CSPro

User's Guide

Version 6.0.1

International Programs Center for Technical Assistance
Population Division
U.S. Census Bureau
Washington, DC 20233-8860

Phone: 1-301-763-1451
Fax: 1-301-763-4282
E-mail: CSPro@lists.census.gov

25 September 2014

Table of Contents

CSPro Users Guide 1

The CSPro System 1

What is CSPro? 1

CSPro Capabilities 2

CSPro Applications 3

Data Entry Applications 3

Batch Edit Applications 3

Tabulation Applications 4

Data Dictionary 5

Forms Design 5

Tool List 5

CSPro General Concepts 7

CSPro Initial Screen Layout 7

Trees 8

Windows 8

Unicode Primer 9

How To 11

Create a CSPro Application 11

Open an Existing Application 12

Change the View 13

Designer Font Preferences 13

Change Windows 14

Add Files to an Application 14
Drop Files from an Application 14
Change the Print Page Setup 14
Print All or Part of a Document 15
Get Help 15
Save an Application 15
Close an Application 16
Save an Application with a New Name 16
Specify Application File Names 16
Pack an Application 17
Data Dictionary Module 18
Introduction to Data Dictionary 18
Organization 18
Questionnaire and Dictionary Organization 18
Data File Type Structure 20
Dictionary Hierarchy 21
Dictionary Concepts 22
General 22
Labels 22
Names 22
Notes 22
Levels 22
Level Description 22

Level Properties 23
Records 24
Record Description 24
Record Properties 25
Record Type 25
Required Record 26
Maximum Number 26
Items 26
Item Description 26
Identification Items 27
Subitems 27
Item Properties 28
Starting Position 29
Length 29
Data Type 29
Occurrences 30
Decimal Places 30
Decimal Character 30
Zero Fill 31
Value Sets 31
Value Sets Description 31
Value Set Properties 32
Values 32

Value Description 32
Value Properties 33
Relations 33
Relation Description 33
Relation Properties 34
Data Dictionary Application 34
Creating a Dictionary for a New File 34
Creating a Dictionary for an Existing File 35
Data Dictionary Screen layout 36
Data Dictionary Tree 37
Relative and Absolute Mode 38
Dictionary Types 38
Reconciling Dictionary Changes 39
How to 39
Open an Existing Dictionary Application 39
Move Around a Dictionary 39
View the Dictionary Layout 40
Add Dictionary Elements 41
Modify Dictionary Elements 41
Add or Modify Levels 41
Add or Modify Records 41
Add or Modify Items 42
Add or Modify Value Sets 42

Linked Value Sets 42
Generate Numeric Value Set 43
Add or Modify Values 44
Undo and Redo Changes 44
Select Several Dictionary Elements 45
Insert Dictionary Elements 45
Delete Dictionary Elements 45
Move Dictionary Elements 45
Find Dictionary Elements 46
Document Dictionary Elements 46
Convert Items to Subitems 47
Select Relative or Absolute Positioning 47
Create Dictionary with No Record Types 47
Occurrence Labels 47
Add or Modify Relations 48
Print the Dictionary File 48
Save Dictionary As New File 48
The CSPro Language 49
Introduction to CSPro Language 49
Data Requirements 49
The CSPro Program Structure 50
Debugging CSPro Programs 51
Declaration Section 51

Compiler Mode 51
Variables 52
Arrays 52
Files 53
User-Defined Functions 53
Alias Statement 53
Procedural Sections 54
Statements 54
Proc Statement 55
Preproc Statement 55
Postproc Statement 56
Logic 56
View Logic 56
Create and Edit Logic 57
Find and Replace Logic 58
Set Compiler Defaults 58
Compile Logic 58
Language Elements 59
Delimiters 59
Comments 59
Variables and Constants 59
Data Items 59
This Item ($) 60

Subscripts 60
Numbers 61
Text Strings 61
Expressions 62
Expressions 62
Substring Expressions 62
Special Values 63
Operators 64
Operators 64
In Operator 65
Has Operator 65
If and Only If Operator <=> 66
Operator Precedence 66
And/Or Truth Table 67
Files 67
External Files 67
Lookup Files 68
Working Storage File 69
Message File 69
Program Information File 70
Data Entry Module 70
Introduction to Data Entry 70
Data Entry Application 70

General Data Entry Concepts 71
Data Entry Philosophies 71
Skip Issues 71
Errors at Data Entry 72
Adding Logic 72
CSPro Data Entry Concepts 72
Operator vs. System Controlled 72
Data Entry Path 73
Data Entry Elements 73
Issues to Consider When Designing a Form 75
Create a Data Entry Application 76
Create a New Data Entry Application 76
Generate Default Data Entry Forms 77
The Drag Option Menu 78
Data Entry Forms Screen Layout 79
Data Entry Tree 81
Run a Data Entry Application 82
Run Production Data Entry 82
Generate Binary Data Entry Application 85
Change Data Entry Characteristics 85
Change the Order of Entry 85
Change Data Entry Options 86
Change Default Text Font 89

Change Field Font 90
Change Error Sound 90
Forms Designer 90
Introduction to Forms Design 90
Add Things to a Form 91
Modify Things in a Form 94
Change Form Properties 101
Data Entry Editing 111
Introduction to Data Entry Editing 111
Editing Concepts 111
Type of Edits in Data Entry 111
Structure Edits 112
Consistency Edits 113
Checking errors 113
Writing Logic 114
Data Entry Logic Screen Layout 114
Moving Around a Logic Application 116
Order of Executing Data Entry Events 116
Sequence dictated by designer 118
Compile an Application 118
Run as Batch 119
CAPI Data Entry 120
Introduction to CAPI 120

CAPI Features 121

Extended Controls 121

Text Box Options 125

CAPI Strategies 126

Forms 126

Fields 127

Questions 127

Values 128

Organization of the Instrument 128

Using Multiple Languages 128

Breaking Off the Interview 129

Coming Back Later 129

How to 129

Create a New CAPI Application 129

Define Languages 130

Organize Forms 131

Enter Question Text 131

Create Fills In Questions 131

Create Standard Forms 132

Change Formatting 132

Add Pictures 133

Use Multiple Language 133

Create Conditional Questions 133

Display Questions Without Scrolling 134
Structure Movement..... 134
Create Helps for Fields 134
Show Values for Selection..... 135
Show Images and Values for Selection 135
Show Full Screen Value Sets..... 135
Handle Don't Know and Refused..... 136
Handle Multiple Answers 136
Choose Topic Sections..... 137
Create General Helps 138
Test Application..... 138
Android Data Entry..... 138
Android Development..... 138
Android Navigation 139
Pocket PC (PPC) Data Entry..... 145
Introduction to Pocket PC (PPC) Data Entry..... 145
PPC Requirements 145
How To 146
Write PPC Data Entry Application 146
Install CSPro on a Pocket PC..... 148
Deploy PPC Application to the Pocket PC 150
Run PPC Application on the Pocket PC 152
Retrieve Data Files from the Pocket PC 154

Batch Editing Applications 154
Introduction to Batch Editing 154
Create a Batch Edit Application 154
Create a New Batch Edit Application 154
Batch Application Screen Layout 156
Batch Edit Tree 158
Run a Batch Edit Application 159
Order of Editing 160
Order of Executing Batch Edit Events 160
Batch Edit Order 160
Change Edit Order 161
Correcting Errors 161
Methods of Correcting Data 161
Guidelines for Correcting Data 162
Imputation 163
Static Imputation 163
Dynamic Imputation (Hot Deck) 164
DeckArrays 165
DeckArray Leftover Rows 168
Types of Edits in Batch Editing 169
How to 170
Manipulate Automatic Reports 170
Create a Specialized Report 171

Use Hot Decks 172
Initialize Hot Decks In Program Logic 172
Initialize Hot Decks Using Saved Arrays 173
Interpret Reports 174
Run Production Batch Edits 175
Steps in Developing a Batch Editing Program 177
General Issues 177
Review Edit Specifications 178
Define Coding Standards 178
Code Edits of Individual Data Items 179
Develop Comprehensive Test File 179
Test CSPro Program 179
Re-Test with Live Data 179
Begin Production Editing 180
Tabulation Applications 180
Introduction to Tabulation 180
Parts of a Table 181
Common Uses of Tabulation Applications 184
Capabilities of Tabulation 185
Cross Tabulations 185
Tabulate Counts or Percents 186
Tabulate Values and/or Weights 186
Produce Summary Statistics 186

Restrict a Universe 187

Format Tables for Printing 187

Load and Save Formatting Preferences 187

Produce Tables by Area 188

Save Tabulations in Different Formats 188

Copy Table to Other Formats 188

Copy and Paste Table Specification 188

Map Results by Geographic Area 189

Create Multiple Subtables 189

Change Unit of Tabulation 191

Tally Items from Related Records 191

Creating Tables 191

Create a New Tabulation Application 191

Create a Table 192

Create Tables with Multiple Variables 194

Implications of Data Dictionary Value Sets 196

Tabulate Items with Multiple Occurrences 197

Tally Attributes for a Variable 198

Tally Attributes for a Table 203

Add, Insert, and Delete Tables 206

Move Between Tables 207

Run a Tabulation Application 207

Renaming Tables and Table Applications 208

How To ... 208

Add a Variable to a Tabulation 208

Remove a Variable from a Tabulation 210

Define a Universe for a Table 210

Add Weights to a Table 212

Tabulate Values Instead of Frequencies 213

Include/Exclude Special Values in a Variable 213

Hide or Change the Position of the Total 214

Add Percents to a Table 214

Add Summary Statistics to a Table 216

Include/Exclude Tables from Run 216

Debug Table Totals 217

Formatting Tables 219

Formats for a Part of a Table 219

Formats for a Table 223

Formats for an Application 224

Formats for Printing 226

Views of Tables 229

How To ... 230

Customize Table Text 230

Hide or Show a Row or Column 231

Hide Rows Containing All Zeros 233

Change the Number of Decimal Places Displayed 233

Add a Footnote (Pagenote or Endnote) 234
Add Header/Footer Text to a Table 235
Add a Subtitle 236
Add Stub Leadering 236
Add Borders 237
Add Reader Breaks 238
Change the Way Numbers are Displayed 239
Change the Automatically Generated Text 240
Change Fonts or Colors 241
Change Indentation or Alignment 242
Add Borders to Cells 242
Make Captions Span Data Cells 244
Reset Format of Table Item to Default 245
Change the Repeating of Boxheads 245
Change Stub Column Position 246
Creating Tables by Geographic Area 247
Area Processing 247
Create an Area Names File 248
Area Dialog Box 250
Area Captions 251
Custom Consolidation 253
Create a Thematic Map of Results 255
How To 256

Display Results for One Geographic Area 256
Tabulate only Certain Levels of Geography 257
Printing Tables 257
Using Print Preview 257
Navigating Between Pages, Tables, and Areas 258
Viewing Multiple and Facing Pages 259
Modifying Row and Column Spacing for Printing 260
Print Preview Options 260
Print Setup 263
Sending Tables to the Printer 264
How To 265
Undo or Reset Changes in Print Preview 265
Print Only Selected Tables or Pages 266
Tabulation Preferences 266
Preferences and Default Formats 266
Modifying Preferences 267
Loading and Saving Preferences 268
How To 269
Share the Same Format on Multiple Computers 269
Saving and Copying Table Data 269
Save Tables for the Table Viewer 269
Saving Tables as Text, HTML or Rich Text 270
Select and Copy Table Data to Other Applications 271

Using Table Data in the Map Viewer 272

How To 273

Distribute Finished Tables to Other Users 273

Copy Table Data to a Spreadsheet or Word Processor 274

Prepare Tables for Posting to the Web 274

Table Post Calculation 274

Introduction to Table Post Calculation 274

Adding Rows and Columns For Post Calculation 275

Post Calculation For Individual Cells 276

Post Calculation For Rows, Columns and Ranges 279

Row and Column Indexing for Post Calculation 281

Run Production Tabulations 283

Introduction to Production Tabulations 283

Run All in Batch 284

Run in Parts 285

Introduction to Run in Parts 285

Run Tabulate Interactively 286

Run Tabulate in Batch 286

Run Consolidate Interactively 288

Run Consolidate in Batch 289

Run Format Interactively 290

Run Format in Batch 291

Advanced Table Topics 292

Using Subtables 292
Changing the Unit of Tabulation 293
Table Logic (tablogic) 294
Tabulations Using Relations 298
How To 299
Tabulate Items in Relations 299
Table Tips and Tricks 300
Add Subtotals to a Table 300
Tabulate Categories With Disjoint Values 302
Format/Hide Rows and Columns in Subgroupings 303
Recodes in Tables Using Value Sets and Subtables 307
Use Expressions in Universe and Value Tallied 311
CSPro Statements and Functions 312
Alphabetical List 312
Statement Format Symbols 317
List of Reserved Words 317
Declaration Statements 318
Set Statement 318
File Statement 319
Numeric Statement 320
Alpha Statement 320
String Statement 321
Array Statement 321

Relation Statement 323

Function Statement 324

Program Control Statements 326

Break Statement 326

Do Statement 327

Exit Statement 328

For Statement 328

If Statement 329

Next Statement 329

Universe Statement 330

While Statement 330

Assignment Statements 331

Assignment Statement 331

Recode (Box) Statement 331

Impute Function 334

Setvalue Function 335

Getvalue Function 336

Data Entry Statements and Functions 336

Accept Function 336

Advance Statement 337

Changekeyboard Function 338

Demode Function 339

Display Orientation 339

GetOrientation Function 339

SetOrientation Function 339

Editnote Function 340

Endlevel Statement 340

Endgroup Statement 341

Enter Statement 341

Getcapturetype Function 341

Getlanguage Function 342

Getnote Function 342

Getocclabel Function 343

Getoperatorid Function 343

Getos Function 344

Getrecord Function 344

Getusername Function 345

GPS Function 345

Highlighted Function 347

Ispartial Function 347

Killfocus Statement 348

Move Statement 348

Noinput Statement 349

Onchangelanguage Global Function 349

Onchar Global Function 350

Onfocus Statement 351

Onkey Global Function 351

OnKey Character Map 355

Onstop Global Function 356

Putnote Function 357

Randomizevs Function 357

Reenter Statement 358

Savepartial Function 358

Selcase Function 359

Set Attributes Statement 360

Set Behavior Canenter Statement 362

Setcapturepos Function 362

Setcapturetype Function 363

Set Errmsg Statement 364

Setlanguage Function 366

Setvalueset Function 366

Setvaluesets Function 367

Format: 367

Show Function 368

Showarray Function 369

Skip Statement 370

Userbar Function 371

Visualvalue Function 375

Batch Edit Statements 376

Endcase Statement 376
Export Statement 376
Getdeck Function 377
Putdeck Function 378
Set Behavior Export Statement 379
Setoutput Function 379
Skip Case Statement 379
Numeric Functions 380
Abs Function 380
Cmcode Function 380
Countnonspecial Function 381
Exp Function 382
High Function 382
Inc Function 382
Int Function 383
Log Function 383
Low Function 383
Random Function 384
Randomin Function 384
Seed Function 385
Sqrt Function 385
Set Behavior SpecialValues Statement 385
String Functions 386

Compare Function 386
Concat Function 387
String Concatenation Operator 387
Edit Function 388
Getbuffer Function 389
Length Function 389
Maketext Function 390
Pos Function 391
Poschar Function 392
Strip Function 392
Tolower Function 393
Tonumber Function 393
Toupper Function 394
Multiple Occurrence Functions 394
Average Function 394
Count Function 395
Curocc Function 395
Delete Function 396
Insert Function 396
Max Function 397
Maxocc Function 398
Min Function 398
Noccurs Function 399

Seek Function 399
Seekmax Function 400
Seekmin Function 400
Soccurs Function 401
Sort Function 401
Sum Function 402
Swap Function 402
Totocc Function 403
General Functions 403
Errmsg (Display) Function 403
Execsystem Function (Windows) 406
Execsystem Function (Android) 406
ExecPFF Function 407
Getlabel Function 408
Getsymbol Function 408
Invalueset Function 409
Pathname Function 409
Special Function 410
Stop Function 410
Sysparm Function 411
Trace Function 411
Date and Time Functions 413
Dateadd Function 413

Datediff Function 414

Datevalid Function 415

Sysdate Function 415

Systime Function 416

External File Functions 416

Clear Function 416

Close Function 417

Delcase Function 417

Fileconcat Function 418

Filecopy Function 418

Filecreate Function 419

Fileempty Function 419

Fileexist Function 420

Filedelete Function 420

Filename Function 420

Fileread Function 421

Filerename Function 421

Filesize Function 422

Filewrite Function 422

Find Function 423

Key Function 423

Loadcase Function 424

Locate Function 424

Open Function 425
Retrieve Function 426
Setfile Function 426
Writecase Function 427
Write Function 428
Appendix 429
Appendix A - Installation 429
Hardware and Software Requirements 429
Uninstalling CSPro 430
Installing a Newer Version 430
Installing Data Entry Applications 431
Creating Limited CSPro Installations for Redistribution 432
Appendix B - Keys Summary 432
Data Dictionary Keys 432
Data Entry Keys 433
Batch Edit Keys 435
Tabulation Keys 436
Appendix C - Menu Summary 437
CSPro Menu 437
Data Dictionary Menu 437
Data Entry Menu 438
Batch Editing Menu 440
Tabulation Menu 440

Appendix D - Toolbar Summary 441
CSPro Toolbar 441
Data Dictionary Toolbar 442
Data Entry Toolbar 443
Batch Editing Toolbar 444
Tabulation Toolbar 444
Appendix E - Converting Within IMPS or ISSA 445
Converting a data dictionary 445
Converting within IMPS 446
Converting within ISSA 447
Converting an IMPS Data Entry Application 447
Converting an ISSA Data Entry Application 447
Appendix F - Errors in Censuses and Surveys 447
The Nature of Census and Survey Data 448
Errors in Censuses and Surveys 448
Appendix G - File Types 450
File Types 450
Importing Data to CSPro Format 451
Locking Application Files 453
Temporary Data File 454
Files Description 455
Data Dictionary File (.DCF) 455
Binary Data Entry Application File (.PEN) 456

Data Entry Application File (.ENT) 456

Form File (.FMF) 456

Logic File (.APC) 456

Messages File (.MGF) 457

Question File (.QSF) 457

Data File 457

Data File Index (.IDX) 457

Notes File (.NOT) 457

Data File Status (.STS) 458

Listing File (.LST) 458

Tabulation Application File (.XTB) 458

Table Specifications File (.XTS) 459

Tables File (.TBW) 459

Area Names File (.ANM) 459

Table Matrices File (.TAB) 459

Table Matrices Index File (.TAI) 459

Batch Edit Application File (.BCH) 460

Edit Order File (.ORD) 460

Frequency file (.FRQ) 460

Map Data File (.MDF) 460

Map File (.MPC) 460

Program Information File (.PFF) 460

Frequency Specification file (.FQF) 461

Operator Statistics File (.LOG) 461

Saved Arrays File (.SVA) 462

Synchronization File (.PNC) 462

Index 467

CSPro Users Guide

The CSPro System

What is CSPro?

