uses the file buffering in stdio (the I/O library), recording and playback cannot descend into a subsurvey.

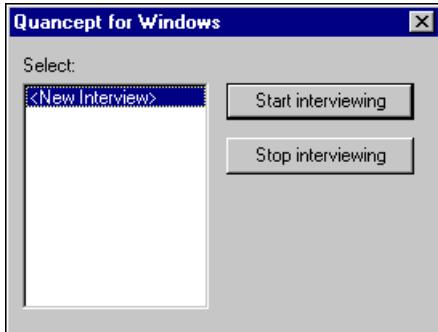
35 Testing CAPI scripts

Once you have parsed a Quancept CAPI script successfully, you can test it by running it as an interviewer would see it. This chapter explains briefly how to do this. If you need more detailed information on how to deal with specific types of questions, refer to the *Quancept CAPI Interviewer's Guide*.

-
- ☞ For information on the error messages that are displayed by the interviewing program, see section 33.32, qtip error messages.
-

35.1 Starting new interviews

To run test interviews using QCompile, select a script and click the Run button. The interviewing program starts and displays the interviewing window overlaid by the following dialog box:

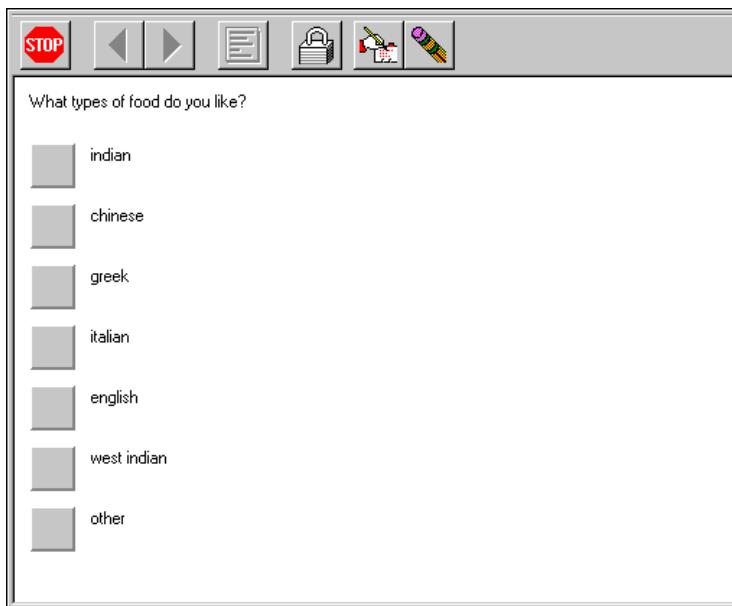


Click the Start interviewing button. The dialog box closes and the first text in the interview is displayed in the interviewing window.

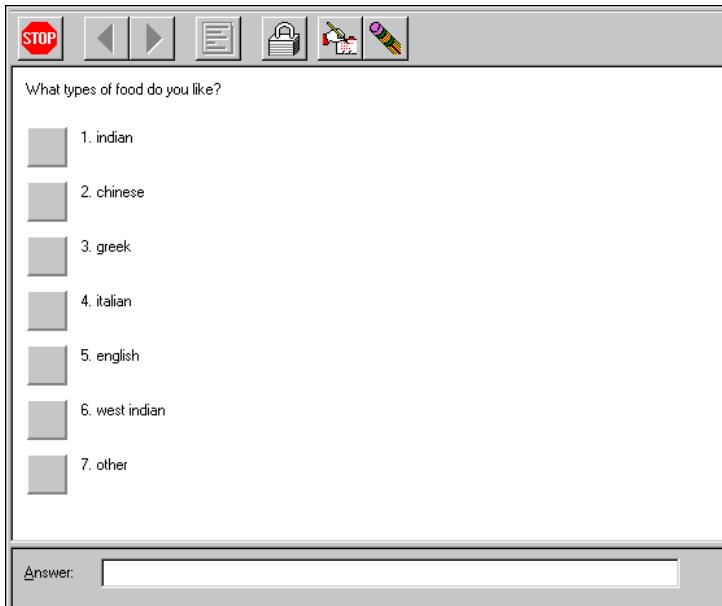
-
- ☞ You can suppress this dialog box and go straight into a new interview by selecting the 'Test interview' option on the Run tab in QCompile. When the interview ends you are returned to the main QCompile window and will need to click Run again to run another interview.
-

The interviewing window

The interviewing window for use with a mouse or light pen looks like this:



If you are using the keyboard interface, each single-punched or multipunched response text is preceded by a number and an answer box is displayed at the foot of the window:

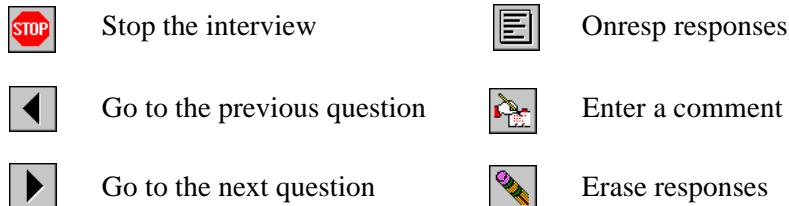


In both cases, the blank box below the response frame is where the interviewing program displays the answers given to previous questions. You'll use this box if you need to check the answers to those questions.

The toolbar

Unless you request otherwise, the toolbar is displayed at the top of the window and with six buttons as shown in the examples. You can position the toolbar at the bottom of the window or on the left or right, and can display only the first four buttons, by setting options on QCompile's Run tab.

Buttons in the standard six-button toolbar are as follows:



35.2 Usage summary

The table below summarizes the facilities you have for testing interviews using a mouse, a light pen or the keyboard. Unless otherwise stated, actions for light pens are the same as those for the mouse, except that instead of clicking you point instead.

To	Mouse or light pen	Keyboard
Enter a single-punched or multipunched response	Click on the response's check box.	Type the response number. Follow numbers for multipunched responses with a space.
Enter a numeric response	Select the number using the numeric keypad that the interviewing program displays.	Type the number.
Enter a numeric slider response	Move the marker to the appropriate point on the slide bar.	Either type the number or use ALT sequences to move the marker on the slide bar. On laptops, type ALT, Fn + to move the marker to the right or ALT, Fn, – to move it to the left. On desktop PCs, hold down the ALT key and press + to move the marker to the right, or hold down the ALT key and press – to move it to the left
Enter an open-ended response	Type the text in the box provided.	Type the text.

To	Mouse or light pen	Keyboard
Enter a specified other response	Click the Other check box and then type the response text in the box provided.	Type the Other response number, then a space, and then the response text enclosed in double quotes.
Enter responses to <i>multitask</i> questions	Select or enter a response for each question according to its type.	Type the question letter followed by the response to that question according to its type. Responses that contain spaces and the special responses <i>null</i> , <i>dk</i> and <i>ref</i> must be enclosed in parentheses. For example, a(1 5) b(6 "Japanese food").
Select responses from a response list database	Click on the response or responses in the list.	Type the response number. If multiple selections are allowed, type a space after each number.
Reduce a database response list to show only responses that start with a specific letter or string	Type the letter or letters in the blank input box and click the Find button.	Move the screen focus away from the answer box by clicking the interview page using a pointing device. Then tab between the Find box, the Find button and the list. A dotted rectangle around the word 'Find' indicates when focus is on the button, at which point you press Enter to start the search. Once the search is complete, press ALT+A to return to the Answer box..
Cancel a database search pattern and return to the full list	With a mouse, highlight the text and click Delete then Find. With a light pen, touch the asterisk button in the letter list and then touch Find.	Backspace over the search text, then tab to the Find button and press Enter.
Enter a No answer response	Click the No answer check box.	Type <i>null</i> .
Enter a Don't know response	Click the Don't know check box.	Type <i>dk</i> .
Enter a Refused response	Click the Refused check box.	Type <i>ref</i> .

To	Mouse or light pen	Keyboard
Select responses in a multicolumn grid	Click on at least one check box in each column.	For each column, type a response row letter followed by a response column number, for example, A5 for the first response in column five, or C1 for the third response in column one.
Write a response on the screen (<i>scribble</i>)	Draw in the box provided using the mouse or light pen.	Not applicable.
Play or stop a video clip	Click the Video button.	Type ALT+V.
Play or stop a sound file	Click the Audio button.	Type ALT+P.
Move to the next question	Click the Next button.	Type SHIFT+ENTER.
Return to the previous question	Click the Previous button.	Type SHIFT+BACKSPACE.
Record or stop recording an audio response	Click the Record button.	Type ALT+R.
Display the cursor in a text or answer box	Click in the box.	Type ALT+A.
Scroll up and down the screen	Use the scroll bar.	Press Page Up or Page Down.
Cancel an incorrect response	Click the response's check box.	Backspace over the response number.
Erase all responses chosen for a question	Click the Erase button.	Press F10.
Stop an interview	Click the Stop button. See also 'Stopping and restarting interviews' below.	Press F4. See also 'Stopping and restarting interviews' below.
Use fast jumps (<i>onresp</i> s)	Click the Onresp button and select an item from the list.	Press F7. Use the arrow keys to highlight an item in the list and then press ENTER to select it. To close the list without making a selection, press Esc.
Lock the session	Click the Lock button and enter a password of up to 16 characters in the Lock Session dialog box. Click OK to continue.	Press F8 and enter a password of up to 16 characters in the Lock Session dialog box. Press ENTER to continue.

To	Mouse or light pen	Keyboard
Enter a comment	Click the Comment button and enter a comment in the Comments dialog box. Click OK to continue.	Press F9 and enter a comment in the Comments dialog box. Press ENTER to continue

35.3 Stopping and restarting interviews

Quick Reference

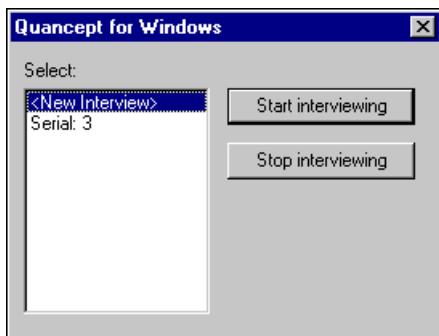
To stop an interview:

Mouse: Click the Stop button.

Keyboard: Press F4.

To restart a stopped interview, select its serial number on the initial dialog box and then click the 'Start interviewing' button.

When you stop an interview, the interviewing program displays a dialog box asking you to confirm your action. If you select Yes, a second dialog box asks whether you will want to restart this interview. If you select Yes, Quancept CAPI saves the data gathered so far in the file stopped\resp_num.stp in the project directory, where resp_num is the respondent's serial number. The interview closes and you are returned to the initial dialog box. This now displays the serial number of the interview you have just stopped:



When you restart an interview, the first question displayed is the one at which you stopped the interview. The Previous button is activated in case you want to step back and review the saved portion of the interview. To complete the interview, answer the remaining questions in the usual way.

35.4 Automatic testing

If you want to do bulk testing of your script you can run it in automatic mode. The interviewing program will run the number of interviews you request, each time selecting a different combination of answers.

To run interviews in test mode:

1. From the menu select Script, Options.
2. Select the Special tab.
3. In the Autotest box, click Enable and enter the number of interviews you want to generate.
4. Click OK.
5. Select the script and click Run.

36 Testing Quancept Web scripts

This chapter explains how to install a script on the Quancept Web server ready for testing, and how to stop and restart interviews. Apart from these topics, testing a script simply consists of selecting the responses you want and clicking on the Next, Previous or Stop buttons to move to the next or previous question or to stop the interview.

36.1 Files required for interviewing

The Quancept Web interviewing program requires a number of files to run successfully. Some of these files are required for any Quancept script and others are specific to each project. You must ensure that all the files your project needs from this list have been copied onto your Quancept Web server.

Filename	Description
<i>project.sif</i>	The compiled version of the Quancept script.
<i>project.qti</i>	Question and response texts for <i>project.sif</i> .
<i>filename.gif</i> / <i>filename.jpg</i>	Picture files. These files may be shared by multiple projects and can be stored in a directory other than the project directory.
script files	Files dependent on the script, for example, databases for questions of type <i>resp dbase</i> .

36.2 Running the script

This section lists the minimum you need to do in order to install your script on the Quancept Web server and test it.

1. Write your script and then compile it with QCompile. Make sure that you have the SIF file option selected when you compile the script.
2. On your Quancept Web server create a directory for the project in the qcweb\qcserver\projects directory. The directory must have the same name as the compiled script file so, if the compiled script file is called *qctest.sif*, you'll create a directory called qcweb\qcserver\projects\qctest.
3. Copy the *project.sif* and *project.qti* files that QCompile created into the new project directory. The way you do this will depend on the way your network is set up and the type of networking software your company is using.
4. Open your Web browser and test the script by entering the location (URL) as:

`http://server_url/scripts/qcweb.dll?starthtml&qcserver_name&project_name`

where:

- server_url* is the address of your Quancept Web server (when you're testing your own scripts you can enter the name of your server here)
- scripts* is the name of the directory in which the file qcweb.dll is installed (usually scripts). Just type the directory name, not the full pathname.
- qcserver_name* is the name of your Quancept Web server
- project_name* is the project's name

For example:

`http://zeus/scripts/qcweb.dll?starthtml&zeus&qctest`

In this example, zeus is the name of the Quancept Web server and the project is called qctest.

Once the survey is live, outside users can run it by typing, for example:

`http://www.spss.com/scripts/qcweb.dll?starthtml&zeus&qctest`

36.3 Stopping and restarting interviews

Quick Reference

To restart a stopped interview, type:

`.../scripts/qcweb.dll?starthtml&server_name&project_name&unique_ref`

or:

`.../scripts/qcweb.dll?starthtml&server_name&project_name&unique_ref&both`

where *scripts* is the name of the directory containing qcweb.dll, *server_name* is the name of your Quancept server, *project_name* is the project name, and *unique_ref* is the respondent's unique reference code.

Quancept Web does not automatically display a Stop button. If you want to test the stopping and restarting of interviews you must request a Stop button. You do this by placing the statement:

StopButtonEnabled=Yes

in the [HTML] section of the project's initialization file. When you click this button, the data gathered so far is saved in a *resnum.stp* file, where *resnum* is the respondent serial number, so that the interview can be restarted.

When a project allows respondents to stop and restart interviews, you need to start Quancept Web with a command that caters for two different types of respondent. These are respondents who are coming to the project for the first time and will be starting a new interview, and respondents who want to continue interviews that they have previously started and stopped part-way through.

Normally, neither you nor your prospective respondents will type these commands. Instead, you'll create a CGI program that runs them for you. For example, your program may contain a form that prompts for the user's name and then runs one of these commands, with the user's name substituted in place of *unique_ref*.

Writing the CGI program is your responsibility, but to help you SPSS MR distributes some **unsupported** sample scripts with suggestions on how you can provide a simple stop-start mechanism for Web interviews.

-
- ☞ Other less frequently used methods of restarting stopped interviews are described later in this chapter.
-

Behind the scenes with the stop-start mechanism

The Quancept Web mechanism for stopping and restarting interviews is based on a cross-reference table held in the idm file in the project's directory on the Quancept Web server.

When a respondent starts an interview he/she is allocated two identifiers. The first is the respondent serial number. This is used for marking the respondent's data in the data file and, if the respondent stops the interview, as part of the filename for the stopped portion of the interview. The second identifier is a name or code that the respondent enters when prompted to do so at the start of an interview. The idm file keeps track of which respondent serial numbers belongs with each unique reference code.

When a respondent starts an interview and enters his/her unique reference code, Quancept searches for this code in the idm file. If it finds a match it checks the related respondent serial number and looks for a file with that name in the stopped data directory. If the respondent has a stopped interview awaiting completion, Quancept restarts that interview. If not, it starts a new interview.

For example, suppose the respondent's unique reference code is Ben123. Quancept searches for Ben123 in the idm file and finds a cross-reference to respondent serial number 2019. It then looks for the file *project\STOPPED\2019.stp* in the qcweb\projects directory on the server and restarts the interview using the data held in that file. If there is no cross-reference for Ben123, Quancept generates a new respondent serial number, updates the idm file with this information and then starts a new interview.

36.4 Displaying the respondent's interview ID

Quancept Web assigns each respondent two numbers: a respondent serial number and an interview ID. The respondent serial number is a sequential count of interviews started and is included in the respondent's data record; the interview ID is a unique character sequence assigned to each interview for the purposes of stopping and restarting interviews. Quancept Web maintains a cross-reference table of serial numbers and interview IDs in the file *project.idm* in the project directory on the Quancept Web server.

When a respondent stops an interview and then attempts to restart it, Quancept Web prompts for the respondent's interview ID so that it can retrieve the correct interview. If you want the respondent to be able to restart a stopped interview, you need to tell him or her what ID Quancept Web has assigned to the interview. There are two ways of doing this: using variables in the project's ini file and placing statements in the script.

Variables the ini file

Quick Reference

To display the respondent ID, or indeed any other text, at the start of the interview type:

ShowStopStringAtStart=*text*

To display text, including the interview ID, when the respondent stops an interview, type:

ShowStopStringAtEnd=*text*

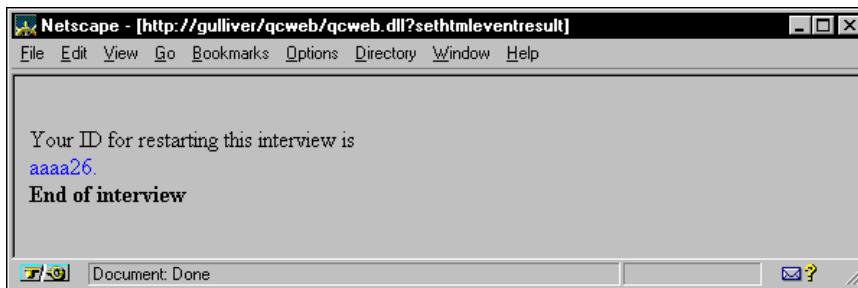
In both cases the default text is 'no' which turns off these features.

The previous sections have talked a lot about interview IDs but have made no mention of how the respondent knows what his/her ID is. If you want the respondent to have this information in order to restart an interview you must display it during the interview. There are two variables you can use for this, depending on when you want to display the information. Both are declared in the *project.ini* file.

As an example, the definition:

```
[HTML]
ShowStopStringAtEnd=Your ID for restarting this interview is<BR>
```

generates the following display if the respondent stops the interview. Notice how Quancept Web has inserted the respondent ID at the end of the *ShowStopStringAtEnd* variable even though there is nothing in the text that says to do so.



Statements in the script

Quick Reference

To display the respondent's interview ID, type:

```
set variable = intname
display ['text']+variable
```

- ☞ This use of *intname* is unique to Quancept Web. *intname* also exists in other variants of Quancept but with a different meaning.

36.5 Other methods of restarting stopped interviews

This section explains other methods of restarting stopped interviews that you may find useful in particular situations. These are:

- Have each attempt generate a new interview even if the respondent has a stopped or completed interview. Stopped interviews cannot be restarted.
- Start interviews for new respondents only. Respondents who have completed or stopped interviews are rejected. Stopped interviews cannot be restarted.
- Restart a stopped interview for this respondent if one exists, otherwise do nothing. This prevents new interviews taking place.

Always start a new interview

Quick Reference

To start a new interview, type:

```
.../scripts/qcweb.dll?starthtml&server_name&project_name
```

where *scripts* is the name of the directory containing qcweb.dll, *server_name* is the name of your Quancept server and *project_name* is the project name.

Quancept generates a new interview ID automatically and associates it with a respondent serial number. Stopped interviews cannot be restarted unless you also provide users with access to one of the other commands listed below.

Start interviews for new respondents only

Quick Reference

To start interviews for new respondents only, type:

```
.../scripts/qcweb.dll?starthtml&server_name&project_name&unique_ref&new
```

where *scripts* is the name of the directory containing qcweb.dll, *server_name* is the name of your Quancept Web server, *project_name* is the project's name and *unique_ref* is the unique reference code (for example, the respondent's name) for this interview.

This option gives you tighter control over exactly when a new interview is started. If the respondent has already completed an interview for this project, or has a stopped interview waiting to be restarted, Quancept will issue the message 'The Respondent ID is already stopped or complete' and stop. New interviews occur only for respondents who have not run the survey before.

You can define the message that the respondent sees when an interview is not possible by using the *ErrStopRecordExists* variable in the HTML section of the *project.ini* file. For example:

```
[HTML]
ErrStopRecordExists=You cannot start a new interview because a completed or
stopped interview already exists for your respondent ID.
```

Always restart stopped interviews but never start new ones

Quick Reference

To restart stopped interviews but never start new ones, type:

```
.../scripts/qcweb.dll?starthtml?server_name?project_name?unique_ref&restart
```

where *scripts* is the name of the directory containing qcweb.dll, *server_name* is the name of your Quancept Web server, *project_name* is the project's name and *unique_ref* is the unique reference code (for example, the respondent's name) for this interview.

This is the opposite of the previous option as it will only restart stopped interviews and never start new ones. You might use this near the end of a survey when you are trying to complete stopped interviews but do not want to start any new ones.

If there is no stopped interview for the current respondent Quancept issues the message 'The Respondent ID was not found' and stops. You can define your own message using the *ErrNoStopRecord* variable in the *project.ini* file's [HTML] section. For instance:

```
[HTML]
ErrNoStopRecord=Unable to continue as there is no stopped interview for your
respondent ID
```

Prompt the user for an interview ID before looking for a stopped interview

Quick Reference

To prompt the user for an interview ID before looking for a stopped interview, define a suitable prompt using the *RestartGreeting* variable in the project's initialization file. Then type:

```
.../scripts/qcweb.dll?starthtml?server_name?project_name?unique_ref&restart
```

where *scripts* is the name of the directory containing qcweb.dll, *server_name* is the name of your Quancept Web server, *project_name* is the project's name and *unique_ref* is the unique reference code (for example, the respondent's name) for this interview.

This variation of the previous restart option allows you to prompt the user for a respondent ID and to search for a stopped interview with that ID. If Quancept finds a suitable stopped interview it restarts it, otherwise it stops with the message 'The Respondent ID was not found'. You can customize this message using the *ErrNoStopRecord* variable as described earlier.

The prompt for a respondent ID is:

```
Restarting your questionnaire<BR>
Please enter your Respondent ID:<PRE>           </PRE>
```

Notice the use of `<pre>...</pre>` to insert spaces between the end of the prompt and the point at which the user enters his/her ID. You can change this prompt by defining your own with `RestartGreeting` in the [HTML] section of `project.ini`.

The user types a respondent ID and then presses a button whose default label is Start. You can change this with the `RestartSubmitButton` variable in the [HTML] section of `project.ini`.

Here is an extract from a `project.ini` file and the screen the respondent sees when he/she starts the interviewing program:

```
[HTML]
RestartGreeting=Please enter your respondent ID
RestartSubmitButton=Begin
```



37 Testing mrInterview scripts

This chapter explains how to test mrInterview scripts and how to check that the data being written to the database is correct.

37.1 Testing your script

Open your Web browser and type:

`http://server_url/scripts/mrwebpl.dll?project=project_name`

where:

server_url Is the address of your Interviewing server (if the whole mrInterview system is installed on the machine you're using for scriptwriting, you may be able to enter *localhost* here).

scripts Is the name of the directory in which the file mrwebpl.dll is installed (usually /InetPub/scripts). Just enter the directory name, not its full pathname.

project_name Is the project's name.

For example:

`http://localhost/scripts/mrwebpl.dll?project=tea`

In this example, the user's own computer is being used as the Interviewingt server. The project is called tea.

The first question is displayed in your browser window. Use the Next, Previous and Stop buttons to navigate through the interview.

-
- » If you are using DimensionNet, you can test the script by selecting the project and then clicking Test Interview.
-

Stopping and restarting interviews

The following points are mentioned at various points in this Guide, and are reproduced here for easy reference.

- The standard navigation bar does not display a Stop button unless you request one by placing the statement *unset hidestop* in the script at the point you want the button to appear. You can hide the Stop button again using *set hidestop=1*.

- If you stopped the interview by clicking Stop and the project does not use sample management, use your browser's Back button to display the page on which interview stopped, and continue the interview from that point.
- If the project uses sample management, the sample management system will usually provide instructions on restarting the interview. For example, when the interview stops you may be given a special restart code. You may then have to re-open the project's authentication page and enter this code to restart the interview. Failing to enter a valid restart code may result in a new interview or no interview at all.
- If the interview is stopped by a *stop* statement in the script, you will not be able to continue past this point unless the script contains code that bypasses the *stop* statement on restarted interviews.

Running test interviews on active projects

If you want to run test interviews on an Active project, add the parameter *test=1* to the URL that you type to run the interview. For example:

`http://localhost/scripts/mrwebp1.dll?project=tea&test=1`

to run test interviews on the tea project. The data for these interviews will be marked as Test in the case data.

37.2 Checking the data

mrInterview writes its data to a relational SQL database with the same name as the script. The structure of this database and the tables it contains are described later in this guide, but you do not normally look at the data in this format. Instead, you are more likely run a program that extracts the data into a more easily readable layout, where the data for each respondent is presented in a linear form. Your company may have a particular program that is used for this; if not, you can use the Excel macro that comes with mrInterview to extract data from the database and present it in a worksheet.

Data structure

Responses and other values that are written into the data are held in variables that are named according to the question or variable names in the script. Different types of response require different types of variables:

Response type	Case data variables
Single and multiple categoric responses	A categoric variable whose name matches the question name in the script. The variable stores a code that represents the response text. Each categoric response in the questionnaire is assigned a unique code.

Response type	Case data variables
Numeric responses	A long (<i>resp num</i>) or double (<i>resp real</i>) variable whose name matches the question name in the script.
Open-ended responses	A text variable whose name matches the question name in the script, and a categoric variable named <i>qname</i> .Codes that can be used by the Verbastat coding program for storing the codes assigned to the response.
Specified other	A text variable named <i>qname</i> .Other to hold the text entered by the respondent, and a categoric variable named <i>qname</i> .Other.Codes that can be used by the Verbastat coding program for storing the codes assigned to that response.
<i>Null, dk and ref</i>	A categoric variable named <i>qname</i> .Codes if the question has a numeric or open-ended response list. If chosen from a categoric response list, these responses are stored in the standard question variable in the same way as ordinary categoric responses.

☞ See section 38.4, The mrInterview case data file, for further information about the structure of the database.

As an example, consider the following script:

```

q1 ask 'Which of the following types of tea do you drink?'
      resp mp 'Assam' ('assam') / 'Darjeeling' ('darjeeling') /
             'Lapsang Souchong' ('lapsang') / 'China' ('china')
             other null
q2 ask 'Which other teas do you drink?'
      resp coded
q3 ask 'How many cups of tea do you normally drink each day?'
      resp num 1 to 30

```

Question q1 requires three variables: q1 to store the categoric responses chosen (*null* is stored as a category), q1.Other to store the text entered for *other*, and q1.Other.Codes to store the codes assigned when the other responses are coded.

Question q2 requires two variables: q2 to store the respondent's answer and q2.Codes to store the codes assigned to the response during coding.

Question q3 requires just one variable called q3 to store the number.

Looking at the data

When you're at the testing stage, you'll want a quick and easy method of checking that the answers you enter in your browser are being stored correctly in the database, and that any questions that are not answered have blank variables in the database. You can always open the database using Microsoft SQL Enterprise Manager or use a Data Source Component (DSC) to export the data in a particular format, but these methods do not always produce data that can be easily checked.

mrInterview has its own tool for checking data, which is part of the Dimensions Development Library. This is an Excel macro called GetMRData that extracts variables from the database and presents them as separate columns in an Excel worksheet. The data for each test interview forms a row in the worksheet.

-
- ❖ Do not use this macro for exporting live data into Excel. The macro has been designed with simplicity of layout rather than running speed in mind, and will run extremely slowly on large databases.
-

Here is an example that illustrates the way the worksheet is formatted. The script that produced the data is as follows:

```
q1  ask 'Which types of tea do you drink?'
    resp mp 'Assam' ('assam') / 'China' ('china') / 'Darjeeling' ('darj') /
          'Lapsang Souchong' ('lapsang') / 'Orange Pekoe' ('opekoe') /
          'Puerh' ('puerh') / 'Rooibos' ('rooibos') /
          'Fruit/Herbal' ('fruit') other
q2  ask 'And do you normally use tea bags or loose tea?'
    resp sp 'Tea bags' ('bags') / 'Loose tea' ('loose') /
          'Use both equally' ('both')
q3  ask 'How many cups of tea do you normally drink in a day?'
    resp num 1 to 30 dk
    set herbal = bit(q1/8)
    if (herbal) {
q4      ask 'What do you think are the benefits of drinking
        fruit or herbal teas?'
        resp coded
    }
end
```

GetMRData produces a worksheet that contains columns for all variables in the database, including system variables that store general information about the interview. System variables appear at the start of the record and have names that start with Respondent or DataCollection. You can delete the columns for all system variables except the first column which contains the respondent's serial number. This leaves you with just the columns associated with questions in the questionnaire. The illustration below refers to the example script and contains data for three interviews.

	A	B	C	D	E	F	G	H	I
1	Respondent.S	q1.Other.Code	q1.Other	q1 {assam,darjeeli}	q2 {loose_tea}	q3.Codes	q3	q4.Codes	q4
2		1					5		
3		2		{fruitherbal}	{tea_bags}		8		Has no caffeine and is a healthy alternative to
4		3	English breakfast;	{assam,lapsang_souchong,otally}	{use_both_equal}	{dont_know}	0		
5		4	English breakfast	{darjeeling,lapsang_souchong,fruitherbal,other}	{tea_bags}		3		Less tannin;Low caffeine
6									

-
- ☞ For information on using GetMRData, refer to your Dimensions Development Library documentation. If you do not have the Dimensions Development Library but you have Excel 2002, you can import the mrInterview test data directly into Excel 2002. See your SPSS MR Data Model documentation for details.
-

38 File formats

This section describes the files created by the Quancept and mrInterview parsers, compilers and interviewing programs.

38.1 Files created by the parser

All files created by the parser are named *script.xxx*. The mrInterview version of QCompile is based on QCompile for Quancept CAPI and Quancept Web, but because of the differences between Quancept and mrInterview some of the files that QCompile creates are not applicable to mrInterview. These files are identified by the letters NA in the table below and are not flagged with the mrInterview icon in the rest of this chapter.

Extension	Description	qpars	QCompile CATI/Web	QCompile mrInt.
acl	Script in a form suitable for creating flowcharts with Quanflow	X	✓	NA
bqt	Designed to hold barcode information, but always empty	✓	X	X
bt8	8-bit characters found in the script	✓	X	X
err	Error summary	✓	✓	✓
exe	Quancept CAPI interviewing program	X	✓	X
exp	Intermediate file created during the creation of the sif file	X	✓	X
fr1	Information about questions allowing open ends for use by QCformat when reformatting data	✓	X	X
fr2	Information on precoded responses for questions with open ends	✓	X	X
ipk	Names of files called by the script	X	✓	X
lib	Intermediate file created during the creation of the sif file	X	✓	X
lst	Script compilation listing	✓	X	X
map	Data map	✓	✓	NA
maq	Always empty	✓	X	X
mdd	Questionnaire definition	X	X	✓

Extension	Description	qparsE	QCompile CATI/Web	QCompile mrInt.
mdm	Intermediate file used in the creation of the mdd file	✗	✗	✓
mqw	Data map of the script for use by Quanquest when creating Quantum analysis specs	✓	✓	NA
psm	Always empty	✓	✗	✗
ptf	Project text file	✓	✓	✗
qc8	Script with all 8-bit characters stored as 7-bit characters	✓	✓	NA
qdf	List of question names appearing in the data file, for use by QCformat	✓	✗	✗
qdi	Data map of the script for use by Qwincode and QDITum	✓	✓	NA
qdt	Intermediate file containing texts from the script for use in the creation of the qdi file	✓	✓	NA
qoc	Quancept CATI compiled script	✓	✗	✗
qtv	Maps temporary variable names to numbers	✓	✗	✗
quo/quiz	Quota file for Quancept Web	✗	✓	✗
ref	Miscellaneous reference information for use by the quac open-end coding program	✓	✗	✗
sif	Quancept Web interviewing program	✗	✓	✓
sum	Compilation summary	✓	✓	NA
tx0	A numbered list of all the texts that appear in the script	✓	✗	✗
txx	The index that has been assigned to the script's unique IDs, <i>rotranfix</i> , <i>rotransub</i> , <i>sp</i> , and <i>promptoth</i> keywords, and a secondary index to the text to which the ID/keyword applies, as listed in the tx0 file	✓	✗	✗
xin	Information for the <i>eximport</i> statement for use by qtip when importing or exporting GNU databases	✓	✗	✗

Intermediate files



Qcompile creates a number of intermediate files while it runs. You will normally choose to have these files deleted automatically at the end of the compilation, by selecting the appropriate options on QCompile's Parse/Compile tab. However, if there are problems with your script that are not simply the result of mistypings or incorrect usage, your support representative may ask you to keep these files for investigation.

The following files contain various types of messages that would otherwise have been displayed on the screen (the *id* in the filenames is your two-character user ID):

wconv8*id*.out wconv8*id*.err wqcpar*id*.out wqcpar*id*.err

The following files are created by wqcparse.exe and contain C++ code:

ccdes1.esr ccnew2.esr ccnew3.esr cclist3.esr
ccinit4.esr cclogic5.esr cemapfil.esr cctexts.esr

Quancept Web also creates the intermediate files sif.obj and sync.obj, but these cannot be deleted automatically.

mrInterview also creates a file called Qdimdm.err that lists any errors that occurred during the creation of the *script.mdm* file.

Directories



Qcompile creates the following subdirectories in the project directory ready for use by the interviewing program.

Directory	For storing
audio	Wav files for recorded open ends
bitmap	Bmp files created for <i>scribbled</i> responses
info	Messages that interviewers leave for supervisors (not created for Quancept Web)
RESULTS	Data files and the timing file if timings are held separately
stopped	Data files for stopped interviews

☞ These directories and the files they contain are discussed later in this chapter.

The acl file



This file holds stores question names and question texts, and is used by Quanflow to present a Quancept script as an allCLEAR 3.5 flowchart. For example, the script:

```
color  define 'red' / 'blue' / 'green' / 'yellow' other
favc  ask 'What is your favorite color?'
      resp sp color other
      set x = nbit(favc)
      if (x<4) {
          goto ctn
      }
rbgy  ask 'Which of the following colors do you like?'
      resp mp 'red' / 'blue' / 'green' / 'yellow'
ctn   continue
name  ask 'What is your name?'
      resp coded
age   ask 'How old are you?'
      resp num 10 to 16 go exit / 17 to 99
car   ask 'What car do you drive?'
      resp dbase('cars')
exit  end
```

appears in the acl file as follows:

```
{*hide all}
Start of Study.
-color
-favc
favc: What is your favorite color?.
{*shape 72}
    Is (x<4)?
    (
    >ctn
?end
-rbgy
rbgy: Which of the following colors do you like?.
-ctn
{*shape 60}
...
-name
name: What is your name?.
-age
age: How old are you??
(..->exit)>exit
( )
?end
-car
car: What car do you drive?.
-exit
END OF STUDY.
```

The bqt file



This file is designed to hold barcode information, but as Quancept does not yet generate barcodes this file is always empty.

The bt8 file



This file exists only if the script contains 8-bit characters. 8-bit characters are those with accents such as: ä, á or ø. It is created by the program conv8 which is run automatically by the qparse shell script.

The err file

This file lists any error messages produced when your script is checked. Error messages are identified as follows:

—Error at | *keyword* | in statement at line *n*: *Explanation*

where *keyword* is the keyword at which qparse recognized there was an error. It may be this keyword itself that is incorrect or you may have made a mistake just before it — for example, a missing quote or parenthesis — that causes the keyword to be incorrect.

The exe file



This is the Quancept CAPI interviewing program.

The exp file



This is an intermediate file created during the creation of the sif file.

The fr1 file



This file contains information on column allocations for use by quac and vquac. It is always created, but only contains information if at least one response list is open-ended or contain specified other. If qparse needs to write anything to this file, it writes entries for every question whether or not its response is open-ended or uses specified other, otherwise it is empty.

There are one or two lines in the file for each question, with each line consisting of six fields separated by spaces. The fields are as follows:

Field	Description
1	question name
2	iteration number

Field	Description
3	response type
4	card number
5	start column number
6	end column number

The response type field (field 3) tells you what type of data the column will contain. Data types are as follows:

1	single punched	5	resp list
2	multipunched	8	fix statement
3	numeric	9	database
4	open end	10	real

Questions whose response lists contain specified other have at least one column for the precoded responses, plus three columns for coding specified other. qparse generates two lines for these types of questions; the first refers to the precodes and the second to the open ends.

Questions whose response type is coded have one line only which refers to the columns in which the open ends will be coded. Questions whose responses are not open ended and which do not use specified other have one line for the precoded responses only.

Let's look at a simple script and then see what its fr1 file contains.

```
master define 'one'/'two'/'three'
q1      ask 'first question'
        resp sp master
        for i = 1 to 2
q2      ask 'second question'
        resp mp 'red'/'blue'/'green'/'yellow' other
        next
q3      ask 'third question'
        resp coded(15)
        route (q1='one') go exit
q4ask   'last question'
        resp coded
exit    end
```

The fr1 file for this script will be:

```
q1 1 1 1 8 8
q2 1 2 1 9 9
q2 2 2 1 10 10
q2 1 4 1 11 13
q2 2 4 1 14 16
q3 1 4 1 17 18
q4 1 4 1 19 21
```

The first line refers to question q1. It tells you that q1 has 1 iteration, has a single-punched response list, and its data is stored in column 8 of card 1.

There are four lines for question q2 because it is in a loop. The first two lines refer to the precoded columns for iterations 1 and 2 respectively, while the second two lines refer to the columns for open ends.

Questions q3 and q4 both have one line only because they are ordinary open ends with no precodes. The script requested 15 codes only for q3, so to the column allocation is set accordingly.

The fr1 file is used by the open-ended coding programs quac and vquac to find out which columns have been allocated to each question, and which of those columns are for precodes and which are for open ends.

The fr2 file



This file is a variation of the fr1 file which concentrates on precodes for questions which have coded or specified other responses. If there are no questions with these types of responses, the file is empty.

For each question with an open end of any kind (*resp coded* or specified other), this file contains a line of six fields separated by spaces. The fields are as follows:

Field	Description
1	question name
2	an offset value which you can ignore
3	response type (see the fr1 file)
4	maximum number of iterations
5	number of precodes
6	number of open-ended codes

- | Field | Description |
|-------|--------------------------------------|
| 1 | question name |
| 2 | an offset value which you can ignore |
| 3 | response type (see the fr1 file) |
| 4 | maximum number of iterations |
| 5 | number of precodes |
| 6 | number of open-ended codes |

If the question is single-punched or multipunched, the line is followed by one line for each precoded response in the list, showing the response code and the response text. These lines are indented by one character. Here is the fr2 file for the script in the previous section:

```
q2 102 2 2 4 29
  1 red
  2 blue
  3 green
  4 yellow
q3 158 4 1 0 15
q4 205 4 1 0 29
```

Question q2 is multipunched (response type 2), has two iterations and four precodes (listed on the next four lines), and may have up to 29 open-ended codes.

Questions q3 and q4 are both coded (response type 4) and have one iteration and no precodes each. q3 may have up to 15 codes while q4 accepts the default of 29 codes.

This file is used by quac and vquac for building basic code frames for coding.

The ipk file



This file contains a list of all files called by the script, including bitmaps, audio files, video files and database files. These files need to be copied onto the interviewing PCs with the exe file.

The lib file



This is an intermediate file created during the creation of the sif file.

The lst file



This file is created when you use qparse with the -f option. It contains a listing of your script with line numbers. If errors are found, the messages are displayed beneath the offending statements.

The map file



This file tells you how the data will be laid out in the data file. In Quancept CATI it is used by the program qcset as the basis for the extended data map file required by qcprt (questionnaire printing), qctum (create Quantum axes) and qct (topline tabulations).

The map file contains a table showing, for each question, the question name, the card type, the columns in which the response appears, and finally the response type. If the script is:

```

color  define 'red' / 'blue' / 'green' / 'yellow' other
favc  ask 'What is your favorite color?'
      resp sp color other
      set x = nbit(favc)
      if (x<4) {
          goto ctn
      }
rbgy  ask 'Which of the following colors do you like?'
      resp mp 'red' / 'blue' / 'green' / 'yellow'
ctn   continue
name  ask 'What is your name?'
      resp coded
age   ask 'How old are you?'
      resp num 10 to 16 go exit / 17 to 99
car   ask 'What car do you drive?'
      resp dbase('cars')
exit  end

```

the data map will be:

```
##Data map created for TEST on Wed Aug 26 15:38:35 1987
```

qcp_ser_	Numeric	(1)	1	1	5
qcp_crd_	Numeric	(1)	1	6	7
cardtype 1 ...	type	iter	crd	frm	to
favc	Single	(1)	1	8	8
favc[other]	Other	(1)	1	9	11
rbgy	Multi	(1)	1	12	12
name	Coded	(1)	1	13	15
age	Numeric	(1)	1	16	17
car	Dbase	(1)	1	18	22

It tells us that the respondent's favorite color is single punched and will be stored in column 8 of card 1. The response list allows specified other in case an unlisted color is chosen, and columns 9 to 11 have been reserved for coding these colors.

The answer to rbgy may be multipunched. It has one iteration and will be stored in column 12 of card 1.

Name is a coded response which has been allocated the default of 3 columns for 29 codes.

Age has a response range of 10 to 99, so two columns have been allocated to store the response.

Responses picked from Quancept databases are coded using the record's five digit record number, so the answer to car has been allocated columns 18 to 22 on card 1.

-
- ❖ The way that questions are listed in the map, and the way columns are allocated in the data file depends on the way you parse your script.

When you run qparse, it allocates columns to each question either when a value is assigned to the question using a *set* or *unset* statement or when qparse reaches the definition of the question, whichever comes first.

However, if you use the *-m* option when you parse your script, qparse ignores *set* or *unset* statements and allocates columns in the order in which the questions are defined in the script. For instance, consider the following script:

```
unset q2
q1  ask 'text for q1'
     resp sp 'yes'/'no'
q2  ask 'text for q2'
     resp sp 'yes'/'no'
```

If you parse the script with qparse *-m*, the data map will contain the questions in the order q1 then q2. Without *-m*, the *unset* statement will cause qparse to allocate columns to q2 first, even though q1 is defined before q2.

- ❖ There are other options for parsing your script that can affect the data map; you can also define your own data map.
-

The maq file



This file is not currently used and is always blank.

The mdd file



This file, known as the questionnaire definition file, holds your questionnaire in a special format that allows for more than one version of the questionnaire to be stored and available at a time. This allows you to create different versions of the questionnaire for different markets, for example, using different brand names for the same product, or to translate the questionnaire to produce versions in different languages.

The parser creates the mdd file the first time you parse the questionnaire and adds a new version to the file each time you reparse the questionnaire. If you translate the questionnaire, the translations are added to the file and are then passed forward to any new versions created by reparsing after translation. You can, of course, update the translations or add new ones to reflect the changes you have made to the questionnaire.

The mdd file is written using the extensible mark-up language (XML). You can look at this file using any text editor, but as the lines tend to be very long and there is no formatting it is not always easy to follow.

The mdm file

 This is an intermediate file used in the creation of the mdd file.

The mqw file

 This is a data map file used by Quanquest when creating Quantum output.

The psm file

 This file is no longer used and is always blank.

The ptf file

 This file contains the script's texts in the original language. You can then use the Quancept translation program, Qolyglot, to translate these texts into other languages. The translations are then available when you use the script with other Quancept software; for example, when you run an interview with the interviewing program.

The file starts with a line defining the default language. This is followed by two lines for each text. The first line in each pair defines the variable's name, its data type and its record number in the file. Text are numbered sequentially from 1 and each text has its own unique number. With responses, the first line also tells you the response's position in the response list.

The second line in each pair consists of a language flag and the text in that language. As texts are translated, the translation program inserts additional lines for each language.

The question:

```
like      ask 'What types of food do you like?'
           resp mp 'indian' / 'chinese' / 'greek' / 'italian' / 'english'
```

appears like this in the ptf file:

```
LABEL=like;TYPE=ASK;RECID=0;
ENGTEXT="What types of food do you like?"
LABEL=like;TYPE=RESP;INDEX=1;RECID=1;
ENGTEXT="indian"
LABEL=like;TYPE=RESP;INDEX=2;RECID=2;
ENGTEXT="chinese"
LABEL=like;TYPE=RESP;INDEX=3;RECID=3;
ENGTEXT="greek"
LABEL=like;TYPE=RESP;INDEX=4;RECID=4;
ENGTEXT="italian"
LABEL=like;TYPE=RESP;INDEX=5;RECID=5;
ENGTEXT="english"
```

The qc8 file



This is an intermediate file created and used by the parser. It is a version of your script with any references to 8-bit characters written in such a way that Quancept can translate them into 7-bit references.

The qdf file



This file contains a list of the question names in the script, and is used by QCformat when filtering and writing out data.

The qdi file



The qdi file holds data mapping information for the project. It starts with a FILE section that describes the qdi file version and the maximum record size:

```
FILE=[  
    VERSION=0.2;  
    MAXRECSIZ=80;  
]
```

This is followed by one DATA section for each question or *fix* statement. If the question is:

```
like      ask 'What types of food do you like?'  
           resp mp 'indian' / 'chinese' / 'greek' / 'italian' / 'english'
```

the corresponding section on the qdi file will read:

```
DATA=[  
    VARNAME=like;  
    VARITER=[1]/[1];  
    VARTYPE=MP;  
    TEXT="What types of food do you like?";  
    RESPONSES=[  
        [ 1,"indian",[1,8]|1 ]  
        [ 2,"chinese",[1,8]|2 ]  
        [ 3,"greek",[1,8]|3 ]  
        [ 4,"italian",[1,8]|4 ]  
        [ 5,"english",[1,8]|5 ]  
    ];  
]
```

Notice that the whole data definition is enclosed in square brackets.

The VARNAME, VARITER and VARTYPE lines define the question name, its iteration and its response type. The TEXT line defines the question text.

The RESPONSES subsection defines the responses to the question. Each response starts with a response number. This is followed by the response text, the card and column in which the answer is coded, and the response code. The ‘greek’ response, for example, is coded with code 3 in column 8 on card 1.

-
- ❖ Qwincode, the open-end coding program, writes its code frames to the qdi file. If you reparse the script once the code frames have been written you will need to copy the code frames from the old qdi file into the new one.
 - ☞ Section 32.3, ‘Reparsing a CAPI or Web script once coding has started’, explains how to do this and provides a more detailed description of the qdi file to back this up.
-

The qdt file



This file contains a list of all question and response texts, and the names of any callable functions used in the script. Each text is written on a separate line.

The qoc file



This file is qparses version of your script. It is used by qtip for interviewing and by qcset, which prepares an extended data map for various Quancept utility programs.

The qtv file



All variables in the script have an internal number. Questions are created with built-in numbers, but temporary variable are not. The qtv file lists the names of the temporary variables in the script and gives them numbers according to their positions in the list. If the script contains:

```
color  define 'red' / 'blue' / 'green' / 'yellow' other
favc  ask 'What is your favorite color?'
      resp sp color other
      set x = nbit(favc)
      if (x<4) {
rbgy   ask 'Which of the following colors do you like?'
      resp mp 'red' / 'blue' / 'green' / 'yellow'
}
exit  end
```

the qtv file will contain one line as follows:

X 3

because x is the third variable in the script (color and favc are variables 1 and 2).

-
- ❖ The qtv file allows the substitution of temporary variables in response lists to work. If a response list has responses created using temporary variables, the values of temporary variables will appear in the response list only if the qtv file is present in the project directory.
-

The quo and quz files



If the script contains *quota* statements, the parser creates a single quz file containing the question names and response texts for questions named on those statements. When you are ready to define the quota targets you rename this file so that it has a quo extension and define the quota targets using quotedit.

If the script contains the following lines:

```
age      ask 'What age group do you belong in?'
        resp sp 'Under 18' / '18 to 34' / '35 to 54' / 'Over 54'
gender   ask 'And are you ... '
        resp sp 'Male' / 'Female'
        quota age/gender file='a1' pass=ok
```

the quz file will be as follows:

```
#set 'a1'
#quota
#dimension 'age'
#label 'Under 18'
#label '18 to 34'
#label '35 to 54'
#label 'Over 54'
#dimension gender
#label 'Male'
#label 'Female'
```

If the script contains more than one *quota*, these lines will be repeated for each *quota* statement.

If we then rename the file with a quo extension and run quotedit to define quota targets, the quo file will contain the following:

```
#set 'a1'
#quota
#dimension 'age'
#label 'Under 18'
#label '18 to 34'
#label '35 to 54'
#label 'Over 54'
#dimension 'gender'
#label 'Male'
#label 'Female'
#cell 0,0,0,0,0,0 100 0
#cell 1,0,0,0,0,0 100 0
```

```
#cell 2,0,0,0,0,0 100 0
#cell 3,0,0,0,0,0 100 0
#cell 0,1,0,0,0,0 100 0
#cell 1,1,0,0,0,0 100 0
#cell 2,1,0,0,0,0 100 0
#cell 3,1,0,0,0,0 100 0
```

The six comma-separated numbers in each #cell line refer to the dimensions for the current quota. and identify an individual quota cell. In this example, the third, fourth, fifth and sixth numbers in the group are always zero because the quota is only two dimensional.

The first #cell line starts with the numbers 0,0 and, in this example, defines the quota for men aged under 18. The second #cell line starts with the numbers 1,0 and defines the quota for 18 to 34 year old men. Notice that cell numbering starts from 0 for each dimension. The last #cell line starts with the numbers 3,1 and defines the quota for women aged over 54.

The last two numbers in each line are respectively the target and number of completes for each cell.

The first time that the interviewing program uses the quo file, it inserts the line:

```
#locked FALSE
```

at the beginning of the file. When the interviewing program has control of the quo file, the setting changes to TRUE and prevents you changing any of the completed counts using quotedit. As soon as the interviewing program releases the file the setting reverts to FALSE and you may change the completed counts as necessary.

The ref file



The ref file contains miscellaneous information about defined lists, questions with any type of open-ended response, and single-punched or multipunched questions used in *route* statements. It is used by the coding programs quac and vquac. The file contains one line for each such entry and fields within the lines are separated by spaces.

Entries for *define* statements consist of six fields:

Field	Description
1	Always 5
2	The label number
3	An offset value which you can ignore
4	The label name
5	Always DEFINE followed by a backquote
6	The name of the current file enclosed in single quotes

Entries for *resp coded* questions and single punched or multipunched questions whose response lists allow specified other have six fields:

Field Description

-
- 1 The question name preceded by an asterisk
 - 2 An offset value which you can ignore
 - 3 The question type (1=sp, 2=mp, 4=coded)
 - 4 The number of iterations
 - 5 The number of precodes
 - 6 The number of codes reserved for coding open ends

Entries for single-punched or multipunched questions used in routing statements have six fields:

Field Description

-
- 1 A reference value
 - 2 The line number of the statement in the script
 - 2 An offset value that you can ignore
 - 4 The label name
 - 5 The value to be tested for followed by a backquote
 - 6 The name of the current file enclosed in single quotes

Here is an example. If the script is:

```
master define 'one'/'two'/'three' other
q1      ask 'first question'
        resp sp master other
        for i = 1 to 2
            q2      ask 'second question'
                    resp mp 'red'/'blue'/'green'/'yellow' other
                    next
            q3      ask 'third question'
                    resp coded(15)
                    route (q1='one') go exit
            q4      ask 'last question'
                    resp coded
exit      end
```

the ref file will be:

```
5 1 41 master DEFINE 'tt'
2 10 56 q1 one 'tt'
*q1 56 1 1 3 29
*q2 102 2 2 4 29
*q3 158 4 1 0 15
*q4 205 4 1 0 29
```

Notice that the question entries come at the end of the file.

-
- ❖ If there are no defined lists, open ends or routing with single-punched or multipunched response lists, this file is empty.
-

The sif file



This is the Quancept Web and mrInterview interviewing program. For Quancept Web you will need to copy this file to the *projects* directory if you want to use it for interviewing. mrInterview does this automatically as part of the activation process.

The sum file



This file contains a summary of the types of variable found in the script, and the amount of space required in the data file to store an interview. If the script is:

```
color  define 'red' / 'blue' / 'green' / 'yellow' other
favc  ask 'What is your favorite color?'
      resp sp color other
      set x = nbit(favc)
      if (x<4) {
          goto ctn
      }
rbgy  ask 'Which of the following colors do you like?'
      resp mp 'red' / 'blue' / 'green' / 'yellow'
ctn   continue
name  ask 'What is your name?'
      resp coded
age   ask 'How old are you?'
      resp num 10 to 16 go exit / 17 to 99
car   ask 'What car do you drive?'
      resp dbase('cars')
exit  end
```

the summary file will contain:

```
17 Line read
 5 Questions
 8 Variables
 3 Temporary variables
 7 Labels
 2 Gotos
 0 Loops
Qoc size 344
Data recs 1
Data cols 15
```

The number of variables is the total number of different variables used in the script. It is broken down into question variables and temporary variables. The latter includes not only variables used with *set* and *for* but also the names of defined lists and *fix* statements and the names of databases used with *resp dbase*. This script has a defined list, a *set* statement and a database.

The number of labels refers to question and statement names only; that is, names starting in column 1. The names of defined lists count as variables so their names are not included in the count of labels even though their names start in column 1.

The number of gotos includes *goto* statements and *go* used in response lists or with *route*.

The Qoc size is the number of words used to store the information in the script as special Quancept code in the qoc file.

The Data recs line reports the number of records (cards) that each respondent will have in the data file, and the Data cols line tells you how many columns are required to store the data from the script. It does not include the serial number and card type columns added automatically by qtip.

The tx0 file



This file contains a numbered list of all the texts that appear in the script. For example, for the following script:

```
eatout ask 'Do you ever eat out?'
      resp sp 'Yes'/'No' ref
favor  ask 'What is your favorite type of food?'
      resp sp 'Greek' / 'Indian' / 'Chinese' / 'Italian' /
            'Turkish' / 'English'
browns ask 'Have you ever eaten at Brown''s restaurant?'
      resp sp 'Yes'/'No' ref
```

qparse creates the following tx0 file:

```

0 Do you ever eat out?
1 Yes
2 No
3 What is your favorite type of food?
4 Greek
5 Indian
6 Chinese
7 Italian
8 Turkish
9 English
10 Have you ever eaten at Brown's restaurant?
11 Yes
12 No

```

The txx file



The txx file is an important file that should be treated in the same way as a qoc file; that is, it must not be deleted or moved from the project directory. The information held in this file is:

- The index that qparse has assigned to the script's unique IDs, *rotranfix*, *rotransub*, *sp (^s)* and *promptoth (^o)* keywords, starting at one.
- A secondary index to the text that the ID or keyword applies to, as listed in the tx0 file.
- the unique ID or one of the following flags:

x	for <i>rotransub</i>	o	for <i>promptoth (^o)</i>
f	for <i>rotranfix</i>	s	for <i>sp (^s)</i>

↖ The secondary index in this file is linked to the primary index in the tx0 file, which is in turn linked to the record ID within the ptf file.

The xin file



This file contains information about GNU databases used in the script. For each database you have an introductory line whose information comes from the *eximport* statement that establishes a connection to the database. This is followed by a number of lines related to the lines in the database description file. These blocks will be repeated for each database used.

The following example shows a database description file, the eximport statement that connects to the database and the xin file that qparse creates. The database description file is called ratedes and is as follows:

```
lowrate 1 5 n
midrate 6 5 n
highrate 11 6 n
```

The *eximport* statement that connects to the database that this file describes is:

```
eximport(1,'ratedb','ratedes','gdbmfix',0)
```

The xin file is as follows:

```
**1 ratedb ratedes gdbmfix 0 1024
1 5 1 n
2 6 5 1 n
3 11 6 1 n
```

The first line is virtually the same as the *eximport* statement. Fields are as follows:

Field	Contains
1	Database identification number
2	Database name
3	Name of the database description file
4	Database type, either <i>gdbmfix</i> for a database with fixed length fields or <i>gdbm</i> for a database with variable length fields
5	Always zero
6	The maximum number of characters per record — always 1024

The remaining lines correspond to lines in the database description file:

Field	Contains
1	Line number in the database description file
2	In <i>gdbmfix</i> databases, the start column of the current field; in <i>gdbm</i> databases, the field's position in the record
3	In <i>gdbmfix</i> databases, the field width; in <i>gdbm</i> databases, always 1
4	Always 1
5	The field type — numeric , text or real

38.2 Files created by the CATI, CAPI and Web interviewing programs

Many of the files created by the interviewing programs are called *script.xxx*. The mrInterview interviewing program does not create any of these files, instead, it writes the data to an SQL database.

-
- ☞ See section 38.4, The mrInterview case data file, for information on how mrInterview data is stored.
-

Extension	Description	CATI	CAPI	Web
act	Accounting information	✓	✗	✗
com	Messages from interviewers to supervisors	✓	✗	✗
dat	Card-column data file	✓	✓	✓
drs	Data file	✓	✓	✓
drx	An index to the drs file	✓	✗	✗
idm	Cross-reference table of interview IDs and respondent serial numbers	✗	✗	✓
msg	Messages from interviewers to supervisors	✗	✓	✗
qca	Accounting information	✓	✗	✗
qsh	Memory address of shared qoc file	✓	✗	✗
ser	Serial numbers	✓	✓	✓
tbl	Screen layout parameters	✗	✓	✗
tex	Open-ended response texts	✓	✓	✓
tim	Interview timings	✗	✓	✓

The interviewing programs also create files with names that do not include the name of the script. These are:

Name	Description	CATI	CAPI	Web
qid.bmp	Scribbled responses	✗	✓	✗
respono.stp	Stopped data files	✗	✓	✓
qid.wav	Recorded open ends	✗	✓	✗
tiperrs	Error messages passed to qtip	✓	✗	✗

- ☎ In Quancept CATI, qtip may also create one or more of the following directories if they are needed:

Name	Stores
comments	Comments entered by interviewers
recording.dir	Recorded keystrokes for interviewers
script.ddr	Individual data files
script_name	Data for CATI interviews stopped by the script
script_stop	Data for CATI interviews stopped by the interviewer or the script
script_auto	Back-up data files generated by <i>autostop</i> statements in the script

The act file

- ☎ This file contains basic details about each call in a form suitable for analysis with a standard tabulation package. Each time you run qtip, a line is added to the file containing the following information:

Columns	Contain
1 – 4	Interviewer's number.
5 – 24	Blank.
25 – 30	Interview serial number (respondent ID).
31 – 34	Always zero.
35 – 50	Date and time of call in the format ddd mmm dd hh:mm. For example, Mon Oct 5 17:21.
51 – 66	Date and time of callback in the format ddd mmm dd hh:mm. If there is no callback this field is blank.
67 – 71	Always zero.
72 – 76	Always zero
77 – 81	Screen time of call in format mmmss. If there is no screener section in the script (marked by the presence of <i>screener</i> and <i>main</i> statements) this value is 00.
82 – 86	Mark time in format mmmss. This is not implemented so the value is always 00.
87 – 91	End time of the call in the format mmmss.
92 – 103	Interviewer's name.

Columns	Contain
104 – 105	The result of the interview: 25 <i>signal</i> statement with values 3, 4 or 7 encountered 34 stopped interview (the script stops because of an error) 36 stopped interview (you type ‘stop’, or a <i>stop</i> statement is read, or a <i>signal</i> statement with values 9 or 10 is reached). 40 completed interview (<i>end</i> or <i>signal 1</i> statement reached) 41 early completion (you type ‘complete’ or a <i>signal 6</i> statement is read) 51 terminated by <i>abandon</i> or <i>signal 8</i> 53 terminated by quota control or <i>signal 5</i> 54 terminated in screener (even if by <i>quit</i> or <i>abandon</i>) 56 terminated by <i>quit</i> or <i>signal 2</i>
106 – 107	Local status code from <i>privsig</i> statement in script, or 00.
108 – 117	Number of seconds since 1 January 1970.
118 – 125	Name of last question asked.
126 – 129	Number of questions asked. Questions asked twice due to snapbacks are counted twice.
130	Always 0.
131 – 135	Quota cell number. <hr/> <p>☞ This field is updated only if a <i>quota</i> statement is executed in the script. If the quotas are SMS quotas based on interaction between SMS and the quota files, this cell will show zero.</p> <hr/>
136 – 137	Number of snapbacks. This count is incremented each time qtip displays a question and asks whether the current answer is correct.

The auto directory



This directory contains temporary data files created by the *autostop* callfunc. One file is created for each interview and is overwritten each time the interview is saved. It is removed at the end of the interview or if data is written to the data files by a *stop* statement in the script, or by the interviewer typing *stop*, *quit* or *abandon*.

If the computer crashes while an interview is in progress you will be able to continue the interview after rebooting by moving its file into the stopped data directory and entering the respondent serial number when qtip asks whether you would like to restart a stopped interview. The format of the autostop files is the same as that of the drs file, except that each file contains only one record. You can find the interview serial number by looking at the first line in the file.

The bitmap directory and bmp files



Files named *qcid.bmp* contain a bitmap picture, drawn in response to a *resp coded scribble* question. Each file has a unique name created by inserting the open end's text serial number in the filename; for example, *qc123.bmp* for the open end whose serial number is 123. These files are created in the project directory in a subdirectory called **bitmap**.

The com file



This is the CATI communications file used for passing messages from interviewers to the supervisor. Messages are appended to it whenever you type **msg* to send a message to the supervisor, or use **note* to enter an open end instead of a single punched, multipunched or numeric answer.

The layout of the communications file is:

Columns	Contain
1 – 10	Interviewer's name
11 – 15	Interviewer's number
16 – 25	Respondent serial number
26 – 33	Question name
34+	The message type followed by a space and then the message text

For example:

```
ben      9      102Q9      MSG Typed *msg and then this text
barbara  4      170Q1      NOTE Typed *note and then this open end
```

The comments directory



When you select a call outcome from the SMS menu that prompts for a comment, or use the special response **comment* whether or not the project uses SMS, whatever you type is saved in a file in a subdirectory of the project directory called **comments**. Within this directory, files are grouped into subdirectories named *grp_n*, where *n* is a unique number allocated by Quancept. Each file within these subdirectories is named with the record key of the record to which it belongs. If several interviewers enter comments for the same record, they will be appended to the file so that it contains a complete list of all the comments entered for that record. The SMS supervisory program reads these comments when requested to do so, as does the SMS server when it displays them at the start of a call.

Each comment creates at least two lines in the file. The first reports the date and time at which the comment was entered, the name of the interviewer who made it and the result code which prompted for it. Subsequent lines report the text of the comment, as shown below:

```
***Tue Oct 16 20:40:44 1990:barbara Result(11)"Make an appointment"
Ask for Mrs Caroline Jones (NOT Carol Jones).
***Tue Oct 16 21:54:40 1990:barbara Result(11)"Make an appointment"
Caroline Jones not yet available, but will definitely be interviewed.
```

The dat file



-
- ☞ This file is not created for Quancept Web unless you have the line WriteDatFile=Yes in the [DATA] section of the project's initialization file.
-

This is the card-column format data file which stores single-punched, multipunched and numeric answers collected during the interview, and other data inserted by *fix* statements in the script. Each respondent's record is stored according to the layout given in the data map created by the qparse or QCompile.

Columns 1 to 5 contain the respondent serial number and columns 6 to 7 contain the card number. The remaining columns, up to a maximum of 80 for each card number, contain the data for that interview. Columns for open ends are left blank.

If the data contains multipunches and you list the file on your screen, you will see asterisks in the multipunched columns with extra characters representing the multipunches at the ends of those lines. To see the multipunches displayed in the correct format use the data editor, ded.

-
- ☞ For information on using ded, see the *SPSS MR Utilities Manual*.
-

If a card contains only a serial number and card type, Quancept CATI still writes this card to the data file, whereas Quancept CAPI and Quancept Web do not.

The ddr directory



Besides writing data to the dat, drs and tex files, qtip can also create a separate file for each respondent in a subdirectory of the project directory called *script.ddr*. Within this directory qtip creates subdirectories which may store data for up to 100 respondents. The subdirectory for respondents 1 to 99 is called sub0, the subdirectory for respondents 100 to 199 is called sub100, and so on. The files in these subdirectories are named with the serial numbers of the respondents to whom they belong, so the data file for respondent 256 is *script.ddr/sub200/256*.

If an interview is stopped and the data is written, qtip will overwrite the respondent's file with the complete interview when the interview is restarted and completed.

When you review an interview, qtip places a copy of the interview file in the directory *script.ddr/USED*. If you try to review this interview a second time, you will be asked to remove or rename the file in the USED subdirectory first.

-
- ❖ These data files are used only by the completed interview reviewing facility within qtip and by the data formatting program QCformat. If you would prefer not to have them created at all, set the environment variable QCNODDR to 1.
 - ☞ For further information on the contents of files in the ddr directory, see 'The drs file' below.
-

The drs file



This file is a text version of each respondent's data. The information in this file about single-punched, multipunched and numeric responses should be identical to that in the dat file. Unlike the dat file, the drs file also contains the texts of open ends or specified other responses. These texts and the information about the columns in which they will be coded should be the same as that in the tex file.

The easiest way of describing a record in the drs file is to look at one created by a specific script. Here is our script:

```
colors    define 'red' / 'blue' / 'green' / 'yellow' / 'black'  
first     ask 'Which color do you like best?'  
          resp sp colors null dk ref  
          quota first file='zz' pass=ok  
          route (.not. ok) go exit  
more      ask 'And which other colors do you like?'  
          resp mp not first in colors null dk ref  
childs    ask 'How many children do you have?'  
          resp num 0 to 9  
awake     ask 'What keeps you awake at night?'  
          resp coded  
exit      end
```

If the respondent chooses green for the first question, red and blue for the second, has no children and is kept awake by things that go bump in the dark, the respondent's data file will be:

```
##recstart 2  
##ac barbara 901108 151010 151037 27 658077010 658077037  
##init 2 2 0 0 1 1 3 2 0 9392  
##qzz 3 1  
##v2 FIRST 1 1 7 =3'green'  
##v3 MORE 1 2 7 =1'red';2'blue'  
##v4 CHILDS 1 3 7 =0  
##v5 AWAKE 1 4 7 ='SER           2/Things that go bump in the dark'  
##end 2
```

The first and last lines mark the start and end of the record and define the respondent's serial number.

The line starting ##ac provides basic accounting information. Sometimes the way this is stored varies between Quancept CATI and Quancept CAPI and Quancept Web.