The **C**ensus and **S**urvey **Pro**cessing System (CSPro) is a software package for entering, editing, tabulating, and disseminating data from censuses and surveys. CSPro combines the features of the **I**ntegrated **M**icrocomputer **P**rocessing **S**ystem (IMPS) and the **I**ntegrated **S**ystem for **S**urvey **A**nalysis (ISSA).

CSPro runs under Windows XP, Vista, 7, and 8. It does not run under other operating systems such as Linux or Mac OS. It is a public domain product, so it can be used and distributed at no cost.

CSPro can be used to process data from censuses and surveys, both small and large. Typical subject areas include:

• Housing and Population
• Demographic Characteristics
• Health and Nutrition
• Agriculture
• Labor Force
• Business Establishments
• Education
• Living Standards
• Energy
• Immigration
• Household Income and Expenditure
• Community
• Institutional
• Post-Enumeration
• Vital Statistics

CSPro uses data dictionaries to provide a common description of each data file used. All data files in CSPro are text files encoded in UTF-8 format. CSPro provides tools to view data and other text files, to view tables and thematic maps created by CSPro, to convert IMPS and ISSA data dictionaries to and from CSPro, and to convert ESRI shape files (maps) to CSPro map files.

CSPro is not intended to provide database management capabilities. However, the data generated and/or manipulated by a CSPro application may be imported into a database system. While CSPro provides some tabulation capabilities, it is not intended to replace more sophisticated statistical analysis software such as R, SAS, SPSS, Stata, etc. In addition, even though CSPro includes a module for generating thematic maps, it cannot be considered a geographical information system (GIS) because the maps cannot show the multiple layers available in a true geographical information system.

CSPro includes the following modules:

• Data Entry Applications

• Batch Edit Applications

• Cross Tabulation Applications

• Tools

If you have never used CSPro before, you can refer to the Getting Started Guide, a tutorial that gives you an overview of CSPro's capabilities.

This section includes the following information:

CSPro Capabilities
CSPro Applications
CSPro General Concepts
How To …

CSPro Capabilities

• Process Census or Survey Data
Given an existing data file, a user can develop a CSPro application that will examine the file for inconsistencies, structural defects, or other errors. CSPro permits the user to generate detailed reports on all errors found; the user may also create subfiles from the original data, and may use multiple look-up files during the validation and/or report-generation process.

• Enter, Modify, and Verify Data
CSPro users can create data entry forms (screens) for data capture. The application designer has full control over form layout. CSPro supports rosters, consistency checks and skip patterns of unlimited complexity, user-defined messages and menus, multiple lookup files, and produces operator statistics.

Once a case has been completely entered, the operator can modify any part of the existing data and can add or remove information, as well (subject to application constraints).

CSPro supports both dependent and independent verification (double keying) to ensure the accuracy of the data entry operation. Using independent verification, operators can key data into separate data files and use CSPro utilities to compare them. Using dependent verification, operators can key data a second time and have CSPro immediately compare it to what was keyed the first time on a field-by-field basis.

• Manipulate Data Files
CSPro permits the user to re-structure existing data files and to create subsets of data in separate files. New files can be created by merging two or more case-related files. Data files in software-specific formats may be created for import into spreadsheets and some statistical packages.

• Tabulate Data
The user can create an application to produce frequency distributions or cross-tabulations using two to four variables. Results can be displayed either globally (for the totality of the data file) or according to one or more elements of the geographic hierarchy. Tabulations may show only percentages, or percentages in conjunction with counts. Data may be weighted or unweighted.

• Create Thematic Maps
If computerized maps are available for the relevant geographic areas, CSPro can generate cross-tabulations whose results can be joined to the map files to produce thematic maps for display of information. Thematic map display parameters permit a high degree of customization in the presentation of these data.

• Use and Share External Files
When a data file is to be used by more than one person, a CSPro dictionary can be created and distributed among users of the data to facilitate access. Including multiple value sets for variables can cater to the different needs of individual users so that each user's requirements are met.

- **Examine Data Files**
  CSPro provides language elements that will permit the specification of logic to carry out a detailed examination of a data file. Elements of the file may be tested against other elements of the same file or against elements of one or more other files, and the user may generate reports showing the results of the examination. CSPro also provides a facility for comparing the contents of two data files. This utility will generate a detailed report to the user documenting any differences found.

- **Interactive Editing**
  CSPro language elements can be used to construct a series of tests to be carried out on a case-by-case basis using the CSEntry module. Whether adding a new case or modifying an existing case, CSPro instructions permits interactive editing and correction of data elements. If the user desires, a report on editing activity may be generated and saved for printing after the session is completed.

- **Examine Results of Editing**
  Whether the user is carrying out interactive or batch editing, the CSPro language permits the preparation of reports with detailed information on cases tested, errors found, and errors corrected. These reports are written to disk in text format and may be viewed (or printed) with any text viewer, such as CSPro's utility Text Viewer. They provide documentation of work carried out and permit analysis of types and frequency of errors.

CSPro Applications

Data Entry Applications

A Data Entry application contains a set of forms (screens) and logic that a data entry operator uses to key data to a disk file. Data entry applications can be used to add new data and to modify existing data.

You can have the following run-time features in your data entry application:

- Add new cases (questionnaires) or retrieve and modify existing cases

- Logic can be executed and messages displayed after any field is entered

- Consistency checks and skip patterns of unlimited complexity

- Multiple look-up files

- Cases indexed to avoid duplication and for easy retrieval

- Operator statistics

You use CSPro to develop the data entry application. You use CSEntry to run the data entry application. For small surveys and for testing applications, you can run CSEntry directly from CSPro, on the same computer. For large surveys and censuses, which require a production environment, you can transfer the application files to other computers and run CSEntry on them.

See also: Create a New Data Entry Application

Batch Edit Applications

A **Batch Edit** application contains logic that you can apply against one set of files to produce another set of files and reports. Batch editing applications can be used to gather information about a data file.

You can incorporate the following run-time features in your batch editing application:

• Write edits (logic) using powerful CSPro language

• Validate individual data items

• Test consistency between items

• Check case/questionnaire structure

• Modify data values

• Use arrays for hot deck or cold deck imputation

• Generate imputation statistics

• Generate edit reports automatically or create a customized report

• Create additional variables

• Read/write to multiple look-up files

You use CSPro to develop the batch editing application. You use CSBatch to run the application. For small surveys and for testing applications, you can run CSBatch directly from CSPro, on the same computer. For large surveys and censuses, which require a production environment, you can transfer the application files to other computers and run CSBatch on them.

See also: Create a New Batch Edit Application

Tabulation Applications

A **Tabulation application** contains a set of table specifications and a data dictionary describing a data file to be tabulated. When you create your application, you can use an existing data dictionary or you may create one as you create the application.

In a Tabulation application, you can:

• Cross-tabulate a virtually unlimited number of variables.

• Tabulate variables created "on the fly" under program control.

• Select the universe of tabulation.

• Tabulate values and weights.

• Tabulate simple counts and percents.

• Tabulate mean, median, mode, standard deviation, variance, n-tiles, proportions, min, max.

• Perform table cell manipulation after tabulation.

• Define detailed table formatting.

• Save tabulations in several file formats.

• Copy tables to spreadsheets or word-processing documents.

• Produce tables by geographic area.

• Map results by geographic area.

See also: Create a New Tabulation Application

Data Dictionary

A Data Dictionary describes the overall organization of a data file, in other words, it gives a description of how data are stored in a file. CSPro requires that a data dictionary be created for each different file being used. A Data Dictionary file has the extension .DCF.

In the Data Dictionary you can:

• Define simple or complex hierarchical file organization.

• Define hierarchical levels, identification items, records, items (fields or variables), value sets (categories of values), and values.

• Create descriptive notes for documentation.

• Define multiple-occurring items.

• Produce reports of file organization.

See also: Creating a Dictionary for a New File, Creating a Dictionary for an Existing File

Forms Design

You can create an unlimited number of forms (screens) for data entry. These can be designed independently or as part of the data entry application.

• Forms may be any size; CSEntry will scroll as necessary

• Forms may contain fields from different physical records

• Physical records may be split among different forms

• Forms may contain individual fields or rosters

There is usually one Form File (.fmf) per application, but there may be multiple forms files. Each forms file contains one Data Dictionary File (.dcf) that represents the primary data file that is being created or modified.

See also: Introduction to Forms Design

Tool List

To run a tool, open the **Tools** menu and select one of the tools listed below:

• Text Viewer
The Text Viewer will display the contents of any text file up to a maximum of 32,000 characters wide and up to 2 gigabytes in size. You can copy, save, or print all or part of the contents of the text file. You can also find text in the file, identify line and character position in the file, and copy tabular reports to spreadsheet programs. The file cannot be modified within the Text Viewer utility.

- Table Viewer
  The Table Viewer allows you to examine, but not change, the contents of any CSPro tables file. A table file (extension .tbw) is produced by running CSPro tabulation applications or using the Tabulate Frequencies tool. You can copy, save, or print all or parts of the tables in RTF (for word processing programs), or HTML (for Internet), or TAB delimited (for spreadsheet) formats. You can also create and view thematic maps of selected cells.

- Map Viewer
  This tool allows you to create and manipulate thematic maps of data. Thematic maps can be created as part of CSPro tabulations. Thematic maps can be generated for a selected variable at a selected geographic level or as a combination of two variables as a difference, percent change, ratio or percent ratio. Multiple variables may be associated with a single map data file to permit greater flexibility to users of the map(s). You can vary the number of intervals, size of the intervals, colors, titles and legends; change the lowest geographic level shown; copy maps to a word processor; or save maps in GIF (for Internet) format.

- Table Retrieval
  Retrieve and display tables, maps, and other previously prepared documents from a large database of documents based on geography, subject matter, and title. It is very useful as a data dissemination tool.

- Tabulate Frequencies
  This tool allows you to produce frequency distributions of all or some of the variables in a data file. You simply select the variables (value sets) you want to tabulate and provide the name of the data file. More than one data file can be tabulated. See Section x.x.x for more information.

- Sort Data
  Sort a data file by questionnaires or by records. Sort key may be identification fields or data fields on single required record types.

- Export Data
  Export selected data records or parts of data records to tab- or comma- delimited files. These files can be imported into spreadsheets or databases. It also allows you to export data records or parts of records to data files for which descriptions are created for SPSS, SAS, or Stata.

- Reformat Data
  Reformat data from one file format to another using an input and output data dictionary. Fields with corresponding names are copied from the input to output file. This is useful for reorganizing data records or lengthening data items.

- Compare Data
  Compare the contents of two data files and identify the differences. The data files must have the same structure, that is, the same CSPro dictionary must describe both data files.

- Concatenate Data
  Concatenate (join end-to-end) two or more CSPro data (or other text-based) files. You do not need a dictionary for this utility. You only need to know the name and location of the files you wish to combine.

- Table Retrieval Setup
  Create and modify a set of tables and other documents organized by geographic area, subject and title for use by CSPro Table Retrieval tool.

- Convert Dictionary
  Convert IMPS and ISSA data dictionaries to CSPro data dictionaries, or convert CSPro dictionaries to IMPS or ISSA dictionaries. Convert ISSA dictionaries to CSPro data dictionary and data entry forms.

• Convert Shape to Map
  Convert ESRI ArcView or ArcInfo polygon shape files to CSPro map files. Map files can be thinned to reduce the number of points in the polygons.

• Pack Application
  Pack all the files in a CSPro application into a zip file so the application can be moved to another computer or sent as an email attachment.

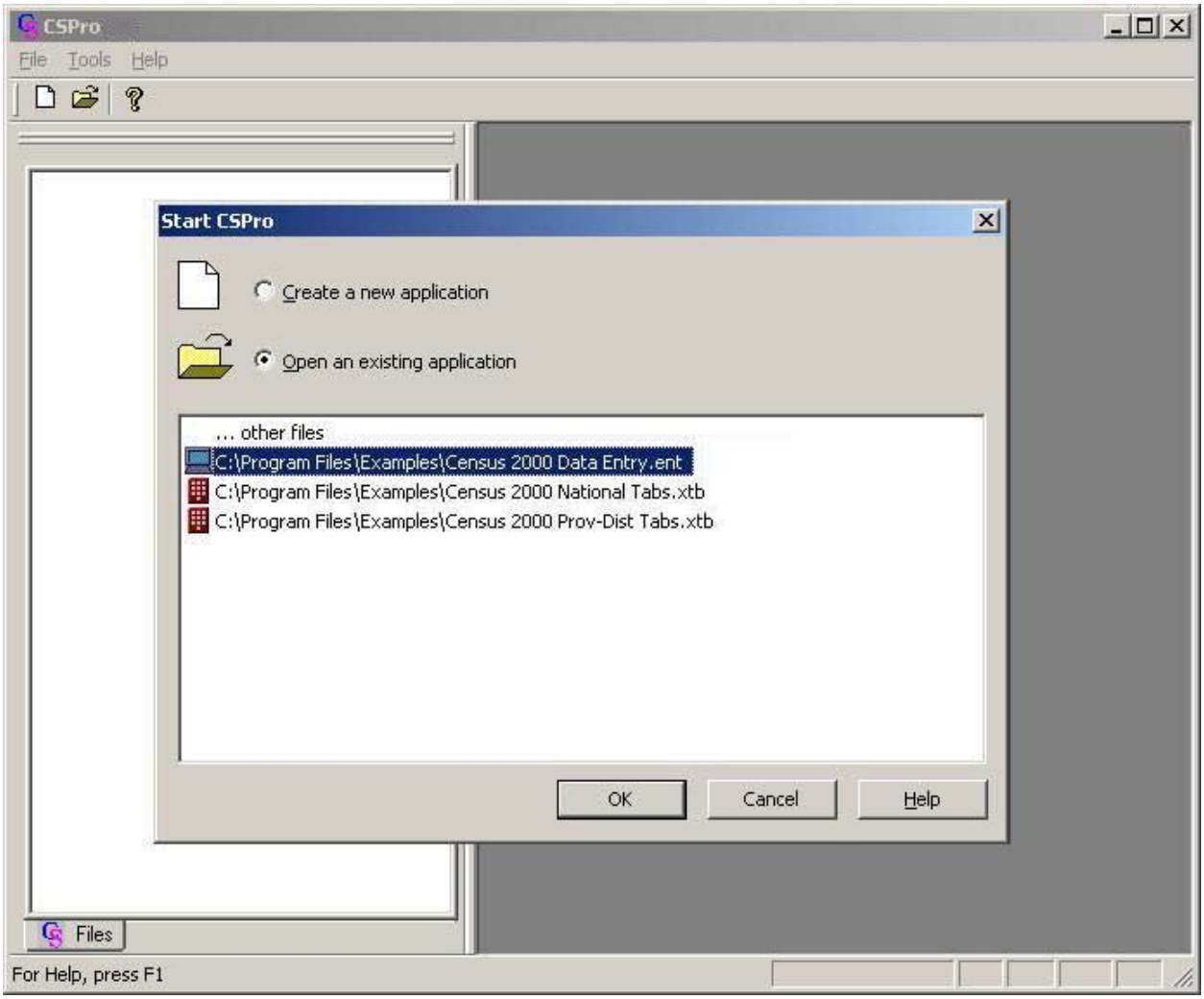
• Index Files
  Generate indices for data files or identify duplicate cases in a data file.

There is a user's guide for each of the tools.