Data	Description
barbara	In Quancept CATI, the name of the interviewer who conducted the interview. In Quancept CAPI and Quancept Web this is always qcw.
901108	The date of the interview. In Quancept CATI the date is in the format yyymmdd; in Quancept CAPI and Quancept Web the date is in the format mmddyy.
151010	The interview start time in the format hhmmss.
151037	The interview end time in the format hhmmss.
27	The length of the interview. In Quancept CATI the length is shown in the format hhmmss (values of less than three digits are seconds only). In Quancept CAPI and Quancept Web the duration is shown in seconds only.
658077010	The interview start time in seconds since 1 January 1970.
658077037	The interview end time in seconds since 1 January 1970.

The line starting ##init contains initialization information for the interview:

Data	Description
2	The respondent serial number (field 1 in the ser file).
2	The serial number assigned to the last open end in the script.
0	Always zero.
0	Always zero.
1	The number of calls to this respondent.
1	The number of quotas in the script.
3	The respondent's quota cell number.
<hr/> ↗ This field is updated only if a <i>quota</i> statement is executed in the script. If the quotas are SMS quotas based on interaction between SMS and the quota files, this cell will show zero.	
2	The number of interviews started (field 3 in the qtip ser file)
0	A pointer used for randomization.
9392	A value used in randomization.

The line starting ##q is the quota initialization line; in Quancept CATI and Quancept Web you will have one entry for each *quota* statement in the script (Quancept CAPI does not support quotas).

-
- ❖ This line is present only if a *quota* statement is executed in the script. If the quotas are SMS quotas based on interaction between SMS and the quota files, no ##q line is generated.
-

Data	Description
zz	The quota file suffix.
3	Which quota cell was decremented (-1 if <i>quota</i> statement not executed; 0 if cell was full).
1	Execution flag (1 if <i>quota</i> statement was executed, 0 if it was not).

This is followed by one ##v line for every question which the respondent answers. These have the general format:

var_num qname iteration resp_type how_used =answer

var_num is the variable's number or position in the script. Notice that temporary variables such as the name of the defined list are included in the count of variables. This is followed by the question name, converted to uppercase, and the iteration number. This is always 1 unless the question is part of a loop.

Next you have the response type. The ones you should expect to see are:

1	single punched	5	resp list	101	<i>null</i> (CATI only)
2	multipunched	8	<i>fix</i> statement	102	<i>dk</i> (CATI only)
3	numeric	9	database	109	<i>ref</i> (CATI only)
4	open end	10	real		

- 💻 In Quancept CAPI and Quancept Web, *null*, *dk* and *ref* take the response type of the question to which they belong. For instance, a *null*, *dk* or *ref* response to a single-punched question has a response type of 1 whereas a *null*, *dk* or *ref* response to a multipunched question has a response type of 2.

This is followed by a flag telling you exactly how the question was used in the interview. The variable itself is an octal variable in which each bit represents a different characteristic. The decimal values for these characteristics are:

- 1 The question was asked
- 2 The question was answered
- 4 The question was on the interview path

- 8 The question was encountered as qtip walked forward through the script after a snapback
- 16 The interview was stopped at some point after question

If a question has more than one characteristic, qtip will set all the bits that are necessary to represent that characteristic. You are therefore more likely to see values that are combinations of these basic values rather than these values themselves. Let's look at some common examples.

If the interview is not stopped and there are no snapbacks, all ordinary questions asked during the interview will have a value of 7. This means that they were asked, answered and on the interview path. This is the most common setting. Dummy questions have a value of 6 because they are answered (that is, they have a response assigned to them) and on the interview path. Bit 1 is not set because the question is not asked.

Now suppose that when the childs question is displayed, the interviewer stops the interview. When the interview is restarted and completed, questions first and more will have a value of 31 (asked, answered, on interviewing path, walked through, and stopped). The childs question will have a value of 15 (asked, answered, on interviewing path and walked through). The awake question will have the standard value of 7 (asked, answered and on interviewing path).

The final entry on each line is the response given. You will see the different formats in the example above. Single punched and multipunched responses appear as pairs of codes and quoted texts with each pair separated by semicolons. Numerics appear as numbers. Open ends are shown enclosed in single quotes, with the serial number separated from the response text by a slash. For database responses, the record selected is stored enclosed in single quotes. The opening single quote is followed by a [sign. Responses chosen from *resp list* statements are stored as texts enclosed in single quotes.

-  In Quancept CAPI and Quancept Web you may choose whether to place timing information for each question in the drs file or in the tim file. If timings are written to the drs file, they appear as a ##t line after the ##v line for the question to which they refer.

 See 'The tim file' below for information about ##t lines.

-  In Quancept CATI, the drs file is a concatenation of the data in the files in the ddr subdirectories. The only difference between the two is that where there is only one file per respondent in the ddr/subx directories, the drs file may contain many versions of a respondent's data. If an interview is stopped, restarted or reviewed, a new copy of the data for that respondent will be appended to the drs file.

 You can prevent qtip creating a drs file by setting the environment variable QCNODRX to 1.

Differences between CATI records and records for CAPI and Web

There are a number of differences between the way certain types of data are written for Quancept CATI and the way the same data is written for Quancept CAPI and Quancept Web. They are as follows:

Data type	CATI	CAPI and Web
<i>resp coded(0)</i> <i>resp ... nodata</i>	Serial numbers of -1 are assigned and appear in the drs file	Serial numbers are assigned as for standard open ends; these are shown in the drs file.
<i>resp sp ... other</i>	The response code for specified other is not included in the open-end text.	The notation < <i>resp_num:Other</i> > appears at the start of the open-end text.
Specified other	Appears in the drs record as OTHER.	Appears in the drs record as Other
Formatting characters with specified other	Formatting characters do not appear in the drs record.	If the script shows formatting characters around specified other, these appear in the drs record at the same point as in the script.
<i>^o</i> in response texts	The <i>^o</i> is omitted from the response text written to the drs record.	The response text is written to the drs record exactly as it appears in the script, including the <i>^o</i> .
Multiple responses flagged with <i>^o</i> and more than one <i>^o</i> response chosen	A single text serial number is assigned to the question. The <i>^o</i> texts are listed with this serial number at the end of the record.	Separate text serial numbers are assigned to each <i>^o</i> response. The <i>^o</i> texts appear at various places in the list of answers chosen, depending on the positions that <i>^o</i> appears in the response list.
<i>null, dk and ref</i>	The response type is 101 (<i>null</i>), 102 (<i>dk</i>) and 109 (<i>ref</i>).	No special response types: the response type matches that of the question.

SMS variables in the .ddr and .drs files

The values of all variables present in the sample record are added to the end of each interview's entry in the *project.drs* file, and to the end of each interview's file in the *subn* directories of the *project.ddr* directory. Each variable name is prepended by the letters sms so, for example, the record key is labeled smskey.

Here is an example of a typical interview:

```
##recstart 7
##ac barbara 940725 134719 134735 16 775140439 775140455
##init 7 0 0 0 1 0 0 10 0 63545
##v6 NAMEYN 1 1 7 =1'yes'
##v8 TELYN 1 1 7 =1'yes'
##v12 FX1 1 8 6 ='0'
##v13 WORK 1 1 7 =8'Retired'
##v22 AGE 1 1 7 =6'56 to 65'
##v23 NUCLEAR 1 1 7 =3'Expensive but the only alternative to oil/gas/coal'
##v24 TRANSPT 1 1 7 =3'Mostly on trains and some on motorways'
##v25 TAXES 1 1 7 =3'privatised'
##smskey=002
##smstipcode=0
##smsdataser=0
##smstimestried=0
##smsstoploc=0
##smshascomment=0
##smskeeprecs=0
##smsstatus=0
##smstelnumb=0207-234-5671
##smsregion=Inner London
##smsname=Claire Holland
##end 7
```

The sms variables contain the following information:

Variable	Contains
smskey	The record key of the sample record associated with this interview.
smstipcode	The call result code for the previous call to this record.
smsdataser	The data serial number of the record's data.
smstimestried	The number of times this record has been issued to qtip or the autodialer before the current call.
smsstoploc	The identification of the associated stopped record, if any (see 'The stopped interview directory' later in this chapter for details).
smshascomment	1 if the record has a comment associated with it or 0 if not.

Variable	Contains
smstelnumb	The telephone number passed to the autodialer. 0 if the project does not use an autodialer.
smskeeprecs	1 if the keeprecords variable has been set within SMS or 0 if not.
smsstatus	The QSAMP code returned to qtip by the autodialer interface.
smsregion, smsname	These are record variables that are defined in the SMS recvars file and which appear in the sample record. They will almost certainly vary between projects.

-
- ☞ The values shown for these variables are the values that the variables had at the time the sample record was passed to qtip. The smstipcode variable is therefore unlikely to contain the final call result code for the record.
 - ☞ See the *Quancept Sample Management and Telephony Systems User's Guide* for further information on these variables.
-

The drx file



The drx file is an index to the data in the drs file, and tells qtip where each new record begins. This is a binary file so you will not be able to list it on your screen.

-
- ☞ You can prevent qtip creating this file by setting the environment variable QCNODRX to 1.
-

The idm file



This file contains a cross-reference table linking interview IDs with respondent serial numbers. It is used for restarting stopped interviews in Quancept Web.

The msg file



This file holds comments or other messages that the interviewer enters during the course of an interviewing session. It is created the first time the interviewer clicks the Comment button during an interview, and is appended to thereafter.

The file contains three lines per message. The first line shows the date and time at which the message was entered, the second line is blank, and the third line shows the message text.

The qca file

-  This is a Quancept CATI accounting file. It contains one record per interview in which the fields are separated by colons. For example:

```
acct:barbara:5:878297205:878297241:'Fri_Oct_31_11:27:21_1997':36:1:3:0:1:0:0:0:  
5:DISLIKE:NONE:0:15:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:  
0:afternoon:1:::::::
```

The data in this file can be formatted into reports using options in the supervisory program, supermen. Fields in each record are as follows. The field names shown in column 2 are the names by which the fields are known in supermen.

Field	Field name	Contains	
1	studyname	Name of the project.	
2	uname	Interviewer's Quancept login name.	
3	unum	Interviewer's number from the /ip/users file.	
4	istime	Start time of interview in seconds since 1 January 1970.	
5	ietime	End time of interview in seconds since 1 January 1970.	
6	etime	End time of interview in the format ddd mmm dd hh:mm:ss yyyy.	
<hr/>			
` There are two colons appear in this field. If you are using other software to read information from the qca file, you must add two to obtain the correct field number for fields 7 and above unless the software takes into account the single quotes surrounding the etime field.			
<hr/>			
7	durint	If the survey is using SMS, the length of the interview in seconds from the time the record is passed from SMS to the time the question 'Another interview?' is displayed. If the survey is not using SMS, the value in this field will include the time taken to answer the questions 'Do you want to restart a stopped interview?' and 'Do you want to review a completed interview?'. Length of time in seconds spent in the telephone contact section before the first question is displayed.	
8	durtel	Length of time in seconds taken to answer 'Another interview?' at the end of the interview.	
9	durend	Result code returned to SMS, or 0 if SMS is not used.	

Field	Field name	Contains
11	signal	Signal value sent by a <i>signal</i> statement in the script or the exit status if signals are not used: 1 complete 8 abandoned 2 quit 9 stopped with no data written 5 quota stop 10 stopped with data written 6 early completion
12	isrestart	1 if this is a restarted interview, or 0 if it is not.
13	isstopped	1 if, in this attempt, the interview was stopped, or 0 if it was not.
14	prevtipcode	Result code of the previous call to this record, or 0 if SMS is not used.
15	serial	Interview serial number; that is, the respondent ID.
16	lastq	Name of the last question asked, if any.
17	smskey	The SMS key, or NONE if the project does not use SMS.
18 – 49	sect_times[0] to sect_times[31]	<p>Timing information written by <i>timesect</i> statements in the script. If there are no <i>timesect</i> statements these cells are empty.</p> <p>If the project uses SMS, sect_times[0] contains the length of time spent in the telephone contact section and sect_times[1] contains the length of time between qtip starting to execute the qoc file and returning the record to SMS.</p> <p>If the project does not use SMS, sect_times[0] is always zero except for the first interview in each qtip session. For the first interview, the cell contains the number of seconds that the question ‘Do you want to restart a stopped interview?’ was displayed. sect_times[1] contains a timing for the period between qtip starting to execute the qoc file and reaching the first <i>timesect</i> statement or, if there are no such statements, reaching the end of the script.</p> <p>sect_times[2] to sect_times[31] always contain whatever the scriptwriter places in them.</p>
50 – 59	acctinfo[0] to acctinfo[9]	Accounting information written by <i>acctinfo</i> statements in the script. If there are no <i>acctinfo</i> statements these cells are empty. The information written to these cells is entirely under the control of the scriptwriter.

The qsh file



This file is created when you run shareqoc to place a qoc file in shared memory and contains the memory address at which the qoc is located. When you start a qtip session and have the environment variable QCSHARE set to 1, qtip looks for this file and, if the file exists, reads the qoc file from the given area of memory.

The qsh file is a binary file, so it will appear empty if you try to list its contents on the screen.

The recordng.dir directory



If the project directory contains a file called recordng.qct (note the missing 'i'), qtip will record the keystrokes of all interviews that take place. Transcripts are created in files in the directory recordng.dir. The directory contains one file for each qtip process containing the interviews conducted by that process. The files are named *recpid*, where *pid* is the qtip process id. For example, if your qtip session has a process id of 1234, your recording file will be called rec1234.

The ser file



This is the serial number file used for assigning serial numbers to respondents and open ends. In Quancept CATI it contains three numbers, while in Quancept CAPI and Quancept Web it contains only two. Here is an example of a Quancept CATI ser file:

resp	open	intv
28	34	30

The first field is the number of respondents who have been contacted. It is incremented regardless of whether the interview was completed successfully or abandoned with no data saved, so each new respondent is guaranteed a unique serial number.

The second field contains a count of open ends. It is incremented each time you enter an open-ended response or enter a response after selecting specified other. In Quancept CATI is also incremented when you request a serial number for manual recording of open ends, force an open-ended response, or send a message to the supervisor by entering the response *msg.

The last field is the number of interviews started and appears for Quancept CATI only. This counts all interviews regardless of how they finished. Normally this is the same as the respondent count, but if an interview is stopped and restarted, this number is incremented once for each time it is started.

In our example, the serial number for the last respondent was 28, the serial number for the last open end was 34, and when the last interview was started it was the 30th interview altogether.

The stopped interview directory



Quancept CATI creates a stopped interview directory the first time an interview is stopped. If the interview is stopped by a *stop* statement in the script the directory is called *script_ _name*, where *name* is defined as part of the *stop* statement. If an interviewer stops the interview by typing in the special response *stop*, the directory is called *script_ _stop*.

In all cases, the stopped directory contains a file for each stopped interview. Files in the directory are usually named according to the respondent serial number of the interview whose data they contain — for example, *cars_ _stop/123* contains data for interview 123 of the cars survey — but the script may contain a *stopdata* statement which defines a different name for these files. These files have the same format as the records in the drs file.

When interviews are restarted, the stopped data files are transferred into a subdirectory of the stop directory called USED. So, when interview 123 is restarted the file *cars_ _stop/USED/123* is created.

The stopped directory and stp files



Quancept CAPI and Web store stopped interview data in *respno.stp* files in the ‘stopped’ subdirectory of the project directory. These files are named using the respondent’s serial number to ensure that the correct file is reloaded when an interview is restarted.

Records in the stopped data files are similar to records in the drs file, and may contain timing records if the project has been set up to place timing information in the drs file. However, a stopped data record contains an extra line that does not appear in a drs record. This line appears as the third line in the record as is as follows:

```
##cq iteration question_name
```

It shows the question name and the iteration at which the interview was stopped.

The tbl file



This file is created only if you use QCompile’s design mode to change the screen layout of the script. The explanations below are provided for information only, as you are not intended to alter a screen’s appearance by altering this file.

The overall structure of the tbl file is similar to that of a standard Windows initialization file, where the file is broken down into sections, and each section consists of lines that define the values of variables.

The tbl file has just one section called Layout. It contains lines of the form:

Object_name.Type.Property = Value

where *Object_name* is the name of the question or other script component whose screen appearance has been changed; *Type* is a keyword identifying an item on the screen; and *Property* is a keyword identifying the item’s size or position on the screen. For example:

```
[Layout]
q1.Text.X=100
q1.Text.Y=150
```

defines the horizontal and vertical position of the question text for q1.

Keywords that define item types are as follows:

Keyword	Description
Text	An <i>ask</i> or <i>display</i> text.
DK	The Don't know button.
NULL	The No Answer button.
REF	The Refused button.
MM nn	A multimedia item such as a picture, video or audio file, where nn is the position of the item within the <i>ask</i> or <i>display</i> statement.
Respn n	A response button for a precoded question, where nn is the position of the response in the response list.
Other nn	Specified other button in a precoded question, where nn is the position of <i>other</i> in the response list
Entry	The position of one of the following: editable text, a <i>scribble</i> box, the audio button, the slider for a numeric or real question, or a text question.
Numpad	The numeric keypad for a numeric or real question.
Edit	The editable portion of a database question.
KeyPad	The alphabetic keypad for a database question.
FindButton	The Find button for a database question.
Grid	The grid window.

Words that define properties are:

Keyword	Description
X	The horizontal position.
Y	The vertical position.
width	The width of the item.
height	The height of the item.

The tex file



-
- ☞ This file is not created for Quancept Web unless you have the line WriteTexFile=Yes in the [DATA] section of the project's initialization file.
-

This file contains the text and serial numbers assigned to open ends which you type in during or at the end of the interview. For example:

```
000001000001Q1      001080901It is too sweet for my liking  
000001000002Q2      001101201It was chewy and tasted good
```

The format of this file is:

Columns	Contain
1 – 6	Respondent serial number.
7 – 12	Response serial number.
13 – 20	Question name.
21 – 23	Subscript value.
24 – 25	Start column for coding.
26 – 27	End column for coding.
28 – 29	Card number.
30+	Whole response as a single text string. If the response has been recorded as an audio file or is a scribbled response held as a bitmap, this field contains the name of the file in which the response is held.

Open ends of any sort are numbered sequentially from 1 in the order in which they are passed to qtip. Therefore, if we have two interviews running concurrently and each one has two open ends, the response serial numbers 1 to 4 will be spread across the two interviews according to the order in which qtip received the response text.

Differences between Quancept CATI and Quancept CAPI and Quancept Web

Data type	CATI	CAPI and Web
Specified other	Specified other response texts in columns 30 onwards are preceded by the notation <code><resp_code:OTHER></code>	Specified other response look like any other open-ended response.

Data type	CATI	CAPI and Web
<i>resp coded...null dk ref</i>	Serial numbers are not assigned and no record is created if null, dk or ref is chosen	Serial numbers are assigned to all <i>resp coded</i> questions and a record is always written to the tex file. However, columns 30 onwards are blank if null, dk or ref is chosen.

The tim file



Timings for Quancept CAPI and Quancept Web interviews are normally written to the drs file, but you can choose to store them in a separate file if you wish. You do this by selecting the ‘Separate TIM file’ on QCompile’s Special tab when you create the interviewing program.

The format of the tim file is the same as that of the drs file, although it contains lines starting with ##t rather than ##v. These take the following format:

var_num qname iteration first_time total_time

where *varnum* is the variable’s number or position in the script. This is followed by the question name in upper case and the iteration number. This is always 1 unless the question is part of a loop.

The fourth field holds the number of seconds the interviewer spent on the screen in which the question appears the first time he or she reached it. The fifth field holds the total number of seconds spent on the screen in which the question appears. The values in these two fields differ only if the interviewer has snapped back and so looked at that screen more than once. Timing records are generated for each interviewing session, so if an interview is stopped and restarted, timings for any questions snapped back to are saved for the current interview only; time details previously saved for the question in another interview are not taken into account.

To illustrate, here is an example of a timing record:

```
##t3 work 1 12 19
```

In this example, a question called ‘work’ was on the interviewing screen for 12 seconds the first time the interviewer reached it and for a further seven seconds later in the interview. This may have been on one or more further occasions. The total time spent on the question was 19 seconds.

-
- ❖ For *multitask* questions that appear on the same screen, the same time totals appear in the timing record. This is because it is the time a question appears on screen that is recorded, rather than the time it took for the interviewer to enter a response.
-

The tiperrs file



This file contains messages passed to qtip while it is running. These generally consist of messages reporting the quota cell number in which the respondent belonged and the status of the return flag after the quota cell was tested.

Occasionally you may find other errors placed in the file by qtip itself. These refer to inconsistencies in the script which qparse was not able to trap; for example, variables on quota statements which are not single punched, multipunched or numeric, or failure to open a database as the response list to a question.

If the script does not work as you expected, even though it parsed without errors, this is the file to check for help in solving the problem.

The wav files



Files named qc*id*.wav contain open ends that were recorded in response to a *resp coded audio* question. The filenames are made unique by inserting the open end's text serial number in the filename; for example, qc234.wav is the audio recording for the open end whose serial number is 234. In Quancept CAPI, these files are created in the project directory in a subdirectory called audio.

Quancept CATI with the QTS creates the following files:

/hostname/project/qcid.wav for open-ended and other specify responses

/hostname/project/respserno_varname_iter.wav for other questions, where *respserno* is the respondent serial number, *varname* is the question name and *iter* is the iteration number

In both cases, *hostname* is the name of the telephony computer.

38.3 SMS files



☞ For full details of the accounts.sms and serverlog.sms files, see the *Quancept Sample Management and Telephony Systems User's Guide*.

accounts.sms

This file stores information about records returned to SMS from qtip. It contains one entry for each record returned from qtip, formatted in such a way as to be easily used by other reporting programs. Each record consists of a single long line in the file.

serverlog.sms

This file keeps a record of information passed from qtip to the server and from SMS back to qtip. Each entry in the file contains one message, consisting of date and time, message identification codes and text.

38.4 The mrInterview case data file

- mrInterview writes its data to a relational SQL database whose name matches the name of the script. The database stores response data in three tables called Variables, Responses and OtherData, each of which is described below.

The illustrations and descriptions of these tables are based on the following script:

```

q1 ask 'Which types of tea do you drink?'
resp mp 'Assam' ('assam') / 'China' ('china') / 'Darjeeling' ('darj') /
       'Lapsang Souchong' ('lapsang') / 'Orange Pekoe' ('opekoe') /
       'Puerh' ('puerh') / 'Rooibos' ('rooibos') /
       'Fruit/Herbal' ('fruit') other

q2 ask 'And do you normally use tea bags or loose tea?'
resp sp 'Tea bags' ('bags') / 'Loose tea' ('loose') /
       'Use both equally' ('both')

q3 ask 'How many cups of tea do you normally drink in a day?'
resp num 1 to 30 dk

s1 set herbal = bit(q1/8)
i1 if (herbal) {
q4   ask 'What do you think are the benefits of drinking fruit or
herbal teas?'
     resp coded
}
exit end

```

The Variables table

VariableID	VariableName	Type	Project	ParentID
1	Respondent.Serial	1	<NULL>	<NULL>
2	Respondent.Serial.SourceFile	2	data	<NULL>
3	Respondent.Origin.Other	2	data	<NULL>
4	Respondent.Origin	3	data	<NULL>
5	DataCollection.Status	3	data	<NULL>
6	DataCollection.InterviewerID	2	data	<NULL>
7	DataCollection.StartTime	5	data	<NULL>
8	DataCollection.FinishTime	5	data	<NULL>
9	DataCollection.MetadataVersionNumber	1	data	<NULL>
10	DataCollection.MetadataVersionGUID	2	data	<NULL>
11	DataCollection.RoutingContext	3	data	<NULL>
12	DataCollection.EndQuestion	2	data	<NULL>
13	DataCollection.TerminateSignal	1	data	<NULL>
14	DataCollection.SeedValue	1	data	<NULL>
15	DataCollection.InterviewEngine	2	data	<NULL>
16	DataCollection.CurrentPage	1	data	<NULL>
17	DataCollection.Debug	2	data	<NULL>
18	q1.Other.Codes	3	data	<NULL>
19	q1.Other	2	data	<NULL>
20	q1	3	data	<NULL>
21	q2	3	data	<NULL>
22	q3.Codes	3	data	<NULL>
23	q3	1	data	<NULL>
24	q4.Codes	3	data	<NULL>
*	q4	2	data	<NULL>

The Variables table holds the names and data types of all variables in the database. The table does not hold data. In this example, the variables that are directly associated with questions in the script are those on lines 17 to 24. The variables on lines 1 to 16 are system variables that refer to the interview as a whole.

☞ See the SPSS MR Data Model documentation for more information about system variables.

Responses and other values that are written into the data are held in variables that are named according to the question or variable names in the script. Different types of response require different types of variables:

Response type	Variables created
Single and multiple responses	A categoric variable whose name matches the question name in the script. The variable stores an integer that represents the response text. Each categoric response in the questionnaire is assigned a unique code.
Numeric response	A long (<i>resp num</i>) or double (<i>resp real</i>) variable whose name matches the question name in the script.

Response type	Variables created
Open-ended responses	A text variable whose name matches the question name in the script, and a categoric variable named <i>qname</i> .Codes that can be used by the Verbastat coding program for storing the codes assigned to the response.
Specified other	A text variable named <i>qname</i> .Other to hold the text entered by the respondent, and a categoric variable named <i>qname</i> .Codes that can be used by the Verbastat coding program for storing the codes assigned to that response.
Null, dk and ref	A categoric variable named <i>qname</i> .Codes if the response is given to a numeric or open-ended question. For questions with categoric response lists, these responses are stored in the standard question variable in the same way as the categoric responses.

As the illustration shows, question q1 in the sample script requires three variables: q1 to store the categoric responses chosen, q1.Other to store the text entered for *other*, and q1.Other.Codes to store the codes assigned when the other responses are coded.

Question q2 requires a single variable called q2 to store the categoric response.

Question q3 requires two variables: q3 to store the number and q3.Codes to store any *dk* responses.

Question q4 also requires two variables: q4 to hold the text that the respondent enters and q4.Codes to the codes assigned to this response during coding.

When a question appears in a loop, the question name component of the variable name is modified so that it includes the label of the loop's *for* statement and the unique ID of the item in the loop's control list to which this question relates. For example:

```
tchar  for ttype = 'Lapsang Souchong' ('lap') / 'Orange Pekoe' ('opek')
q6      ask 'What would you say are the main characteristics of '+ttype+ tea?'
        resp coded
next
```

generates variables called tchar[{lap}]q6.Codes and tchar[{opek}]q6.Codes. If you do not define unique IDs, mrInterview uses the IDs that it generates automatically from the first 50 characters of the response text. It therefore makes sense to define unique IDs that are short so that the variable names do not become too long. It is also important to assign labels to each *for* statement so that the variable names clearly identify the question to which they refer.

The Responses table

Serial	VariableID	OrderChosen	Response	OtherID
4	0	0	0	<NULL>
4	3	0	2	<NULL>
4	8	0	1	<NULL>
4	9	0	0	23
4	10	0	14	<NULL>
4	6	0	0	24
4	13	0	636114	<NULL>
4	15	0	4	<NULL>
4	19	0	17	<NULL>
4	19	1	18	<NULL>
4	19	2	22	<NULL>
4	19	3	3	<NULL>
4	18	0	0	25
4	20	0	23	<NULL>
4	22	0	3	<NULL>
4	24	0	0	26
4	5	0	0	27
4	12	0	0	<NULL>
4	11	0	0	28
4	16	0	0	29
4	7	0	0	30
4	4	0	12	<NULL>
4	4	1	4	<NULL>

The Responses table stores interview data and, for open ends, specified other and real numbers, a reference to the OtherData table where those responses are stored. Each response is a separate line of the database, even if it is part of a multiple response categoric list (a multipunched response list). The Serial column shows the serial number for this interview. The VariableID column contains the variable's reference number that links it back to the Variables table. The OrderChosen column shows the order in which multiple categoric responses were chosen. The Response column contains the code that identifies the response chosen. The OtherID column contains a reference code that identifies an entry in the OtherData table for open ends.

The data in this table refers to an interview in which the following response were given. At q1 (variable number 19) the respondent chose Darjeeling (17), Lapsang Souchong (18), Fruit/Herbal (22) and Other (3) in that order. Variable 18 contains the reference code to the text of the Other response that is held in the OtherData table.

At q2 (variable 20) the respondent chose response number 23, Tea bags, and at q3 (variable 22) he/she said three cups a day. Variable 24 contains the reference code to the respondent's answer to q4 which is stored in the OtherData table.

Variables 17, 21 and 23 are missing from the Responses table. Variable 17 will be added when the specified other responses to q1 are coded; variable 21 is not required because the respondent did not select *dk* at q3; and variable 23 will be added when the open ends for q4 are coded.

As this example illustrates, when mrInterview writes response data to the Responses column, it uses a unique number for every categoric response in the questionnaire. The numbering sequence does not necessarily start with 1 for the first categoric response in the questionnaire, nor is it always totally sequential. This scheme makes it very easy for the data to be extracted and manipulated, but has the disadvantage that it is not easy to match data in the data file with response texts in the questionnaire simply by looking at the tables in the database.

SPSS MR provides a selection of tools for viewing data in a more easy-to-read form and for extracting it for analysis or other purposes. One such tool is an Excel macro that has been designed solely for displaying test data in a form that more closely matches the texts and values used in the questionnaire.

☞ See section 37.2, Checking the data, for details.

The OtherData table

	OtherID	TextVal	RealVal	DateVal	ObjectVal
	25	English breakfast	<NULL>	<NULL>	<Binary>
	26	Less tannin;Low caffeine	<NULL>	<NULL>	<Binary>
	27	gmya4	<NULL>	<NULL>	<Binary>
	28	q4	<NULL>	<NULL>	<Binary>
	29	cur=q4	<NULL>	<NULL>	<Binary>
	30	<NULL>	<NULL>	10/04/01 10:53:10	<Binary>
*					

This table stores open-ended texts awaiting coding, real values for *resp real* questions, and the values of some system variables, including the date and time that the interview began. The OtherID column holds a unique reference for each text which refers back to the OtherID column in the Responses table. If you look at the illustration of the Responses table you will see that OtherID 25 relates to variable 18. If you then locate variable 18 in the Variables table you will find that it is associated with variable q1 and therefore question q1 in the script. In this example, the respondent chose *other* and entered ‘English breakfast’ at q1. Similarly, OtherID 26 relates to variable 24, which is associated with variable q4. This is question 4 which has an open-ended response, and the respondent’s words were ‘Less tannin;Low caffeine’.

38.5 Other mrInterview files

The mqd file

- ☞ The mqd, or quota definition, file is created in the project’s source directory when you save quota definitions and targets that you have set up using Quota Setup. Like the mdd file, the mqd file is written in the extensible mark-up language (XML) and can be viewed using a text editor.

The file contains information about the quota tables and expressions you have defined and the targets for each quota. When you activate a project for the first time, the activation process uses the mqd file to determine the structure and content of the quota database files that it needs to create within the project's database. Whenever you re-activate the project, you can choose whether or not to reread the mqd file. If you have not changed the quota definitions or targets using Quota Setup, there is no need for the activation process to reread the mqd file.

-
- 『 If you change targets using ChangeQuota, these changes are made in the quota database but not in the mqd file. If you re-activate using the mqd file after changing targets or quota flags using ChangeQuota you will overwrite the new information with whatever settings are in the mqd file.
-

The activation process copies the mqd file to the FMRoot directory, assigns it a version number, and then copies it to the project's working directory on the various interviewing servers. However, the file is not used by the interviewing program.

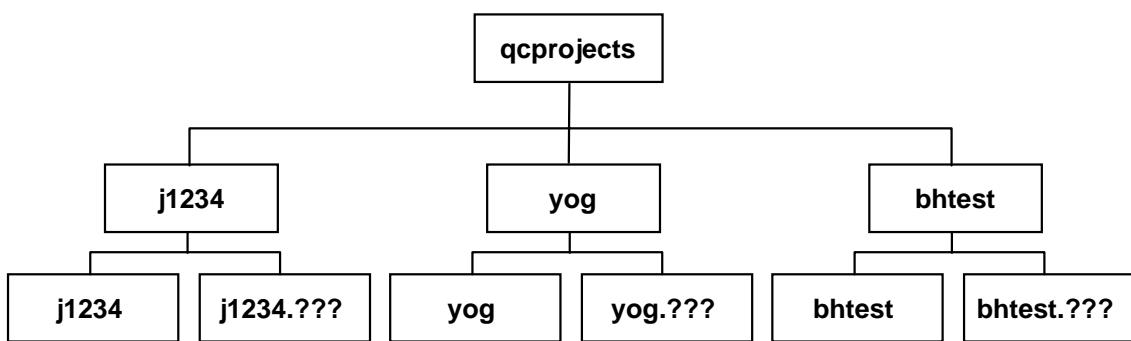
39 Managing CATI projects

This chapter discusses general topics to do with managing CATI projects, including the structure of the project directory and the use of environment variables to control the way a project operates.

39.1 Storing your script on the computer

Each time you write a new script, start by creating a new directory in which the script and the files it generates may be stored. You should name this directory according to the project name or job ID — for example j1234 for the project whose ID is j1234, or yog for a survey about yogurt. We call this directory the *project directory*.

The text of a script is stored in a file inside the project directory. You should name the script file with the same name as the project directory. Thus, the yogurt script would be called yog, and the script for j1234 would be called j1234. Script file names may be up to eight characters in length: the first character must be a lowercase letter, the others may be lowercase letters or numbers. Spaces are not allowed in directory names or filenames; so the name ‘job 123’ is illegal. Here is a diagram of a typical Quancept CATI directory:



The file names containing question marks represent the files which are created by various programs in the Quancept suite. Each program creates files with different extensions. The parser, for example, creates the files j1234.map and j1234.qoc, amongst others, for the j1234 project. These files store the data map and the script in a computer-readable form.

If you have very large or complex scripts, you may prefer to create several script files rather than one very long one. In this case, the main script (that is, the one that contains the start of the script) should be named with the project name. The other scripts should be named with some logical name according to their function or their position in the script as a whole and will be read by the main script at the appropriate point in the interview. For instance, if you have a section of the script which deals with demographics, you might place this in a separate file and name it demog.

39.2 Environment variables

The interviewing environment determines the manner in which qtip operates for each user. It covers such things as whether you will be prompted to enter your interviewer name at the start of each session, whether it will be necessary to press Return after answering y or n to one of qtip's internal questions, and whether you may press Return to indicate a null response. These and other requirements are defined with environment variables.

The environment variables that qtip recognizes are given in two subsections in this section, divided by the type of environment in which qtip can run:

- ‘General environment variables’ describes those settings most likely to be used by all interviewers and scriptwriters regardless of their working environment.
- ‘SMS and QTS environment variables’ describes settings relevant to sites using Quancept with SMS or the QTS.

Setting environment variables

Quick Reference

To define or set an environment variable, type:

setenv *env_var value*

where *env_var* is the name of the environment variable and *value* is often 1 to switch the feature on, but other values are possible. Environment variable names are always entered in uppercase regardless of the operating system you use.

To unset an environment variable, type:

unsetenv *env_var*

Most environment variables require a setting of 1 to work; for example, setting QCASKNUMBER to 1 forces qtip to ask you which record you wish to call, where normally it would issue you with the next eligible record. When we say, therefore, that an environment variable is ‘set’, we usually mean it is ‘set to 1’ or switched to be ‘on’. Some variables have other numeric settings or text values.

You can add the definition to an interviewer’s .login or .profile file.

☞ See the next section, ‘The environment file’, for more information about creating an environment variable definitions file with QCENV. Creating this file allows you to set environment variables for projects or groups of interviewers.

See the *Quancept Sample Management and Telephony Systems User’s Guide* for information about setting the values of QTS environment variables.

The environment file

Environment variables are typically set in your .login or .profile file but you may also define them in a file specified by the environment variable QCENV. This file may exist in your home directory or, if a group of interviewers shares the same settings, the file may exist in some central location. It may also exist in a project directory, where the settings apply only to that project.

Entries in the file consist of an environment variable name and a value separated by at least one space (or one tab). For example:

```
QUANCEPT    tvi925
QCWHOAMI    barbara
QCNULL      1
QCUSCORE    1
QCUPDATE    2
```

This says that you are using a tvi925 terminal and that your interviewer name is barbara. During interviews you may press Return instead of entering the keyword *null* and you want a line to be drawn between question texts and response lists. If the qoc file is updated while you are interviewing, the next interview that you start will be conducted using the modified qoc file.

-
- ↖ You cannot define the QCENV or TZ environment variables in an environment file; they must be defined separately in each interviewer's .login or .profile file.
-

The full pathname of the file to read is defined in your .login file with the environment variable QCENV, for example:

```
setenv QCENV /usr/users/barbara/ENVIRON
```

Because environment variables are used to change the way qtip operates, you may wish to set certain environment variables on specific projects only. If you want to set project specific environment variables:

1. Decide on a common name for the environment variable file, such as ENVFILE.
2. Set the environment variable QCENV to be, for example, ./ENVFILE (you can substitute whatever name you choose for ENVFILE). The . / notation means that the file is located in the current (project) directory.
3. Create the environment variable file in the project directory.

-
- ↖ If an environment variable is set in the .login (or .profile) file and also in an environment file, the value in the environment file overrides the local settings.
-

General environment variables

The environment variables described in this section control settings for interviewers and scriptwriters who work under all operating systems.

Variable	Value to set	Description
F_VERB	1 (on)	At the end of an interview qtip normally asks whether you would like to append verbatims. When this variable is set to 1, you do not have this choice. Instead, open ends are displayed one at a time and you may append to them or press Return twice to move to the next one. If you are using subsurveys, then you will have to review any verbatims contained in the secondary script when the subsurvey ends, not at the end of the main script.
LANG	text	Defines the locale for the language in which the interviewer is working. Used only if QCLANG is not defined.
LANGUAGE	text	Defines the default language in which qtip's messages are displayed. Translated texts for Quancept programs are read from \$QCHOME/\$LANGUAGE.
LOGNAME	text	Defines the interviewer's user name to be displayed in qmonitor if the USER environment variable is not set.
QCAMPNAMES	1 (on)	Causes variable references of the form & <i>varname</i> to be treated the same as [+ <i>varname</i> +] references.
QCAUTODIAL	1 (on)	If this variable is set to 1, qtip assumes that you have a modem attached to your terminal and that all numbers will be dialed by the modem. qtip then expects to find the environment variables QCAUTOSTART and QCAUTOEND defining the command sequences which tell the modem to dial the number at the start of the call and to drop the line at the end of the call.
<hr/>		
☞ See section 39.3, Dialing with modems, for further details.		
QCAUTOEND	text	Defines a string that is sent to the terminal at the end of each interview, just before the 'Another interview?' prompt. Only used if QCAUTODIAL=1.
QCAUTOSTART	text	Defines the dialing string for modem dialing via from a terminal. It should contain everything that the modem requires for dialing the number, including the number itself. qtip passes this string to the terminal at the start of a call, so you will need to include a command sequence which tells the terminal to pass the command on to the modem. Only used if QCAUTODIAL=1.

Variable	Value to set	Description
QCBELL	0 (off) 1 (audible) 2 (visible)	When an interviewer makes a mistake, for example, by entering a non-existent response code, qtip issues an audible warning. Set this variable to 0 to turn the audible warning off, or to 2 to use a visible warning (usually the screen flashes) instead.
QCCOMMENTS	<i>number</i>	Defines the number of comments to be displayed when the interviewer uses *comment. Comments are normally displayed in the order they are entered, so the most recent comment is shown last. To reverse the order so that the most recent comment is shown first, set QCCOMMENTS to a negative number.
QCCOMP	1 (end) 2 (start)	Before the start of each interview, you will be asked whether you wish to review a completed interview. This is only possible if you allow qtip to create individual data files in the ddr directory (that is, you do not have QCNODDR set to 1). The value of QC COMP determines whether the interview to be reviewed opens at the first (2) or last (1) question in the interview.
		<p>☞ See section 34.3, Reviewing completed interviews, for notes on this subject.</p>
QCDUMP	1 (on)	If there is a logic problem in the script which qparse was unable to detect, it is likely that qtip will fail when it reaches this point in the script. Any data gathered so far will be written to a file in the <i>project_</i> _stop directory. If this happens and you are unable to locate the fault, your support representative may ask you to set this variable to 1 so that programming tools can be used to find the problem. This variable is not useful to interviewers.
QCEDITALL	1 (on)	Allows the interviewer to use *e to edit open-ended responses after answering 'y' when asked whether they want to append or change verbatims.
QCEDITOR	<i>text</i>	Defines the editor to use when appending or entering verbatims at the end of an interview; if this variable is not set, 'vi' is assumed.
QCENV	<i>text</i>	Environment variables for qtip may be defined in each user's .login or .profile file, or in a separate environment file. If you define your qtip environment variables in a separate file, you must define its location with the environment variable QCENV. This variable must be defined in your .login or .profile file.

Variable	Value to set	Description
QCERRSLEEP	<i>number</i>	Specifies the number of seconds that error messages will be displayed at the foot of the screen. The default is two seconds. qtip uses this variable when it displays multiple messages and when it needs to clear the screen.
QCEUROPEAN	1 (on)	Dates entered in the form aa/bb are assumed to be in European format; namely dd/mm. The default is the American notation mm/dd.
QCFLUSH	1 (on)	If you type more quickly than qtip can refresh the screen, qtip remembers everything that you type and applies it as the answers to questions and other prompts as they appear. If you set QCFLUSH to 1, this 'type-ahead' facility is disabled. qtip will discard any characters typed before a question appears on the screen.
QCGROW	0, 1, 2 or 3	Defines the maximum number of temporary variables and the amount of heap space to use. Possible values are: <ul style="list-style-type: none"> 0 the number of temporary variables is 997 and the amount of heap space is controlled by the QCHEAPMAX environment variable. 1 the number of temporary variables is unlimited and the amount of heap space is controlled by the QCHEAPMAX environment variable. 2 the number of temporary variables is 997 and the amount of heap space is unlimited. 3 the number of temporary variables and the amount of heap space are both unlimited.
QCHEAPMAX	<i>number</i> >50,000	Increases the 'heap size' or number of characters available for storing variables and their values; the default is 50000.
<hr/>		
QCHOME	<i>text</i>	Defines the pathname of the Quancept home directory; that is, the directory containing the programs and other information needed in order for the current release of Quancept to work.

☞ You can use the *heaptop* keyword with *set* to find out how much heapspace is used. See section 12.7, Special keywords with set.

Variable	Value to set	Description
QCID	1 or 2	When qtip displays the answers to the current or previous questions, it prints the response texts but not the response codes for single-punched and multipunched questions. Set QCID to 1 to see the response code and the response text, or to 2 to see the response code only.
QCKBDELAY	number	Specifies the maximum number of seconds to wait for the rest of an incomplete escape sequence during an interview. The default is 1 second.
QCLANG	text	Defines the locale for the language in which the interviewer is working. QCLANG controls the following aspects of multilingual interviewing: <ul style="list-style-type: none"> • Keyboard input — that is, which characters acts as punctuation. • Screen output — for example, where to break long lines. • Case conversion when comparing texts — for example, in list-in-list selections. • Collation sequence for alphabetic sorting. • Pattern matching when % is used with the <i>strngchk</i> callfunc to match any letter. If QCLANG is not defined, qtip uses the standard Unix LANG variable. You can override both LANG and QCLANG with the <i>setlocale</i> callfunc.
<hr/> <p>☞ See ‘Setting the language for alphabetic sorting, punctuation and line breaks’ in chapter 27, Callfuncs, for information about <i>setlocale</i>.</p> <hr/>		
QCLOCALE	number	Defines your international dialing code (for example, 44 for the UK, 49 for Germany) and controls the output formats for dates and the default setting for QCEUROPEAN. If this variable has a value between 30 and 49 or 300 and 449 inclusive, the variable QCEUROPEAN is set ‘on’ automatically.
QCLOOPLONG	number	Determines the minimum number of characters that are guaranteed to be preserved on text which drives <i>for</i> loops. Texts longer than the specified limit may be truncated. The default is 2048 characters.

Variable	Value to set	Description
QCNOAPPEND	0 (always) 1 (sub) 2 (main) 3 (sub and main)	Suppresses the prompt ‘Do you want to append verbatims?’ at the end of secondary scripts (subsurveys) and/or main surveys. If you have this variable turned on, you must either enter the open ends in full as they are given, or write them down since you will not be able to access them once you leave the subsurvey. The prompt to append verbatims at the end of the main script refers to open ends in the main script only. The value 0 always prompts, the value 1 suppresses the prompt in the subsurvey only, the value 2 suppresses the prompt in the main survey only, and the value 3 suppresses the prompt in both the main and the subsurvey.
QCNOCHANGE	0 (always) 1 (sub) 2 (main) 3 (sub and main)	Suppresses the prompt ‘Do you want to inspect or change anything?’ at the end of secondary scripts (subsurveys) and/or main surveys. If you have this variable turned on, you will not be able to go back and change or inspect answers in the survey once you have entered the answer to the last question in that script. The prompt to inspect the data at the end of the main script refers to questions in the main script only. The value 0 always prompts, the value 1 suppresses the prompt in the subsurvey only, the value 2 suppresses the prompt in the main survey only, and the value 3 suppresses the prompt in both the main and the subsurvey.
QCNOCORR	1 (on)	Allows you to correct selections from multipunched response lists using <code>!code</code> . This variable is switched on by default causing qtip to display the response codes chosen so that they can be corrected. If you switch this variable off, qtip does not display the list of codes chosen and you will have to retype all the response codes for the question.
QCNODDR	1 (on)	Do not create individual data files in the ddr directory. If QCNODDR is ‘on’, you will not be able to review completed interviews.
QCNODRX	1 (on)	Do not create a drs file in the project directory.
QCNONULL	1 (on)	Code null responses as blank (not asked) rather than – (no answer).
QCNOPTF	1 (on)	When this variable is set to 1, translations are read from the tx0 file rather than the ptf file.
QCNORESTART	1 (on)	Do not offer the option of restarting stopped interviews.

☞ See also QCCOMP.

Variable	Value to set	Description
QCNOSNAPBACKS	0 (off)	<p>When set, allows only the last question to be changed if the interviewer answers ‘y’ to the question ‘Do you wish to change or inspect anything?’. The default is to allow interviewers to snap back to any question.</p> <hr/> <ul style="list-style-type: none"> ❖ If the last question answered allows specified other and the respondent has chosen this response with or without other responses, qtip automatically presents the open-ended text for editing, and does not allow any other answers to the question to be changed.
QCNOSTOPCHANGE	1 (on)	Suppresses the prompt to inspect or change responses when the interview is stopped by a <i>stop</i> statement in the script or by the interviewer typing <i>stop</i> .
QCNULL	1 (on)	Allows you to press Return instead of typing <i>null</i> . We recommend you set this variable if using QTS Record and Play.
QCOLDLOOPS	1 (on)	<p>Requests pre-version 7.8 loop and list behavior as follows:</p> <ul style="list-style-type: none"> • <i>rotranfix</i> and <i>rotransub</i> in defined lists are not recognized when the defined list is used to drive a loop. • <i>atoz</i> does not work on <i>for</i> statements, nor does a negative <i>step</i> size for numeric ranges. • A deselected response causes a <i>rotransub</i> group to be split when evaluating <i>list in master</i>. Groups always remain together in version 7.8. • When evaluating <i>not list1 in list2</i>, <i>rotranfix</i> and <i>rotransub</i> are taken from <i>list1</i> rather than from <i>list2</i>. • <i>other</i> is included in the loop control list if it is present in either of the lists named on the <i>for</i> statement (that is, in either <i>list1</i> or <i>list2</i>). In version 7.8, <i>other</i> is included in the control list only if it is present in the list that drives the loop. • <i>other</i> is included in the loop if it is selected in <i>list1</i> but qtip displays its value as it is set in <i>list2</i>. In version 7.8, <i>other</i> is included only if it present in <i>list2</i>.
QCPROJECTS	<i>text</i>	Defines the pathname for a projects file to be used instead of the default /ip/projects.

Variable	Value to set	Description
QCQUOTALOG	1 (long) 2 (concise)	Generates a log file of actions taken when a <i>quota</i> statement is encountered in the script. If QCQUOTALOG is set to 1, the information for each action is laid out over two lines. If the variable is set to 2, each action is recorded as a single line in a form that can be easily read by other programs. The log file is called <i>script.qdbxx</i> , where <i>xx</i> is the quota suffix.
		<p>☞ See ‘Creating a quota log file for debugging purposes’ in chapter 30, Quota control, for further information.</p>
QCRETAIN	1 (on)	When a response list is re-ordered using <i>atoz</i> , <i>rot</i> or <i>ran</i> , <i>qtip</i> normally sorts the list and then numbers responses sequentially from 1 in the order they appear on the screen. This means that the code that the interviewer types to select a particular response will vary between interviews. If <i>qretain</i> is set to 1, responses will always be numbered according to their position in the original list, regardless of the order in which they are sorted. This variable is ignored by <i>qtip -t</i> .
QCSEGUE	1 (on)	Suppresses the ‘End of subsurvey’ message when the interviewer leaves a subsurvey. The interviewer is returned to the main script immediately rather than having to press Return at the message.
QCSET	0 (pre-v7.8 behavior) 1 (codes) 2 (texts)	Determines how <i>sp</i> , <i>mp</i> and <i>list</i> assignments will work. When QCSET is set to 1, assignment is based on the position of responses in the response list. This is the default in Quancept CATI version 7.8. When QCSET is set to 2, assignment is based on response texts. When QCSET is 0, assignment is based on response texts if identical response texts exist in the two variables named on the assignment statement, or on response position if not.
		<p>☞ See ‘Question labels’ in chapter 12, Assignment, for further information.</p>
QCSHARE	1 (on)	Tells <i>qtip</i> to look for a shared <i>qoc</i> file for the current script before loading a new one into memory. If you do not have this variable set, <i>qtip</i> will load the <i>qoc</i> file into memory even though there may already be a shared version there.
QCSHMPATH	<i>text</i>	Defines the search path for <i>resp dbase</i> databases.

Variable	Value to set	Description
QCSLEEP	<i>number</i>	Defines the number of seconds to pause after each response when simulating responses in automatic test mode. If the number given is negative, Quancept converts the number to a positive value and then selects a random number between 0 and that number (for example, -10 results in a delay of between 0 and 10 seconds).
		<p>☞ See section 34.1, Automatic testing, for more information about testing your scripts.</p>
QCSNAP	1 (on)	Causes qtip to write debugging information about snapbacks to the <i>script.snp</i> file.
QCSNAPCOUNT	<i>number</i>	Specifies the loop limit when snapping. The default is 99000.
QCSUBSCRIPT	0 or higher	Determines what happens when qtip encounters an out-of-range subscript:
	0	(or not set) Accept negative subscripts on all variable types, and do not reset the value of the * subscript when used in an inner loop to refer to a variable defined in an outer loop. This is subscription as in versions prior to 7.8. Refer to the version 7.8 release notes for further details.
	1	Display a warning. The subscript is changed to 1 and the interview continues.
	2	Display a warning and write a message to the tiperrs file. The subscript is changed to 1 and the interview continues.
	3+	Stop with a run-time error (errorstop).
QCTTINTIME	0, 1 or 2	The number of deciseconds to wait between bytes before satisfying a read from the terminal. Suggested values are 0 if the speed is greater than 19200 bytes per second, 2 if the speed is 110 bytes per second, otherwise 1.
QCTTOUTBUF	<i>number</i>	The terminal output buffer size. The default is 2200 bytes.

Variable	Value to set	Description
QCUPDATE	0, 1 or 2	Determines what should happen if the qoc file changes while interviewing is in progress. The variable may have one of three possible settings: 0 (or not set) Carry on with the original qoc. 1 Terminate the session at the end of the current interview with the message 'New version of study available — please contact supervisor'. 2 Start the next interview using the new version. You will see the message 'Reading in new version of survey. Please wait' while the new qoc file is being read.
QCUSCORE	1 (on)	Displays a separator line of underscores between the question text and the response list.
QCWHOAMI	<i>text</i>	Defines your interviewing name. If this variable has a value, qtip does not prompt for your name at the start of each session.
QCYNRTN	1 (on)	Normally there is no need to press Return after answering one of qtip's internal questions which expects a y/n response. If you set this variable to 1, qtip will wait for you to press Return after every response. Setting this variable provides more consistent keystrokes for data entry because prompts defined in a script always require a Return.
QUANCEPT	<i>text</i>	Defines the type of terminal you are using. The default is QUANCEPT=\$TERM. The terminal type must be defined in /etc/termcap.
QCZEROVARS	1 (on)	Resets all temporary variables to zero when snapping.
TERM	<i>text</i>	Defines the type of terminal you are using. Only used if the QUANCEPT environment variable is not set.
USER	<i>text</i>	Defines the user name to display in qmonitor.

SMS and QTS environment variables

This section contains a summary of environment variables that are used in versions of Quancept where SMS and/or the QTS is running.

Variable	Value	Description
QCANOTH	1	Suppresses the ‘Another interview (y/n)’ prompt at the end of an interview. When an interviewer knows that he/she is about to start the last interview before taking a break, the interviewer should select the call result code from the menu and follow it with the letter ‘p’ (for example, 5p). qtip reminds interviewers of this by appending the text ‘(append p for pause)’ to the standard prompt to select a code from the call result menu. If QCANOTH is set to 1 and either modem dialing is used or the variable QCSTATION is set to 0, the prompt for a call result code becomes ‘Please enter a choice (append p for pause), or r to redial:’
QCASKNUMBER	1 (on)	Rather than being issued with the first eligible sample record to call, the interviewer is prompted to enter the key of the record to call. The record key is a unique number representing a record to be called. This may be the telephone number, although this may not always be the case. If the interviewer does not wish to request a record, he or she can press Return to be issued with a record in the usual way.
QCDIALER	<i>text</i>	Defines the interviewer’s extension number and the dialer’s hostname and TCP/IP port number. Mandatory in QTS projects when the script contains #audio. Also required by the dapi, dcontrol, qtsclean, qtssline, emon, tmon and dmon programs. See the <i>Quancept Sample Management and Telephony Systems User’s Guide</i> .
QCDIALERPROPS	<i>text</i>	Defines autodialer properties.
QCIVR	<i>text</i>	Defines the initial configuration properties for connecting to the IVR system. See ‘Defining the initial IVR configuration’ in chapter 27, Callfuncs, for details.
QCPRINTCMD	<i>text</i>	Defines the command to be used for printing files generated by options in the SMS supervisory menu. The default is ‘lpr %s’ for runsms supervisor and ‘lp –c %s’, where %s represents the name of the file to be printed
QCSMSRIGHTS	<i>text</i>	Limits or expands the options available to a supervisor; accessed if the supervisor’s entry in the global or project permissions file contains an &.

Variable	Value	Description
QCSTATION	<i>text</i>	Defines the interviewer's station name. See the <i>Quancept Sample Management and Telephony Systems User's Guide</i> .
QCTIMEOUT	<i>number</i>	Defines the number of seconds that qtip and SMS should wait when attempting to contact an SMS or dialer server before deciding that the server has crashed. The default is 300 seconds (five minutes). If this limit is exceeded when qtip has requested a record from the server, qtip exits with the message 'Server request timed out'. If this limit is exceeded when SMS is communicating with qts-sms, SMS returns the message 'Request to dialer timed out. Inform supervisor' to qtip.
QCTIMEOUT2	<i>number</i>	The maximum number of seconds that qtip should wait for a reply from SMS, other than when waiting for a sample record. When this time expires, qtip proceeds with the interview but does not contact SMS for the following ten seconds. (This gives the script time to complete, say, a loop with many <i>querysms</i> statements.) If the request to return the sample record times out at the end of the interview, qtip writes the message to SALVAGE.SMS and exits. The default is 30 seconds.
<hr/>		
» The return value for a timed-out <i>querysms</i> statement is null.		
<hr/>		
SMVARVALS	<i>text</i>	Defines the pathname of the a file containing initialization values for global SMS variables.
TZ	<i>text</i>	Defines the interviewer's time zone. Do not set in the Quancept environment file (QCENV).

39.3 Dialing with modems

Most interviewers dial numbers by hand or have them dialed by an autodialer. With Quancept CATI, you can also attach a modem directly to the aux port of your terminal or to the serial port of your PC and have the modem dial the numbers for you. To do this, you need to define the character sequences which tell the modem to dial the number at the start of the call and to drop the line at the end of the call.

The environment variables you use for this are QCAUTODIAL, QCAUTOSTART and QCAUTOEND. Set QCAUTODIAL to 1 if you will be using a modem. This prompts qtip to read the other two variables.

Exactly what you type in the string depends on what modem and terminal you are using, and you may need to refer to the manuals for this equipment to find out what to type.

QCAUTOSTART defines the dialing string. It should contain everything that the modem requires for dialing the number, including the number itself. qtip passes this string to the terminal at the start of a call, so you will need to include a command sequence which tells the terminal to pass the command on to the modem. Typically, you will need:

- The string to turn the aux/serial port on (the value of *po=* in /etc/termcap for the terminal type).
- The modem initialization string (ATDT...).
- The telephone number to be dial.
- The string to turn the aux/serial port off (the value of *pf=* in /etc/termcap for the terminal type).

If you wish to pass the values of SMS variables such as the telephone number, you must type them in the string as:

%variable_name

For example:

%telnumb

to pass the telephone number to be called.

Many command sequences include special characters such as new lines or octal values. Some special characters are represented by backslash followed by a letters; for example, the Return key sends a carriage return and new line sequence which you write as *\r\n*. Others are octal values which you enter as **%onumber**; for example, **%007** for the octal value 7.

If you need to type an octal value followed by a number which qtip could read as being part of the variable name, you may mark the end of the octal value with a | sign. The | sign acts as a delimiter and has no effect on the values themselves. For example:

%053

means the octal value 53 whereas:

%05|3

means the octal value 5 followed by the decimal value 3. The same applies to variable names followed by other characters. If the string could become ambiguous when the variable's value is substituted in the string, follow the variable name with | (for example, **%telnumb|**).

To refer to the phone number, type **%#** in the string. This picks up the phone number after qtip has checked it against the area code transformation file (see below), rather than using the phone number as it appears in the sample file.

To insert a one-second pause in the command string, type **@**.

If you need to insert ****, **%** or **@** as characters in their own right, type ****, **%%%** or **\@** as appropriate.

The string for QCAUTOEND is similar except that it tells the modem to drop the line (hang up) at the end of the call. Typically, it will contain:

- The string to turn the aux/serial port on (the value of *po*= in /etc/termcap for the terminal type).
- The modem hang-up string.
- The string to turn the aux/serial port off (the value of *pf*= in /etc/termcap for the terminal type).

Here are some examples that work with Wyse 60 terminals and PCs. %033 is the escape character and %015 is carriage return:

```
setenv QCAUTOSTART '%033|d#^M@@@ATDT1%telnumb^M@@@%024'  
setenv QCAUTOEND '%033|d#^M@@@+++@@@@ATH^M@@@%024'
```

or

```
setenv QCAUTOSTART '%033|5i^M@ATDT91%telnumb;H^M@%033|[4i'  
setenv QCAUTOEND '%033|5i^M@ATH^M@%033|[4i'
```

or

```
setenv QCAUTOSTART '%033[5iATDT1%telnumb;H%015%033[4i'  
setenv QCAUTOEND '%033[5iATH0%033[4i'
```

The strings used in these examples are as follows:

%033[5i ESCAPE[5i turns on the printer port.

ATDT%telnumb;H%015 dials the number, returns to command mode and places the modem on-hook (hangs up), followed by a carriage return.

%033[4i ESCAPE[4i turns off the printer port.

Optional area codes

Most telephone numbers start with an area code. When you dial a number with the same area code as yourself, the phone system often responds with a message telling you that it is ignoring the area code or asking you to redial without the area code. This message can make the modem think that the phone has been answered, so Quancept allows you to ignore these area codes without actually editing the sample as it appears in the sample files.

When you dial numbers with a modem, qtip checks the area code for the number in the file \$QCHOME/include/transform. If the file does not exist or does not contain the number's area code, the number is left unchanged. If the file contains an entry for that area code, the area code in the number will be modified as defined in this file.

Entries in this file have the following format:

old_code:new_code

old_code is the area code as it appears in the sample files, and *new_code* is the area code you wish to use as a replacement. Both or either parameter can be blank. If interviewing is taking place in the 01342 area, the transform file may contain the line:

```
01342:
```

This tells qtip (and therefore the modem) to ignore the 01342 area code in numbers. If the number in the sample file is 01342 123456, qtip will pass the number on as 12345, but numbers with different area codes will still be passed in full.

qtip reads the file sequentially from top to bottom, and once it finds a match it does not continue reading the file. If you have area codes which are subsets of one another (for example, 0207 and 02072), make sure that the more specific entry comes first. In this example, 02072 should come before 0207.

40 Managing Quancept Web projects

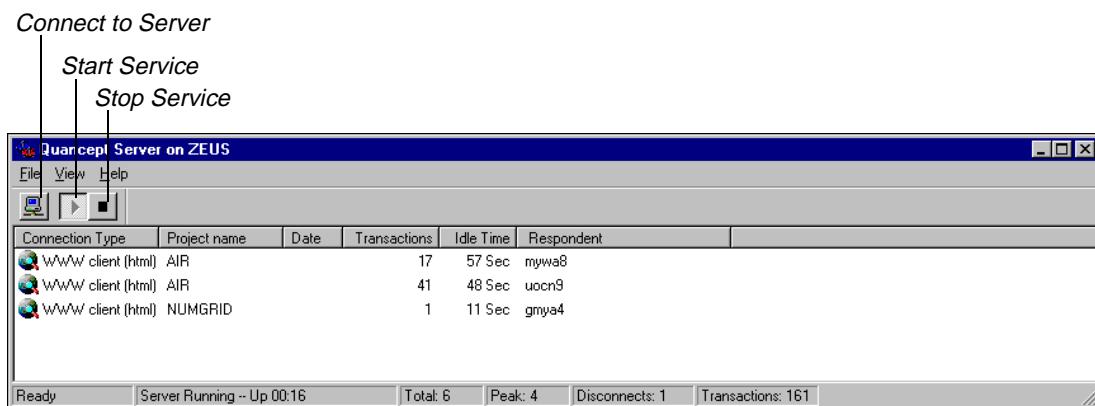
Quancept Web is unlike Quancept on any other platform because software other than Quancept and Quancept-related tools is involved. Inevitably, this means more work in setting up the project and checking that it works before it becomes ‘live’. This chapter explains these steps.

40.1 The server manager program

The Quancept Web server manager provides facilities for stopping and starting the Quancept Web server, and for monitoring the number of interviews currently in progress for each project.

The server manager also allows you to set a time-out delay for interviews that have been idle for a given period of time. Timing out an interview does not terminate the interview: instead, it creates a stop file to hold the data gathered so far, and then releases the resources that were allocated to that interview. When the respondent selects another response or clicks on a button, the interview is restarted and continues as normal. This temporary suspension of the interview is seamless and is completely invisible to the respondent.

The window that the server manager displays is as follows:



Quancept Web Server Manager runs on your Web server. All instructions in this section assume that you are logged in on this machine using a user name with the appropriate privileges.

To start the server manager:

1. From the Start menu select Run...
2. In the Open box, type `drive:\qcweb\qcserver\servermgr` and then click OK.

The server manager window is displayed.

To start the server:

- If the server is not running, click the Start Service button to start it.

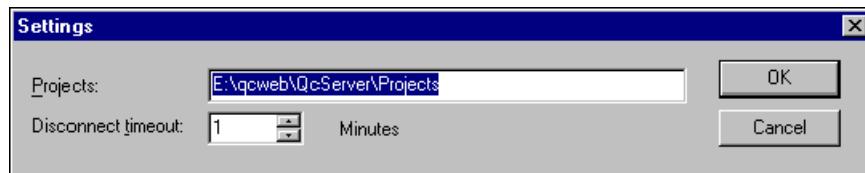
To stop the server:

- Click the Stop Service button.

To set the time-out delay for a project:

1. From the File menu select Settings.

The Settings dialog box is displayed:

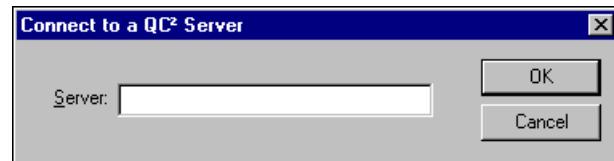


2. In the Disconnect time-out box, type or select the number of minutes idle time you will allow before a project is timed out.
3. Click OK to return to the main window.

To connect to a different server:

1. Click the Connect to Server button.

The Connect to server dialog box is displayed:



2. Type the server's name in the Server box and click OK.

40.2 Making projects available on the Web

Quick Reference

To make a project available on your Web site, type:

`selection_text`

where *script_filename* is the URL of the script file and *selection_text* is the text the user will click to start an interview.

Once you have tested your script and are sure that it works as you intended you're ready to make it available to potential respondents. The best way to do this is to create a link on your company's home page, or a page specifically designed for starting surveys, so that all respondents have to do is go to this page and click on or select a text.

You use the HTML Anchor command for this. For example:

```
<A HREF="http://www.spss.com/scripts/qcweb.dll?starthtml&zeus&cars">  
Start an interview</A>
```

In this example the text 'Start an interview' will be shown in red or blue or underlined or in italics according to the respondent's browser's style. When the respondent clicks on this text, the browser will run the cars script named in the first part of the command.

Respondent information limited to 128 characters in URL

The URL that the user types or clicks on to start an interview may contain information after the project name. This is often used when restarting stopped interviews for passing the respondent's ID to the server, but can also be used for passing other information to the server for use in the interview.

You can pass a maximum of 128 characters in this portion of the URL. Information that exceeds this limit will be truncated at 128 characters. If this limit is too small for your requirements, you should consider using the ODBC read facility for passing the information to the script instead.

Data security with Web interviews

The Internet is not a secure medium for data transfer. To maximize security you should contact Verisign at <http://www.verisign.com/> in order to obtain a security key. This means that the Secure Sockets Layer (SSL) can be used with Quancept Web, thereby allowing all data transfers to be encrypted when leaving your Web server and decrypted on arrival at the respondent's browser.

Please be aware that the level of protection provided by Verisign is sufficient to deter the casual hacker; more determined hackers may be able to bypass the security and access the data.

40.3 The project initialization file

Each project may have an initialization file. This file is stored in the project's home directory on the Quancept Web server and is called *project.ini*. A file for the cars project would be called qcweb\projects\cars\cars.ini, for example.

Initialization files are optional, but using them allows you to customize the appearance of the interviewing screens using HTML code, and to control more precisely how an interview runs. Examples of the types of things you can do are as follows:

- Define text or an image to be displayed before or after each question.

- Define what should happen at the end of the interview.
- Define your own texts for No Answer, Don't Know and Refused to be used in place of the standard texts.
- Define text to be displayed next to open-ended response boxes.
- Define messages to be displayed on specific occasions such as wrong answers or server crashes.
- Name a file containing your own navigation buttons.
- Control whether and how interviews may be stopped and restarted.
- Determine whether or not dat and tex files are created in addition to the drs data file.