CSPro General Concepts

CSPro Initial Screen Layout

To open CSPro, click on the CSPro icon on your desktop. The screen will be subdivided into two parts: the left is reserved to display file trees; and the right window is reserved to display the actual application. Initially both windows are empty.



Create a new application
This allows you to create a new application when CSPro is launched. After you specify the names of the applications files, the new application is opened.

Open an existing application
This allows you to open an existing application either by selecting a recently used application from the list provided or to select, **using … other files**, any CSPro application available on the computer or connected servers.

If you cancel the dialog box, CSPro will remain open so that you can use CSPro tools or at a later time open an application or create a new application.

Trees

After you create an application, the tree will display the application(s) that are currently open, all the files that belong to that application and their relationships with one another. The files tree is always present. When you close the application or file, it is removed from the files tree.

There are five kinds of trees in CSPro:

• Files tree shows all the applications that are open, and the files they contain.

• Dictionaries tree shows all the dictionaries that are open, and their contents.

• Data entry forms tree shows all the form specifications that are open, and their forms and fields.

• Batch edits tree shows all the edits specifications that are open, and the order of edits.

• Tables tree shows all the table specifications that are open, and their contents.

The files tree is always available. The other four trees are available only if appropriate applications are open.

To change the tree on the left side, click the tab of the tree you want to see.

The tabs at the bottom of the tree indicate which file tree is displayed. To change the tree, click the tab of the tree you want to see. Use **Ctrl+T** to see the full file names (labels) of the files you have open. Double-click on the files tree to switch the frame on the right side of the screen.

See also: Data Dictionary Tree, Data Entry Tree, Batch Edit Tree

Windows

The window on the right side of the screen allows you modify the contents of a dictionary or application. Each different window has different functions associated with it. That is, you will see a different menu and toolbar with each different window.

Part of the toolbar to the left of the Help button shown below allows you to switch between different types of windows: Dictionary, Forms Design, Batch Editing, and Tabulation.

To change the contents on the right side of the screen press the button of the type of window you want to view. If there is more than one window of that type, the most recent one viewed will be displayed.

If you need to select a particular window, from the **Window** menu, select the file name you want to view.

Unicode Primer

Beginning with version 5.0, CSPro is Unicode compliant.

What is Unicode?

Unicode is a widely adopted system for representing characters for all languages currently in use. Early computer programs generally used only one byte to represent a character, which led to a limit in the number of characters that could be displayed on screen and used in computations. This limit of 256 characters was used effectively by people who only required English characters, or characters from most European languages, but it could not represent the languages used by more than a billion people, including most notably, many Asian languages.

The Unicode standard now defines more than 100,000 characters, but many of these characters represent extinct languages. For that reason, Windows only provides native support for a subset of Unicode characters. In a Unicode program, a character is represented by two bytes, which allows for up to 65,536 characters.

Whereas in English it is very clear what makes up one character—one keystroke—in other languages it is not as straightforward. For example, in Chinese, typing two characters, "天津," requires seven keystrokes (“Tianjin”). In Bangla, the character পি requires two keystrokes (প + ি), which combine to create one character. Both the Chinese and Bangla examples require four bytes in memory and return a string length of 2.

What is UTF-8?

With each character stored as two bytes in memory, there are several ways to write characters to a disk. One way is to write two bytes for every character, but this is costly for most users, the bulk of whom only use characters that can be expressed properly with either the ASCII or ANSI encoding systems. The computer world has settled on using UTF-8, a variable-length encoding scheme that uses between one and four bytes to represent a character. ASCII characters are represented using one byte, other ANSI characters are represented using two bytes, and most Asian characters are represented using three bytes. To identify a file as encoded in UTF-8, a three-byte BOM (byte order mark) is placed at the beginning of a text file. For example, here is the UTF-8 representation of: "I am François from 法国."

Character	(BOM)			I		a	m		F	r	a	n	ç		o	i	s		f	r	o	m		法			国			.
Hex	EF	BB	BF	49	20	61	6D	20	46	72	61	6E	C3	A7	6F	69	20	20	66	72	6F	6D	20	E6	B3	95	E5	9B	BD	2E
ANSI	ï	»	¿	I		a	m		F	r	a	n	Ã	§	o	i	s		f	r	o	m		æ	³	•	å	›	½	.

With a UTF-8 encoding, a file's size is not equal to the number of characters in the file. An empty file is three bytes because of the BOM. (The fileempty can be used to determine whether a file is actually empty.) An ASCII file converted to UTF-8 will generally be the size of the original file plus three bytes. An ANSI file that uses non-English characters will be even larger. The name François takes eight bytes to represent in ANSI but uses nine bytes in UTF-8.

How Will CSPro Work with Unicode Files?

Versions of CSPro up to 4.1 used ANSI characters for data files and all specification files. Starting with version 5, CSPro uses the UTF-8 encoding scheme for representing Unicode characters. CSPro will always be able to read both ANSI and UTF-8 files, so any old files will work in the new version, but when files are modified and saved the files will be rewritten in UTF-8. All new data files and listing reports will be created in UTF-8. Because earlier versions of CSPro only supported ANSI files, any file created in CSPro 5.0 or later, including data files, is no longer automatically backwards compatible with older versions of CSPro.

How Will UTF-8 Affect My File Sizes?

If you only use ASCII characters in your specification and data files, your new files will be only three bytes larger than the CSPro 4.1 equivalents. These three bytes reflect the size of the UTF-8 BOM. If you use many accented characters that are valid in ANSI (like à and û), then your files will increase in size, though only slightly, because the vast majority of characters in any CSPro file are still digits and Latin characters. In general your files will increase in size by an inconsequential amount.

What if I Want to Use a Data File With an Older Version of CSPro?

The CSPro installation includes a tool, the Unicode Text Converter, that allows you to change the encoding of text files. This utility allows you to convert ANSI files to UTF-8, though this is not necessary because CSPro 5.0 or later does this automatically. You will more likely use the tool to convert UTF-8 files to ANSI. If you use non-ANSI characters in your specification or data files, these characters will be converted to question marks during the conversion to ANSI. For instance, "I am François from 法国." will become “I am François from ??.” Converting files from UTF-8 to ANSI is not necessary a lossless conversion, so it should be done only if absolutely necessary of if you are confident that you did not use any Unicode-only characters.

Some text editors, including Notepad, allow a user to change the encoding of a text file. These editors can be used to convert text files from UTF-8 to ANSI. If using Notepad, open the file and then select Save As. At the bottom of the dialog box is an option to change the encoding.

What about CSPro's Binary Files?

The important binary files created by CSPro are compiled data entry applications (.pen), file indices (.idx), and intermediary tabulation files (.tab and .tai). These formats have changed and CSPro can no longer read binary files written by CSPro 4.1 or previous versions. CSPro can determine if a compiled data entry application is of the correct format and will present an error message to an operator who tries to open an old version in CSEntry. CSPro can also determine if an old index is being used and will automatically generate a new index. If you use intermediary tabulation binary files (e.g., if you run tables in parts), you should delete and regenerate them when upgrading to a Unicode version of CSPro. The Unicode Text Converter will not work correctly on binary files.

Can I Still Write and Save to ANSI Format?

Outside of the Export Data tool, CSPro no longer supports writing to ANSI format, though it can still read files encoded in ANSI. CSPro automatically converts files from ANSI to UTF-8 formats as needed. CSPro will convert only files that may be modified by a program. For example, if you use an external file in your application but only call fileread and never filewrite, CSPro will keep the input file in its original format. However, if there is a filewrite statement and the file is encoded in ANSI, CSPro will convert the file to UTF-8 before opening it. The old ANSI file will be sent to the Recycle Bin in case you want to recover it.

Distributing CSPro Data Files

Most applications support Unicode so you may not have a problem using external applications with your data or specification files. However, if you must use an application that is not Unicode compliant, run the Unicode Text Converter on the necessary files.

If you are distributing files to a broad audience and if your data file does not contain any non-ANSI characters, you might consider converting it to ANSI for maximum compatibility. To illustrate an example of what might happen with a non-Unicode program reading a UTF-8 file, imagine a CSPro data file that has two record types, P (population) and H (housing). The ANSI file might look like this:

```
P<case id><data>
P<case id><data>
P<case id><data>
H<case id><data>
```

The UTF-8 file might look like this:

```
<BOM>P<case id><data>
P<case id><data>
P<case id><data>
H<case id><data>
```

An older version of CSPro would regard the first row of data as an invalid record because the BOM was in the place of the record type. It would thus register the household as having two members instead of three. If any non-ASCII characters existed in the data file, it would further shift the data and then items would not be read correctly.

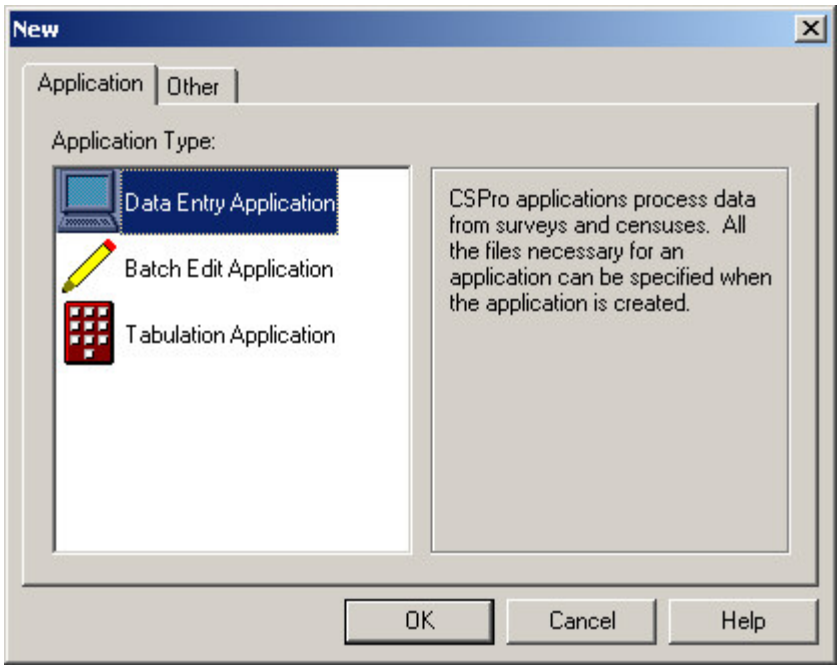
If you release a UTF-8-encoded data file widely, you may want to include a documentation file with text similar to the following:

> **Warning: This CSPro data file is encoded in UTF-8. UTF-8 is a text encoding format widely used to represent characters from any language. If you use this data file with a software package that does not support Unicode and UTF-8, you must first convert the data file to an ANSI encoding. Without this conversion there is a risk that the software package will misread parts of the data file.**

How To ...

Create a CSPro Application

Click on the toolbar; or from the **File** menu, select **New**; or press **Ctrl+N**. The following dialog box will appear.



Select the type of application you want to create and press **OK**. You can create one of the following applications:

- Data Entry Application – to enter data from a paper questionnaire or though a computer assisted personal interview (CAPI)

- Batch Edit Application – to detect and correct errors in data file

- Tabulation Application – to produce publication ready tables

You will then be prompted to enter the file name of the application. Enter the file name of the application and press **Create.**

The File Associations Dialog will be displayed. Specify the names of other files that make up the application, such as the data dictionary files.

You are given a name for the input dictionary. You can change the name of the input dictionary and/or add the names of additional external lookup dictionaries.

Open an Existing Application

Click on the toolbar, from the **File** menu select "**Open**," or press **Ctrl+O**. Select from the "Files of type" at the bottom of the dialog box. In CSPro you can open either an application or a file. Select the name of the file you want to open. Each time you open or create an application or file, it is added to the files tree. Any other files belonging to the application or file will also be opened and added to the appropriate trees.

You may open a data dictionary and make changes to it, even if it already belongs to an application. Be aware that if you later open an application to which it belongs, CSPro will automatically make necessary adjustments in other files. For example, if you delete or rename a dictionary item, then later open an application that contains the data dictionary, any corresponding fields on forms will be deleted.

You may open a forms file and make changes to it, even if it already belongs to an application. However, you will not have access to the associated Logic file and you will not be able to run it.

Any changes you make to applications and files are not made permanent until you save the file or application that you modified.

Change the View

- **In the File Tree**
  The file tree can display the label or name of the file(s) and the contents of the applications currently open. To toggle between labels and names in trees open the **View** menu, select "Names in Trees", or press **Ctrl+T**. A check mark appears next to the "Names in Trees" menu item when names are displayed instead of labels. The setting of "Names in Trees" affects **all** the trees.

- **Screen**
  To toggle between trees on left [i.e., split screen] and full screen open the **View** menu, select "Full Screen", or press **Ctrl+J**. A check mark appears next to the "Full Screen" menu item when the display is in full screen mode. The setting of "Full Screen" affects **all** applications.

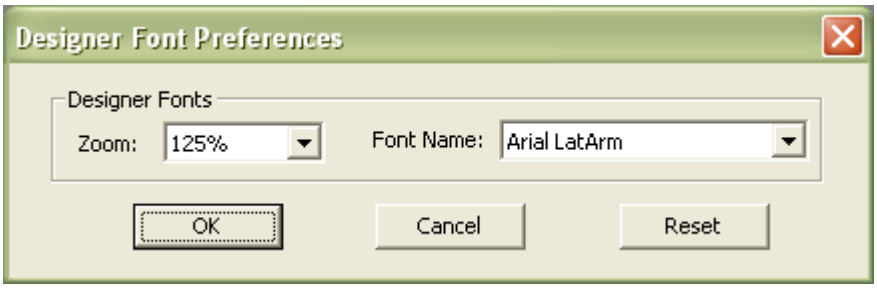
Designer Font Preferences

Versions of CSPro prior to 5.0 supported some languages that used non-Latin alphabets such as Armenian and Russian. With the switch to Unicode support in CSPro 5.0, this feature has been removed. Old applications that used these language specific ANSI fonts will no longer look correct in the CSPro Designer. For example, this is a value set in an Armenian dictionary:

N	Value Set Label	Value Set Name	Value Label	From	To	Special
☐	ê»é	SEXVT				
☐			îÕ³Ù³ñ¹ (m)	1		
☐			ÎÇÝ (f)	2		
☐			âÉñ³ó³Í	▯		NotAppl

To view these old applications correctly, you can manually change the font used to render the text in the dictionary grid, the dictionary and form tree, and, if the font is monospaced, the application's logic.

With no application open, select Tools -> Preferences -> Fonts.



In this example, an Arial Armenian font has been selected, at an increased zoom rate. After this selection, the dictionary editor displays the contents better:

N	Value Set Label	Value Set Name	Value Label	From	To	Special
	Սեռ	SEXVT				
			Տղամարդ (m)	1		
			Կին (f)	2		
			Չլրացված	□		NotAppl

Ultimately you will want to convert your application to Unicode, as these font preferences are disregarded by CSEntry. Many tools exist to convert ANSI language scripts to their Unicode equivalents.

Some users may find that the CSPro Designer renders the Unicode characters for their language at a very small size. It is possible to change the zoom factor while leaving the font name blank, thus making the characters easier to read.

Change Windows

• **Cascade**
Use this command to arrange multiple opened windows in an overlapping fashion.

• **Tile Top-to-Bottom**
Use this command to arrange multiple opened windows one above the other in a non overlapping fashion.

• **Tile Side-by-Side**
Use this command to arrange multiple opened windows one beside the other in a non-overlapping fashion.

• **1,2,…**
View displays a list of currently open files at the bottom of the **Window** menu. A check mark appears in front of the name of the file in the active window. Activate a window by choosing the name of its file from this list.

Add Files to an Application

You may add dictionaries and forms files to a data entry application. Additional dictionaries represent data files used by the application, such as look-up files. Multiple forms files are sometimes used in advanced applications. You may not add files to tabulation applications.

To insert a file, click on the **Files** tab to bring up the files tree. From the **File** menu, select **Add File**. Select an existing file or enter the name of the file to be created.

Drop Files from an Application

To drop one or more files from an application, the **Files** menu, select **Drop Files**. Select the files you want to drop from the application and press OK. These files will be dropped from the application, but will not be deleted from the disk.

Change the Print Page Setup

To change the page headers, footers, or margins, click on the toolbar or, from the **File** menu, select "Page Setup". The changes will remain in effect until you change them again. In the page setup dialog box make changes to the page headers, footers, and margins.

- **Header**
  Edit the text to be placed at the top left, top center, and top right of each page. You can use the **Date**, **Time**, **File**, and **Page** buttons to insert the current date, time, file name, and page number.

- **Footer**
  Edit the text to be placed at the bottom left, bottom center, and bottom right of each page. You can use the **Date**, **Time**, **File**, and **Page** buttons to insert the current date, time, file name, and page number.

- **Margins**
  Change the size of the top, bottom, left and right margins. Your printer may not allow margins below certain values.

To change the page orientation or size, open the **File** menu and select "Print Setup". In the print setup dialog box make changes to orientation (portrait or landscape) and paper size.

Print All or Part of a Document

To print an entire document click on the toolbar; or from the **File** menu, select "Print"; or press **Ctrl+P**.

To print part of a document select the text you want to print then click on the toolbar; or from the **File** menu, select "Print"; or press **Ctrl+P**.

To preview the printing, click on the tool bar; or from the **File** menu, select "Print Preview."