HTML commands

All HTML commands must be entered in a section labelled [HTML]. Each line in this section has the following format:

variable=*text*

where *variable* is a keyword from the list below and *text* is a string of text optionally containing HTML commands. Variables you may use are:

Use	To define
AnswerPrompt=	Text to display before boxes for open-ended, numeric and real responses. The default is 'Answer:'. If you do not want a prompt displayed at all, just type <i>AnswerPrompt</i> = with no value on the right of the = sign.
Body=	Additional body attributes to be applied to each page of the interview.
ButtonMapCoords=	The start positions of the buttons in the navigation bar
DK=	A text to use for Don't Know instead of 'Don't know'.
EndHeader=	What happens at the end of the interview; for example, start the export postprocessor.
EndTrailer=	What happens at the end of the interview; for example, end the export postprocessor.
ErrNoStopRecord=	A message to display if a respondent tries to restart a stopped interview when he/she does not have one. The default is 'The Respondent ID was not found'.

Use	To define
ErrStopRecordExists=	A message to display if a respondent tries to start a new interview when he/she already has a stopped or completed interview. The default is 'The Respondent ID is already stopped or completed'. This message is displayed only if the interview is started using the &new tag in the URL. This tag causes the server to start new interviews only, and not to restart stopped interviews.
Footer=	A text or an image to display after each question.
Header=	A text or an image to display at the top of every page. Only one header is allowed per project and it is displayed for the whole questionnaire: you cannot switch it off and on or change it between questions.
ImageMap=	The name of the file containing the navigation bar.
InvalidAnswer=	A message to display if the respondent enters an incorrect answer; for example, chooses two answers from a single-punched list. The default is 'Incorrect answer(s), please answer again'.
MissingAnswer=	A message to display if the respondent tries to move to the next question without answering the current one. The default is 'Missing answer(s), please try again'.
NextButton=	The text to display on the Next button.
NULL=	A text to use for No Answer instead of 'No answer'.
PrevButton=	The text to display on the Previous button.
QCServerFailure=	A message to display if the Quancept server is not running. The default is 'Quancept server not responding'.
REF=	A text to use for Refused instead of 'Refused'.
RestartGreeting=	A message to prompt for the respondent's unique reference code when he/she restarts a stopped interview. The default is 'Restarting your questionnaire. Please enter your Respondent ID'.
RestartSubmitButton=	The text to display on the Restart button instead of Start.
ShowStopStringAtEnd=	The text that displays the respondent's unique reference code when he/she stops the interview by pressing the Stop button.
ShowStopStringAtStart=	The text that displays the respondent's unique reference code at the start of the interview.
StopButton=	The text to display on the Stop button.
StopButtonEnabled=	Whether to display a Stop button.
Timeout=	A message to display if the connection times out.

Use	To define
Title=	The text to be printed in the title bar of every page.
UsedBrowserButtons=	A message to display if the respondent uses the browser's navigation buttons. The default is 'You have used the browser buttons, please use the Next/Previous buttons below'. The message is displayed in the following situation only. Answer some questions; use the browser's Back button to return to an earlier question; change the answer to that question; then use Quancept Web's Next button to go forwards.

Here is an example that illustrates many of these settings:

```
[HTML]
Header=<center><b><i>SPSS MR Web Survey</i></b></center><br><br>
DK=No idea whatsoever
NULL=No Answer
REF=Don't want to answer this question
InvalidAnswer=Quancept cannot accept your answer. Please try again.
UsedBrowserButtons=Please use Quancept's navigation buttons, not those
belonging to your browser
AnswerPrompt=Please type your answer here
```

-
- ❖ Further information about navigation buttons and the variables associated with them is given later in this chapter.
 - Further information on the variables associated with certain methods of stopping and restarting interviews is given in section 36.3, Stopping and restarting interviews.
-

40.4 Navigation buttons or icons?

By default, Quancept Web displays two buttons labeled Next and Previous for moving to the next or previous question in the questionnaire. (The Previous button is not shown for the first question.) You can define your own labels for these buttons using the variables *NextButton* and *PrevButton* in the [HTML] section of the *project.ini* file. For example:

```
[HTML]
NextButton=Forwards
PrevButton=Backwards
```

If you want respondents to be able to stop interviews, you'll need to request a stop button. To do this, add the command:

StopButtonEnabled=Yes

to the [HTML] section of *project.ini*.

The button will be labeled Stop but you can change this with a *StopButton* variable in *project.ini*:

```
StopButton="Stop Interview"
```

¶ Notice that the button label is enclosed in double quotes because it contains spaces.

If you would prefer to display navigation icons rather than buttons you may do so. You can define your own navigation icons by drawing them using a drawing package that saves images as GIF files. When you draw the icons, make a note of the X pixel values of the top left and right corners of the three icons as you'll need to specify these in *project.ini*.

Create an images directory on your Quancept server and copy the gif file into this directory. Then edit the project's ini file as follows:

1. Define the location of your navigation bar:

ImageMap=/image_dirname/filename

where *image_dirname* is the full pathname of your images directory relative to the document root.

2. Insert a *ButtonMapCoords* statement with the X pixel values you noted when you drew the buttons:

ButtonMapCoords=prev_left, prev_right, next_left, next_right, stop_left, stop_right

You must always define all six positions even if you will not be displaying the Stop button.

You may omit this statement if the positions of your icons are identical to those in the standard navigation bar.

40.5 Sharing initialization settings between projects

Often you'll want to use the same initialization settings for a number of projects. You can either type the texts into each project's ini file or you can create a set of master files, one for each setting, that you then refer to whenever you need to use those texts. For example, if you always want to use the same header text you could create a file called *header.htm* containing:

```
<center><b><i>SPSS MR Web Survey</i></b></center><br><br>
```

Then, in each project's ini file you would type:

```
Header=@\qcweb\projects\standard\header.htm
```

in each project's ini file.

If you do this there are two points you must remember:

- Always give the full pathname of the file
- Always start the pathname with @

40.6 dat and tex files as well as the drs file

Quancept Web writes all interview data (apart from the stopped portions of stopped interviews) to the drs file in the project directory. This file is a text file that stores data in a format that can easily be manipulated and used by other programs in the Quancept Web suite.

If necessary, Quancept Web can create separate files of precoded data and open-ended texts. The dat file stores just the precoded data in a card-column format, while the tex file holds the open-end texts with various information on the field in the data allocated to those texts.

You can have Quancept Web create one or both these files by placing the following entries in the [DATA] section of *project.ini*:

```
[DATA]
WriteTexFile=Yes
WriteDatFile=Yes
```

40.7 Using interview data outside Quancept

↖ The notes in this section are designed for users with some experience of HTML and CGI programming. Since the Quancept documentation is not intended to teach or explain these areas, only brief explanations and examples have been given.

At times you may want to use your interview data somewhere other than in a Quancept application. For example, you might have a script that displays boxes for respondents to enter their names and addresses in order to receive free samples of shampoo. The name and address data is written to the project's drs file with the rest of the interview data and can be extracted for creating a mailing list or for printing labels.

The job of extracting and reformatting the data needs care since the data is scattered throughout the file and each name and address entry is preceded by other information on the line. A more efficient approach is to pass the name and address information to a postprocessor that does the formatting for you straight away. (This is in addition to having the names and addresses written to the drs file in the usual way.)

The steps involved in this are as follows:

1. At the end of the script, insert *display* statements naming the variables containing the data to be exported.

2. In the project's ini file, define a form that runs a CGI program to receive and process the exported data.
3. Write the CGI program to do whatever processing you require.

You can do these steps in any order.

Exporting the data

Quick Reference

To export data from the interview, type:

```
display '<INPUT TYPE=HIDDEN NAME=exp_name VALUE="text">'
```

or:

```
display '<INPUT TYPE=HIDDEN NAME=exp_name VALUE=""+qc_var_name+">'
```

where *exp_name* is the variable name that will be used to represent this piece of data in the CGI program, *text* is a fixed text to be exported, and *qc_var_name* is the name of a variable used earlier in the script.

Note that all quotes are single quotes.

The code inside the angle brackets is HTML code. The *TYPE=HIDDEN* parameter tells Quancept Web not to display anything mentioned inside the brackets. Instead, the value defined inside the brackets is passed to the postprocessor with the name defined by the *NAME* parameter.

Here's how you'd write the code to export the names and addresses for the free samples of shampoo.

```
        set htmlfmt='textline,45,noprompt'
name      dummyask 'Name' resp coded
addr1    dummyask 'Address1' resp coded
addr2    dummyask 'Address2' resp coded
city      dummyask 'City' resp coded
PCODE     dummyask 'Postcode' resp coded
info      multiask 'Please type your name and address in the boxes below
           if you would like to receive a free sample of this shampoo.'
name / addr1 / addr2 / city / pcode
        display '<INPUT TYPE=HIDDEN NAME=name VALUE=""'+name+">'
        display '<INPUT TYPE=HIDDEN NAME=addr1 VALUE=""'+addr1+">'
        display '<INPUT TYPE=HIDDEN NAME=addr2 VALUE=""'+addr2+">'
        display '<INPUT TYPE=HIDDEN NAME=city VALUE=""'+city+">'
        display '<INPUT TYPE=HIDDEN NAME=PCODE VALUE=""'+PCODE+">'
        display 'Thank you completing this questionnaire.'
end
```

The example uses the same variable names for the exported data as are used for the original data, but this is not a rule: you can use different names if you wish.

-
- ❖ *display* statements to export data must come at the end of the script otherwise the data will not be exported.
-

Defining the export form

Quick Reference

To define an export form that names the postprocessor program, type the following lines in the ini file:

EndHeader=<html><body><form action=*pathname* method=post>

EndTrailer=<INPUT TYPE=submit NAME=done VALUE=done></form></body></html>

At the end of an interview, control passes from Quancept Web back to the respondent's browser. The *EndHeader* and *EndTrailer* commands in the project's ini file determine what should happen at this time. The default, if you do not specify anything, is to display an ending page with the words End of Interview.

When you have data that you are passing from the interview to a postprocessor, you need to define a form that names the postprocessor to be run. The *EndHeader* command defines the start of the form and gives the full pathname of the postprocessor. Usually this will be a perl script stored somewhere on the Quancept Web server. If the postprocessor is specific to one project only then you can store it in the project directory along with the other files for this project. If it is more general and can be used with a number of projects, you might store it in the main executables directory instead. This directory is set up when the server is configured and might be called scripts or cgi-bin.

The browser runs the postprocessor which collects the data from Quancept and does whatever processing has been specified. When the program ends the *EndTrailer* command is activated. This simply closes the form and terminates the Quancept session.

-
- ❖ The syntax shown has the *EndHeader* and *EndTrailer* properties contained entirely in the *project.ini* file. In fact, you have two choices:
 - a) put the bulk of the HTML statements in the Quancept script and have extremely brief *EndHeader* and *EndTrailer* statements.
 - b) as in the syntax statement, have the *EndHeader* and *EndTrailer* properties refer to files containing the bulk of the HTML code for the end page, and put only the hidden variable display statements in Quancept.
-

Writing a postprocessor

The postprocessor is a program written in a language such as perl. It is activated by the *EndHeader* command in the *project.ini* file.

Exactly what you write in your program will depend on what data is being passed from the interview and what needs to happen to it. The example below outlines a label printing program that could be run with the shampoo sample script shown earlier.

```
sub array_it
{
    // code to put exported variables into an array called global_array
}

&array_it();

if (open(LABELS, ">>labels") == 0)
    exit(1);

print LABELS $global_array{"name"}, "\n";
print LABELS $global_array{"addr1"}, "\n";
print LABELS $global_array{"addr2"}, "\n";
print LABELS $global_array{"city"}, "\n";
print LABELS $global_array{"PCODE"}, "\n\n";

close(LABELS);

exit
```

40.8 Reusing respondent IDs

If an interview is already in progress Quancept Web will sometimes allow a second respondent to start an interview and use the same ID as the first respondent. This happens when the second respondent enters an ID as part of the URL, or when he/she is prompted to enter an ID at the start of the interview. In these cases, the server is unable to tell the difference between the two respondents and therefore accepts answers from them both, but assumes that all answers apply to the same interview.

The second respondent enters the interview at the point that the first respondent has reached, and from that point onwards a combination of both respondents' answers is added to the drs file. When one respondent finishes the interview the other respondent's interview is terminated.

Although it would be possible to correct this problem, the correction would prevent respondents from restarting interviews from different machines or IP addresses, and would be likely to create more problems than it solves. SPSS MR has therefore decided not to change this aspect of Quancept Web.

41 Managing mrInterview projects

This chapter explains how to:

- Make questionnaires available on your Web site.
- Run the activate wizard from the command line.
- Back up and delete old projects

41.1 Making projects available on the Web

Quick Reference

To make a project available on your Web site, type:

```
<A HREF="script_filename">selection_text</A>
```

where *script_filename* is the URL of the script file and *selection_text* is the text the user will click to start an interview.

Once you have tested your script and are sure that it works as you intended, you're ready to make it available to potential respondents. The best way to do this is to create a link on your company's home page, or a page specifically designed for starting surveys, so that all respondents have to do is go to this page and click on or select a text.

You use the HTML Anchor command for this. For example:

```
<A HREF="http://www.spss.com/scripts/mrwebpl.dll?project=cars">  
Start an interview</A>
```

In this example the text 'Start an interview' will be shown in red or blue or underlined or in italics according to the respondent's browser's style. When the respondent clicks on this text, the browser will run the cars script named in the first part of the command.

Data security with Web interviews

The Internet is not a secure medium for data transfer. To maximize security you should contact Verisign at <http://www.verisign.com/> in order to obtain a security key. This means that the Secure Sockets Layer (SSL) can be used with mrInterview, thereby allowing all data transfers to be encrypted when leaving your Web server and decrypted on arrival at the respondent's browser.

Please be aware that the level of protection provided by Verisign is sufficient to deter the casual hacker; more determined hackers may be able to bypass the security and access the data.

41.2 Activating projects from the command line

You can activate projects from the command line. This is useful in the following circumstances:

- You want to activate a number of projects, but do not want to have to wait while each one is activated.
- You want to activate to a number of clusters or servers.
- You want activation to run unattended and/or at some time in the future.

The Activate Document

Activation from the command line uses an Activate Document that specifies how the project is to be activated. It contains the same information as you would normally enter using the Activate dialog box in Quancept Compiler, but in XML format. This means that if you have a number of projects with identical or similar activation requirements, you can create the Activate Documents by copying and editing an existing file.

The easiest way to create an Activate Document from scratch is to fill in your requirements on the Activate dialog box and then save the specification to a file, optionally without activating the project at all.

If you have pre-version 2.3 activation initialization files, you can convert them into Activate Documents by running the ActivateIniToXml program.

☞ For information about ActivateIniToXml see ‘Converting activation .ini files to .xml format’.

Activating projects

Quick Reference

To activate in console mode, type:

activate -a "actdoc" [-u username] [-t ticketname] [-r] [-c] [-l] [-?]

To activate in dialog box mode, type:

activate -g [-p proj_ID] [-s script_type] [-f "filename"] [-d DPMservername] [-l] [-?]

You activate projects from the command line by running the Activate program in either console or dialog box mode. In both cases you will need to run the command from \Program Files\Common Files\SPSS MR\mrInterview\2.3.0.0.

In console mode, activation is based solely on the contents of the Activate Document named on the command line, and is ideal for activating remotely or on an unattended machine. To activate in this mode type:

```
activate -a "actdoc" [-u username] [-t ticketname] [-r] [-c] [-l] [-?]
```

Parameters are as follows:

Option	Description
-a "actdoc"	Names the Activate Document.
-u username	The user name to use for activating the document. If none is specified, the activation process will use the name specified in the Activate document. (It is recommended that names are not stored in Activate documents.) If there is no name or user ticket specified in the Activate document, the activation process attempts to log in as a trusted Windows user.
-t ticketname	The user ticket to use for activating the document. (It is recommended that tickets are not stored in Activate documents.) This option is valid for command-line activation only.
-r	Display a progress box.
-c	Write information to the screen.
-l	Display a login dialog.
-?	Display help.

In dialog box mode displays a dialog box in which you specify the activation parameters, and is the mode in which Activate runs in Quancept Compiler. You can include options on the command line that pre-populate the dialog box with certain types of information, such as the project name or interview script type. When you have completed the dialog box, click the Activate button to start activation. This mode is ideal for creating new Activate Documents. To activate in this mode type:

```
activate -g [-p proj_ID] [-s script_type] [-f "foldername"] [-d DPMservername] [-l] [-?]
```

Parameters are as follows:

Option	Description
-g	Displays the Activate dialog box. Use this option when you want to create an Activate document. If this option is not present, <i>-a</i> is required.
-p proj_ID	Project ID.
-s script_type	Interview script type. Valid types are Dimensions (the default) and Quancept.
-f "foldername"	Pathname of project source folder.

Option	Description
-d DPMservername	Names the initial DPM server.
-l	Display a login dialog.
-?	Display help.

Example activation commands

The following command displays the Activate dialog box:

```
activate
```

The following command activates using the myactdoc.xml Activate Document:

```
activate -a "c:\myactdoc.xml"
```

The following command activates using the myactdoc.xml Activate Document and writes any output to the log.txt file:

```
activate -a "c:\myactdoc.xml" > log.txt
```

The following command activates using the myactdoc.xml Activate Document but displays a login dialog box before activation starts:

```
activate -a "c:\myactdoc.xml" -l
```

The following command displays the Activate dialog box with the Project Id field containing 'museum'. The activation process is set to activate a Quancept project:

```
activate -g -p museum -s Quancept
```

The following command displays the Activate dialog box with the Project Id field containing 'museum' and the Source Files field containing 'c:\myprojects\museum':

```
activate -g -p museum -f "c:\myprojects\museum"
```

The following command displays help:

```
activate -?
```

Converting activation .ini files to .xml format

Quick Reference

Go to \Program Files\Common Files\SPSS MR\mrInterview\2.3.0.0 and type:

ActivateIniToXml -i "ini_filename" -x "xml_filename"

If you used command line activation prior to version 2.3 and you want to re-use the activation .ini files with Activate, you can convert them using the ActivateIniToXml tool. For example:

```
ActivateIniToXml -i "c:\mrproject\myproject.ini" -x "c:\mrproject\myproject.xml"
```

If you forget the command syntax, type `ActivateIniToXml -?` for a reminder.

41.3 Backing up and deleting old projects

When you are certain that a project is closed, you can back it up and then delete it from the system. To do this, take the following steps.

To back up the project files and delete the project from DPM

1. Back up the project's folder in FMRoot\Master. There are various ways of doing this, one of which is to use a program such as winzip to create a compressed archive file and then write that file to a CD.

 - » If the project uses quota control and you want to keep a copy of the project's mqd file, you will also need to back up the project's source folder because the mqd file is not copied to the Master folder during activation. Similarly, if the project uses any templates, ASP files, or image files that are not held in a central location, back up these files too.
2. If you wish, make a note of the DPM settings for this project. You can do this using DPM Explorer or Project Editor.
3. In DimensionNet, select the project in the project list and click Delete.

This deletes the project from DPM but does not remove any of the project files or the case data files.

4. Delete the project's folder from FMRoot\Master and FMRoot\Shared and from the Projects folders on each of the Interview servers. Also, delete the project from any user folders in which it appears.

To back up the case data database

The following instructions explain how to back up the case data database. If the project uses other databases such as a sample database or an ODBC database, repeat these steps to back up those databases.

1. From the Start menu choose Programs, Microsoft SQL Server, Enterprise Manager
2. Navigate to the server containing the project's database.
3. Expand the Databases directory.
4. In the Tree frame, right click the database you want to back up and choose *All Tasks* and then *Backup database*.

The SQL Server Backup dialog box appears.

5. In Name, type a name for the backup; the default is '*project_name* backup'.
6. In Description, enter a description for the backup.
7. In the list of backup types, choose *Database – complete*.
8. Click Add.

The Select Backup Destination dialog box opens.

9. Choose *Backup device* and then choose the device from the pull-down list. If this is the first time you have used this facility, the list will contain an option for adding a new device. Select this option and then follow the instructions in the SQL Server documentation for adding a new device.
10. Click OK.
11. In the Overwrite section, choose whether you want to append to the backup medium or overwrite it. The default is to append.
12. Optionally, use the Options tab to select backup options such as automatic verification on completion.
13. Click OK.
14. The backup starts and a progress bar appears, followed by a message when the backup is complete.

To delete the case data database

- In the Tree frame on the main Enterprise Manager window, right-click the database you have just backed up and choose *Delete*. Confirm this request when prompted to do so.

If the project uses other databases that you have backed up and these databases are not used by any other projects, you can follow these instructions to delete those databases too

42 Quancept CAPI design mode

The interviewing program that you create by parsing and compiling a script provides a graphical interface for the interviewer or respondent. Once you see your script running an interview, you may want to rearrange or resize the screen *objects* such as sections of text, images, response buttons and so forth.

In order to redesign the layout of the interview screen you need to run the script in ***design mode***, which you can select from QCompile's Options dialog box.

Design mode works by identifying screen objects and writing the changes you make to their position and size to a *tbl* file. The *tbl* file is an editable text file which is written when you exit the interview after design work. Whenever you are satisfied with the design of a section, therefore, you should exit the interview in order to save your work.

The *tbl* file labels objects according to their names in the script where this is possible. Objects that have no name in the script are assigned a temporary label by the *tbl* file. If you edit and recompile the script these temporary labels will be lost and you will have to repeat the design work for these un-named objects. For this reason you should try to ensure that all the *display* and *protect* texts in the script have a name.

It is possible to move objects off the screen entirely so that you can no longer select them. If this happens, exit the interview, open the *tbl* file in a text editor, and delete the section that refers to the object you have moved off screen. It will revert to its default position when you open the interviewing program again.

Here is a script fragment and its corresponding *tbl* file:

```
milk    protect 'Dairy Marketing Association'  
q8      ask 'Which of these logos is more familiar to you?'  
        mmpicture= 'cow1.bmp', mmpicture= 'cow2.bmp', mmpicture= 'cow3.bmp'  
        resp sp 'The first one' / 'The second one' /  
                'The third one' null dk ref  
q9      display 'Thank you for your help. Goodbye'  
        pause  
        end
```

The *tbl* file looks like this:

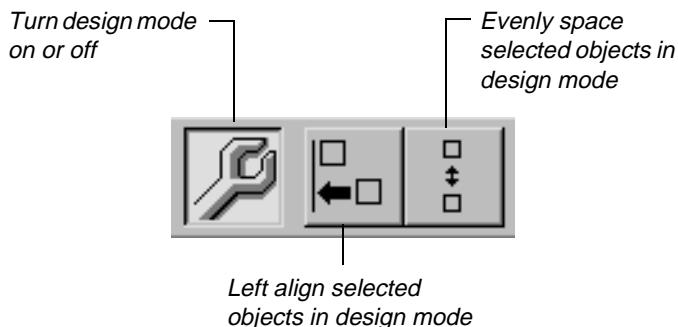
```
[Layout]  
milk.Text.X=334  
milk.Text.Y=9  
milk.Text.Width=140  
milk.Text.Height=45  
q8.Text.X=10  
q8.Text.Y=38  
q8.Text.Width=598  
q8.DK.X=185  
q8.DK.Y=218
```

```
q8.NULL.X=185
q8.NULL.Y=260
q8.REF.X=185
q8.REF.Y=302
q8.Resp0.X=185
q8.Resp0.Y=82
q8.Resp1.X=185
q8.Resp1.Y=124
q8.Resp2.X=185
q8.Resp2.Y=166
q9.Text.X=104
q9.Text.Y=130
q9.Text.Width=199
q9.Text.Height=55
```

Each line describes an object's position or dimension and the file is appended each time you do more design work.

42.1 Working in design mode

When you select design mode, QCompile makes extra buttons available on the interviewing program's toolbar:



While you are in design mode you cannot answer a question or move to the next one. To answer the question or to move on to the next screen, switch out of design mode.

To add design mode buttons to the toolbar:

1. From the Script menu, select Options.

The Options dialog box opens.

2. Select the Run tab.
3. Select Design mode.
4. Click OK.

To turn on design mode:

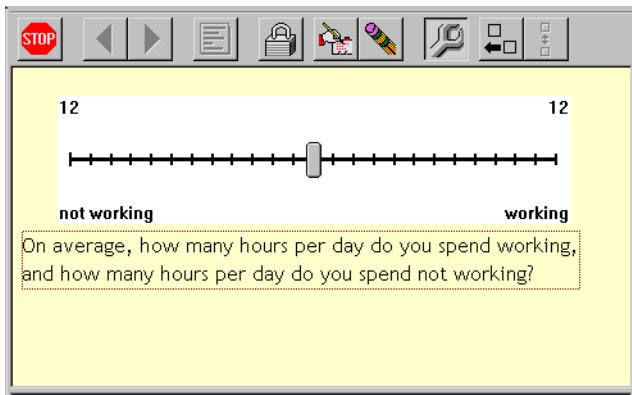
1. While the interview is running, either:
 - Click anywhere on the background of the interviewing screen; or
 - Enter an answer.

The wrench button becomes enabled.

2. Click the wrench button

Selecting objects on the interviewing screen

You can select an object on the interviewing screen by clicking on it or by clicking and dragging a selection box over it. When you select an object, the interviewing program draws a dotted line to indicate the selection area.



You can select one or several objects at a time:

- Question and display texts
- Multimedia elements, such as images, video buttons or boxes, and audio play buttons
- Response texts and buttons
- Response grids
- Objects used for collecting open-ended responses, namely text boxes, scribble boxes or audio recording buttons
- Slide bars and number pads for entering numeric data
- Keypads and Find buttons for database responses

To select a single object:

- In design mode, do one of the following:
 - Click on the object.
 - Click near the object and drag a selection box around it.

To select multiple objects:

- In design mode, hold down the SHIFT key and select the objects.

Moving objects

You can move any object on the interviewing screen.

To move an object:

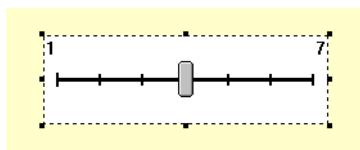
1. Select an object.
2. Click the object and drag it to a new position.

Resizing objects

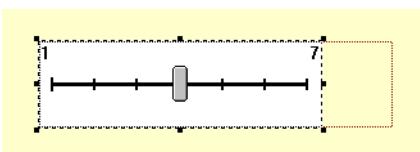
You can resize the following objects:

- Boxes that contain *other*, *resp coded*, *resp coded scribble* and *resp dbase* responses,
- Question, *display* and *protect* texts.

When you select an object for resizing, the selection box has handles. You can drag any handle to resize the selected object.



As you drag, a dotted line shows the new size for the object.



To resize an object:

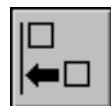
1. In design mode, hold down CTRL and click on an object.
A handled selection box appears over the object.
2. Click on a handle and, holding down the left mouse button, drag it to a new position.
QCompile resizes the object.

Left-aligning objects

When you left-align objects, QCompile aligns the selected objects with the one selected first.

To left-align objects:

1. Select the objects to be aligned.
2. Click on the Left Align button



Adjusting the vertical distribution of objects

You can evenly distribute three or more objects, which is useful when you want to apply even spacing to objects you have moved around the screen. When you adjust the vertical distribution of objects, QCompile spaces them according to the distance between the first and second objects you selected. The first object you select should be at the top or the bottom of a vertical ‘list’ of objects. When you select the topmost object in the list first, QCompile spaces the other objects below that object. When you initially select the bottom-most object in the list, QCompile spaces other objects above that object.

-
- ❖ When you distribute objects, try to avoid selecting an object in the middle of a list of objects before selecting other objects in the list. If you do this, QCompile moves the selected objects so that they appear above or below the first selected object, depending on its list position.

To avoid confusion you should select objects for distribution one by one rather than dragging a selection box across them.

To adjust the vertical distribution of objects:

- With the relevant objects selected, click on the spacing button.



-
- You may find that when you run the interviewing program, repositioned or resized *display* statements do not appear as intended. This may be because the *display* statements do not have unique labels. In such cases, the interviewing program assigns its own labels internally and may not be able to identify which *display* statement was repositioned/resized. If you intend to redesign the layout of *display* statements, ensure that you assign a unique label to each one.
-

A Limits

This appendix lists limits within the Quancept language.

- 80 columns per card (CATI/CAPI/Web)
- 8 characters in a label name
- 8 characters in the project name (CATI/CAPI/Web)
- 16 characters in the project name (mrInterview)
- 1168 characters per response text
- 1000 responses per question (CATI)
- 729 responses per question (CAPI/Web)
- 9929 responses per question (mrInterview)
- 999999 open-ended responses per project
- 1023 characters of *display/protect* text including substituted values (CATI/CAPI/Web)
- 100 *display/protect* statements per page (mrInterview. Use *dsplmax* to increase).
- 2000 characters including substituted values for each *ask*, *dummyask*, *multitask*, *display* and *protect* (mrInterview)
- 14 lines of *display/protect* text with a question text (CATI)
- 21 lines of *display/protect* text without a question text (CATI)
- 2047 characters in an open-ended response defined using a *set* statement. Otherwise the length is controlled by the amount of space available on the screen, as interviewers can only type as much text as will fit into the blank lines on the current screen. (CATI)
- 4000 characters in an open-ended response text (mrInterview)
- 10 columns per integer question variable (including a minus sign). If the integer is negative, you can have 9 digits and the minus sign.
- 14 columns per real question variable
- 60 responses with *freeze*
- 5 columns per screen display with *setcols*
- 2047 characters copied with *substr*
- 2047 characters copied with *setstr*
- 2047 characters copied with *selrec*

- 16 characters per SMS key
- 72 columns per *fix*
- 200 *resp dbase* database files per script
- 99999 records per *resp dbase* database file (qparse always reserves 5 columns in the dat file for a resp dbase question, so the limit is 99999 unless you write only drs/ddr files, in which case the limit is 100000)
- 2047 characters per line in *resp dbase* database files (CATI, but only the first 72 characters are displayed)
- 80 characters per line in *resp dbase* database files (CAPI)
- 1024 characters in a GNU database record (CATI)
- 8 GNU database records updated with *impupdate* per interview (CATI)
- 1024 characters in a GNU database field updated with *impupdate* (CATI)
- 63 characters in a GNU database record key when using *impupdate* (CATI)
- 63 characters in a GNU database filename (CATI)
- 3 decimal places per real value, but only 2 when the value is displayed in qtip (values with 3 decimal places are rounded)
- 10 *if ... else* levels of nesting
- 10 languages per multilingual survey (CATI)
- 99 maximum value for *privsig*
- 6 quota axes per quota
- 99 *quota* statements per project
- 999 text or numeric values in the value list for a *for* statement (CATI)
- 298 text or numeric values in the value list *for* a for statement with qcset (CATI)
- 999 temporary variables currently assigned per script if QCGROW=0 (CATI)
- 300 characters substituted in a text (see next entry for an exception)
- 255 characters substituted in a text inside a loop when the loop is based on a defined list
- 30 variables combined with *longtext*
- 128 characters passed in the Quancept Web URL
- 99 characters in the pathname of a QTS sound file
- 4095 characters per response in the ddr and drs files

999 characters in a response if no unique ID is defined (mrInterview)

2000 cards with qcformat

B Keywords in questionnaires created with Build

When you design a questionnaire using Build, all the information about the questionnaire is written directly to a questionnaire definition (mdd) file. When you test the questionnaire, an intermediate program is run to generate a Quancept script file from the information in the questionnaire definition file. If you look at the Quancept script you will see three keywords at the start of the script that you would not normally use in questionnaires that you create manually using a text editor. These keywords control the following aspects of the script generation procedure:

UseCurrMdd	Whether to create a new version of the questionnaire in the questionnaire definition file.
UseMapFile	Whether to use mapping for variable names and responses.
usepictf	Whether to look for picture information in the questionnaire definition file.

When a Build questionnaire uses an If...Goto, the expression that the user creates to define the condition is converted into Quancept code using an **eval** keyword. You can use this keyword in your own questionnaire scripts when you want to evaluate an expression and substitute the result of the evaluation in a text.

B.1 Using the current version of the mdd file

When you create a questionnaire using Build, the questionnaire content is saved automatically in a questionnaire definition (mdd) file. To use the questionnaire for interviewing, you need to create a Quancept script from the mdd file and then parse and compile that script to create a sif file that can be activated and used for interviewing. You can either do this automatically using Build's Test option, or you can run the steps manually using the Quancept activity.

When you parse a Quancept script, the parser normally creates a new version of the questionnaire in the mdd file to match the current contents of the Quancept script. In this case, however, the Quancept script has just been created from the current mdd file so there should be no need to update it. To prevent this happening, the application that generates the Quancept script from the mdd file inserts a **UseCurrMdd** statement near the top of the script.

Occasionally, you may want to force the mdd file to be updated, in which case you can remove this statement either by commenting it out or by deleting it. Examples of when you may want to do this are as follows:

- When you create a Quancept script from a project's mdd file, and then edit the script. Any changes you make by manually editing the Quancept code will not be transferred to the mdd file unless you allow the parser to update this file.

- When your Build questionnaire has a Quancept Script item that contains *fix* or similar statements Build creates the mdd file, it creates variables for all questions in the questionnaire but not for any variables that are generated by statements in QCScript items. It simply copies the code in these items so that it can be placed in the Quancept script. So, although these statements appear in the Quancept script, the variables will not appear in the mdd file unless you force the parser to recreate it.)

UseCurrMdd interacts with the *UseMapFile* keyword (see below) as follows:

	UseMapFile exists	UseMapFile does not exist
UseCurrMdd exists	Do not generate a new version in the current mdd file.	Do not generate a new version in the current mdd file.
UseCurrMdd does not exist	Generate a new version in the current mdd file using the map file.	Ignore the map file when generating a new version in the current mdd file.

B.2 Mapping for variable and response names

With Build you can create questionnaires in any language that your computer supports, and can use question names longer than the eight characters allowed by the Quancept parser. In order for the parser and the sif interviewing program to accept names that are not valid Quancept names, the application that writes the metadata to the mdd file generates a mapping file — *script.qqc.map* — that maps these names to valid Quancept names. The **UseMapFile** statement indicates whether or not a mapping file should be used. The default is always to use a mapping file.

If you edit a Quancept script that has a mapping file, you can add routing and other non-question-and-response statements to the script without having to alter the metedata.

-
- *UseMapFile* does not control whether the interviewing program uses mapping. If the mapping file is present in the project source folder, it is always compiled into the project's sif file.
-

Refer to the table at the end of the previous section for information on the interaction between *UseMapFile* and *UseCurrMdd*.

B.3 Questionnaires with pictures

Build lets you associate pictures with questions and responses, and this information needs to be transferred to the Quancept script. The **set usepictf=1** statement near the top of the script tells the script generator that it should look for picture information for every question and response in the questionnaire. This process inevitably causes the script generation to be a little slower than it would otherwise be. If your questionnaire does not contain images, you may remove this statement from the questionnaire script.

B.4 Evaluating expressions

When a user creates an If...Goto, he/she defines the condition that controls the Goto by selecting questions and responses from dropdown lists. When the Quancept activity converts the questionnaire file into a Quancept script, it replaces the condition with an expression that can be evaluated programmatically. The expression appears in the Quancept script as an **eval** text substitution:

[\$eval([+expression+])\$]

When the Quancept parser reads the character string `[$eval(` it scans the rest of the script looking for a matching `)$]`. When the parser finds these characters, it copies everything between the start and end markers into the interviewing program. During the interview, the expression is parsed and evaluated using the Evaluate component and its value is substituted in the text.

If the Quancept parser cannot find the closing character string then no substitution is made and no evaluation is done.

+ Refer to Dimensions Development Library for information about the Evaluate component.

You can use *eval* in the following statements:

- question, response, *display* and *protect* texts; for example:

```
qdisp    display 'The value of [$eval(2 + 3)$] should be 5'
qexpr    ask 'Enter and expression to be evaluated'
          resp coded
qevel    ask 'The expression [+qexpr+] evaluates to [$eval([+qexpr+])$].
          Is this correct?
          resp sp 'OK' / 'Not OK'
```

- *dbquery* and *selrec*
- *msgnull*, *msgref*, *msgdk* and *msgoth* definitions
- *set* statements for *fix* variables
- *pgtitle* definitions

You cannot use *eval* with *nodata* questions. You should also test your script carefully to ensure that it does not produce erroneous results if the variables used in the expression change from being on the interview path to being off-path.

Index

Symbols

!, cancel selections from multipunched response list 807
!, filter response list database 809
!, function of in interview 41
!,function of in interview 810
", accept current responses 821
", displaying in text 41
", function of in interview 41
#, function of in script 41
\$, function in script/interview 42
%, function of in script 41
%, print with *rprint* 429
&, display in text 41
(), function of in script/interview 40
*, current subscript value 265
*, function of in script 39
+, function of in script/interview 39
+, scroll forwarded in a response list database-, scroll backwards in a response list database 809
+, scroll forward in response list database 810
+, variable name in text 196
-, function of in script/interview 39
-, scroll backwards in response list database 810
., function of in script/interview 42
/, function of in script 39
/, list question names on interview path 818
/autojump, automatically display next question when single-punched question answered 769
/ip/projects, define alternative for 913
:, check answer to a previous question 818
:, function of in interview 41
;:, request serial number for handwritten open end 41, 808
<, check for responses before given one 325
<, display in text 39
<, function of in script/interview 39
<, snap back to previous question 820
<<, snap back to start of interview 820
>>, function of in script/interview 40
>>, list names of snappable questions 819
<SmallCaps>CATI interviewing, texts for 305
=, append to existing open end text 811
=, function of in script/interview 39
=, select response from response list database 809
> snap to next question 820
>, check for responses after given one 325
>, display in text 40
>, function of in script/interview 40
>>, snap as far forward as possible 820
?, function of in interview 41
?help for appointment setting 825
@, function of in script 41
@, new line in text 46
@, request serial number for handwritten open end 808
[], function of in script 40
[+...+], text substitution in loop value lists 277

[+...+], variables in response lists 198
\, display in text 39
\, function of in script 39
\n, new line in report file 428
\^, function of in script 41
_stop directory, stopped interview directory 813, 817, 894
_stop/USED directory, restarted interviews 894
\{ }, function of in script 40
\#0a, new line in report file 428
\\$, display in text 42
\\$, function of in script/interview 42
\', function of in script 40
\,, function of in script/interview 42

A

<a href>, anchor interview on Web page 924, 935
abandon, terminate interview without data 816
 save data after 389
aca, automated conjoint analysis 620
Accounting information
 custom 614
 file for qtip 880
 identify sections in a script 612
 private status codes 419
 time sections in a script 611
accounts.sms, information on records returned from SMS to qtip 898
acctinfo, write custom accounting information to the qca file 614
Accuracy with real numbers 87
acl file, script in format for creating flowcharts 862
act file, qtip accounts 880
Activate Document 936
Activate program, run from QCompile 780
Activation
 base language 781
 cluster to activate on 781
 project status 781
 with Quota Control 783
Active projects, running test interviews on 854
adddatepart, increment current date/time by given value 599
Aided/unaided awareness, script for 346
<align>, horizontal alignment in grid cells 547
<align>, horizontal position of grid on page 563
Aligning cell contents in grids 547
Aligning objects in design mode 947
<allCells>, settings for all grid cells 544
allow, allow early completion 380
 effect of in qtip 815
Alphabetic sorting of responses in multilingual scripts 115
Alphabetical ordering of responses 113

Alphabetic ordering of responses, unsorted responses 127
alphdate, convert text dates to operating system format 600
<altBodyCol>, format for alternate grid columns 544, 558
<altBodyRow>, format for alternate grid rows 544, 558
Ampersand, display in text 41
.and., both/all expressions true 327
and, all responses required 338
Answer prompt, define for Web interviews 297, 300
AnswerPrompt, open-end, numeric & real prompt 926
Answers *See Responses*
Apostrophes in texts 34
append, append one text to another 583
 restrictions when using 584
Appointments
 arrange with respondent 824
 assign to specific interviewer 826
 comments for 826
 entering time ranges for 825
 help with setting 825
 make in script with *makeappt* 601
 set in script with *setdatepart* 599
 time differences with 825
Area codes *See also* International dialing code
 ignore when dialing with modem 921
Arithmetic expressions 361
 accumulating totals in loops 271
 data type hierarchy in 365
 integers in 372
 mixing variable types in 364, 365
 multiple operands on a line 362
 negative numbers in 362
 real numbers in 365
 save in question variables 364, 366
 save in temporary variables 364, 365, 367
 temporary variables in 361, 363
 text numbers in 370
 uses of 362
 variable types with 363
 with real numbers 371
Arrays
 definition of 26
 maximum array size 272
 subscription of 272
ask, define question text 80
 with *mintime* and *maxtime* 190
 See also Questions, Labels, Responses
ask #SUB, call subsurvey script 570
Assignment
 abbreviating long response texts 287
 arithmetic with temporary variables 363
 behavior for *sp/mp/list* assignments 291, 914
 longtext, combine responses 312
 one question to another 289
 response positions 288
 response texts 285
 responses to questions 284
 temporary variables 281
 text 283
 variable types with 363
with subscripted variables 292
with *substitute* 315
Asterisk, function of in script 39
At sign
 new line in text 46
request serial number for handwritten open end 808
atoz, display responses in alphabetical order 113
 language setting in multilingual scripts 603, 911
 response groups with 128
 with QCRETAIN 114
atozflgs, alphabetic sorting flags 116
atozlang, language for sorting in multilingual scripts 115
atozmode, type of alphabetic sorting to use 116
atozsort, sorting order for multilingual scripts 115
#audio=playback, play sound files 160
#audio=record, record open ends as sound files 161
audio directory, CAPI recorded open ends 898
Audio files *See Sound files*
Audio recordings *See Sound recordings*
auto directory, *autostop* data files 881
Autocorrection, control editing of multipunched responses 912
Autodialer properties, defining 917
autojump, automatically jump to next question 138
Automated Conjoint Analysis 620
Automatic open ends
 for *sp/mp* responses 107, 109
 how stored for *promptoth* 108
Automatic script testing
 CAPI 768, 843
 CATI 831
Automatic test mode, check for 295
autostop, save response data periodically during an interview 617
 data files created by 881
avi files, video clips 153
Axes specs for Quantum 770

B

..., bold text 486
..., bold text 495
Background
 color for grids 545
 color for interview screen 491, 503, 535
 picture for interview screen 491
 picture style for interview screen 491
Backing up and deleting old projects 939
Backslash
 display in text 39
 function of in script 39
Backtracking *See* Snapbacks
Barcode information file 863
<bgColor>, grid background color 545
bit, check responses by position in list 342
 how differs from *nbit* 343
 with *elm* 287, 346
 with *null, dk* or *ref* 343
 with specified other 343

bitmap directory, scribbled open ends 882
bkcolor, background color for interview screen 491
bkground, background image for interview screen 491
bkstyle, layout style for interview screen background picture 491
 Blank cards, suppressing 420
 Blank fields in response list databases 438, 445
 bmp files, scribbled open ends 157, 882
<body>, set background & foreground colors 503
Body, additional attributes for each interview page 926
 Bold text 486, 495
 Bold text in grids 549
 Boolean variables, quotas for 738
 Borders in grids 564
<borderSize>, border size in grids 564
 bqt file, barcode information 863
**
**, line break 496
 Braces, function of in script 40
 Brackets, square, function of in script 40
 Browser
 run Web scripts using 845
 text to display when browser buttons used 928
 Browser buttons, error message when used 297, 300
 bt8 file, 8-bit characters found in script 863
 Busy calls, with qtip and SMS 824
ButtonMapCoords, positions of buttons in Web navigation bar 926, 929

C

Callable functions *See* Callfuncs

Callbacks *See* Appointments

Callfuncs

- aca 620
- acctinfo 614
- adddatepart 599
- alphdate 600
- append 583
- autostop 617
- chgstop 384
- commitivr 633
- convdata 134
- datepart 597
- datetext 597
- datetime 596
- definition of 26
- disable_audio_record 622
- enable_audio_record 622
- getdbvar 667
- getenv 605
- getsmvar 661
- how seen by qparse 648
- in grids 254
- keepstopped 619
- limitresp 92
- list of those used in script 871
- local 647, 650
- longmath 372
- longtext 312

- makeappt 601
- nsubstr 586
- prepareivr 627
- putdbvar 668
- putsmvar 662
- qc_hangup 623
- querysms 664
- quorat 677, 685
- quotacell 682
- quotamap 682
- quoval 687
- readfile 194
- realmath 371
- redial 624
- rfclose 426
- rfopen 426
- rfprint 427
- setcols 201
- setdata 389
- setdatabaser 604
- setdatepart 599
- setinteger 368
- setivr 625
- setlang 602
- setlocale 603
- setreal 368
- setstr 587
- showinf 51, 615
- smscript 656
- stopdata 388
- strngchk 350
- subprog 608
- substr 588
- summary list 70
- testmode 606, 607
- timesect 611
- valstring 372
- varlist 608

Calls

- arrange appointment for 824
- busy 824
- making 823
- no answer 824
- outcome, coding in qtip 823

Cancel incorrect selections in qtip 807

CAPI

- automatic script testing 843
- automatically display next question when single-punched question answered 769
- display name of current question during interviews 192
- enter responses in 839
- enter responses using keyboard 766
- files created by interviewing program 879
- files used by script 866
- interview language 769
- interview screen characteristics 491
- interview toolbar 839
- interviewer's name 767
- interviewing window 838
- options for testing scripts 765

CAPI *contd.*

- parse scripts 759
- position of toolbar on interviewing screen 766
- restart stopped interviews 842
- run interviewing program full screen 766
- screen layout 894
- start an interview 837
- stop interviews 841, 842
- suppress opening dialog box 766
- test interviews with 837
- text formatting overview 485
- timing responses 769

Captions below grids 543

Card number, increase number of columns for 407

cardcols, number of columns for card number 407

Cards

- avoid conflict with *qparsel -u* 406
- data map, predefined 759
- definition of 397
- increase number of available 569
- restore default data mapping 405
- specify map for multiple responses 405
- specify map for one response 404
- start new before old is full 402
- suppress blank 420

Caret, function of in script 41

Cascading style sheets 533

- dropdown list box tag 534
- error message tag 534
- format and location of 533
- list box tag 534
- mp* response button tag 534
- multiple response text tag 534
- name in template 535
- numeric grid cell tag 534
- question text tag 534
- single response text tag 534
- sp* response button tag 534
- tag for *display* statements 534
- tag for numeric/open-end answer boxes 534
- tag for *protect* statements 534

Case data file for mrInterview 899

Case sensitivity in qtip 42

Categorical values, assigning to temporary variables in multilingual scripts 304

CATI

- automatic script testing 831
- correcting mistypings 807
- environment variables 906
- error messages 826
- files created by interviewing program 879
- highlighted text on screen 200
- how to deal with changes to the qoc file 916
- interview duration, where stored 880
- interview result, where stored 881
- parse scripts 757
- quota maintenance 714
- recorded keystrokes, directory for 893
- recording interview keystrokes 835, 893
- recreate interviews from keystroke recordings 834
- rescheduling stopped interviews 813, 817

- responses echoed back 811
- return to SMS menu 826
- review completed interviews with quotas 675
- reviewed interview directory 883
- reviewing completed interviews 833
- save data for interviews terminated by interviewers 387
- save data for incomplete interviews 389
- share qoc file 832
- stop recording keystrokes 835
- stopped interviews, directory for restarted 894
- t flag 831
- test interviews 803
- time sections of interview 611
- using a shared qoc 832
- using qtip 804
- with SMS 823

cell, find respondent's quota cell 682

<**cellPadding**>, space around cell contents in grids 565

Cells, combine quota 719

<**cellSpacing**>, cell spacing in grids 564

<**center**...</**center**>, centered text 500

ChangeQuota program 755

Character set 38

Characters

- number in defined list names 166
- number in question names 79
- number in script names 905
- number in statement names 43

Check mrInterview test data 856

Check text format 350

Checking data 854

Checking previous responses 818

Checking statements, introduction to 33

checkpars, check CATI script syntax 758

chstop, user-definable filenames for stopped data 384

CI-2 interviewing package 620

Circumflex symbol *See* Caret

Code frames, copy between qdi files 775

coded, open-ended response list 88

- prompt text for 926

Coded responses *See* Open ends

codscheme, override data format for one question only 409

col, card and column for one response 404

colgrid, display a loop as a grid 249

- <*mrPageTitle*> with *multitask* 526

Colon, function of in interview 41, 818

Color

- background for interview screen 491, 503
- foreground for interview screen 503
- of text 488, 496
- using RGB values 492, 497

<**color**>, color value tag in grid template 545, 551

<**colorFormat**>, color format tag in grid template 545, 551

Colored text on non-white background 488

Column 1 in script file 44

column 100, restore default data mapping after *col* or *column* 405

Column headings, repeating in grids 557

Column widths in grids 251, 554
column, card and column for multiple responses 405
 Columns
 allocation to questions in loops 248
 avoid conflict with *qparse -u* 406
 display responses in 204
 increase for response list display 201
 number for defined lists 172
 number for integer responses 85
 number for multipunched responses 82
 number for real responses 88
 number for response list databases in data file 440
 number for single punched responses 82
 number for specified other 103
 restore default data mapping after *col* or *column* 405
 set width of response 204
 set widths of in grid 251
 specify map for multiple responses 405
 specify map for one response 404
 wrap heading texts in grids 547
<colWidths>, columns widths in grids 554
 com file, messages to supervisor 882
 Comma, function of in script/interview 42
 Command line arguments for CAPI 769
<comment>, grid caption 543
comment, add explanations to scripts 48
***comment**, number of comments to display with 822, 909
 Comments
 append to during call 822
 enter in CAPI interviews 842
 for appointments 826
 number to display to with ***comment** 822, 909
 with *else* 218
 comments directory 882
commitivr, transfer current call to IVR system 633
 Comparing logical expressions 327
complete, early completion 815
 Completed interviews, reviewing 833
 Concatenating texts 583
 Conditional routing 34
 Conditions, maximum per *if* statement 216
connect, connect to ODBC database 454
 Connections, actions on time-out 927
context, context from which to read interview texts 307
 Continuing statements and texts 44
 with library files 569
continue, place holder statement 49
control=, drop-down lists and combo boxes for response lists 205
 conv8, convert 8-bit CATI scripts into 7-bit format 758
convdata, apply scaling factors 134
 with question variables 135
 with temporary variables 134
 Conversion
 currency, example of 132
 numeric, writing in script 132
 real numbers to integers 368
 text numbers to integers 372
 Conversion characters with *rfprint* 429
 Copying texts 588
 Corrupt screen displays, correcting 818

Counting responses
 in response list 344
 number chosen 344
 Crashes, actions to take 927
 CSS *See*Cascading style sheets
 CTRL+X, cancel incorrect multipunched response in qtip 807
 Curly brackets *See* Braces
 Curly quote *See* Single quotes
curpgstyle, page style for mrInterview questionnaire 530
 Currency conversion, example in script 132
 Customizable error messages 297, 298
 Customized status codes 419

D

dat file, card-column format data 883
 create for Web interviews 930
 Data
 check mrInterview test data 856
 checking 854
 clearing from off-path questions 308, 309
 coding *dk* 82, 101
 coding *null* 82, 100
 coding *ref* 102
 coding *rot/ran* responses in 122
 coding specified other 82, 103
 column allocation with loops 248
 columns allocated to defined lists 172
 columns allocated to multipunched responses 82
 columns allocated to numeric responses 85
 columns allocated to open ends 88
 columns allocated to single-punched responses 82
 columns, avoid mapping conflicts 406
 columns, specific mapping for 404, 405
 copy into sample record 662
 for subsurvey scripts 571
 generate automatically with QCompile 768
 how database responses are coded 440
 number of columns for response list databases 440
 numeric responses in mrInterview 86
 order of questions in 868
 pass between main/subsurvey script 574
 read into script from SMS database 667
 real responses in mrInterview 87
 reserve space to extend defined lists 172
 retrieve from sample record 661
 save all even after snapbacks 416
 save for interviewer-stopped/terminated interviews 387
 save for stopped interviews 388
 save for terminated interviews 389
 save for uncompleted interviews 389
 save rotation number in 125
 store SMS record key in 661
 transfer between SMS and script 661
 variables in mrInterview case data 854
 write into SMS database from script 668
 write out to temporary file 608

Data *contd.*

- write to log file when SQL connection fails in
 mrInterview 304
- Data directory
 - contents of 883
 - suppressing 884
- Data entry fields, automatic completion of 528
- Data format
 - allocate columns in order values are assigned to
 questions/variables 757
 - changing 407
 - column allocation for *sp/mp* responses 82, 83
 - description of 867
 - increase columns for serial number/card number 407
 - predefined map file (*qparse -u*) 759
- Data map
 - define your own 759
 - move serial number in 759
 - reduce size of with loops 273
- Data security for Web interviews 925, 935
- Data types
 - numeric assignments 363
 - temporary variables 282
- Databases
 - cancel filters in CAPI interviews 840
 - filter in CAPI interviews 840
 - ODBC 453
 - select responses from in CAPI interviews 840
 - set SQL cursor type 461
 - Unix, summary of commands for 78
 - See also* Gnu databases, Response list databases, SMS databases
- datamap othzero**, code specified other as zero 104
 - in secondary scripts (subsurveys) 104
- datamap**, data layout commands 407
- *date**, display current date in qtip 822
- Date format, set for interviews 824, 910
- Date manipulation callfuncs 596
- date**, date of start of interview 293
- Date/time
 - convert operating system to text 597
 - convert text to operating system format 600
 - entering with qtip 825
 - extract items from string 597
 - get in operating system format 596
 - setting within the script 599
- datepart**, extract items from date/time string 597
- datetext**, convert operating system date/time into a text 597
- datetime**, get date/time in operating system format 596
- dayofweekn**, day of week as number 294
- dayofweeks**, day of week as string 294
- dbase**, response type for database 439
- dbclose**, close ODBC database connection 455
- dbquery ... insert into**, add records to ODBC database 460
- dbquery ... select**, select records from an ODBC database 456
- dbquery ... update**, update ODBC database record 458
- dbquery**, query an ODBC database 455
- dbrecno**, save response list database record number 442
- dbsqlcsr**, set SQL cursor type 461
- ddr directory, individual respondent data files 883
 - suppressing 884, 912
- ddr file
 - maximum response length 950
 - sample variables in 889
- ddr/USED directory, reviewed interviews 834, 883
- Decimal numbers *See Real numbers/responses*
- Decimal places for real responses 86
- decrm**, decrement quota when interview is stopped 383
- Deduplicating data *See Quancept Utilities Manual*
- Define environment variables 906
- define**, define a response list 165
- Defined lists
 - associate responses with pictures 149
 - atoz & QC retain* with 114, 176
 - blank responses in 181
 - codes allocated to responses in 166
 - coding of other in 172
 - columns allocated in data file 167
 - defining 165
 - expand list names 178
 - find which sublist to use 178
 - keep original codes when displaying 914
 - keywords valid with 171
 - length of list name 166
 - loops with specified other 241
 - master list as a response list 167
 - null, dk, ref* with 168
 - null/dk/ref* with *list* 179
 - number of responses in 172
 - order of responses in 166
 - order of responses in sublists 166
 - position in script 166
 - promptoth* in 171
 - reserve space for extending 172
 - response substitution in 199
 - responses not chosen from variable sublist 180
 - rotranfix* in 171
 - rotransub* in 171
 - ^s* in 171
 - sp* in 171
 - specified other in 169, 177
 - sublists as response lists 173
 - use responses excluded from a sublist 175
 - uses of 165
 - with loops 238
- Dependent quotas
 - checking 678
 - explanation of 674
- Design mode 943
 - add design buttons to toolbar 766
 - display* statements and 188
 - distributing objects vertically 947
 - left-aligning objects 947
 - moving objects 946
 - resizing objects 946
 - selecting objects 945
 - turning on 945
- Designing the screen layout 493
- dfmt**, *display/protect* text global format 498

Dial call from within script 624
 DimensionNet, project and file names for 52
 DimensionNet, working on Quota projects in 705
 Directories
 auto 881
 comments 882
 data 883
 recorded keystrokes of interviews 893
 restarted interviews 894
 stopped interviews 894
 suppressing ddr 912
 Dirty data, saving 416
disable_audio_record, switch off recording of open ends 622
disallow, disallow early completion 380
display, display text temporarily on screen 185
 export data with 931
 global format for text 490, 498
 in design mode 188
 maximum amount of text with 186
 maximum variables per statement 198
 pictures with 187
 position on mrInterview page 524
 removing displayed text 186
 tag in cascading style sheet 534
 text formatting with 187
 variable values with 186, 197
 with *mmaudio* 154
 See also xdisplay
 Display cursor in answer box in CAPI interviews 841
 Display text in a message box 188
display, maximum per page 308
 distributing objects vertically in design mode 947
dk, don't know response 101
 check for 324
 check whether chosen 350
 coding 82, 101
 don't know with qtip 813
 formatting in mrInterview 498
 routing within response list 120
 text to display for 105, 106
 with *bit* 343
 with defined lists 168, 179
 with *elm* 345
 with *nbit* 336
 with *numb* 344
 with *numv* 344
 with quotas 676
 with *set* 292
dk, text to display for Don't know 926
 Dollar sign, function in script/interview 42
 Don't know
 coding in qtip 813
 enter in CAPI interview 840
 how to allow for 101
 text to display for 105, 106, 926
 See also dk
 Double quotes
 accept current responses 821
 displaying in text 41
 function of in interview 41

typing in scripts 40
 Drop-down lists of responses 205
dropdown, display responses in drop-down list 205
 drs file, data in ASCII format 887
 differences between CATI & CATI/Web 888
 do not create 912
 maximum response length 950
 sample variables in 889
 drx file, index to drs file 890
 do not create 912
displaymax, maximum number of *display/protect* statements per page 308
 dum file, dummy subsurvey script 571
 Dummy questions 91
 assign responses to 274, 287, 289, 290, 445
 build response with *append* 584
 definition of 26
 on interview path 416
 pass data between main/subsurvey scripts 576
 points to remember when using 290
 with *condata* 135
 with *mention* 341
 with quotas 684, 685
 Dummy statements 49
dummyask, define unasked question 91
 See also ask
dummyother, automatic open ends without allowing specified other 109

E

***e**, enter/change open ends with editor 812
 Early completion
 allowing 380
 effect of snapbacks on 380
 requesting in qtip 815
echo, echo responses during interview 117
 effect of during interview 811
editopen, edit open ends 352
 Editor
 define default for appending verbatims at end of interview 909
 enter/change open ends with ***e** 812
elm, find text by response position 345
 with *bit* 287, 346
 with *null, dk or ref* 345
 with specified other 345
else, conditional actions 216
 comments with 218
 if without 219
 leaving 219
 nesting 219
 with *goto* 219
 Empty questions, testing for 354
empty, test for unanswered questions 355
emptyok, all blank cells in numeric grids 297
enable_audio_record, switch on recording of open ends 622
end, end of script 50

EndHeader, invoke export postprocessor 926, 932
EndTrailer, close export postprocessor 926, 932
Environment variables
 define/set 906
 definition of 27
 F_VERB 908
 file to set in 907
 LANG 908
 LANGUAGE 908
 LOGNAME 908
 QCAMPNAMES 908
 QCANOTH 917
 QCASKNUMBER 917
 QCAUTODIAL 908
 QCAUTOEND 908
 QCAUTOSTART 908
 QCBELL 909
 QCCOMMENTS 822, 909
 QCCOMP 805, 833, 909
 QCDIALER 917
 QCDIALERPROPS 917
 QCDUMP 909
 QCEDITALL 909
 QCEDITOR 812, 909
 QCENV 909
 QCERRSLEEP 910
 QECEUROPEAN 600, 824, 910
 QCFLUSH 910
 QCHEAPMAX 910
 QCHOME 910
 QCID 911
 QCIVR 917
 QCKBDELAY 911
 QCLANG 911
 QCLOCALE 911
 QCLOOPLONG 239, 911
 QCNOAPPEND 572, 811, 912
 QCNOCHANGE 572, 912
 QCNOCORR 912
 QCNODDR 884, 912
 QCNODRX 912
 QCNONULL 813, 912
 QCNOPTF 912
 QCNORESTART 912
 QCNOSNAPBACKS 913
 QCNOSTOPCHANGE 913
 QCNULL 913
 QCOLDLOOPS 913
 QCPRINTCMD 917
 QCPROJECTS 913
 QCQUOTATALOG 688, 914
 QCRETAIN 114, 914
 QCSEGUE 914
 QCSET 291, 914
 QCSHARE 804, 832, 893, 914
 QCSHMPATH 438, 914
 QCSLEEP 915
 QCSMSRIGHTS 917
 QCSNAP 915
 QCSNAPCOUNT 915
 QCSTATION 918
 QCSUBSCRIPT 915
 QC TIMEOUT 918
 QC TTINTIME 915
 QC TOUTBUF 915
 QC UPDATE 916
 QC USCORE 916
 QC WHOAMI 805, 916
 QC YNRTN 916
 QC ZEROVARS 916
 QUANCEPT 916
 SMS and QTS 917
 SMVARVALS 918
 TERM 916
 TZ 918
 unset 906
 USER 916
Environment, setting for interviewing 906
Equals sign, function of in script/interview 39
err file, parser error messages 762, 863
ErrNoStopRecord, text to display when no stopped interview exists 851, 926
Error messages
 CATI 826
 change text of during interview 302
 customize 297, 298
 how long displayed 910
 parser 795
 position of on mrInterview page 519
 tag in cascading style sheet 534
 translating in mrInterview 301, 302
ErrorMessage.mdd, standard mrInterview message texts 301
ErrQuotaNotFound, error message when interviewing program unable to locate quota target 677
ErrStopRecordExists, text to display when stopped interview exists 850, 927
eval, expression evaluation and substitution 955
Exclamation mark
 cancel selections from multipunched response list 807
 filter database response list 809
 function of in interview 41
exe file, CAPI interviewing program 863
eximport, open GNU database 475
exp file, intermediate sif file 863
Export interview data to postprocessor 930
Export postprocessor, writing 933
export, update GNU records manually 476
Exporting data
 close postprocessor 932
 define the data to be exported 931
 invoke postprocessor 932
 the export form 932
 writing the postprocessor 933
Expression quotas 738
 defining 748
Expressions
 evaluating and substituting result in texts 955
 logical, mixing operators in 329
External programs, run from within interview 608
Extra questions, add to script 398

F

F_VERB, always display open ends at end of interview 908
file=, quota files to read 679
Filenames, suffixes for quotas 679
Files
 __stop directory 813, 817, 894
 __stop/USED 382, 817, 894
 accounts.sms 898
 acl 862
 act 880
 Activate Document for mrInterview command-line activation 936
 audio directory 898
 auto directory 881
 avi 153
 bitmap directory 882
 bmp 157, 882
 bqt 863
 bt8 863
 com 882
 comments 882
 created by parser 859
 dat 883, 930
 ddr directory 883
 ddr/USED 834, 883
 display text from 193
 drs 887, 930
 drx 890
 dum 571
 environment variables 907
 err 762, 863
 ErrorMessage.mdd 301
 exe 863
 exp 863
 fr1 863
 fr2 865
 gdb 472
 gdbm description 470
 idm 848, 890
 ini 503, 925
 ipk 866
 lib 866
 lst 866
 map 867
 maq 868
 mdd 29, 868
 mdm 869
 mpg 153
 mqd 733, 903
 mqw 869
 mrInterview case data 899
 msg 890
 options for parsing/compiling 792
 psm 869
 ptf 869
 qax 770
 qc_idout.txt 608
 qc8 870

qca 891
qcw.lng 105
qdb 688
qdf 870
qdi 870
qdt 871
qoc 757, 871
qsh 832, 893
qtt 713
qtv 871
quo 713, 722, 872
quz 722, 872
read lines from 194
recordng.dir 835, 893
recordng.qct 835, 893
recpid 835, 893
ref 873
report 425
required for Web interviewing 845
run 770
ser 893
serverlog.sms 899
shm 446
sif 875
snp 915
stp 894
sum 875
tab 770
tbl 894, 943
tex 896, 930
tim 897
tiperrs 898
tx0 876
txx 877
wav 153
xin 877
Filtering response list databases 810
 canceling 809
Find respondent's quota cell 682
fix, insert text in the data file 413
Flowcharts, input file for 862
****, font changes in text 496
****, font specifications for grids 549, 551, 553
<fontColor>, font color type for grids 551
<fontEffect>, special text effects for grids 553
<fontPoint>, text size in grids 549
<fontStyle>, text style in grid template 549
Font changes in text 496
Font for text 487, 496
<fontFace>, typeface for grids 549
Footer, screen footer 927
for, start a loop 238
 number of characters retained in loop 239
force, data for extra questions at end of record 398
Foreground color for interviewing screen 503
Form name of interview page 536
Format of response list databases 438
Formatting text 485
Forward slash, function of in script 39
fr1 file, column allocations for quac and vquac 863

fr2 file, info on precodes for questions allowing open ends 865
 Free Software Foundation 467
freeze, response list common to several questions 117
 rating scales with 117
 with questions in loops 248
 Frozen response lists
 defining 117
 definition of 27
 keywords allowed with 118
 Full stop, function of in script/interview 42

G

getdbvar, read data into script from SMS database 667
getdbvar, read data passed from subsurvey to main script 574
getenv, find value of environment variable 605
getsmvar, copy sample info into script 661
getsmvar, read data passed from main to subsurvey script 574
 Global text formatting commands 490, 498
 Glossary, list of Quancept terms 26
 GNU databases
 add changes to original file 473
 add records to 476
 create 472
 create for qtip 472
 data files, creating original 469
 database description files 470
 defining 468
 duplicate keys in data file 473
 error messages 473
 file types 468
 Free Software Foundation 467
 gdbm format 468
 gdbmfix format 468
 information file for 877
 key fields 468, 473
 NFS drive, possible setupdbm error 472
 open 475
 parsing for qtip 472
 read in record 475
 record formats 468
 update records in 476, 477
 view data in 474
 write data back to 473
go, routing in response list 119, 212
 with *mmpicture* 149
goto, routing outside response lists 213
ifsnap with 226
 skip out of a loop 244
 with *if/else* 219
gotosms, return to SMS menu 227
 effect of in CATI 826
 Greater than sign
 display in text 40
 function of in interview 820
 function of in script/interview 40