Get Help

Click on the toolbar; or, from the **Help** menu, select "Help Topics"; or press **F1**. Most dialog boxes have a **Help** button.

To contact us about problems

International Programs Center for Technical Assistance
Population Division
U.S. Census Bureau
Washington, DC 20233-8860

Phone: 1 (301) 763-1451
Fax: 1 (301) 763-4282
E-mail: CSPro@lists.census.gov

Visit: http://www.census.gov/population/international/software/cspro/

Please contact us for any problem or to get more information on an application. When you contact us, please indicate the version number of the software you are using. You can obtain the version number from the top of the **About** box. From the **Help** menu, select "About."

Save an Application

Click on the toolbar; or, from the **File** menu, select **Save**; or press **Ctrl+S**.

The file associated with the current frame (right side of the screen) will be saved. If that file belongs to an application that is open, the entire application will be saved. If the file belongs to more than one application, CSPro will ask you which one you want to save. In that case, select the file or files you wish to close or save and click on "OK."

To choose all of the files, click on the **Select All** button. To choose several files, hold down the **Ctrl** key and click on files you wish to select.

See also: Open an Existing Application

Close an Application

From the **File** menu, select **Close**.

The file associated with the current frame (right side of the screen) will be closed. If that file belongs to an application that is open, the entire application will be closed. If the file belongs to more than one application, CSPro will ask you which one you want to close. In that case, select the file or files you wish to close or save and click on "OK."

To choose all of the files, click on the **Select All** button. To choose several files, hold down the **Ctrl** key and click on files you wish to select.

See also: Open an Existing Application

Save an Application with a New Name

To save an application under a new name:

- Make sure the application is showing in the current frame (right side of the screen)
- From the **File** menu, select **Save As**.
- Enter the file name for the new application in the file open dialog box.
- Using the File Associations Dialog specify the names of other files in the application.

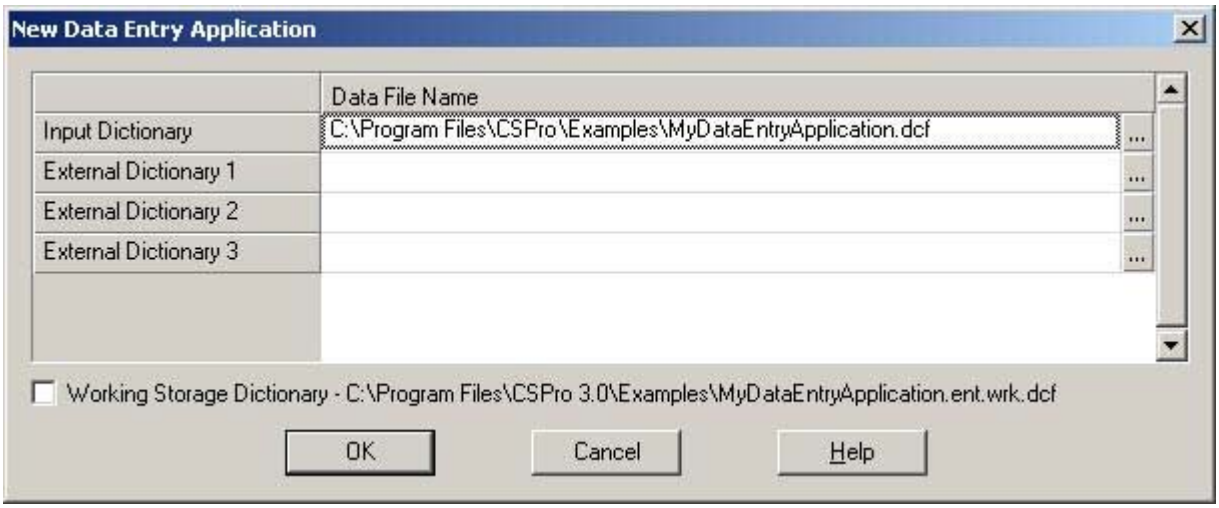
When the Save As is complete you will be editing the new dictionary.

See Also: Save Dictionary As a New File

Specify Application File Names

New Application

When a new application is created, a files dialog box like the one below is displayed.

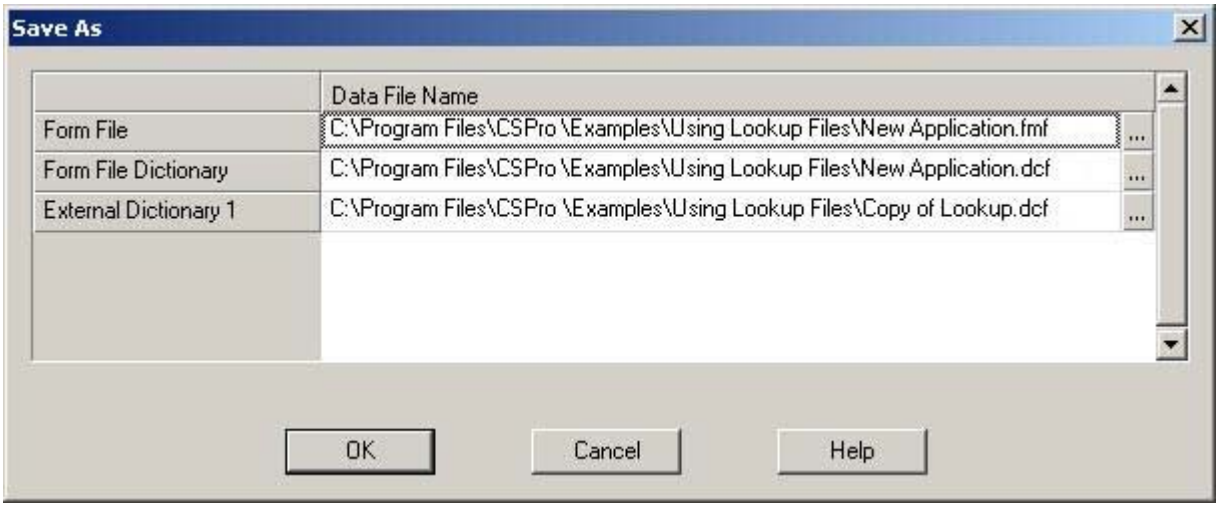


Only the name of the input dictionary must be supplied. A default input dictionary file name is given, but it may be changed. Other external dictionary file names are specified as needed.

If any of the files names specified already exist, those files will be attached to the application. If any of the file names specified are new, those files will be created.

Save As Application

When an application is saved under a new name, a files dialog box like the one below is displayed.



Names for all the files requested must be supplied. Default file names are given, but they may be changed.

If any of file names specified are new, those files will be created for the new application. If any of the files names is the same as the name in the original application, those files will be shared with the new application. If any other of the files names specified already exists, those files will be overwritten.

See also: New Application, Save As Application

Pack an Application

There are instances when it is helpful to collect all the files in an application to:

- Move them to another computer (for example to move data entry applications to all your data entry computers).
- Give the application to a colleague to use.
- Send the application when requesting help.

There is a CSPro tool to perform this function. This tool copies all the files in the application into a ZIP file.

To use this tool go to the **Tools** menu and select **Pack Application**.

See also: Pack Application

Data Dictionary Module

Introduction to Data Dictionary

A Data Dictionary describes the overall organization of a data file. In other words, it gives a description of how data are stored in a file. CSPro requires that a data dictionary be created for each different file being used.

Each dictionary will allow you to give text labels for all levels, records, items, and value sets in the file. It also allows you to describe the organization of each record in the file and the characteristics of each item in the record: names, position in the data record, type of data, length, number of decimal places, valid values, and other documentation.

Before you convert the information from a questionnaire to computer-readable form, you usually create a data dictionary. You can also create a data dictionary for an existing data file if you have a description of its contents showing the location of each item. .

CSPro requires that a Data Dictionary be created for each different file being used.

This section contains the following information:

Organization
Dictionary Concepts
 Levels
 Records
 Items
 Value Sets
 Values
Data Dictionary Application
How to ...

Organization

Questionnaire and Dictionary Organization

• **<u>Questionnaire</u>**
- **Form**
A questionnaire is a collection of information relating to the same unit of observation (such as a household, person, or factory). A typical questionnaire consists of an identification section followed by other sections grouped by topic. Each section includes a set of related questions, each of which is associated with a list of response values. A questionnaire usually constitutes a case.

- Section

Any type of questionnaire will have an identification section that uniquely identifies the form, as well as one or more sections on different topics. Some sections may occur once per questionnaire while other sections are repeated many times. For example, in a typical housing and population census, a questionnaire would contain a section for the housing questions, and a section for the population questions. The questions in the housing section will be answered once per questionnaire [household], while the questions on the population section will be answered by every person in the household. If the census is collecting information on vacant housing units then the questions on the population section will not be answered. In a school survey, for example, the questionnaire would have an identification section and only one section to collect basic information for each student. The questionnaires for the different students are not related.

- Questionnaire Identification

The identification section identifies the questionnaire, usually with numeric geographic codes. The combination of identification codes (such as province, district, village, household) on a questionnaire uniquely identifies the form. These are the codes you would need to locate a specific questionnaire.

- Questions

The basic element of the questionnaire is the question. Each section of the questionnaire contains a set of one or more questions being asked for this census or survey.

- Responses

The valid options in response to a question are usually listed in the questionnaire. Some responses are quantitative, such as "size of farm" or "age of person," and some are qualitative, such as "relationship to head of household" or "crop grown." Responses can be numeric or alphanumeric. Most descriptive responses are equated to numeric codes that are placed on the questionnaire. However, some descriptive responses remain as alphabetic text.

• **<u>Data Dictionary</u>**

- File

In the data dictionary a topic section is usually equivalent to a record. A record includes data items (questions) that are associated with one or more value sets (response values). Records with the same identification codes (i.e., Questionnaire Ids) comprise a single questionnaire.

- Records

Similarly, a data dictionary may have records that occur once and records that repeat many times. The typical housing and population census will have one housing record and as many population records as people in the household; the housing and population records will equate to one questionnaire, and these records are related. If our study permits vacant housing units, then the data file will not include a population record for an unoccupied housing unit. In the school survey each questionnaire will only have one record, and there is no relationship between records in the data file. This type of data file is known as a "flat data file." The records for the different sections will most likely have different structures. Using the data dictionary module you can identify each record structure using a record type name and code.

- File Identification

Similarly in the data dictionary, you first define the identification items to uniquely identify the questionnaire. These data will appear on every record in a data file, as they are "common" to all of the records. In a data file, if a group of questionnaire ID fields uniquely identify the unit under observation, then those records make up one questionnaire. In the case of the student survey the student identification number could serve as the questionnaire identification.

- Data Items

In the data dictionary the data item contains the response to a question, and is therefore the most basic element of a questionnaire — "age", "income", and "crop-code" are all examples of items. Related items should be placed in the same record. And, just like records and levels, data items

possess properties (such as a unique name, label, etc). Items in data files must be fixed format, that is, items must have the same starting position and length in every record where they occur.

- Value Sets
In the data dictionary, the responses in the questionnaire are defined as value sets. A single value set can contain one or more values. The valid values can be defined as individual values or ranges of values.

See also: Record Description, Item Description, Value Sets Description

Data File Type Structure

There are two basic types of data file structures: Those that contain single-record questionnaires, or those that contain multiple-record questionnaires. The following is a brief description of these two types.

• A single record type per questionnaire
In a single-record data file, each line of data from the data file equates to a distinct questionnaire. This means there is no relationship between records in the data file—each record stands on its own and is distinct from another.

One usage for a single-record questionnaire would be a student survey at a university. In this scenario, a single record would be created based on the student. The student identification number could serve as the questionnaire identification. A data file produced from this type of dictionary is known as a **flat data file**. Example:

```
00011122122    ← (first student)
00021122122    ← (second student)
00031122122    ← (third student)
```

`blue` text refers to the student identification number.
`black` text describes the individual data items for each specific record.

Notice there is no need to have a record type identifier.

• Multiple record types per questionnaire
In a multiple-record data file, several lines of data (and therefore several records) from the data file equate to one questionnaire. This means there is a relationship between records in the data file—and information identifying them as such in the form of Questionnaire Ids will be needed.

For example, in a typical housing and population census, a questionnaire might consist of the following records:

- one housing record
- multiple (zero or more) population records

For a given questionnaire there would be one or more population records for one household record, dependent on the number of people in the household. However, if you allowed vacant housing units, then those questionnaires would not have any corresponding population record.

A sample (and recommended) file structure could be as follows (not all fields are defined for this example):

```
11010011122122   ← (household with 3 persons)
21010011201091961 38
21010012121051967 32
```

```
2101001311707199207
11010031211212    ← (vacant household)
11010021111121    ← (household with 2 persons)
2101002110716193069
2101002220812192871
```

In the example above:

Red	text refers to the record type. In our example, 1 is a household record, and 2 is a population record.
Blue	text refers to the (Id Items). Note that the numbers are unique for each questionnaire: the 101001 household contains three people whereas the 101002 household contains two people.
Black	text describes the individual data items for each specific record.

A questionnaire designed for an agricultural census might consist of the following records:

- one farm household record
- multiple (one or more) crop records
- multiple (one or more) farm worker records

A questionnaire for a reproductive health survey might consist of the following records:

- one record for data on the woman
- multiple (zero or more) children-ever-born records
- one contraceptive use record
- one immunization record

Dictionary Hierarchy

A data dictionary is structured in a hierarchical order. The top hierarchy is the case, followed by the level, then record.

• **Case**

A case is the primary unit of data in the data file. A case usually corresponds to a questionnaire. However, some complex applications might have a hierarchical set of questionnaires, or many **levels**. For example, the main questionnaire may consist of a household roster and other household information, and there may be a separate questionnaire for each woman in the household. The data entry application may then contain two levels — one for the household and one for each woman in the household. The set of forms corresponding to the household make up level one. The set of forms corresponding to each woman make up level two. Each case would consist of two type of questionnaires: a single level one and a variable number of occurrences for level two.

Most applications consist of a single level.

• **Level**

A level is a type of questionnaire. By default, all new dictionaries have one level. This is normally sufficient to describe, for example, a population or agriculture census. However, if you have a hierarchically-structured set of questionnaires, you will probably need to use additional levels. A level can have many **records** corresponding to different record types.

• **Record**

A record usually corresponds to a section of a questionnaire, and consists of a group of related data items. For example, data items related to housing would form a housing record; data items related to individuals would form the population records; data items related to production of a particular crop would form the crop record, and so on. If a dictionary contains more than one record, then you must use the record type item to identify one record from another in the data file.

See also: Level Description, Record Description

Dictionary Concepts

General

Labels

Labels are descriptive text used to identify the dictionary and its elements. Labels are required for the dictionary and most of its elements. Labels can contain any printable character and spaces and can be up to 255 characters long.

The dictionary tree displays either the labels or names of dictionary elements. You can press **Ctrl+T** or from the **View** menu, select **Names in Trees** at any time to toggle between labels and names.

Names

Names identify the dictionary and its elements when they are referenced in CSPro procedures. Names are required for the dictionary and most of its elements. Names consist of upper case letters (A-Z), digits (0-9), and embedded underlines (_). The first character of a name must be a letter; the last character cannot be an underline (_).

Names can vary in length from 1 to 32 characters. Names must not be CSPro reserved words. Names cannot be duplicated within a dictionary. However, the same name can be used in different dictionaries, and in some cases, it maybe desirable to do so.

Examples: SEX, RELATIONSHIP, MOTHER_ALIVE, Q102_AGE_CHILD

The dictionary tree displays either the labels or names of dictionary elements. You can press **Ctrl+T** or from the **View** menu, select **Names in Trees** at any time to toggle between labels and names.

Notes

Notes document the dictionary and its elements. The designer may create notes for the dictionary as a whole and/or any of its elements: levels, records, items, value sets, values. Notes may contain any printable character and spaces, and can be up to 65,000 characters in length.

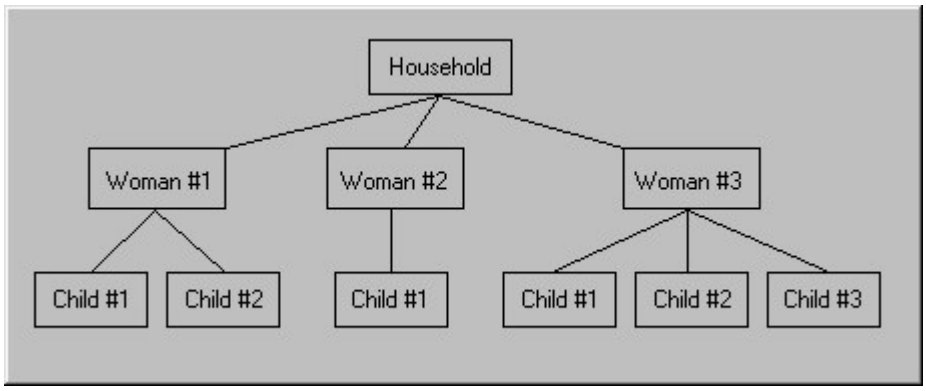
Levels

Level Description

A level is a type of questionnaire. By default, all new dictionaries only have one level. The maximum number is three. A good use for a three-level dictionary might be a reproductive health survey that has the following questionnaires:

- A housing questionnaire
- A questionnaire for each woman of reproductive age in the household
- A questionnaire for each woman's child in the household

A pictorial representation of this scenario is as follows:



In this example, you would want each child to be associated with its mother, rather than the household record. If you were to structure your dictionary in a single level, there would be no way to easily identify which mother and child(ren) belonged together during data entry or during tabulation. To accomplish this, you would want to design your dictionary with three levels, each level containing a single type of record, as follows:

Level 1
Household Record
Level 2
Woman of Reproductive Age Record
Child Record (one for each child)

In the Forms Designer you will be required to place each record's data on different forms. However, this facilitates the desired data entry behavior. You will first be asked to enter information from Level 1, i.e., the household. After completing the household form(s), you will then enter information for the first woman and her children (Level 2). When data entry is finished for this woman record the keyer will advance to the child record, and enter information for each child (if any) for this women.