<**gridFormat**>, start/end grid layout definition 543, 544
 Grid templates 542
 levels in 543
 precedence between levels in 543, 560
 starting/ending 543
 Grids
 background color 545
 bold text 549
 borders 564
 cell formatting for row/column headings 557
 choosing typefaces for 549
 column widths in 251, 554
 display caption below 543
 display responses in 249
 examples of global characteristics 565
 font specifications for 549, 551, 553
 heading row/column settings 545
 horizontal alignment on page 563
 in callfuncs 254
 italic text 549
 <*mrPageTitle*> with *multitask* 526
 naming in grid template 543
 naming in page template 541
 numeric/real questions in 252
 overall characteristics of 562
 page templates for 541
 position on page 542
 repeat row/column headings 557
 routing in 254
 row height 553
 row/column headings 556
 select responses from in CAPI interviews 841
 set column widths in 251
 set default responses for questions in 253
set in 254
 settings for all cells 544
 settings for alternate rows/columns 544, 558
 settings for the whole grid 544
 size limits 254
 space around cell contents 565
 space between cells 564
 special text effects 553
 specified other in 251
 templates for 541, 542
 text alignment in 547
 text color in 551
 text size in 549
 text wrapping in 547
 width of 563
grouped, column allocation in loops 248
 nested loops with 248
newcard with 403

H

Hash symbol, function of in script 41
headCol, heading column characteristics in grids 545, 556
Header, screen header 503, 927

<headerCell>, heading row/column format in grid 557
<headRow>, heading row characteristics in grids 545, 556
 Heap size, increase 910
 Heapspace, memory allocation for qtip 294
heaptop, memory use by qtip 294
<height>, row height in grids 553
 Help, with appointment setting 825
hideprev, show/hide Previous button 507
hidestop, show/hide Stop button 507
 Highlighted text, displaying 200
hsetcols, number of columns for responses 204
 set column width with 204
 HTML tags
 abbreviating long 494
 bold text 495
 centered text 500
 export Quancept data 931
 font 496
 in response lists 494
 indent response lists 502
 italic text 495
 line breaks 496
 paragraphs 496
 screen design using 493
 start interview 924, 935
 text color 496
 text size 496
 underlined text 495
 using 494
htmlfmt, format open end/numeric input boxes 504

I

<i+>...<i->, italic text 486
<i>...</i>, italic text 495
 Identifying sections in a script 611
if, conditional actions 216
if
 leaving 219
 maximum number of conditions per statement 216
 nesting 219
 with *goto* 219
 without *else* 219
ifsnap
 goto with 226
ifsnap, test for snapbacks 223, 294
ImageMap, name of image file for navigation buttons 927, 929
img src, display pictures in Quancept Web/mrInterview 158, 159
import, read in GNU database record 475
impupdate, update GNU records automatically 477
in, defined sublist as response list 173
 specified other with *not qname in list* 177
include, read a library file 569
 continue with 569
 Include files *See Library files, Subsurveys*
 Incorrect responses

cancel in qtip 807
 text to display for 927
 Indent response lists 502
 Independent quotas
 checking 677
 explanation of 674
 Infinite loops, timing out 246
 ini file, project initialization settings 925, 503
 actions at end of interview 926
 actions for connections timeouts 927
 actions for server crashes 927
 background color 503
 content of export form 932
 create dat file 930
 create tex file 930
 define restart greeting 851
 define variables in 926
 display unique ref code 848, 927
 label for Next button 927
 label for Previous button 927
 label for Stop button 927
 message for invalid answers 927
 message for missing answers 927
 message when respondent uses browser buttons 928
 messages for stopped interviews 850, 851, 926, 927
 navigation buttons 926, 927, 928, 929
 open-end, numeric & real prompt 926
 request Stop button 927, 928
 screen footer 927
 screen header 927
 text color 503
 text for don't know 926
 text for No answer 927
 text for Refused 927
 text for restart interview button 851, 927
 Initialization settings, share between projects 929
 Input boxes for open ends, multiple lines 504
 Input from keyboard, routing dependent on 215
 Integers
 arithmetic using 372
 assign to real questions 364
 column allocation for 85
 definition of 27
 maximum range for 84
 num responses 84
 prompt text for 111, 926
 ranges in response lists 84
 scaling factors for 132
 Interactive Voice Response 624
 Internal error, message to display for 300
 International dialing code, defining 911
 Interview data, use outside Quancept 930
 Interview ID, display 848
 Interview page, form name of 536
 Interview path
 definition of 27, 416
 keep data for questions no longer on 416, 619
 InterviewBuilder questionnaires, keywords in 953, 954, 955

Interviewers

- assign callbacks to specific 826
- define special responses for 215
- name to display in qmonitor 916
- name, entering with qtip 805
- name, for CAPI 767
- name, for CATI 294
- name, qtip login name (intname) 294
- name, setting 916
- system login name, using in script 294
- time zone 918
- user number, using in script 295

Interviewing program

- editing layout *See* Design mode
- files created by 879
- see also* qtip, CATI, CAPI, Web interviews, mrInterview

Interviews

- action at end of 926
- add manually recorded to data file 604
- connection ID 536
- count of number started 893
- ending 379
- identify sections in 611
- monitor in mrInterview 536
- recording keystrokes of 835
- restart stopped but do not start new ones 851
- restart with quotas 675
- save data for interviewer-stopped 387
- stop CAPI 841, 842
- stop with *stop* statement 383
- stopped by the script 812
- stopped, directory for 894
- text to display at end of 186
- trying to restart completed 300

intname, find interview's unique ref code 849

intname, interviewer's name 294

InvalidAnswer, text to display for invalid responses 927

ipk files, files used by CAPI script 866

Italic text 486, 495

Italic text in grids 549

iteration, number of times loop started 237, 261

- as subscript 266
- comparison with subscript 267
- effect of *rot/ran* on 262
- loop iteration count 266
- nested loops 263
- value, use of in scripts 263

IVR

- dial 627
- initial configuration 625
- transfer current call to 633

K

keepstopped, keep data for questions not on interviewing path 619

Keyboard input, routing dependent on 215

Keyboard response entry, allow in CAPI 766

Keystrokes, recording of 835, 893

Keywords

- callable functions 70
- summary of 53
- used with *set* 62

L

Labels

- definition of 27
- length of 43, 79
- naming conventions 79
- rules for 43
- statements that must have 43
- with routing 212

labeltyp, label to use for selecting interview text 307

LANG, define interview language for sorting, punctuation and word wrap 908

Language

- choose during CATI interview 822
- default in multilingual scripts 602
- for CAPI interviews 769
- for mrInterview interviews 304
- scripts in more than one 602, 603

LANGUAGE, default interviewing language 908

language, language for mrInterview interview 303

Layout of screen objects 943–948

Left-aligning elements in design mode 947

Less than sign

- display in text 39
- function of in interview 820
- function of in script/interview 39

Less than sign, function of in interview 820

lib file, intermediate sif file 866

Library files

- definition of 27
- ending 569
- how qtip deals with 570
- including in script 569
- parsing 569

limitresp, variable ranges for *num/real* response lists 92

Line breaks in multilingual scripts 603, 911

Line breaks in text 496

Lines, extract from look-up file 194

List box, display responses in 205

listbox, display responses in a list box 205

listdbm, add changes to GNU database file 473

listshm, list shared memory segments 447

***ln**, choose language for interview 822

Local callfuncs 650

- calling 656
- checking for success/failure 656
- create qtip containing 649
- defining property variables 653
- in restarted interviews 658
- naming 648
- passing information between 656
- using 647
- writing 651

- Lock CAPI interview 841
- Logical expressions
 - .not. with other operators 330
 - check for *null*, *ref* or *dk* 324
 - comparing 327
 - definition of 28
 - introduction to 323
 - missing operators in 329
 - more than two sub-expressions 329
 - multipunched responses 325
 - negating 329
 - reserved words with 327
 - route* 214
 - symbols with 324
- Logical values, assign to variables 331
- logical**, assign logical value to variable 331
- LOGNAME, user name for qmonitor 908
- logsql**, write mrInterview data to log file when SQL connection fails 304
- Long response lists *See* Response list databases
- Long response texts, abbreviating 287
- Long statements, continuing 44
- longmath**, arithmetic with integer texts 372
- longtext**, combine responses with 312
- Look-up files
 - response list databases as 444
 - using in scripts 194
 - See also* Response list databases
- Loops
 - assigning values to repeated questions 268
 - column allocation with 248
 - cumulative totals in 271
 - defined lists with 238
 - defined lists with specified other 240
 - definition of 28
 - effect of *rot/ran* on iteration 262
 - for rating scales 126
 - format of 36
 - format of value list 239
 - frozen response lists in 248
 - go to end of 244
 - iteration number 266
 - iteration value as subscript 266
 - iteration with nested loops 263
 - iterations 237, 261
 - multitask* in 242
 - negative subscription in 350
 - nested, restricted questions in 275
 - nesting 243
 - newcard* in 403
 - number of characters retained from *for* 911
 - number of times questions asked 270
 - number of times repeated 238
 - numeric, subscription with 270
 - protected texts in 192
 - purpose of 237
 - randomize value list 247
 - ranges in value list 240
 - reduce size of data map with 273
 - respondent-specific value lists 277
 - rot* and *ran* with 126
 - rotate value list 247
 - routing with 244
 - select items at random from long list 273
 - snapping back into 820
 - specified other in 267
 - store responses to questions in 264
 - subscripted questions in 245
 - subscription 237
 - subscripts for nested 265, 267
 - subscripts for numeric loops 265
 - subscripts for text 265
 - subscripts for unnested 265
 - substitute values in question texts 239
 - substitute variables in value list 277
 - time since last page was displayed 246
 - timing out infinite 246
 - use of iteration value in scripts 263
 - use pre-v7.8 behavior 913
- Lower case
 - distinction from upper case 42
 - GNU database record keys 468
 - lst file, compilation listing 866

M

- Main script
 - pass data to subsurvey script 574
 - pick up data from subsurvey script 574
- main**, main section of script 50
- makeappt**, make appointment in script 601
- makeqtip**, create version of qtip containing local callfuncs 649
- Manually recorded interviews, add to data file 604
- map file, data format 759, 867
 - See also* Cards, Columns, Data, qpars
- mapothzero**, code specified other as zero 104
 - in secondary scripts (subsurveys) 104
 - in subsurvey scripts 571
- maq file 868
- Maximum responses allowed from *mp* list 120
- maxrecord**, maximum recording time for open ends 155
- maxtime**, maximum display time for text 189
 - with *ask* 190
 - with *multitask* 190
- maxvideo**, play video 152
- mdd file
 - check language codes in 794
- mdd file, questionnaire definition 29, 868
- mdm file, intermediate file for mdd file 869
- Memory, show use by qtip 294
- mention**, order of mention for multipunched answers 340
 - 'argument incorrect' message 342
 - dummy questions with 341
 - points to remember when using 342
- messagebox**, display text in a message box 188
- Messages to supervisor, file containing 882, 890
- messages.qqf file, translating messages 302

mintime, minimum display time for text 189
 with *ask* 190
 with *multitask* 190
Minus sign, function of in script/interview 39
MissingAnswer, text to display when no response selected 927
mmaudio, play sound files 153
mmautoplay, play sound file automatically 154
mmcaption, position of response text relative to picture button 150
mmpicsize, define picture size 151
mmpicture, display a picture 148
mmpicunit, unit of measurement for picture sizing 151
mmvideo, play video clip 152
Modem
 area codes, ignore 920
 autodial numbers 908
 send dial string to 908
 send termination string to 908
Moving objects in design mode 946
mp, multipunched response list 81
 multipunched response lists 82
 with frozen response lists 118
mpg/mpeg files, video clips 153
mqd file
 reusing 754
mqd file, quota definitions 733, 903
mqw file, data map for Quanquest 869
<mrAutoComplete>, switch autocomplete on/off 528
<mrConnectionID>, interview connection ID 536
<mrData /Controls>, position of response text on page 519
<mrData /ErrorField>, position of error messages 519
<mrData /TextField>, position of question text on page 519
<mrData>, position of question & response text on mrInterview page 519
<mrData0>, position of *multitask* text 522
.mrDisplayText, displayed text tag in CSS 534
<mrDisplayText>, position of *display* text on mrInterview page 524
.mrDropdown, dropdown list box tag in CSS 534
.mrEdit, tag for numeric/open-end answer boxes in CSS 534
.mrErrorText, error message tag in CSS 534
mrFORM, form name of interview page 536
<mrGridFormat>, name grid template in page template 541
mrInterview 936
 actions for test interviews only 230
 activating projects 780, 936
 activation
 activate to file 788
 CATI projects 784
 changing the project's status 789
 cluster name 788
 page templates 785
 partial 788
 presets 790
 tasks to carry out 788
 to another site 788
 with Sample Management 784
 associate templates with scripts 530
 bypass *stop* in restarted interviews 384
 case data file 899
 convert activation ini files to xml 939
 creating Activate Documents 788
 customizable error messages 298
 define page title 531
 display standard navigation bar 526
 format for *null/dk/ref* 498
 format for specified other 498
 hide/display Stop button 526
 maximum number of responses per question 84
 message for internal errors 300
 message when browser buttons used 300
 monitor interviews 536
 multilingual scripts 303
 position of error messages on page 519
 quota system 691
 rename navigation buttons 526
 restarting stopped interviews 853
 Sample management
 setting up 784
 set page style 529, 530
 text to display at end of interview 186
 translatable error messages 301, 302
 user-defined functions 650
 variables set by 73
.mrListbox, list box tag in CSS 534
.mrMultiple, *mp* response button tag in CSS 534
.mrMultipleText, multiple response text tag in CSS 534
<mrNavbar>, display standard navigation bar 526
<mrNavbutton>, rename navigation button 526
<mrPageTitle>, display question name on mrInterview page 525
 define page title for 531
.mrProtectText, protected text tag in CSS 534
<mrProtectText>, position of *protect* text on mrInterview page 524
.mrQText, question text tag in CSS 534
.mrSingle, *sp* response button tag in CSS 534
.mSingleText, single response text tag in CSS 534
.mrText, numeric grid cell tag in CSS 534
***msg**, message from interviewer to supervisor 817
msg file, messages 890
msgalrcmp, message when trying to restart a completed interview 300
msgansp, answer prompt 111, 297, 300
msgbrbut, error message when browser buttons used 297, 300
msgcntac, **msgcntag**, **msgcntam**, **msgcntan**, **msgcntar**, message when too many responses chosen from *mp* list 299
msgdk, text for don't know 106
msgend, message for end of interview 300
msgintrn, message when internal error occurs 300
msginvan, message when invalid responses given 297
msglang, current language for message texts 301
msglangd, first language for message texts 301
msgmdidx, element index for message texts 301

- msgmdord**, order in which to search mdd files for message texts 301
- msgmisac**, **msgmisag**, **msgmisam**, **msgmisan**, **msgmisar**, message when no response selected 299
- msgmisan**, message when no responses given 297
- msgnull**, text for no answer 106
- msgnumac**, **msgnumag**, **msgnumam**, **msgnuman**, **msgnumar**, message when non-numeric value entered 299
- msgoth**, text for specified other 106, 300
- msgothal**, replacement text for o specified other 107, 110
- msgotiam**, **msgotian**, message when open-end entered without Other being selected 300
- msgotlam**, **msgotlan**, message when Other specify response is too long 300
- msgotmam**, **msgotman**, message when Other selected but no open-end entered 299
- msgprefx**, prefix for message variables 302
- msgref**, text for refused 106
- msgrngac**, **msgrngag**, **msgrngam**, **msgrngan**, **msgrngar**, message when out-of-range value entered 299
- msgrngor**, separator for ranges in lists displayed by *msgrng* variables 300
- msgrngto**, min/max separator for ranges displayed by *msgrng* variables 300
- msgsgac**, **msgsgag**, **msgsgam**, **msgsgan**, **msgsgar**, message when more than one answer chosen from *sp* list 299
- msgsolac**, **msgsolag**, **msgsolam**, **msgsolan**, **msgsolar**, message when *sp* chosen with *mp* answers 299
- msgtxlam**, **msgtxlan**, message when open-end response is too long 300
- multitask**, display more than question on screen 95
 an alternative to using 500
 enter in CAPI interviews 840
 in loops 242
mintime and *maxtime* with 190
 $<mr\ PageTitle>$ with 526
 position on mrInterview page 522
 $<table>...</table>$ with 500
 width of open-end text boxes with 505
- Multicoded responses *See Multipunched responses*
- Multilingual scripts
 alphabetic sorting of responses 115
 assigning categorical values to temporary variables 304
 changing after translation 304
 check language codes in mdd file 794
 choose language in qtip 822
 default language 602
 for mrInterview 303
 writing 602, 603
- Multiple scripts, using 569
- Multipunched quotas 676
- Multipunched responses
 all chosen from given list 338
 assign by response number 288
 assign by response text 285
 at least one chosen from given list 337
 automatic open ends for 107, 109
- cancel incorrect selections 807
 check for a specific response 325
 check for responses after a given one 325
 check for responses before given one 325
 check which chosen 337, 342
 columns allocated to 82, 83
 control editing of, QCNCORR 912
 define 81
 defined lists 165
 displaying 197
 enter in CAPI interviews 839
 entering in qtip 806
 logical expressions with 325
 maximum that may be chosen from list 120
 maximum to be selected 120
 $mp\ n$, effect on qtip 807
 net with set 289
 number chosen from list 344
 number in list 344
 only one chosen from given list 339
 order of mention 340
 position in list 342
 read response text from list 345
 rotation/randomization of 122
 routing within response list 119
 save rotation number in data 125
 single punched responses in *mp* list 807
 symbols for checking 325
- Multipunched, definition of 28

N

- <name>**, grid name in grid template 543
- Navigation buttons
 define your own 929
 display standard in mrInterview 526
 display Stop button 928
 displaying 928
 labels for 928
 positions in user-defined bar 929
 rename in mrInterview 526
- nbit**, check single punched responses 335
 how differs from bit 343
 with *null*, *dk* or *ref* 336
 with specified other 336
- Negating logical expressions 329
- Negative numbers in arithmetic statements 362
- Negative subscription 349
 loops in 350
- Nested grouped loops 248
- Nested loops 243
 restricted questions in 275
- Nesting
 definition of 28
 $if\ and\ else$ 219
- Netting
 definition of 28
 responses 289
- New interviews, always start in CAPI 766

New lines in text 46
 character to represent 47

New paragraphs in text 496

New projects, how to create 905

newcard, start a new card 402
 effect of *grouped* on 403

newline, define newline character 47

newran, begin subgroup in randomized block 221

newrot, begin subgroup in rotated block 221

Next button, label for 927, 928

Next question, move to next in CAPI interviews 841

next, end a loop 238

NextButton, label for Next button 927, 928

NFS drive, GNU databases on 472

No answer

- coding in qtip 813
- enter in CAPI interviews 840
- how to allow for 99
- text to display for 105, 106, 927
- See also* null

noaneql treat unanswered questions as null response
 questions 354, 355

No-answer calls, with qtip 824

noblank, suppress blank cards 420
 effect on qccon of using 420

noclean, save all data 416

nodata, do not save data for this question 121

nolimoth, *not* does not apply to *other* in defined lists 308

noprompt, suppress Answer prompt for numeric/open-end input boxes 504, 506

.not., negate a logical expression 329
 with other operators 329

Not equal to sign, function of in script/interview 40

not, responses excluded from defined sublists 175

***note**, forced open-end 814

notemp, do not allow temporary variables in scripts 311

<nowrap>, text wrapping switch for grids 547

now, appointment relative to current time 825

nselrec, select record from response list database

nsubstr, extract characters, convert to number 586

null, no answer response 99

- advantages of using 100
- automatically allow in numeric grids 297
- check for 324
- check whether chosen 350
- code as blank 912
- coding 82, 100
- effect of QCNONULL on 100
- formatting in mrInterview 498
- no answer with qtip 813
- press Return for 813, 913
- routing within response list 120
- select when no response chosen 296
- text to display for 105, 106
- with *bit* 343
- with defined lists 168, 179
- with *elm* 345
- with *nbit* 336
- with *numb* 344
- with *numv* 344
- with quotas 676

with *set* 292

Null, testing for 354

null, text to display for no answer 927

num, integer responses 84
 prompt text for 111, 926

numb, number of responses chosen 344
 with *null* or *dk* 344
 with specified other 344

Number symbol *see* Hash symbol

Numbers

- arithmetic with text 370
- convert from text to integer 372
- convert integers to reals 369
- convert reals to integers 368
- displaying 197

Numeric conversions, writing in script 132

Numeric grids, allow blank cells in 297

Numeric questions in grids 252

Numeric quotas 676

Numeric responses

- arithmetic with 361
- checking 327
- enter in CAPI interviews 839
- entering in qtip 807
- more than one range with 85
- negative numbers as 84
- quotas for 738
- ranges with 84
- routing with ranges 212

See also Integer responses, Real responses 84

Numeric responses, format input boxes for 506

numv, number of responses in response list 344
 with *null*, *dk* or *ref* 344
 with specified other 344

O

^o, automatic open end for an *sp/mp* response 109
 display responses given with 347
 replacement text for 107, 110

Objects

- moving 946
- resizing 946
- selecting in design mode 945

ODBC databases

- add new records to 460
- connect to 454
- disconnect from 455
- extract single record from selection 457
- general notes on working with 463
- make public 454
- multiple record insertions per interview 460, 461
- number of script replays after error 461
- query 455
- select field from record 458
- select records from 456
- set SQL cursor type 461
- update records in 458

ODBC databases *contd.*
 using 453

ofmt, format for specified other 498

ofpathem, make off-path questions appear empty 308

ofpathmd, clear data from off-path questions 309

onresp, routing dependent on keyboard input 215
 effect of in interview 813
 jumping with in CAPI interviews 841
 position of in script 216

Open ends
 add to data file during interview 415
 append to at end of interview 811
 appending/changing at end of interview 909
 associated with specified other 347
 automatic with *^o* 109
 automatic with *promptoth* 107
 check format of 350
 clear scribble box 157
 columns for coding 896
 concatenating 583
 definition of 28
 disable recording of 622
 display at end of interview 908
 displaying 197
 don't store answer or allocate columns 121
 edit during interview 352
 enable recording of 622
 enter at end of interview 811
 enter in CAPI interviews 839
 entering in qtip 808
 extract characters and convert to a number 586, 587
 files of recorded 898
 forcing with *note 814
 how to code 90
 input boxes for 504, 506
 maximum recording time 155
 nodata with 121
 number of codes required 89
 number stored in tex file 893
 overwrite at end of interview 811
 print with *rprint* 432
 prompt text for 926
 record as sound files 161
 reserve columns for 89
 resp coded 88
 save as bitmap pictures 157
 size of text box 504
 specified other 103
 store text of 896
 store without coding 121
 store without saving columns in data file 89
 type in 808
 using without saving 89, 90
 where stored 90
 width of text boxes with *multitask* 505
 write down 808

Options
 parsing/compiling in QCompile 763
 script testing 765
 special 767

.or., at least one expression true 327

or, at least one response required 337

Order of mention for multipunched responses 340

Other
 coding 103
 coding of in defined lists 172
 edit response text 352
 quoted text in response list 102
 specified other 103
 See also Specified other

other, specified other 103
 for individual responses 107
 formatting in mrInterview 498
 prompt text for 111
 text to display for 106

OtherData table 903

othertext, display with *prompth*/*^o* 347

P

<**p**>, new paragraph 496

Page layout using templates 516

Page style, set for mrInterview 529, 530

Page templates, choosing for questions 528

Page titles
 for mrInterview 531
 for Web interviews 928
 with *multitask* 526

pagstyle, page style for mrInterview questionnaire 529

Parentheses
 function of in script/interview 40
 in *sp* response lists 119, 213

Parse, definition of 28

Parser error messages 795

Parser error messages file 863

Parsing
 library files 569
 subsurvey scripts 571

partofday, part of day 295

pass=, quota full variable 680, 703

Pattern matching in multilingual scripts 603, 911

pause, wait until interviewer presses a key 186

Pen Windows options 768

Pending counts 691

Percent sign, function of in script 41
 print with *rprint* 429

pgstyle, associate template with script 530

ptitle, display question name/page title on interview page 525

ptitle, page title for mrInterview 531

Phone numbers, enter during interview 826

Pictures
 background for interview screen 491
 define size of 151
 display 187
 display in mrInterview 159
 display in Quancept Web 158
 displaying with *mmpicture* 148
 in questions and response texts 158, 159
 layout for screen background image 491

Pictures *contd.*

- save responses as 157
- storing scaled with QCompile 768
- valid formats for 158, 159
- Play sound files 153, 160
- Plus sign, function of in script/interview 39
- Point size of text 487
- Pound sign
 - display in text 42
 - function of in script/interview 42
- Precedence, order of in logical expressions 329
- Premature termination of interview
 - with data 816
 - without data 816
- prepareivr**, dial an IVR system 627
- PrevButton**, label for Previous button 927, 928
- Previous button
 - label for 927, 928
 - show/hide in Web interviews 507
- Previous question, move to in CAPI interviews 841
- Private status codes
 - defining 419
 - where stored 881
- privsig**, customized status code 419
 - where codes are stored 881
- Product test, sample script for 289
- Programs
 - activate 780, 936
 - ActivateIniToXml 939
 - ChangeQuota 755
 - chekpars 758
 - conv8 758
 - listdbm 473
 - listshm 447
 - makeqtip 649
 - qcformat 884
 - qcparse 758
 - qtip 804
 - quac 863
 - quota 713
 - Quota Setup 729
 - quotedit 722
 - rmshm 448
 - run external during interview 608
 - setupdbm 472
 - shareqoc 832, 893
 - shm 446
- Project directory
 - creating 905
 - definition of 28
- Projects
 - backing up and deleting 939
 - make available on Web server 924, 935
 - selecting in QCompile 761
 - sharing quotas between 700
- Prompt, suppress for numeric/open end input boxes 504, 506
- promptoth**, automatic open end for an *sp/mp* response 107
 - disallow specified other with 109, 110
 - display responses given with 347
 - in defined lists 171

Property collections

- creating multiple 654
- naming 653
- storing same data in more than one 655

Property variables

- converting to temporary variables 655
- defining 653, 654

protect, display text until explicitly removed 190

- global format for text 490, 498

- in loops 192

- maximum amount of text 191

- maximum per page 308

- position of on mrInterview page 524

- tag in cascading style sheet 534

- variable values with 191, 197

See also xprotect, unprotect

psm file, text for CAPI(DOS) 869

ptf file, project text file 869

- create with different name 758

Punctuation characters in multilingual scripts 603, 911

putdbvar, pass data from subsurvey to main script 574

putdbvar, write data into SMS database from script 668

putsmvar, pass data from main to subsurvey script 574

putsmvar, update sample record during interview 662

Q

qax file, Quantum axes specs 770

qc_hangup, hang up call from within script 623

qc_idout.txt, output from *varlist* 608

qc8 file, 8-bit script conversion 870

qca file, qtip accounting information 891

QCAMPNAMES, variable text substitution with

- &varname* 908

QCANOTH, pause automatic issuing of records from SMS 917

QCASKNUMBER, prompt for record key to call 917

QCAUTODIAL, dialing via a modem 908

QCAUTOEND, termination string for modem dialing 908

- modem termination string 908

QCAUTOSTART, dialing string for modem dialing 908

QCBELL, audible/visual warning for interviewer mistakes 909

QCComments, number of comments to display with
**comment* 822, 909

QCComp, review completed interviews 805, 833, 909

qccon, *noblank* with 420

QCCTTOUTBUF, terminal output buffer size 915

QCDIALER, station number 917

QCDIALERPROPS, autodialer properties 917

QCDUMP, dump core when interview stopped 909

QCEDITALL, edit open-ends at end of interview 909

QCEDITOR, editor for appending/entering verbatims at
 end of interview 812, 909

QCENV, set interviewing environment 907, 909

QCERRSLEEP, how long to display error messages 910

QEUEUROPEAN, European date format 600, 824, 910

QCFLUSH, disallow type-ahead 910

qcformat, list of question names for 870

QCHEAPMAX, increase heap size 910
QCHOME, Quancept home directory 910
QCID, report response code & value 911
QCIVR, IVR configuration properties 917
QCKBDELAY, time to wait for completion of incomplete escape sequences 911
QCLANG, define interview language for sorting, punctuation and word wrap 911
QCLOCALE, international dialing code 911
QCLOOPLONG, extend number of characters retained in loop value lists 239, 911
QCNOAPPEND, suppress append verbatims prompt 572, 811, 912
QCNOCHANGE, suppress inspect/change interview prompt 572, 912
 suppress inspect/change interview prompt 572
QCNOCORR, control editing of multipunched responses 912
QCNODDR, suppress ddr directory 884, 912
QCNODRX, do not create a drs file 912
QCNONULL, code null as blank 813, 912
QCNOPTF, read translations from tx0 file rather than ptf file 912
QCNORESTART, don't restart stopped interviews 805, 912
QCNOTSTOPCHANGE, do not inspect/change anything for stopped interviews 913
QCNULL, press Return for null response 913
QCOLDLOOPS, use pre-v7.8 loop behavior 913
QCompile
 check language codes in mdd file 794
 files created by 859
 functions of 778
 main menu overview 760
 options when parsing/compiling 763, 764, 792
 overview of functions 759
 Pen Windows options 768
 starting 760, 779
 starting Quota Setup from 791
 types of compilation 763
QCompile, parse CAPI/Web/mrInterview scripts 759
qcparse, check CATI script syntax 758
QCPRINTCMD, command for printed output 917
qcproject, get project name in script 295
QCPROJECTS, projects file location 913
QCQUOTATALOG, create quota log files 688, 914
QCRETAIN, keep original response codes when re-ordering responses 914
 use of with *atoz* 114
QCSEGUE, suppress End of subsurvey message 914
QCSERVERFAILURE, text to display when server crashes 927
QCSET, behavior for *sp/mp/list* assignments 291, 914
QCSHARE, look for shared qoc file 804, 832, 893, 914
QCSHMPATH, search path for databases 438, 914
QCSLEEP, response delay in automatic test interviews 915
QCSMSRIGHTS, supervisory permissions 917
QCSNAP, write snapback debugging information to.snp file 915
QCSNAPCOUNT, loop limit when snapping 915
QCSTATION, station number 918
QCSUBSCRIPT, use pre-v7.8 subscription behavior 915
 example of setting to 0 269
QC TIMEOUT, time-out for contacting SMS server 918
QC TTINTIME, interbyte delay for terminal input 915
QCUPDATE, how to deal with qoc changes 916
QCUSCORE, print line between question and response list 916
QCWHOAMI, interviewer name 805, 916
QCYNRTN, press return after y/n responses 916
QCZEROVARS, reset temporary variables to zero when snapping 916
qdb file, quota log 688
qdf file, question names 870
qli file, data map 870
 copy code frames between 775
 update after reparsing 771
qdt file, question & response texts and names of callfuncs used 871
qfmt, question text global format 498
qmonitor
 user name for 908
qmonitor, user name to display in 916
qname, display name of current question 192
qnameful, full question name 192, 532
qoc file, compiled script for CATI interviewing 871
 creating from script 757
 handling changes to during interviews 916
look for shared 914
reduce size of for CAPI(DOS) 193
shared memory address of 832
sharing 832, 893
qparse, parse CATI scripts 757
 avoid mapping conflicts with *col* and *column* 406
-d flag 758
 debugging facilities 758
files created by 859
-m option 868
 use predefined map file 757, 759
-y flag 758
qsh file, memory address of shared qoc file 832, 893
qtip, CATI interviewing program 804
 case sensitivity 42
 environment variables for 906
 summary of special responses 73
 version containing local callfuncs 649
 See also CATI
qtt file, CATI quota texts 713
qtv file, numbers for temporary variables 871
Quancept home directory, define for CATI 910
QUANCEPT, terminal type for interviewing 916
Quantum specs, creating 770
querysms, send request to SMS server 664
Question mark, function of in interview 41
Question names
 display in browser's title bar 526
 in loops 264
 length of 79
 listing in CATI 818
 order of with *quota* 678
 position on mrInterview page 525

Question names *contd.*
subscription of in loops 264
with defined lists 175

Question text, tag in cascading style sheet 534

Question variables
assign arithmetic expressions to 364
definition of 30
save arithmetic expressions in 366

Questionnaire definition file 29, 868
check language codes in 794

Questionnaire file *See* Script 905

Questions
add extra to script 398
allow respondents to not answer 296
allowin respondents to not to answer 297
ask without saving data 121
assign responses from other questions to 289
assign responses to 284
automatic jumps to next 138
backtracking to previous 820
choosing page templates for 528
clearing data from off-path 308, 309
column allocation in loops 248
common response lists 117
common to many scripts 569
concatenating variables in 284
conditional 216
data for extra at end of record 398
difference between unanswered and null 354
display more than one per screen 500
display name of current in CAPI & Web interviews 192
display responses to snappable 819
displaying more than one on screen 95
dummy 26, 91
dummy, example of use 274, 287, 289, 290, 341, 584, 683, 684, 685
empty, testing for 354, 355
format of 31, 79
full name including iteration number 192, 532
global text format 490, 498
in grids 249
keep data for question removed from path 619
keywords for 95
maximum display time for 189
minimum display time for 189
naming 79
number of times asked in loops 270
numeric/real in grids 252
order of in data map 868
pictures in 158, 159
position on mrInterview page 519
print line between question and response list 916
randomize groups of 220
repeated, assigning values to in loops 268
repetitive 237
restricted in nested loops 275
rotate groups of 220
rotate/randomize 221
single quotes with 32

subscripting in loops 245
subscription of 264
substitute loop values in 239
tabular layout of 500
testing whether unanswered 354
unanswered, testing for 354, 355
variable text in 284
See also Text

quit, premature termination with data 816
save data after 389

quitdata, always save data for quit/abandoned interviews 389
effect of in qtip 816
effect of signal on 386

quo file, CATI quota counts 713
quo file, Web quota information 722, 872

quorat, check quota ratios 685

quota, check/decrement quotas 677
See also Quotas

Quota control
definition of 29
setting up during activation 783

Quota files, reusing in different projects 754

Quota Setup, mrInterview quota definition program 729
choosing working language 735
creating new files 733
Details Pane, docking/undocking 731
Details pane, moving 730
Details pane, resizing 730
displaying numeric/text variables in List pane 731
displaying sample variables in List pane 732
displaying system variables in List pane 731
hiding components of Quota Setup window 733
List pane, changing layout of 731
List Pane, docking/undocking 731
List pane, moving 730
List pane, resizing 730
opening existing files 733
quotas using sample variables 732
returning to default window layout 733
showing/hiding Details pane 733
showing/hiding List pane 733
showing/hiding status bar 733
showing/hiding toolbar 733
starting 730
starting from QCompile 791
undoing/redoing actions 734
window 730
See also Quotas

quota, quota maintenance program 713

quotacell, check quota cells 682

quotactn *See* quoval

quotamap, take quota snapshot 682

Quotas
adding tables 738
And operator in expressions 752
AnswerCount function 752
arithmetic operators in expressions 753
based on sample data 696
categorical 694
cell, find respondent's 682

Quotas *contd.*

change cell text/target 717
 change count of completes after testing 721
 change counts 727
 change individual 718
 change targets and behavior flags 755
 check 725, 755
 check for unfilled cells 682, 685
 check in script 677
 check ratios between cells 685
 checking for missing specifications 704
 combine cells 719
 comparison operators in expressions 749
 completed counts 691
 ContainsAll function 751
 ContainsAny function 750
 counter quotas 695
 create log files 688, 914
 decrement with stopped interviews 383
 define targets 723
 defining expression 746
 defining quota texts 716
 defining table quotas 736
 defining targets 717
 delete counts 728
 delete targets 728
 deleting expression 754
 deleting tables 738
 dependent, checking 678
 display cell target during interview 687
 display number of completes during interview 687
 display number of respondents rejected from full cells 687
 display values used in combining cells 688
 dummy questions with 683, 684, 685
 editing expression 754
 effect of snapbacks on 677
 errors during interviews 677
 expression 694, 738
 expression, advanced features 748
 filename extensions 679
 Filter On dialog box 740, 741, 743, 744, 745
 for combinations of responses 694
 for combined responses 681
 for restarted stopped interviews 675, 695
 for reviewed interviews 675
 functions in expressions 750
 how they work 673
 in CATI 714
 in mrInterview 691, 729
 in Quancept Web 722
 independent, checking 677
 inspecting 718
 intercept termination message 680, 703
 link between responses and quotas 713
 link between statement and files 674
 logical operators in expressions 752
 more than one per script 679
 multipunched 676, 694
 Not operator in expressions 753
null, dk and ref with 676

numeric 676, 694
 operator precedence in expressions 754
 Or operator in expressions 752
 order of question names with 678
 over quota interviews 695
 pending counts 691
 polling for up-to-the-minute counts 727
 prioritization of multipunched 698
 projects in DimensionNet 705
 quota files 713
 quota program 713
 quotedit program 722
 quoval 687
 remove hung temp entries 721
 request status information in script 687
 restart stopped interviews with 675, 695
 reusing mqd files in different projects 754
 review completed interviews 675
 rollback counts 692
 setting up 715
 sharing quota files between projects 700
 single punched 675, 694
 stopped interviews 695
 table 735
 targets 717
 termination of interviews by 812
 text 694
 time when cell became full 688
 timed out interviews 695
 types of 673
 when decremented 674
 with reviewed interviews 834
 with *stop* 674

quotaval *See quoval*

quotedit, Web quota maintenance program 722
quotetxt, quota check error message 704
quotrval, quota check return value 704
quoval, request status information 687
 quiz file, Web quota information 722, 872

R**ran**, randomize responses 125

effect on loop iteration 262
 in stopped interviews 384
 response groups with 128
 with loops 126, 247
 Randomization, effect on loop iteration value 262
 Randomize questions 220, 221
 Randomized lists, fixed position responses in 127
ranend, stop randomizing questions 220, 221
 statements valid with 220
Ranges
 in loop value lists 240
 in numeric response lists 84
 routing with 212
 separate in response list 85
 variable in num/real response lists 92

- ranstart**, start randomizing questions 220, 221
statements valid with 220
- Rating scales**
writing with *freeze* 117
writing with loops 126
- Ratios**, check for quota cells 685
- Read a look-up file** 194
- readfile**, read lines from a file 194
- real**, real (decimal number) responses 86
prompt text for 111, 926
- Real numbers**
accuracy with 87
arithmetic using 369
as responses 86
assign to integer questions 369
definition of 29
force rounding with *setinteger* 369
print in report files 431
quotas for *See Numeric quotas* 676
- Real questions in grids** 252
- Real responses**
decimal places for 86
prompt text for 111, 926
ranges in *resp real* list 86
scaling factors for 132
- realmath**, real arithmetic 371
- Record number**, save for response list database records 442
- Record/stop recording responses in CAPI interviews** 841
- Recording keystrokes of interviews** 835
directory for recorded keystrokes 893
flag for 893
replay keystrokes 834
stop recording keystrokes 835
- recording.dir**, directory for recorded interviews 835, 893
- recording.qct** file, record keystrokes of interviews 835, 893
- Records**
select fields from 443
select from response list databases 444
- recpid**, recorded interviews for one interviewer 835, 893
- redial**, generate dialing instruction from script 624
- Redrawing the screen** 818
- ref**, refused to answer 101
check for 324
check whether chosen 350
coding 102
formatting in mrInterview 498
refused with qtip 813
routing within response list 120
text to display for 105, 106
text to display for refused 927
with *bit* 343
with defined lists 168, 179
with *elm* 345
with *nbit* 336
with *numb* 344
with *numv* 344
with quotas 676
with *set* 292
- ref file**, information for quac 873
- Refusals**
coding in qtip 813
enter in CAPI interviews 840
how to allow for 101
text to display for 105, 106, 927
- regn**, redraw screen 818
- remove**, remove response from multipunched selection 310
- Rename navigation buttons in mrInterview** 526
- <repeat>**, repeat row/column headings 557
- Repeated questions** *See Loops*
- Replay keystrokes of recorded interviews** 834
- Report files**
closing 426
conversion characters with 428
defining format when printing to 428
newline in 428
open 426
print integers in 429
print percent sign in 429
print real numbers in 431
print single-punched responses 431
writing to 427
- Requests**, pass to SMS server from script 664
- Resizing objects in design mode** 946
- resp coded**, open-ended response list 88
- resp dbase**, display response list database 439
- resp list**, find which defined sublist to use 178
- resp mp**, multipunched response lists 82
- resp multidbase**, multiselection response list databases in CAPI 441
- resp num**, integer responses 84
- resp real**, decimal number responses 86
- resp sp**, single punched response list 81
- resp**, responses in a response list 81
- Respondent IDs**, reusing in Web interviews 933
- respondent**, respondent ID of current respondent 295
- Respondents**
arrange appointment with 824
increase number of columns for serial number 407
number contacted 893
screening 228
serial number for subsurvey data 571
serial number, set inside script 604
texts for specific, in loop value lists 277
- Response code & value**, make qtip report 911
- Response list databases**
blank fields in 438, 445
cancel filtering 809
code answers in data file 439
code using codes in records 443
columns allocated in data 440
cross-references to a second response list database 444
display with *resp dbase* 439
filtering 810
format of 438
how displayed during interview 440
keys to look-up files 443
location of 439
maximum per script 441

- Response list databases *contd.*
 - multiple selections from same file 440
 - multiselection in CAPI 441
 - names of those in shared memory 446
 - numbering of records on screen 440
 - remove from shared memory 448
 - save number of record selected 442
 - scrolling through 809, 810
 - search path for 438, 914
 - select fields from records 443
 - select from 809
 - select records from 444
 - select responses from 810
 - shared memory, address of 446
 - store response chosen from 439
- Response lists
 - abbreviate long texts 287
 - cancel frozen 117
 - define a defined list 165
 - display other in 809
 - HTML tags in 494
 - integer (*num*) 84
 - long, how dealt with by qtip 814
 - maximum number of responses in 83, 84
 - multipunched 81, 82
 - number of display columns 201
 - other in 102
 - position on mrInterview page 519
 - random selections from 273
 - ranges in integer 84
 - real 86
 - routing in 119, 212
 - scrolled 814
 - scrolled, defining in script 201
 - separate responses in 85
 - single-punched 81
 - single-punched responses in multipunched 130
 - substitute variables in 198
 - text formatting in 485
 - text substitution in 198
- Response patterns, variance according to position in list 124
- Response substitution, in defined lists 199
- Responses
 - alter with snapbacks 821
 - assign with *set* 284
 - automatic null selection when no value chosen/entered 296
 - automatic open ends for sp/mp 107
 - automatic open ends for *sp/mp* 109
 - blank in defined lists 181
 - cancel in CAPI interviews 841
 - cancel incorrect multipunches in CATI 807
 - change previous 821
 - check by position in list 342
 - check previous 818
 - check with logical expressions 323
 - code *null* as blank 912
 - code specified other as zero 104
 - coded 88
 - coded, definition of 28
 - coding of in defined lists 166
 - coding other in defined lists 172
 - combine for quotas 681
 - combine to form multipunch 289
 - concatenating 583
 - database, how coded in data file 439
 - define special for interviewers 215
 - defined lists 165
 - display during interview 196
 - display for snappable questions 819
 - display in alphabetical order 113
 - display in columns 202
 - display in drop-down list 205
 - displaying pictures as 148
 - dk*, don't know 101
 - don't save in data file 121
 - echo during interview 117
 - edit open ends 352
 - edit specified other 352
 - effects of changing during interview 416
 - entering in qtip 805
 - entering *null* with qtip 913
 - erase all in CAPI interviews 841
 - find text by response position 345
 - find which sublist to use 178
 - fixing position in a list 127
 - format of 31, 80
 - formatting commands with 485
 - global format for text 490
 - global text format 498
 - grouping in a *rot/ran/atoz* list 128
 - HTML tags in 494
 - in grids 249
 - indent 502
 - inspect at end of interview 811
 - integer, columns allocated to 85
 - map exact card and column for multiple responses 405
 - map exact card and column for one 404
 - maximum allowed from *mp* list 120
 - maximum number per response list 83, 84
 - maximum range for integer 84
 - message when invalid chosen 297
 - message when none chosen 297
 - multipunched 82
 - multipunched, definition of 28
 - multipunched, entering in qtip 806
 - netting 289
 - non-numeric, checking 335
 - not defined in a response list 215
 - null*, no answer 99
 - number chosen 344
 - number in list 344
 - numeric, entering in qtip 807
 - open end, definition of 28
 - open ended 88
 - checking 350
 - entering in CATI 808, 811
 - order of in defined lists 166
 - order of in sublists 166
 - order of mention for multipunched 340

Responses *contd.*

position on mrInterview page 519
position relative to picture buttons 150
print line between question and response list 916
questions in loops 264
random presentation of 125
real numbers 86
ref, refusal 101
remove from multipunched selection 310
response list databases 437
restored default data mapping after *col* or *column* 405
rotate presentation of 123
rotated/randomized, how coded 122
rotation of more than one list 125
scaling factors for numeric 132
select at random from long list 273
select from a database 809, 810
set defaults for grids 253
set width of display columns 204
single quotes with 32
single-punched 81
 definition of 29
 entering in qtip 806
 in multipunched list 130, 807
slide bars for integer 136
specifying column number for responses 202
suppress blank cards 420
tag for dropdown list boxes in cascading style sheet 534
tag for list boxes in cascading style sheet 534
tag for *mp* responses in cascading style sheet 534
tag for numeric grid cells in cascading style sheet 534
tag for numeric/open-end answer boxes in cascading style sheet 534
tag for *sp* responses in cascading style sheet 534
text to display for invalid 927
text to display when none chosen 927
text wrapping when long 80
text, checking 350
types of 81
unique IDs for 140
variable ranges in numeric 92
verbatim, definition of 28
whole numbers 84

Responses table 902
Restart CAPI interviews 842
Restart interview button, label for 927
Restart Web interviews 846
Restarted interviews
 directory for 894
 quotas in 675, 695
 the USED directory 382

RestartGreeting, text to display when restarting an interview 852, 927
Restarting stopped interviews 384
RestartSubmitButton, text for Restart button 852, 927
Result codes, prompt for information 826
Return, whether to press after y/n answers 916
Review completed interviews 833
 quotas in 675
Reviewed interviews directory 883

rfclose, close report file 426
rfmt, response text global format 498
rlopen, open report file 426
rfprint, write to report file 427
 integers with 429
 open ends with 432
 real numbers with 431
 response list database records with 432
 single punched responses with 431
RGB values for color, using 492, 497
rmshm, remove response list database from shared memory 448
Rollback counts 692
rot, rotate responses 123
 effect on loop iteration 262
 in stopped interviews 384
 response groups with 128
 with loops 126, 247
Rotate groups of questions 220
Rotate items in loop value list 247
Rotate questions 220, 221
Rotated lists, fixed position responses in 127
Rotation
 effect on loop iteration value 262
 group responses together in a *rot/ran/atoz* list 128
 save pattern number 124
 test response patterns with 124
rotation, save rotation number 124
rotend, stop rotating questions 220
 statements valid with 220
rotnwseed, regenerate rotation seed for every rotated response list 124
rottranfix, fix response position in *rot/ran/atoz* lists 127
 in defined lists 171
 with *rotransub* 129
rotransub, response groups with *rot/ran/atoz* 128
 in defined lists 171
 with *rottranfix* 129
rotstart, start rotating questions 220
 statements valid with 220
Rounding real values 369
route, conditional routing 214
 effects of snapbacks on 416
 logical expressions with 323
 skip out of a loop 244
Routing
 conditional 34
 definition of 29
 for *null*, *dk* and *ref* 120
 function of 33
 go to end of loop 244
 goto 213
 groups of *sp* responses 119
 if and *else* 216
 in grids 254
 in response lists 119, 212
 input from keyboard, dependent on 215
 mandatory 33
 multipunched responses 337
 onresp 215
 outside response lists 213

Routing *contd.*

- position of *onresp* in script 216
- route* 214
- to SMS menu 227
- variables, causing unexpected results with 211, 216
- with loops 244
- with *protect* 191
- with rotated/randomized questions 220
- with snappable questions 821
- with *testingrun* 229

Row headings, repeating in grids 557

Row height in grids 553

rowgrid, display a loop as a grid 249
<mrPageTitle> with *multiask* 526

Rows, wrap heading texts in grids 547

rplaycnt, number of script replays after an ODBC error 461

run file, Quantum run file 770

Runtime errors, testing scripts with 834

S

^s, single-punched response in multipunched list 131
 effect of in interview 807
 in defined lists 171

sampbits, sample bits for sound recordings 156

Sample data, quotas based on 696

Sample records

- calling in qtip 823
- copy information into the script 661
- format of 660
- store record keys in data file 661
- update during interview 662

Sample variables in drs/ddr files 889

samprate, sample rate for sound recordings 156

Saving response data 617

Scaled pictures, storing 768

Scaling factors

- applying 132
- define for numeric responses 132
- defining 132

scodscheme, change the data format for single-punched responses 408

Screen background in CAPI 491

Screen corruption, removing 818

Screen design with HTML tags 493

Screen footer 927

Screen header 927

Screen layout and formatting *See* Text formatting, Templates 37

scrneer, start of screener section 50

Screening respondents 228

scribble, write an open end on screen 157

- clear scribble box 157
- enter responses for in CAPI interviews 841
- picture files created by 882

Script

- associate templates with 530
- automatic testing in CAPI 768, 843

- automatic testing in CATI 831
- call subsurvey 570
- comments in 48
- compiled version of 871
- components of 31
- definition of 29
- design CAPI screen layout 943
- dial numbers from 624
- disable recording of sound files 622
- enable recording of sound files 622
- end of 50
- errors in QCompile 760
- formatting text in 485, 493
- global settings for Web 926
- identify sections in 611
- installing on Web server 845
- keywords in 53
- load library file into 569
- multilingual 602, 603
- multilingual, choose language in qtip 822
- multiple, using 569
- naming 905
- options for testing in CAPI 765
- parse in QCompile 761
- parse with qparsc 757
- reparsing once coding has started 771
- running in design mode 944
- sections in 50
- test CATI 803
- testing 803
- testing CAPI 837
- Web, running 845
- written using DOS editor, parse 765

Script name

- accessing via the script 295
- length of 905

Scripts

- naming 52
- testing mrInterview 853

Scrolled response lists

- requesting from script 201
- using 814

Scrolling in CAPI interviews 841

scrtmout, tikme-out delay for infinite loops 246

scrtmpas, time since last page was displayed 246

[+*se+], end highlighted text 200

sect_ids array 612

sect_times array 612

Sections in scripts 50

- identifying 611
- timing 611

Selecting objects in design mode 945

selfid, select field from database record 443, 458

selrec, select single record from database 457

- select record from database 444

See also nselrec

Semicolon

- function of in interview 41
- request serial number for handwritten open end 808

ser file, serial numbers 893

Serial number
increase number of columns for 407
moving columns in data map 759
request for open end 808
set inside script 604
subsurvey data 571

Serial number file 884

serialcols, number of columns for serial number 407

Server manager
connect to a server 924
set project time-out delay 924
start 923

serverlog.sms, information passed between SMS & qtip 899

set, assign a value to a variable 282
effect of snapbacks on 417
ignore when parsing 868
in grids 254
summary of keywords used with 62
with multipunched responses 289
with *null* and *dk* 292

Set environment variables 906

setcols, number of columns for responses 202

setdata, save data for uncompleted interviews 389
effect of in qtip 813, 816
effect of *signal* on 386

setdataloader, set respondent ID 604

setdatepart, set section of date/time to given value 599

setfmt newline, define newline character 47

setfmt, global text formatting commands 490, 491, 498

setinteger, assign real value to integer question 368

setivr, specify initial IVR configuration 625

setlang, change interview language in the script 602

setlocale, change interview language in the script 603

setreal, assign integer to real question 368

setstr, extract characters, copy into variable 587

setupdbm, create GNU database file 472

sfmt, format for *null/dk/ref* 498

Shared memory
list segments in 447
place qoc file in 832
remove response list database from 448

Shared qoc file, memory address of 832, 893

shareqoc, share a qoc file 832, 893

Sharing response list databases 439

Sharing the qoc file 832

shm file, shared memory address for response list database 446

shm, share a response list database 446

showinf, accounting by script section 51, 615

ShowStopStringAtEnd, display unique ref code at end of interview 848, 927

ShowStopStringAtStart, display unique ref code at start of interview 848, 927

sif file, Web interviewing program 875

signal, forced termination code for interview 385
effect on *setdata/quitdata* 386

Silent null 296

silentnl, silent null 296

Single punched, definition of 29

Single quotes
function of in script 40
in defined lists 165
in question texts 80
typing in script 40
with defined list names 173
with items in loops 239
with questions 32
with responses 32
with single punched responses 81
with text filenames 193
with texts 185

Single responses, check which chosen 342

Single-coded responses *See Single-punched responses*

Single-punched quotas 675, 694

Single-punched responses
assign by response number 288
assign by response text 285
automatic open ends for 107
cancel incorrect selections 807
check which one chosen 335
checking 327
columns allocated to 82, 83
define 81
defined lists 165
displaying 197
enter in CAPI 839
entering in qtip 806
in multipunched response lists 807
number in list 344
parentheses in list 119
parentheses with 213
position in list 342
print with *rprint* 431
read response text from list 345
rotation/randomization of 122
routing with 212, 213
routing within response list 119
save rotation number in data 125
separate in list 81
single quotes with 81

Size of text, change 496

Skips *See Routing*

Slash character, function of in interview 818

slider, slide bar for integer responses 136
entering responses in CAPI interviews 839

SMS
comments files 882
record key, store in data file 661
return to menu during interview 227
running qtip with 823
sample record format 660
sample record, write appointment times to 601
server, send requests to from script 664

SMS database
format 666
information, accessing from the script 659
name of 666

SMS database *contd.*
read data into script from 667
write data into script 668

SMS menu, return to during interview 826
smscript, call user-defined function 656
smscrres, function result of function called by *smscript* 657
smsctxt, message generated by *smscript* callfunc 657
smscerval, success/failure of *smscript* callfunc 656
 SMVARVALS, initialization values for global SMS variables. 918
 Snapbacks
 allow for in script 416
 alter questions after 821
 by one question 820
 change responses to previous questions 821
 definition of 29
 disallowing at end of interview 913
 effect on quotas
 effect on *set* statements 417
 effects on routing 416
ifsnap with *goto* 226
 into loops 820
 keywords for 820
 list names of snappable questions 819
 reset temporary variables when snapping 916
 routing with 821
 test for 223, 294
timeofday with 225
 to named questions 820
 with early completion 380
 with subsurveys 572
 write debugging information to file 915
 Snappable questions, display responses to 819
 .snp file, snpaback debugging information 915
[+*so+], start highlighted text 200
 Sound files
 disable recording of 622
 enable recording of 622
 file format 153
 play 153, 154, 160
 play/stop in CAPI interviews 841
 record open ends as 161
 Sound recordings
 maximum recording time 155
 quality control of 156
sp, single-punched response list 81
 in defined lists 171
 with frozen response lists 118
sp, single-punched response in multipunched list 130
 Spaces
 how Quancept deals with 45
 in text 47
 Spacing screen objects vertically 947
 Specified other
 check whether chosen 350
 code as zero 104
 coding 103
 coding in data 82
 columns allocated to 103
 definition of 30
 disallow when $\wedge o$ is used 110
 disallow with *promptoth* 109
 display open-end text 347
 display responses using negative subscription 349
 display with *promptoth* $\wedge o$ 347
 edit response text 352
 enter in CAPI interviews 840
 entering open end with *sp/mp* response codes 809
 format in mrInterview 498
 in defined lists 169
 in grids 251
 in loops 267
 in response lists 103
 loops 267
 loops with defined lists that include 241
 prompt text for 111
 text for 300
 text to display for 106
 with *bit* 343
 with *elm* 345
 with *nbit* 336
 with *numb* 344
 with *numv* 344
 with *qname* in *list* syntax 177
 See also Other
 SQL cursor type, setting 461
 Square brackets, function of in script 40
<st+>...<st->, strikethrough text 486
 Standard questions
 dealing with
 See also Library files, Subsurveys
 Standard questions, dealing with 569
 Standout and standout end 200
 Starting QCompile 760
 Statement names See Labels
 Statements
 arithmetic 361
 continuing 44
 format of 43
 length of 43
 maximum variables displayable in 198
 naming 43
 Station number, define 917, 918
step, ranges in loop value lists 240
stop, stop interview from within script 383
 effect of in *qtip* 812, 817
 interviewer stops an interview 817
 with quotas 674
 Stop and restart interviews
 display unique ref code 848, 849
 for new respondents only 850
 prompt for unique ref code 851
 restart stopped but do not start new ones 851
 start new even if stopped/completed exist 850
 Stop button
 display 927, 928
 hide/display in mrInterview 526
 label for 927, 929
 show/hide in Web interviews 507
 Stop button, display for test interviews only 231
 Stop CAPI interviews 841, 842
 Stop interviews from within script 383
 Stop Web interviews 846
StopButton, label for Stop button 927, 929

StopButtonEnabled, display/hide Stop button 846, 927, 928
stopdata, whether to save data for interviews stopped by interviewers 388
 effect of in qtip 813
Stopped interview directory 894
Stopped interviews
 allow 928
 directory for 894
 directory for restarted 894
 in qtip 812
 message to display when none exists 851
 message to display when one exists 850
 quotas in 675, 695
 rescheduling 813, 817
 restarting 384, 853
 retain original data for 619
 `rot/ran` with 384
 save data for 388
 suppress restart message 805, 912
 the stop directory 382
 user-definable filenames for 384
 with quota 695
Stopped interviews, suppress option to inspect/change 913
Stopping interviews in qtip 817
stp files, stopped interviews 894
String checking 350
Strings
 extract and convert to a number 586, 587
 extract from a text 588
strngchk, check text format 350
ststest, check whether this is a test interview 230
#sub, call subsurvey script 570
Sublists, order of responses in 166
subprog, execute external program 608
Sub-questionnaires *See* Subsurveys
subscript, find subscript 267
 comparison with iteration 267
Subscripted variables
 assignments to 292
 in loops 245
 with *substr* 590
Subscription
 *, current subscript value 265
 explanation of 264
 in numeric loops 268
 iteration value as subscript 266
 negative 349
 negative in loops 350
 nested loops 265, 270
 non-numeric *for* statements 270
 numeric *for* statements 270
 numeric value lists 265
 of questions with *multitask* 242
 question names in loops 264
 referring to repeated questions in other loops 268
 temporary variables 272
 unnested loops 265
 use pre-v7.8 behavior 915
 with loops 237
 with specified other 349

substitute, assign replacement text to a macro variable 315
substr, extract text and copy to variable 588
 subscripted variables with 590
Subsurveys
 calling 570
 choosing which to use 570
 data files for 571
 `datamap othzero` with 104
 definition of 30
 dummy 574
 how qtip locates 571
 `mapothzero` with 104, 571
 messages at end of 572
 parsing 571
 pass data to main script 574
 pick up data from main script 574
 serial numbers for 571
 snapbacks with 572
 suppress append verbatims prompt 912
 suppress automatic prompts at end of 572
 suppress End of subsurvey message 914
 suppress inspect/change interview prompt 912
 using 569
 sum file, script compilation summary 875
Supervisor, send message to from qtip 817
Supervisory program, display timed sections 615

T

tab file, Quantum table specs 770
<table>...</table>, tabular layout of questions 500
<tableDef>, overall grid characteristics 562
Table quotas 735
Table specs for Quantum 770
Tabs, how Quancept deals with 46
Tabular layout
 multitask with 500
 new columns in 501
 new rows in 501
tbl file, screen layout for CAPI 894
<td>...</td>, new column in table 500, 501
Templates
 associate with scripts 530
 contents of 516
 creating 517
 `<!doctype>` tags in 533
 line breaks and blank space in 518
 tags in 533
 name cascading style sheet in 535
 types of 518
Templates for page layout 516
Temporary variables
 advice for using 281
 arithmetic assignment with 363
 assigning categorical values to in multilingual scripts 304
 assign values to 282
 copy into the data 413

Temporary variables *contd.*
 data types for 282
 definition of 30
 parser to fail when encountered 312
 save arithmetic expressions in 364, 365, 367
 warn when encountered 311

Temporary variables, naming 45

TERM, terminal type for interviewing 916

Terminal type
 QUANCEPT 916
 TERM 916

Terminated interviews, save data for 389

Terminating interviews 379
 abandon without data 816
 allow/disallow early completion 380
 early completion 815
 end of script 811
 forced from within script 385
 quit with data 816
 quota control 812
 save data for 386
 stopping and restarting 382
 temporary suspension 817

Test interviews on active projects 854

Test interviews, actions for 230

Test mode, automatic, check for 295

Test runs, actions specific to 229

Testing
 CAPI scripts 837
 CATI 803
 mrInterview scripts 853
 scripts failing with runtime errors 834
 Web scripts 845

testingrun, actions for test runs only 229

testmode, switch into/out of test mode 606, 607

tex file, store open ends 896
 create for Web interviews 930
 differences between CATI & CAPI/Web 896

Text
 apostrophes in 34
 appending to 583
 arithmetic with numbers stored as 370
 assign to variables 283
 bold 486, 495
 center within the line 500
 check format of 350
 color 488, 496, 503
 colored against non-white background 488
 concatenate variables in 284
 concatenating 583
 continuing 44, 46
 convert numbers to integers 368
 define appearance of 187
 display data in 196
 display from a file 193
 display temporarily on screen 185
 enlarging 496
 extract strings from 588
 font 487, 496
 formatting 485
 formatting in response lists 485, 494

global formats for 490, 498
 highlight during interview 200
 insert in data file 413
 italic 486, 495
 length of when displayed from files 193
 line breaks 496
 maximum protected 191
 maximum with *display* 186
 new lines in 46
 numbers in 197
 open ends in 197
 paragraphs 496
 pictures in 148
 plus signs in 196
 protected displays 190
 protected within loops 192
 reduce size of 496
 remove displayed 186
 remove protected 190
 size of 487, 496
sp or *mp* responses in 197
 spaces in 45, 47
 strikethrough 486
 substitute in loop value list 255, 277
 substitute in response lists 198
 underlined 486, 495

<**textColor**>, text color in grids 551

Text alignment in grid cells 547

Text comparisons in multilingual scripts 603, 911

Text formatting
 in CAPI scripts 485
 in Web scripts 493
 introduction to 37

Text variables, quotas for 738

textarea, format multiline input boxes for open ends 504

textline, format numeric & one-line open end input boxes 506

tim file, interview timings 897

***time**, display current time in qtip 822

time, time current statement was executed 295
 differences in appointment times 825
 ranges, entering appointments in 825

Time manipulation, callfuncs for 596

Time zones for interviewers. 918

Timed out interviews with quota 695

timeofday
 snapbacks with 225

timeofday, time of day 295

Timeout, text to display when connection times out 927

timesect, time sections in script 611

Timing sections of a script 615

tiperrs, qtip message file 898

Title, page title 928

to, range separator 84

tomorrow, appointment for, setting 824

Totals
 accumulating in loops 271
limitresp with 94
 loops with non-numeric *for* statements 271

<**tr>...</tr>**, new row in table 500, 501

Translation, changing scripts after 304

Translations, file to read from 912
tx0 file, numbered list of texts in the script 876
txx file, script index file 877
Type-ahead, disallow in interviews 910
Typeface *See* Font
TZ, interviewer's time zone 918

U

<u+>...<u->, underlined text 486
<u>...</u>, underlined text 495
..., indent response lists 502
Unanswered questions, testing for 354
Uncompleted interviews, save data for 389
Underlined text 486, 495
Unfilled quota cells, locate 682
unfreeze, cancel frozen response list 117
Unique IDs, assign to responses 139
Unique reference code for interview

 display 848, 849

 prompt for 851

Unix databases, summary of commands for 78

unprotect, cancel protected text 190

unset, unset a variable 309

 example in script 289

 ignore when parsing 868

Unset environment variables 906

Upper case

 distinction from lower case 42

 GNU database record keys 468

URL, respondent information in 925

UseCurrMdd, do not create new version in mdd file 953

UsedBrowserButtons, text to display when browser buttons used 928

UseMapFile, map variable names 954

usepictf, associating pictures with questions and responses in InterviewBuilder questionnaires 954

User name for qmonitor 908

USER, user name to display in qmonitor 916

User-defined callable functions *See* Local callfuncs

User-defined functions *See*Local callfuncs 656

userno, interviewer's user number 295

V

^v, scaling factor 132

<valign>, vertical alignment in grid cells 547

valstring, convert text numbers to integers 372

<value>, width of grid/column 554, 563

Values, assigning with *set* 282

Variables

 assign logical values to 331

 assign replacement text to macro 315

 assign text to 283

 assign values to 282

 concatenate in texts 284

 define in ini file 926

definition of 30, 281, 363
in displayed text 186, 197
in protected text 191, 197
in response lists 198
longtext, combining responses in one variable 312
maximum displayable per statement 198
mixing types of in arithmetic expressions 365
resetting temporary when snapping 916
results of unsetting 309
set by mrInterview 73
subscripted, assignment to 292
subscripted, with *substr* 590
temporary

 advice for using 281

 copy into data file 413

 data types for 282

 qparse to fail when encountered 312

 save arithmetic expressions in 364, 365, 367

 subscription of 272

 warn when encountered 311

temporary, naming 45

types of 281

types with arithmetic assignments 363

unsetting values 309

with *mmaudio* 154

with *mmpicture* 149

with *mmvideo* 153

Variables table 900

varlist, write data out to temporary file 608

Verbatims *see* Open ends

Verification statements, introduction to 33

Vertically distributing objects in design mode 947

Video clips

 formats for 153

 play 152

 play/stop in CAPI interviews 841

vsetcols, number of columns for responses 204

 set column width with 204

W

warntemp, warn if temporary variables used in script 311

wav files, sound recordings 153

Web interviewing program 875

 files created by 879

Web interviewing, texts for 305

Web interviews

 action at end of 926

 customizable error messages 297

 data security 925, 935

 define answer prompt 297, 300

 display interview ID 848

 display name of current question during interviews 192

 display Stop button 846

 display unique reference code 848

 files required for 845

 for new respondents only 850

 message for invalid responses 297

Web interviews *contd.*

- message for missing responses 297
- message when browser buttons used 297
- quota maintenance 722
- respondent information in URL 925
- restart greeting 852
- reusing respondent IDs 933
- show/hide Previous button 507
- show/hide Stop button 507
- start new even if stopped/completed exist 850
- start server manager 923
- stop and restart 846
- text for Restart button 852
- text formatting overview 493
- time-out for idle 923, 924

Web project, minimum set-up required 845

Web scripts

- parse 759
- test 845

Web server

- connect to 924
- install scripts on 845
- make projects available on 924, 935
- start 923
- stop 924
- text to display when server crashes 927

Web<SmallCaps>CATI interviewing, texts for 305

Whole numbers as responses 84

<**width**>, grid/column width tag in grid templates 554, 563
<**widthFormat**>, grid/column width type tag in grid templates 554, 563
windfmt, *display* text global format 490
winfmt, global format for all texts 490
winpfmt, *protect* text global format 490
winqfmt, question text global format 490
winrfmt, response text global format 490
WriteDatFile, create dat file 930
WriteTexFile, create tex file 930

X

xdisplay, display text from a file 193

- reduce qc size with 193

See also display

xin file, information about GNU databases 877

.xor., only one expression true 327

xor, only one response required 339

xprotect, display text from a file 193

- reduce qc size with 193

See also protect, unprotect