If there are no further children (or no children at all for this female), finish the level by pressing **F12** (EndLevel Occurrence) and resume entering information for the second woman and her children. Continue in this manner until all women and their children have been entered for the household—when finished, press **Ctrl+F12** (EndLevel) from the Woman Form to complete data entry for this case.

Keep in mind that, when using more than one level, there are implications with respect to the order of executing logic in a data entry application or in a batch edit application.

See also: Dictionary Hierarchy, Level Properties, Add or Modify Levels

Level Properties

Level properties are visible when the dictionary has been selected in the tree tab. To reflect your intended usage for a level we suggest you change the level properties pressing **Ctrl+M,** which will activate the appropriate entry in the right-hand screen.

Property	Meaning
Label:	A descriptive text label which identifies this level
Name:	The name given to this level for use in the CSPro language procedures.

See also: Dictionary Hierarchy, Level Description, Add or Modify Levels

Records

Record Description

A record is a group of related data items. In the process of creating a record to define (a portion of) the questionnaire, you will also be defining the physical layout of the data file. For example, suppose your (very simple) population record looks like the following (only item name, starting position, and length properties are shown; starting positions show that ID items occupy the first 9 positions in the record):

```
Item Name     Start Pos  Length
Relationship  10         1
Sex           11         1
Age           12         2
```

If an operator had keyed a questionnaire for a 35-year-old female (Sex = 2) head of household (Relationship = 1), you would see a line in the data file, corresponding to the population record defined above:

```
 1 2
12345678901234567890 <-- position
--------------------
 1235 <-- line in data file
```

In deciding on a file structure, there is often the choice of defining a record type which occurs once within a questionnaire but contains repeating sets of data, or to define a record type which occurs multiple times within a questionnaire, each with a single occurrence of the data. The application designer should take into consideration the amount of information that recurs and the probable number of occurrences.

A common example in a Housing and Population Census is information about deaths in the household during the 12 months prior to the census. If this information (usually sex and age at death of the deceased) is collected during enumeration, the expectation is that 95% of households will have no more than one, or at most two, deaths during the previous 12 months. With this volume of information, it would be practical to have one record type that occurs once within the questionnaire and allows for repeating occurrences of the data, since it is unlikely that even the maximum number of occurrences, multiplied by the number of positions occupied by each occurrence, will exceed the length of the already-existing household and population records.

However, in the case of an agricultural survey, a section on crops may include questions about acreage planted, yields, etc., whose cumulative length for each crop mentioned may be quite large in relation to other records in the file. In such a case, it would be more practical to define a record type that occurs multiple times within the questionnaire. Within each occurrence of the record would be found the information relating to one specific crop.

See also: Dictionary Hierarchy, Record Properties, Record Type, Required Record, Maximum Number, Add or Modify Records

Record Properties

You can view a record's properties by selecting the questionnaire to which it belongs (via the dictionary tree tab). You may change the default record properties by positioning the cursor in the right window and pressing **Ctrl+M**.

Property	Meaning
Label	A descriptive text label which identifies this record
Name	The name given to this record for use in the CSPro language procedures.
Type Value	The record type value (code) that identifies this kind of record
Required	Must a questionnaire contain this kind of record? (Yes/No)
Max	The maximum number of times this type of record can appear in any one questionnaire.

See also: Dictionary Hierarchy, Record Description, Add or Modify Records

Record Type

The Record Type is an alphanumeric item that uniquely identifies a dictionary record, and therefore helps describe your data file's organization.

If your dictionary contains more than one record, CSPro needs to be able to differentiate one record from another in the data file. "Record Type" provides the means for doing this.

For example, a typical Housing and Population census data file would most likely have a housing record (describing details of the living unit) and a person record (to describe details on each individual in the household). You could assign a Record Type of '1' to the Housing record and '2' to the Person record to distinguish between them.

If your dictionary contains only one record, you do not need to use a Record Type. Therefore, you can 'reclaim' the location that was set aside for the Record Type as follows:

- Select the (ID Items) set or the one-and-only record your dictionary contains from the dictionary tree.

- In the view on the right, you'll notice the first line is (record type). Only three values are used, **Starting_Position**, **Length**, and **Data Type**. Of these three values, you can only modify the start position and length. Change the length to 0. This will effectively "remove" the record type. (You can always reinstate it later by resetting the start position and length to non-zero values).

Similarly, if you would like to modify the length of the Record Type, proceed as above.

The record type value is always alphanumeric. Upper- and lowercase letters are distinct Record Type values (i.e., 'A' is not the same as 'a'). Blank is a valid Record Type value.

See also: Record Description, Record Properties, Required Record, Maximum Number, Labels, Names

Required Record

One of a record's properties is whether or not the record is **required**; the options are "**Y**es" or "**N**o". If a record is required ["**Y**es"], it means that for a given questionnaire, there must be at least one occurrence of this record. If there is not at least one occurrence of this record, the questionnaire will not be complete and the system will issue an error message to inform the keyer. If the record is not required ["**N**o"], the questionnaire may or may not contain an occurrence of this record. The questionnaire can be considered complete without an occurrence of this record.

Suppose you are designing a dictionary for a census. You'll probably have at least two types of records: one for the household, and one for each person in that household. You can have four scenarios:

- If you allow vacant housing units (i.e., you collect information on unoccupied housing units), then the household record **is** required and the person record **is not** required.

- If you allow homeless people, then the household record **is not** required and the person record **is** required.

- If you allow homeless persons and vacant housing units, then neither the housing record nor the population record will be required record types, but because the two conditions will never occur simultaneously, you will never have a questionnaire without one of the other type of record.

- If you allow neither homeless persons nor vacant housing units, then both the housing record and the population record will be required record types. This means that a valid questionnaire will always have one housing record and at least one population record.

See also: Record Description, Record Type, Maximum Number. Labels, Names

Maximum Number

This record property specifies, for the given record, the maximum number of occurrences of that record allowed in one questionnaire.

For example, suppose you are designing a dictionary for a census. You will probably have at least two types of records: one for the household, and one for each person in that household. There should be only one occurrence of the household record, but for the person record you will of course need more than one occurrence, as there will likely be more than one person in a household. Thus, the maximum for the person record could be 25, if limiting yourself to a family unit, or larger, if enumerating group facilities (military barracks, hospitals, mental institutions, etc.).

The maximum number of occurrences that may be specified for any record is 9,999. However, for greater program efficiency we recommend that you never have this many occurrences and that you keep the maximum to the lowest value that is appropriate for your particular application.

See also: Record Description, Record Type, Required Record, Labels, Names

Items

Item Description

An item describes the response to a question or helps identify the questionnaire. The item is the most basic element of a questionnaire: "age," "income," and "crop-code" are all examples of items. Related items should be placed in the same record. And, just like records and levels, data items possess properties (such as a unique name, label, etc).

An item can be redefined into sub-items

See also: Identification Items, Item Properties, Add or Modify Items

Identification Items

Identification items (i.e., ID items) are those data items that uniquely identify the questionnaire and define hierarchical levels. They are usually geographic items, such as Province, District, or Enumeration Area, or other unique identification, such as Survey ID Number. These data will appear on every record in a data file, as they are "common" to all of the records. The maximum number of ID items is fifteen.

If you are using absolute mode, you will typically want to structure your dictionary so that the ID items begin in column 2; in this way they will precede a record's data items (column 1 is usually reserved for the record type identifier). If you are using relative mode, you have no choice but to place the ID Items first; the system assigns a consecutive position in the order in which the items are created.

For a pictorial representation, open a dictionary in CSPro and press **Ctrl+L** (L=layout). Press **Ctrl+L** again to toggle this view off.

See also: Item Description, Sub-Items, Item Properties, Add or Modify Items

Subitems

This item property specifies whether the data is an **Item** or a **Subitem.**

Subitems allow items to be broken up into smaller pieces, or across broad categories. In this respect, they let you redefine data items and refer to the same data field in several different ways. The start position of a subitem must be within its parent item (the previous item).

One useful application of subitems involves date and time fields. A date item, for example, could be referred to as a single 8-digit entity: DDMMYYYY. However, this does not allow you to easily manipulate or refer to a portion of the date (such as the day, month, or year itself). Suppose you had the following definition for date (for demonstrative purposes, not all item properties are being shown):

Item Label	Item Type	Starting_Position	Len
Date of birth	Item	20	8

To redefine this item into subitems, you only need to add the following subitems:

Item Label	Item Type	Starting_Position	Len
Day of birth	Subitem	20	2
Month of birth	Subitem	22	2
Year of birth	Subitem	24	4

Another reason for using subitems is to make data references available across larger categories. Censuses and surveys often have items of three or four digits in length representing categories such as industry, occupation, or ethnicity. For occupation codes, the full value refers to a very detailed occupation,

such as bus driver. The first digit alone refers to the 'major' division, such as 'public service'. The first two digits together refer to a more detailed 'major' division, such as 'public transportation'. It may be useful to test the ranges with the CSPro language at the item level. In tabulation applications, tables can be made at the major (1- or 2-digit) or minor (3- or 4-digit) divisions. The following example could represent part of an economic survey:

```
Item Label              Item Type  Starting_Position Len
Occupation              Item       45                4
Occupation, Major       Subitem    45                1
Occupation, Sub-major   Subitem    45                2
Occupation, Minor       Subitem    45                3
```

In IMPS 3.1, the predecessor to CSPro, it was very common to use subitems to redefine data items. This is now more easily accomplished with value sets.

NOTE: Identification items cannot have subitems.

See also: Item Description, Identification Items, Item Properties, Add or Modify Items

Item Properties

You can view an item's properties by selecting the record to which it belongs (via the dictionary tree tab). When creating an item, the following fields must be set:

Property	Meaning
Label	A descriptive text label that identifies the item. It is used as default field text in data entry forms and in default titles in tabulation.
Name	The name given to this item for use in CSPro language procedures.
Start	Indicates the starting position of the item within the record
Len	Indicates the length of the data item (i.e., the number of characters necessary to represent the values for the item).
Data Type	Indicates the type of data (numeric or alphanumeric) that will be found in the item.
Item Type	Indicates whether the item is or is not subordinate to, or part of, another item. If the item is part of another item, it is considered a "subitem". If not, it is identified as an "item". **Identification items cannot have subitems**.
Occ	The number of times this item will repeat within the record. The default value is "1". **Identification items cannot have multiple occurrences.**
Dec	The number of decimal places (if any) in the item. The default number of decimals is "0". **Identification items cannot have decimals.**
Dec Char	This specifies whether the item should be stored in the data file with an explicit decimal character. This applies only to items or subitems which

	have been defined with the "Dec" property grater than zero (i.e., Dec >= 1).
Zero Fill	This item property states whether the numeric data item should contain leading zeros or blanks.

Press the **Esc** key to quit modifying without making changes. Press **Ctrl+Enter** to finish making changes. Use undo if you completed the modification incorrectly. There is no limit on the number of items within a record.

See also: Item Description, Sub-Items, Identification Items, Add or Modify Items

Starting Position

This item property indicates the starting location of a data item. In conjunction with the length property, it specifies the location of the item in a record. In absolute positioning mode, you cannot give a starting position that will cause the item to overlap with another item.

The start position of a subitem must be within its parent item (the previous item).

See also: Select Relative or Absolute Positioning, Item Properties.

Length

This item property indicates the total length of the data item (i.e., the number of characters necessary to represent the values for the item). In conjunction with the "**start"** property, it specifies the location of the item in a record. In absolute positioning mode, you cannot give a length that will cause the item to overlap with another item.

The maximum length of a **Numeric** item is 15 digits. The maximum length of an **Alpha** item is 255 characters.

See also: Item Properties

Data Type

This item property specifies what type of data (numeric or alphanumeric) that will be found in the item. The default item type is "Numeric".

- **Numeric** items can contain numbers or blanks, and they may be negative or positive in value. Numeric values will be right-justified and, if requested, zero-filled.

- **Alphanumeric** items can contain any character, letter, numeric digit(s), blanks, or special characters. These values will be left-justified and are blank-filled, whether or not "zero-fill" has been selected. Declaring 'M' or 'F' for gender is an example of an alphanumeric value.

Some responses are quantitative, such as size of farm, and some are qualitative, such as relationship to head of household. Most descriptive responses, such as 'head of household', are equated to numeric codes which are placed on the questionnaire. However, some descriptive responses remain as alphabetic text.

Numeric responses can be discrete values or continuous values. An example of a discrete value is gender, 1 (male) or 2 (female). An example of a continuous value is yearly income, which can range from zero to a value limited only by the number of digits permitted for the response. A discrete value may be used to represent a grouping of continuous values. For example, when asking income, one may be asked to select from a choice of ranges of incomes rather than specify the exact income. Thus, the possible responses to the income question could, for example, be a code between 1 and 10.

See also: Item Properties

Occurrences

This item property defines the number of consecutive repetitions of the item in the data record. The dictionary will reserve space equal to the product of the length of the item times the declared number of occurrences for the item.

For example: A census collects information on births and deaths, and each questionnaire can list the ages of up to a dozen household members who died during the past year. By defining an item "Age at death" with a length of 2 digits and 12 occurrences, the dictionary will reserve a location 24 characters in length for this item.

Be aware that if fewer than 12 people died in the household, then the unused portion of this item will be blank. If you have several items that use occurrences and they are often unused, you are increasing the size of your data file. Therefore, you should always specify the number of **occurrences** with care.

If an item has multiple occurrences, then its subitems may not have multiple occurrences. Conversely, if a subitem has multiple occurrences, then its parent item may not have multiple occurrences.

NOTE: ID items cannot have multiple occurrences.

See also: Item Properties

Decimal Places

This item property lets you specify how many digits of the numeric item represent the decimal portion of the item. CSPro does not expect the decimal point to be in the data file; if your data file does contain the decimal point, you will need to set the decimal character property. Therefore, the length of the item is not affected by the number of decimal places.

For example: Suppose you had two data files, each containing an item in the format "##.##". One file has an implied decimal point, the other file physically contains the decimal point. Here are the two ways to define the item (using 12.75 as an example)

```
Length  Dec  Dec Char
4       2    No       (decimal implied; number would appear as "1275")
5       2    Yes      (decimal present; number would appear as "12.75")
```

NOTE: ID items cannot have decimals.

See also: Item Properties

Decimal Character

This item property applies to those numbers specified as decimal. If the number is a decimal value, this states whether or not the decimal point is present in the data file. Your valid choices are:

• **Yes** the data file contains a decimal point for this item, or

• **No** the data file does not contain a decimal point for this item.

Note that if your item does not have a "numeric" data type, the Data Dictionary will not allow any value other than **No.** You can set this option for all items by clicking on "DecChar Default 'Yes'" on the Option menu.

See also: Item Properties

Zero Fill

This item property states whether the numeric data item should contain leading zeros or blanks.

For example: During data entry a numeric item with a length of 3 is encountered. A value of '92' was keyed. How will this value be stored in the data file?

- If zero-fill had been set to **Yes**, the value would appear as `'092'`
- If zero-fill has been set to **No**, the value would appear as `' 92'`

You can set this option for all items by clicking on "ZeroFill Default 'Yes" on the Option menu.

See also: Item Properties

Value Sets

Value Sets Description

Value sets let you specify one or more group of values for a data item or subitem. When using Tabulation applications or MapViewer, you will want to choose Value Set labels to tabulate/map, as it will give you more descriptive results. The resulting tables (or maps) will contain row and column labels (or region labels) that correspond to the value labels (or numeric distributions, if no value label is present). In a Batch Edit or Data Entry application, the use of value sets can help you when using the **vset** option to the impute function.

For example, suppose you have a survey that needs to classify people's ages three different ways: by discrete value, by 5-year cohorts, or by category, such as "Child," "Adult," etc. This is easily done by adding value sets for the AGE data item:

Value Set Label	Value Set Name	Value Label	From	To
Age	AGE			
			0	98
		Not Reported	99	
Age by 5 years	AGE_5YRS			
		0 to 4 years	0	4
		5 to 9 years	5	9
		10 to 14 years	10	14
		15 to 19 years	15	19
		20 to 24 years	20	24

```
                                    25 to 29 years      25    29
                                    30 to 34 years      30    34
                                    35 to 39 years      35    39
                                    40 to 44 years      40    44
                                    45 to 49 years      45    49
                                    50 to 55 years      50    54
                                    55 to 59 years      55    59
                                    60 years and over   60    98
Age by Category  AGE_CATEGORY
                                    Infant              0     0
                                    Child               1     12
                                    Teenager            13    19
                                    Adult               20    59
                                    Senior              60    98
```

The AGE item now has three defined value sets: AGE, AGE_5YRS, and AGE_CATEGORY. The first value set defines the acceptable range for data entry, while the second and third value sets give a breakdown as you might want to see the data tabulated.

The value set will always be added to the end of the item's value set listings. If you add to the wrong place, press the **<Esc>** key to stop the add. Use undo if you added at the wrong place.

See also: Value Set Properties

Value Set Properties

You can view a value set's properties by selecting the item to which it belongs (via the dictionary tree tab). When creating an item, the following must be set.

Property	Meaning
Value Set Label	A descriptive text label for a collection of categories of an item. Used by the Tabulation module to select items for tabulation and in table titles.
Value Set Name	The name of this item for use in the CSPro language procedures.

See also: Value Sets Description

Values

Value Description

A single value set can contain one or more values. To add multiple ranges to a value, enter one or more spaces as the value label on the next value(s), and the values that follow will become part of the previous value. Multiple ranges are indicated by the lack of a notes box at the beginning of the value line.

Zeroes should be avoided in assigning codes to identification items that identify geographic areas, because zeroes are used in CSPro to describe summarized geographic levels. If zeroes are already in the data, they can be recoded to other values using the CSPro logic.

You can assign a negative number to a value or the starting and/or ending value of a range. Negative numbers have a leading minus (-) sign. Positive numbers have no sign. The minus sign will be displayed in the data file immediately to the left of the value. If the item is "zero-fill", the minus sign will be displayed in the left-most position.

New values will always be added to the end of the existing value set listings. If you add to the wrong place, press the Esc key to stop the add. Use "undo" if you added at the wrong place.

See also: Value Properties

Value Properties

Property	Meaning
Value Label	The descriptive text for a single value or range of values. This label is used by the CrossTab module when creating column headings and stubs.
From	This is the single value, or starting value of a range associated with the value label. To add multiple ranges to a value, enter one or more spaces as the value label on the next value(s), the values which follow become part of the previous value. Multiple ranges are indicated by the lack of a "notes" box at the beginning of the value line.
To	This value is the upper limit of the range of values being defined. It must always be greater than the "From" value on the same line. Where only a single value is associated with the "value label," the "to" value may be blank.
Special	A numeric data item can be assigned one of three special values in the data dictionary. These are: "missing", "notappl", and "default".

See also: Value Description

Relations

Relation Description

Relations provide a way of linking one multiple record or item to one or more multiple records or items. For example, suppose a questionnaire contains two record types, child records and mother records. Each child record contains a data item that gives the sequence number of the mother record of the child's mother. A relation can be defined which links child records to mother records so that when data items from a child record are processes, the corresponding mother record data items are available for processing. Relations work much like database joins.

Relations have one primary multiple record or item. Each instance of the primary element in a case is processed one at a time. A relation has one or more secondary records or items. The corresponding secondary elements are linked to the primary element during processing.

There are four types of linking that can defined by relations.

Occurrence to Occurrence – Corresponding occurrences of the primary record or item and secondary record or item are linked, that is first occurrences are linked, second occurrences are linked and so on.

Item to Occurrence – The value of an item on the primary record is a pointer to the occurrence of the secondary record.

Occurrence to Item – The value of an item on the secondary record is a pointer to the occurrence of the primary record.

Item to Item – The value of an item on the primary record is compared to the value of an item on the secondary record. If the values are equal, the records are linked.

Relations can be used in **for** statements in batch programs and in the Export Data tool.

See also: Relation Properties

Relation Properties

Property	Meaning
Relation Name	The name of this item for use in **for** statements in the CSPro language.
Primary	This is the name of multiply occurring record or item. Items in the secondary are linked to items in the primary.
Primary Link	This is either (occ) or the name of an item. If the primary is a record, then this is the name of an item within the primary record. If the primary is an item, then this is a the name of a subitem within the primary item.
Secondary	This is the name of multiply occurring record or item. Items within the secondary are linked to item in the primary. The secondary cannot be the same as the primary.
Secondary Link	This is either (occ) or the name of an item. If the secondary is a record, then this is the name of an item within the secondary record. If the secondary is an item, then this is a the name of a subitem within the secondary item.

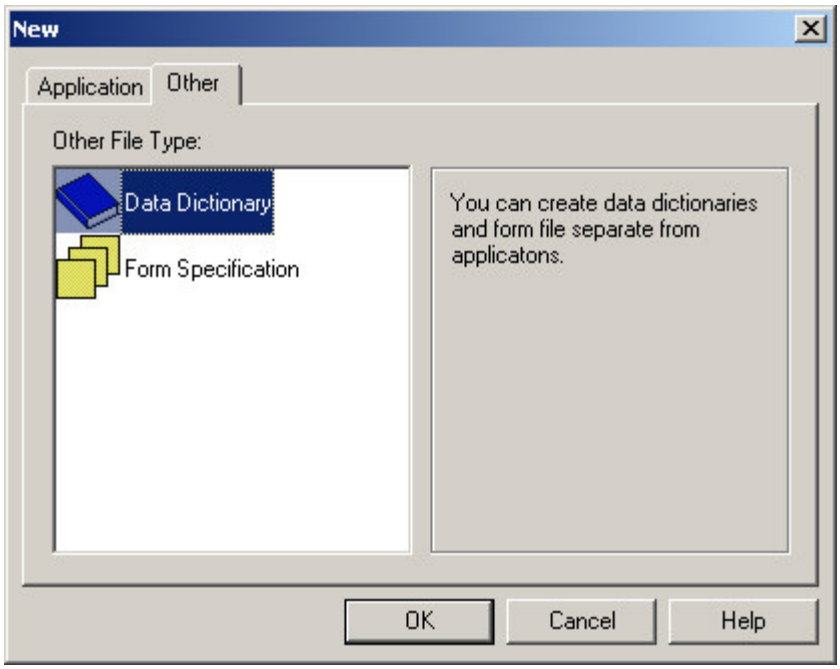
See also: Relation Description

Data Dictionary Application

Creating a Dictionary for a New File

To create a data dictionary application do the following:

• Click on the toolbar, or from the **File** menu, select **New**; or press **Ctrl+N**



• In the Object window select Data Dictionary

• Provide a name for the new dictionary (you need not provide the dictionary extension (.dcf); it will be automatically appended to the name).

• Select the folder where the dictionary (object) is to be stored. You can press the "**Browse**" button to locate an existing folder. If the dictionary file already exists, it will be used. If the dictionary file does not exist, it will be created.

• Press "**Next"** to advance to the Summary Screen and review your choices.

• If everything looks correct, press **Finish** to complete the operation.

If you are creating a dictionary to describe an existing data file, you may want to use absolute mode, in the event there are any "holes" in the data file(s). Or, if you want to use only a subset of the data file's information, using absolute positioning allows you to define only those data items of interest to you. If you are defining a dictionary for a new data file, you should be in relative mode, as this does not allow "holes" in your data.

If you have a long questionnaire, you can split the job and have several persons create data dictionaries for different sections. Later, you can copy and paste the items into one dictionary, making sure that the record type identification is unique.

Creating a Dictionary for an Existing File

To create a dictionary from an existing file you will need written documentation concerning the organization of the data on the file. This is usually presented as a set of record descriptions. These record descriptions identify the different kinds of records, the items on each record, the starting position and length of each item, type of data contained in each item, the values that can appear in each item and the significance of those values. Further, if you only want to use a subset of information in the data file, using absolute positioning allows you to define only those data items of interest to you.

Once you have the record descriptions for the data file you are ready to create the dictionary.

• With CSPro create a new Data Dictionary file.

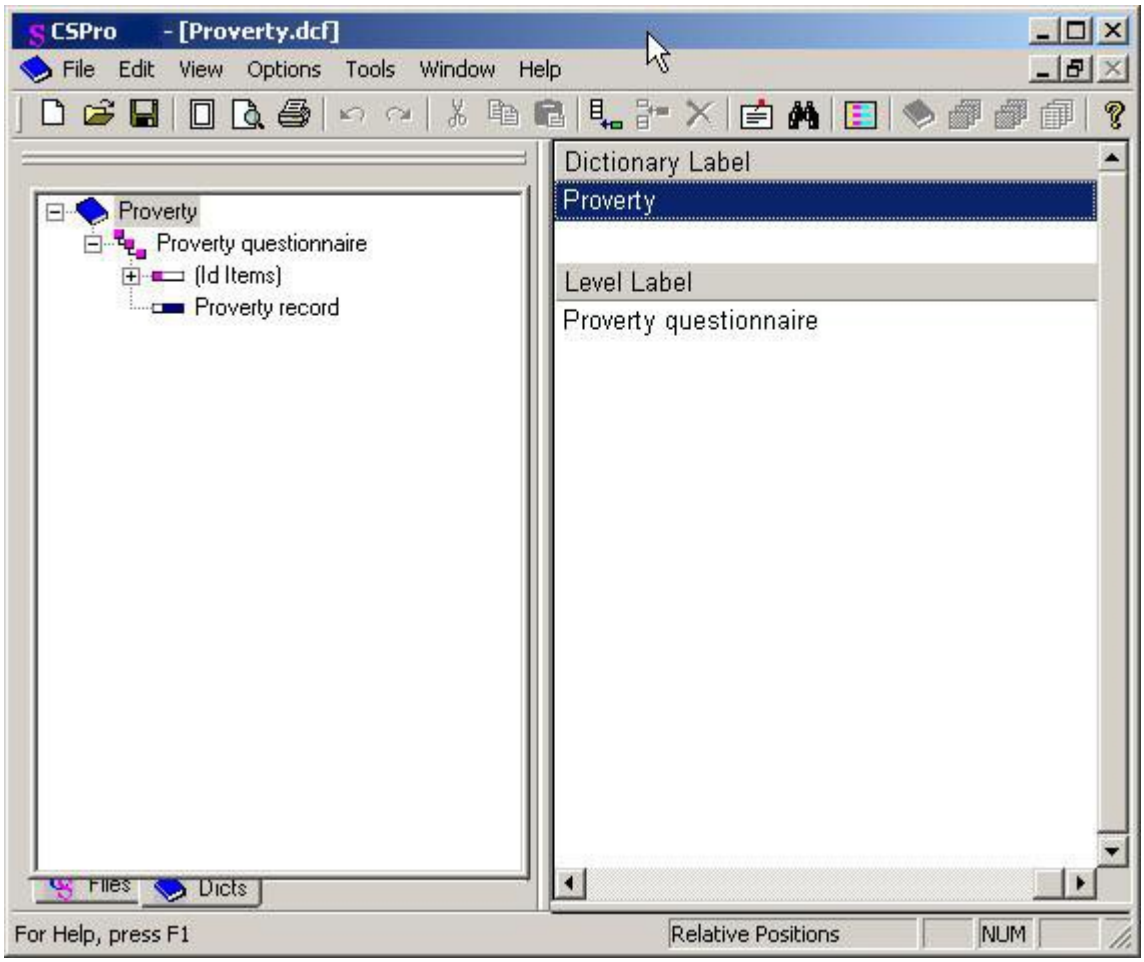
• In the Turn Options menu, uncheck the option for "Relative Positioning" so that you can position each data item according to the written specification.

• In the new data dictionary, define the records, items, and values from the record description.

See also: Select Relative or Absolute Positioning

Data Dictionary Screen layout

When finished creating the data dictionary application, you will see the screen divided into two windows. The screen on the left displays the dictionary tree:



CSPro created a dictionary ("Poverty") with one level ("Poverty questionnaire"), and that level contains a set of ID Items ("(Id Items)") and one record ("Poverty record").

The screen on the right displays detailed information for the highlighted object in the left-hand screen. For example, if in the left-hand screen the focus [cursor or highlight] is on the first line ( ), the right-hand screen will display information about the dictionary [file] as a whole. If the focus in the left-hand screen is moved to the second line ( ) questionnaire or case level, the right-hand screen will display information about the questionnaire [case], which is the basic element of the file. As the focus is moved down the

dictionary tree, the right-hand screen changes to reflect the different items of interest at each successive level.

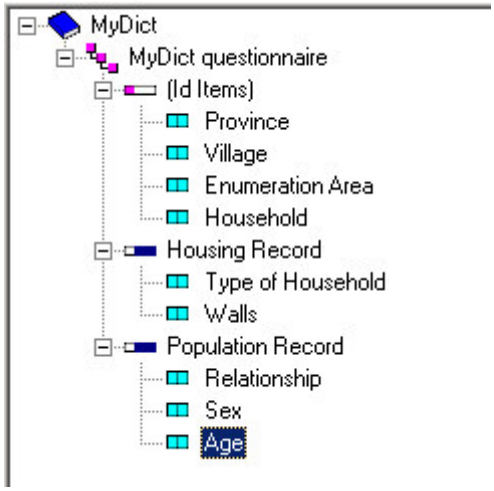
The tabs at the bottom of the right-hand display are marked "Files" and "Dicts". Clicking on either of these tabs will bring up the appropriate tree. In addition, each tree [**File Tree** and **Dictionary Tree**] can be toggled [using **Ctrl + T**] between views. In the case of the File Tree, the views will show either the internal or external names of the files; in the case of the Dictionary Tree, the views will show either the name or the label of each entity in the file.

The first thing we suggest you do is to change the level properties to reflect your intended usage for them. Next, change the record properties [e.g., maximum number of records for each type, whether a given record type is required or optional, etc.] and create any needed additional records for the level. Once you create a record, you can start creating the ID and data item value sets and values.

If this structure is sufficient for your needs, you can begin adding identification items and data items to the set(s) and each record you created. Remember that identification items defined in the ID Items set will appear on **each** record in the current level, as well as each record in lower levels.

See also: Add or Modify Levels, Add or Modify Records

Data Dictionary Tree



The Data Dictionary Tree contain the following items:

• **Data Dictionary File:**
This if the highest node, the identification of the data dictionary file.

• **Level:**
This is the second-tier tree node and identifies the level or questionnaire or case

• **Identification Item(s) or Record:** or
Represent the identification, or the record

• **Field:**

Represent the individual items. If the item has more than one value set, then the field will show the symbol "+" indicating that the item can be expanded. Value sets are always designated with the symbol . If the item contains sub-items, these will be designated with the symbol .

Relative and Absolute Mode

The mode refers primarily to the start positions of your data items. You can create a dictionary either in **relative** or **absolute** mode.

• Relative mode

- Use relative mode when designing a new data file; you do not normally want "holes" (unused space between items) in your data files, as this will increase the size of the file.

- The record type, if present, is always the first item in every record.

- ID items, if any, are always located after the record type (and other ID Items defined at a higher level).

- Each data item will be placed after **all** defined ID items (even those defined at a higher level than the record) in the record.

- There are no gaps or "holes" between items.

- As items are added, inserted, modified, or deleted, other items are automatically moved as needed to maintain the above arrangement.

- Changing the starting position of an item will move it and any following items to give the implied relative arrangement.

• Absolute mode

- Use absolute mode to create a dictionary from an existing data file.

- The record type, ID items and data items can be positioned at any location in a record.

- All items will remain in their assigned locations, unless specifically moved by the user.

- When inserting or adding an item, there must be room (i.e., a "gap") for the item at the specified location.

- When items are deleted, gaps may be created.

- When an item's starting position or length is changed, room for the item must exist.

Dictionary Types

Every dictionary associated with an application has a type value that indicates how it is being used. For the primary dictionary (i.e., the one upon which your application was created), this will be your **main** dictionary. Other dictionaries (ones that are inserted either directly or secondarily via a forms file) can have additional properties, as explained below.

To see your dictionary's type, go to the Files tab, right-click on the dictionary in question, and select "Dict Type." You will then see the following four choices (which may or may not be active, depending on their use):

• Main

This is the principal dictionary upon which the application was built. You cannot give the dictionary another status as it will always be the primary dictionary for the application.

- **External**
  When you add a dictionary to an application, its type can either be "external" or "working". If it is an external dictionary, it must have an associated data file. When external dictionary variables are used in an application, their default values will be Not Applicable (Notappl).

- **Working**
  When you add a dictionary to an application, its type can either be "external" or "working". If it is a working dictionary, it does not need an associated data file. When working dictionary variables are used in an application, their default values will be blank (if the variable type is alphanumeric) or zero (if the variable type is numeric).

- **Special Output**
  This option is provided for backward compatibility with ISSA Batch Edit Applications. Only non-primary dictionaries used in Batch Edit Applications can have a "special output" type. Refer to the ISSA Manual for further instruction.

Reconciling Dictionary Changes

Whenever you make changes to a Data Dictionary, CSPro must reconcile the changes in applications that use the dictionary. If the application is open when you make the change, CSPro automatically makes the change in your application. If the application is not open, CSPro will attempt to make any changes the next time you open the application.

Under some circumstances CSPro will ask you to assist in the reconciliation process. You may asked whether you want to delete item from a form or rename the item, that is, use an item with a different dictionary name.

To rename the item, select "Rename" and then choose the new item name from the list presented. To delete the item, select "Delete."

How to ...

Open an Existing Dictionary Application

You can display several data dictionary files in the tree. Click on the toolbar; or from the **File** menu, select **Open**; or press **Ctrl+O**. Select any file with extension .DCF. If you already have a data dictionary application open, the new dictionary will be added to the Files tree.

You may open a data dictionary and make changes to it, even if it already belongs to an application. Be aware that if you later open an application to which it belongs, CSPro will automatically make necessary adjustments in other files. For example, if you delete or rename a dictionary item, then later open an application that uses this modified data dictionary, any corresponding fields on the data entry forms will be deleted.

Move Around a Dictionary

Press To

Up Arrow Move up one line

Down Arrow Move down one line

Page Up Scroll up one screen (if possible)

Page Down Scroll down one screen (if possible)

Ctrl+Home Jump to first record, item, or value (from the view only)

Ctrl+End Jump to last record, item, or value (from the view only)

Ctrl+Left Arrow Scrolls left (if possible, and from the tree only)

Ctrl+Right Arrow Scrolls right (if possible, and from the tree only)

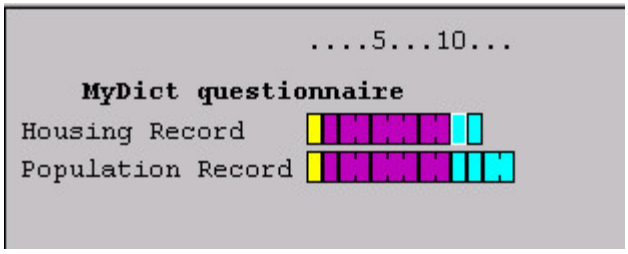
Ctrl+Up Arrow Multi-selects rows (from the view)
Scrolls up (if possible, and from the tree only)

Ctrl+Down Arrow Multi-selects rows (from the view)
Scrolls down (if possible, and from the tree only)

See also: Add, Modify, Delete Dictionary Elements. Toolbar Summary

View the Dictionary Layout

From the toolbar, press , or press **Ctrl+L**, or select Layout from the View menu. Any of these options will toggle the display of the current layout of the data dictionary. The display will appear in the lower half of the right-hand screen.



The layout shows you where, physically, each item in each record is located, how much space has been allocated to it, and if there are any gaps in your file (which is possible when the file's status is absolute). The color scheme used is the following:

denotes the Record Type

denotes Id Items

denotes record Items

denotes Subitems

• Click on the item on the layout window to move the cursor to the specific item in the dictionary window

• Single click on an item to move to the item's definition.

• Double click on an item to show its value set(s).

• Press **Ctrl+L** a second time to close the view.

Add Dictionary Elements

If you wish to add a level, record, item, value set, or value to a dictionary, first position the cursor in the location where you want to add the dictionary element, then click on the toolbar; or press **Ctrl+A**; or, from the **Edit** menu, select the "Add [element]" option. You can add from either the tree or view — in either case, the dictionary's menu bar and pop-up menu listing (displayed if you right-click over a tree item or the view) are context-sensitive. If you add to the wrong place, press the **Esc** key to stop adding.

See also: Modify Dictionary Elements

Modify Dictionary Elements

You can modify any of the dictionary's items (i.e., a level, record, item, value set, or value). You can modify an item from either the tree or view — in either case, the dictionary's menu bar and pop-up menu listing (displayed if you right-click over a tree item or the view) are context-sensitive. Therefore, depending on what you've selected, your choice will be to modify the properties of a level, record, item, value set or value. The **Ctrl+M** key combination invokes the "modify" process; or, from the **Edit** menu, choose the "Modify [element]" option.

See also: Add Dictionary Elements

Add or Modify Levels

Most of the applications will only use one level. If you need to add additional levels (recommended only for more complex censuses and surveys), do the following:

- From the dictionary tree, select any level within the dictionary
- Right-click to get the pop-up menu and select "Add Level", or press **Ctrl+A;** or select "Modify Level", or press **Ctrl+M**.
- Complete the level properties requested
- When you are finished entering the level(s) desired and wish to terminate data entry, press the **Esc** key.

After entering the second level, the 'add level' mode will continue for one additional level. To terminate with just two levels, press **Esc** when you reach the (third) new level entry. Additional levels will have exactly the same structure as the first one, i.e., an (ID Items) set and a record ('New Record'). You can undo modifications if you decide they were incorrect.

If you need additional records for this level, you should create them first. After selecting a level in the tree, you can press to initiate add mode. The level will always be added at the end of the dictionary.

See also: Dictionary Hierarchy, Level Description, Level Properties

Add or Modify Records

- From the dictionary tree, select any record (or (Id Items) set) within the level you wish to add or modify.
- Right-click to get the pop-up menu and select "Add Record", or press **Ctrl+A**; or select "Modify Record", or press **Ctrl+M**.
- Complete the record properties requested.

• When you are finished entering the record(s) desired and wish to stop adding, press the **Esc** key.

If this structure is sufficient for your needs you can begin adding identification items and data items to the record. The item will always be added at the end of the record.

There is no limit on the number of records within a level.

See also: Dictionary Hierarchy, Record Description, Record Properties, Record Type, Required Record, Maximum Number

Add or Modify Items

• From the dictionary tree, select the item within the ID Items or record you wish to add or modify.

• Right-click to get the pop-up menu; select "Add Item", or press **Ctrl+A;** or select "Modify Item", or press **Ctrl+M**.

• Complete the item properties requested.

• When you are finished entering data items and wish to stop adding, press the **Esc** key.

There is no limit on the number of items within a record. The item will always be added at the end of the record. After selecting an item in the tree, you can press to initiate add mode.

If you add to the wrong place, press the **Esc** key to stop adding. Use undo if you added at the wrong place.

Add or Modify Value Sets

• From the dictionary tree, select the item or subitem to which you wish to add a value set.

• Right-click to get the pop-up menu and select "Add Value Set", or press **Ctrl+A**; or select "Modify Value Set", or press **Ctrl+M**.

• Provide the Label and Name for the Value Set.

• Complete the value set properties requested.

• When you are finished entering values and wish to stop adding, press the **<Esc>** key.

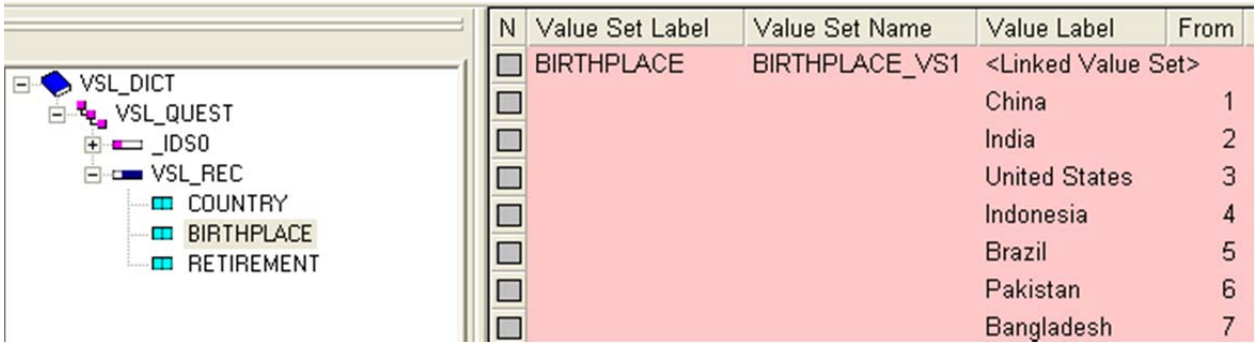
Linked Value Sets

There are times when several items in a dictionary share the same list of possible values. For example in a survey that asks for the country where someone lives as well as the country where that person was born, the value set for both items would be a listing of all countries. A user can enter the possible values for one item and then copy and paste them onto the second item, but the concept of linked value sets is a better approach to the problem. With linked value sets a user pastes a "value set link" which indicates to CSPro that the two items share the same possible values. Then if the user adds or modifies entries for one item, the values for the linked items are also modified. Users do not have to update multiple value sets to make changes.

Create a Value Set Link

Create the value set for one of the items and then copy the value set to the clipboard. Go to the value set window for the item to which you want to link the value set. Right-click on the window and select "Paste Value Set Link."

After pasting the link the value set will turn pink as a way of indicating that the value set is shared across two or more items. Modify or add values to the value set as you would for a regular value set.



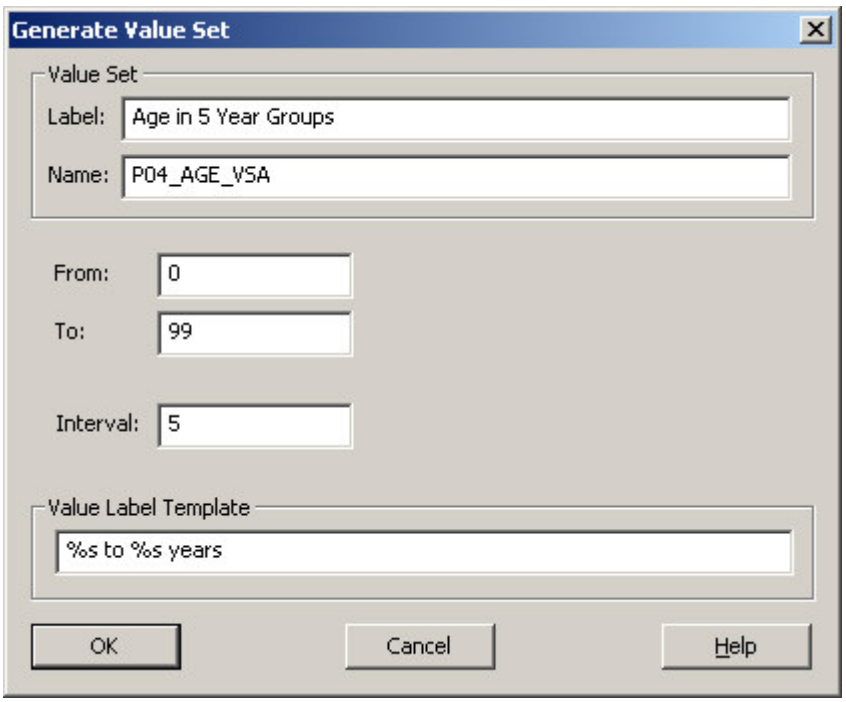
Remove a Value Set Link

Right-click on the value set name and select "Remove Value Set Link." If the value set is linked to more than two items, you can chose to remove the link for only the item that you right-clicked on, or you can remove all the linkages. When a link is removed the value set is no longer colored pink. The values remain but now modifying the value set does not affect the value sets in the items to which it had previously been linked.

Generate Numeric Value Set

To generate a value set for a numeric data item:

- From the dictionary tree, select the desired item.
- Right click on the view on the right, or select the **Edit** menu.
- Select **Generate Value Set**. The following dialog box will appear.



Value Set Label: A descriptive text label for a collection of categories of an item. Used by the Tabulation module to select tabulation categories. This cannot be left blank.

Value Set Name: The name of this value for use in the CSPro language procedures.

From: The smallest value in the value set.
To: The largest value in the value set.
Interval: The size of the interval in each value range generated. For example, if 5 year age groups are desired, then the interval is 5. The interval must be greater than 0. If the interval is the smallest positive number that can be contained in the data item, then single values are produced.

Value Label Template: A model for descriptive text of each value or value range that is generated. The characters **%s** in the template are replaced by the **from** and **to** values for each value range. All other characters are output as entered.

Add or Modify Values

• From the dictionary tree, select the desired value set.

• Select one of the value set's values in the view on the right.

• Press **Ctrl+A** to begin adding a value.

• Complete the Value Properties requested.

• When you are finished entering values and wish to stop adding, press the **Esc** key.

A single value set can contain one or more values. The value will always be added to the end of the value set listings. If you add to the wrong place, press the **Esc** key to stop the add. Use undo if you added at the wrong place.

Undo and Redo Changes

CSPro keeps track (in an "undo stack") of the 12 most recent changes you have made to your application. Please be aware that not all changes can be undone.

If you have made a mistake and want to undo it, press on the toolbar; or from the **Edit** menu, select **Undo**; or press **Ctrl+Z**. CSPro will try to restore your forms to the state previous to last change you made. To undo the next-to-last change, press the **Undo** button again.

Sometimes you may undo several changes and realize you have gone too far back. Press on the toolbar; or from the **Edit** menu, select **Redo**; or press **Ctrl+Y**. Redo is an "undo" of an undo.

Select Several Dictionary Elements

You can select several dictionary elements of the same type from the dictionary window.

• **Using the mouse:**
- Click on the line where you want to start selecting.
- Hold the left mouse button down and drag the mouse up or down until your desired selection is highlighted. Note that the window will automatically scroll if necessary.
- Release the mouse button.

• **Using the keyboard:**
- Using the cursor keys, move to the start of your desired selection, so that the blue highlight bar is on that line.
- Press and hold the **Shift** key.
- Use the **Up** and **Down** arrows to expand your selection. PgUp and PgDn will expand the selected lines a page at a time.

You can use cut and copy to move/copy your selection elsewhere within the dictionary, or to use in another open dictionary. You can delete multiple records, items, or values at the same time. Undo can be a useful feature when dealing with block operations.

Insert Dictionary Elements

If you wish to insert a level, record, item, value set, or value to a dictionary, first position the cursor in the location where you want to insert the dictionary element, then do one of the following: click on the toolbar; or press the **Insert (Ins)** key; or, from the **Edit** menu, select the "Insert [element]" option. You can insert from either the tree or view — in either case, the dictionary's menu bar and pop-up menu listing (displayed if you right-click over a tree item or the view) are context-sensitive. If you insert to the wrong place, press the **Esc** key to stop inserting. Use undo if you completed the insert to the wrong place.

Delete Dictionary Elements

If you wish to delete a level, record, item, value set, or value from the dictionary, first position the cursor in the location where you want to delete the dictionary element, then do one of the following: click on the toolbar; or press **Delete/Del,** or select **Delete** from the Edit menu.

If you delete the wrong object, click on the toolbar to undo the operation and recover the deleted material. You can select multiple lines by dragging the mouse over the desired lines or pressing down on the **Shift** key while you use the up or down arrow to adjust the selection.

Move Dictionary Elements

To move things in the Dictionary around use "cut," "copy," and "paste." "Cut" will delete the material from the dictionary and place it on the clipboard. "Copy" will simply place a copy of the selected material on the clipboard. "Paste" will place a copy of the material on the clipboard into the dictionary. You can paste cut or copied material to more than one location. You can also copy/cut dictionary elements from one dictionary and paste into another.

- **To cut things …**

  Select the material you want to cut, then click on the toolbar; or from the **Edit** menu, select **Cut**; or press **Ctrl+X**.

- **To copy things …**

  Select the material you want to copy, then click on the toolbar; or from the **Edit** menu, select **Copy**; or press **Ctrl+C**.

- **To paste things …**

  Select the place where you want the records, items, or values to be pasted then click on the toolbar; or from the **Edit** menu, select **Paste**; or press **Ctrl+V**.

Find Dictionary Elements

Press on the toolbar, or press **Ctrl+F**; or select **"Find"** from the **Edit** menu to open the "Find" dialog Box. Names and labels of all dictionary entities (for example, levels, record, items, value sets) will be searched. The following options allow you to search for text:

- **Find What**
  Enter all or part of the text string to search for. Text used in previous searches is available by clicking on the down arrow and selecting from the dropdown list.

- **Next**
  Find the next occurrence of the text string, starting from the last one found. If it finds the item, it will be brought into focus in the view; otherwise you will receive a notification that it could not be found.

- **Prev**
  Find the previous text string starting from the last one found.

- **Match Case**
  If this option is checked, the string will match only if the letters are the same case (upper or lower) as in the string you entered as the search key. If this option is not checked, the search will ignore case.

- **Close**
  Close the "Find" dialog box.

Document Dictionary Elements

In the view screen on the right, select the element you want to document or for which you want to modify the documentation. Click on the toolbar; or from the **Edit** menu select **Notes**; or press **Ctrl+D**. You can also press the button at the beginning of the line of the selected element, or right click and select "Notes."

You can use the **Enter** key to end a paragraph and begin a new one within the note. You can use **Ctrl+X**, **Ctrl+C**, and **Ctrl+V** to cut, copy, and paste text in the note.

Convert Items to Subitems

Select the items you want to convert to subitems. From the **Edit** menu, select **Convert to Subitems** or right-click on the item list in the view and select **Convert to Subitems** from the pop-up menu. Enter information about the item that will include these subitem(s).

To convert subitems back to items, delete the item. When asked if you wish to "Delete subitems too?" answer **No**.

Select Relative or Absolute Positioning

To toggle between Relative and Absolute positioning, select **Options** from the Menu bar, then select Relative Positions. A check mark indicates your file is in Relative Positioning; the absence of a check mark indicates the file is using Absolute Positioning.

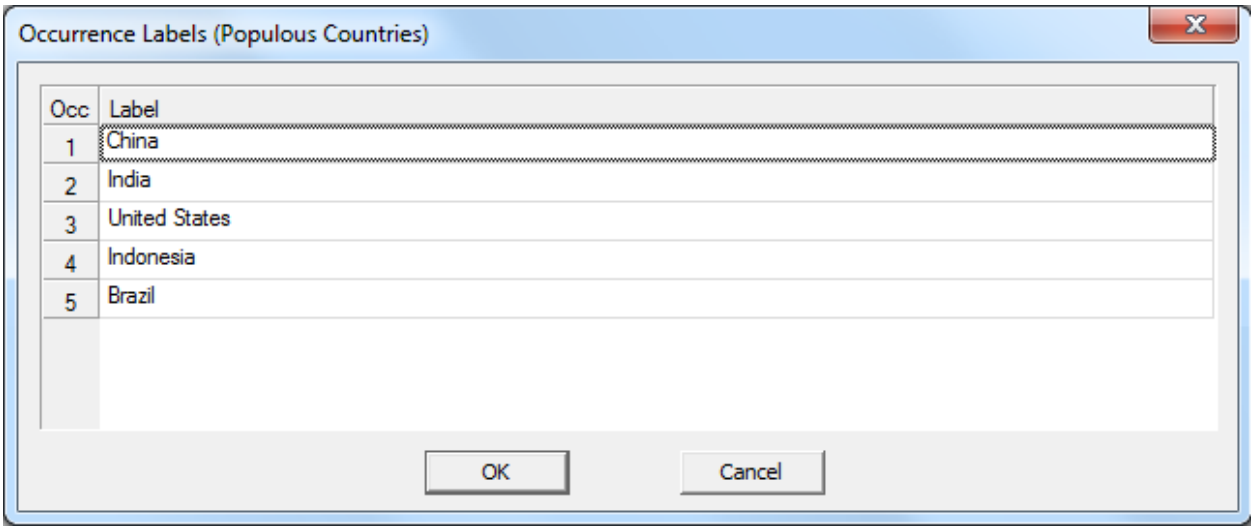
Create Dictionary with No Record Types

If your dictionary has only one type of record, there is no need for a **Record Type** item in the record. In order to remove the **Record Type** item, set its starting position to 0 and it length to 0.

N	Item Label	Item Name	Start	Len	Data Type	Item Type
	(record type)		0	0	Alpha	
	Province	PROVINCE	1	2	Num	Item
	District	DISTRICT	3	2	Num	Item
	Village	VILLAGE	5	3	Num	Item
	Enumeration area	EA	8	3	Num	Item
	Urban/Rural	UR	11	1	Num	Item
	Building number	BUILDING	12	3	Num	Item
	Housing unit	HU	15	3	Num	Item
	Household number	HH	18	1	Num	Item

Occurrence Labels

Occurrence labels can be attached to a repeating item or a repeating roster by right-clicking on the dictionary element and selecting "Occurrence Labels." Alternatively, that option can be found on the Edit menu. To edit an occurrence label, simply click on the appropriate row and add the label.



Occurrence labels can be used as row labels in a roster, in programmed logic, or as replacement text in the CAPI text window.

Add or Modify Relations

- From the **Edit** menu, select **Relations**.
- Press the **Add** button to add a relation at the end of the list or press the **Insert** button to insert a relation at the current highlighted line.
- Enter the Relation Properties across the line.
- Press the **Delete** button to delete the highlighted line.
- Press OK button when you have completed your edits.

Print the Dictionary File

Click on the toolbar; or from the **File** menu, select **Print**; or press **Ctrl+P** to open the Print dialog box and select among the following options:

• **Where**
You can send your output directly to a printer, or save it in a file.

• **Detail**
You can select **Complete** to print the data dictionary with the value sets and notes; or **Brief** to obtain a listing of the items only.

• **Order**
This option allows the user to place either the item name or the item label in the first column of the listing.

You can also print part of a document by selecting the desired text and then pressing **Ctrl+P**. To preview the printing click on the tool bar; or from the **File** menu, select **Print Preview**.

Save Dictionary As New File

To save a dictionary under a new name:

- Open the dictionary alone, that is, not as part of an application.
- From the **File** menu, select **Save As**.

- Enter the file name for the new dictionary in the file open dialog box.

When the Save As is complete you will be editing the new dictionary.

See Also: Save an Application with a New Name

The CSPro Language

Introduction to CSPro Language

The CSPro language lets you write programming logic for your Data Entry and Batch Edit applications. In Data Entry applications you can write logic to control and check the keying operation as it progresses. In Batch Edit applications you can write logic to identify and correct errors after data capture is complete.

This section contains the following information:

Data Requirements
The CSPro Program Structure
Declaration Section
Procedural Sections
Logic
Language Elements
Variables and Constants
Expressions
Operators
Files

For a list of commands see CSPro Statements and Functions

Data Requirements

Data files appear in many different formats and structures, but all data files used by CSPro must be text files (i.e., you must be able to view them in a text editor—they can not be encrypted in any way). If you are using data files created by another software package, you must save the data in a separate text file before you can use it with CSPro. Data files are limited to 2 gigabytes in overall size; the maximum length of any record in the file is 32,000 characters. CSPro encodes data files using UTF-8. Read more about Unicode text files at the Unicode Primer.

CSPro processes one case at a time. Each record must contain a unique questionnaire identification code in the same position in each record. This number must be the same for all records of the same case. If the file is a ‘flat’ file, meaning that each questionnaire contains only one record, the questionnaire identification becomes irrelevant. In this case, any data item can be used as the questionnaire identification, but it should be unique for each record. CSPro uses the case identification values to determine where one case ends, and the next one begins. Records belonging to the same case must be contiguous within the data file, but there is no requirement that the data file be sorted by case identifier.

CSPro can handle a data file with multiple record types -- for example, housing and population -- but a record type code must identify the type of record. This code must be in the same position in each record. Within the same record type, each data field must be in the same position.

CSPro can process one input data file at a time, but it can access one or more external files. These files must also be described in a data dictionary and must be in text format.

In some survey data, especially where the total number of data items is great but only a few responses are expected, the user may choose a format in which each data field is preceded by a ‘source code’

relating it back to the original document. By using this scheme, non-response fields (empty responses) need not be entered. With this type of format, each data field is not in a pre-defined location on the record. Before a file like this can be processed by CSPro, it must be reformatted so that the data fields are in fixed positions. Items in data files must be fixed format, that is, items must have the same starting position and length in every record where they occur.

See also: Data File Type Structure

The CSPro Program Structure

CSPro logic consists of a collection of events defined as procedures. Each procedure performs the operations you specify using CSPro statements and functions written in the CSPro Language. A CSPro program includes a declaration section and one or more procedural sections.

- **Declaration Section (PROC GLOBAL)**

The declarations and definitions are defined in the **global procedure**. In this section you declare the mode of operation (implicit or explicit), variables, arrays, and user-defined functions. The global procedure always appears at the beginning of the logic file and begins with the line "`PROC GLOBAL`". Except for within user-defined functions, there are no executable statements in this section. The global procedure is equivalent to the ISSA application procedure. You can edit the `PROC GLOBAL` section by clicking on the topmost entry of the data entry edits tree or batch edits tree.

Example:

```
PROC GLOBAL

set explicit; {mode}
numeric x, xage;  (numeric variables)
alpha flag; (alphanumeric variable)
array Relly(5); (numeric array)

function InitRellyArray ();  {user-defined function}
 Relly (1) = 3; { child of head }
 Relly (2) = 4; { parent of head }
 Relly (3) = 9; { grandchild of head }
 Relly (4) = 8; { grandparent of head }
end;
```

- **Procedural Section**

This section contains executable statements and assignment statements that can be written before (preproc) or after (postproc) an event. Events always fall under the Proc section, which is followed by the name of the forms file, level, form, roster, or field. Statements are assumed to be in the postproc unless it is explicitly stated that they are in the preproc. The statements can be included in the level, form, roster, or field events.

A data entry application will have, in addition, a forms file procedure. The form file preproc is executed before any data are entered, and the postproc is executed after all data for the file are entered. In a data entry application the forms, rosters, and fields procedures can also have onfocus and killfocus statement.

Example:

```
PROC MYDICT_FF (File proc)
Preproc (File preproc)
InitRellyArray();  (Call Function)

PROC MYDICT_QUEST (Level proc)
```

```
Preproc (Level preproc)
<statements>
Postproc (Level postproc)
<statements>

PROC HOUSING_FORM (Form proc)
<statements> (Form postproc (implicit))

PROC Income (Field proc)
Onfocus (Field onfocus)
<statements>

Postproc  (Field postproc)
<statements>
```

See also: Order of Executing Data Entry Events, Order or Executing Batch Edit Events

Debugging CSPro Programs

There is no structured debugger that works with CSPro. However, there are several techniques that can be used to find problems and solve them. One technique is to "comment out" code. By enclosing specific code between braces { }, you can isolate specific areas of code that may be the cause of your problem.

Another technique is to use error messages, with the errmsg function. The generation of an error message tells you that program control has passed to that point in your application. The error message can also include variable values to indicate the status of those variables at that point.

It may sound obvious, but it is good programming technique to "indent" your code. This will help you find problems caused by unterminated if or do statements, for example. If you consistently indent (or tab) the content within any control structure, Finding a lost endif or enddo will be much easier.

It is a good idea to keep the set Explicit option turned on (on the Option menu, mark Set Explicit). This will force you to declare all of your variables in the PROC GLOBAL section of your application before using them. The Set Explicit option will prevent the use of misspelled variable names, a common cause of subtle errors that are difficult to detect. Note that this option can also be turned on in your code using:

```
SET EXPLICIT;
```

in the PROC GLOBAL section of your application.

You can also use the trace function to determine problems in your application.

Declaration Section

Compiler Mode

The CSPro compiler operates in one of two modes:

- **Explicit mode**
  You must declare all variables not defined in your dictionary; otherwise, the variables will be flagged as errors by the compiler. The advantage of this mode is that you do not have to worry about misspelled names. The default compiler mode is explicit, which means that any variable used in a program must be declared in a numeric or alpha statement.

- **Implicit mode**
  This allows you to declare a variable "on the fly", i.e., anywhere in your program. For example, simply coding "myvar = 3;" in any procedure or function automatically declares a numeric variable "myvar". All such declarations are global in scope, meaning you can assign or get the value from any other procedure. User-Defined Functions, string variables, and arrays must still be declared in `PROC GLOBAL`. The advantage of this mode is that you can write your code more quickly. The danger in using this mode is that you may misspell the name of a variable or dictionary item. If you do this, the compiler will create a separate variable for the misspelled name. For example, you may code "if mivar = 3 then…" and the compiler will create a new variable "mivar", with initial value 0, and therefore evaluate the condition as false.

You can change the default mode on your computer by checking or unchecking the "Option/Set Explicit" setting. This setting will then remain in effect for all applications. Note that this setting is in effect only on your computer; if you move your application to another computer with a different setting, you may get a different result when you compile.

You can override the computer's default setting mode by using the set statement in PROC GLOBAL. In this case your application will always give the same result on any machine.

See also: Set Compiler Defaults

Variables

In CSPro you can declare numeric or alphanumeric variables. Variable names must contain only letters, numbers, or the underscore ('_') character, and must begin with a letter. Names can be up to 32 characters long and are case insensitive, that is, uppercase and lowercase letters are considered the same. For example, "myvar", "MYVAR", and "MyVar" are all equivalent. Variables of all types follow the same rules for names.

- **Numeric Variables**
  In CSPro, numeric variables are stored internally in floating point format. They can accommodate numbers of extremely small or large size, positive or negative. Numeric variables are global in scope, meaning you can assign or get the value of a variable from any other event. A numeric variable may be up to 15 digits in size. It is equivalent to a float or double variable.

  If you include set implicit in the global procedure, then you may declare a numeric variable "on the fly" (i.e., variables not associated with any dictionary) in any part of the program. If you include set explicit (or permit it as the default option) in the global procedure, then you must declare all numeric variables in the `PROC GLOBAL` section using the numeric statement.

- **String Variables**
  String variables in CSPro store alphanumeric data. You must declare a string variable in the global procedure, using either the alpha statement or the string statement. This is true whether you have chosen "Set explicit" or "Set implicit" for numeric variables.

See Also: Mode, Arrays, Set Statement, Text Strings

Arrays

CSPro supports numeric or alphanumeric arrays of up to three dimensions. You must declare arrays in the global procedure, using the array statement. Only one variable can be defined in each array statement.

Array names can contain only letters, numbers, or the underscore ('_') character, and must begin with a letter. They can be up to 32 characters long. Array names are not case sensitive, meaning that uppercase and lowercase letters are considered the same. For example, "myvar" and "MyVar" would refer to the same variable.

Whenever the array variable is used in the application, a value or numeric expression for each dimension must be given. The initial array contents are zero (if numeric) and blank (if alphanumeric) until a value for each dimension is assigned.

Files

Files whose structure is defined by a data dictionary are automatically declared by their dictionary names. Files that do not have associated data dictionaries and need to be named at run time are defined by the file statement. Such files can be used in the file manipulation functions such as filecopy, filedelete, fileread, filewrite, etc. They can also be used in export statements.

The file statement gives a name to a file that is local to that application. The physical name of the file is requested in the files dialog box when the application is run.

User-Defined Functions

User-defined functions are coded in the declaration portion (`PROC GLOBAL`) of an application. Once defined, they can be used anywhere in an application. Functions are used to perform operations that are used in several different places in an application.

Functions are of the form:

Return-value = function-name(parameter-list)

Functions must include a parameter list which can vary depending on the function call's requirements. This list may be null (that is, it contains no parameters between the opening and closing parentheses) or it may contain one or more parameters. Each parameter specifies a variable or array that is used by the statements within the function. Numeric and alphanumeric variables are local to the function. That is, if a variable is passed as a parameter, its value in the rest of the application will not be changed by actions within the function. On the other hand, arrays passed as parameters refer to the source array and changing values of the array within the function also change the source array.

A user-defined function:

- Returns a single value, either numeric or alphanumeric.
- Can contain CSPro statements and functions, as well as other user-defined functions. If no return value is assigned to the function, a **default** value is returned.
- Cannot be recursive (i.e., it cannot call itself though it can call other user-defined or system-supplied functions).

The Function Statement allows the creation of a user-defined function.

Alias Statement

Format:

```
alias aliased-name : dictionary-name;
```

Description:

The **alias** statement allows for the creation of new names to alias, or to provide an alternative reference to, names in a program. This can be used to shorten, or lengthen, variable names, or to match names in a dictionary with preexisting code using a certain naming convention.

Example 1: Shortening dictionary names to quicken the typing of variables

```
alias   P14 : P14_AGE,
        P15 : P15_RELATIONSHIP,
        P16 : P16_SEX;
```

Example 2: Using aliases to standardize variable names so that logic can be reused across programs

```
alias   AGE             : P14_AGE,
        RELATIONSHIP    : P15_RELATIONSHIP,
        SEX             : P16_SEX;
```

In both examples, P14_AGE, P15_RELATIONSHIP, and P16_SEX are the names of items declared in a dictionary. Once the alias statement has been specified (in PROC GLOBAL), the names P14 and P14_AGE can be used interchangeably. For example, these two statements are identical:

```
if AGE < 18 and SEX = 2 then

if P14_AGE < 18 and P16_SEX = 2 then
```

Procedural Sections

Statements

A procedure contains a series of statements. Each statement is a complete instruction to the computer.

- **Executable Statements**
  Begin with a command and ends with a semicolon (;). They are made up of a combination of commands, keywords, expressions, and functions.

  Example:
  ```
  skip to Q103;
  ```

  Skip is a command, **to** is a keyword and "**Q103"** is the name of a data entry field.

- **Assignment Statements**
  The assignment statement sets a variable equal to the value of an expression and do not contain commands. If the expression is a string-expression, then the variable must be alphanumeric. If the expression is numeric or conditional, then the variable must be numeric.

  Format:
  numeric-variable = numeric-expression;
  string-variable = string-expression;

  Examples:
  ```
  age = 10;
  Q102 = prev_age;
  Y = sqrt(X);
  name = "John Doe";
  ```

```
sex_ration = males / females;
```

Proc Statement

Format:
```
PROC procedure-name
```

Description:
The **proc** statement declares the beginning of the procedures for a data entry or batch processing element. The procedure name must always be the name of an object in the forms or edit tree. If you are in the logic view and select a processing element from the forms or edit tree, the logic view will automatically generate the "PROC <item-name>" heading for you.

Example:
```
PROC AGE
```

If you plan to write logic for more than one procedure, the order of procedures must be as follows:

```
PROC <item-name>
 preproc
<statements>
 onfocus { data entry only }
<statements>
 killfocus { data entry only }
<statements>
 postproc
<statements>
```

See also: Preproc Statement, Postproc Statement, Onfocus Statement, Killfocus Statement, Order of Executing Data Entry Events, Order of Executing Batch Edit Events

Preproc Statement

Format:
```
preproc
```

Description:
The **preproc** statement declares that the following statements are executed at the beginning of a run, case, level, record, form, roster, or field.

In data entry applications, the statements in a preproc procedure are executed when you move **forward** onto an object, that is, when the execution flow moves the cursor onto the object, or when the user goes forward to the object by any means (mouse-click, tab, arrow keys, etc.). Preproc statements are NOT executed when you move **backward** onto an object, that is, the keyer reenters value, goes backwards with a mouse click, or uses Shift+tab to move back to it. If you want to execute the statements when you move both forward and backward onto a field, code them in the onfocus procedure.

In Batch Edit applications, the preproc is used to execute logic at the beginning of a run, case, level, or record. For a data item there is no difference between placing all your logic in a preproc or postproc. Remember, if you don’t code a preproc or postproc in a proc, all instructions in the proc are considered postproc statements by default.

Example:
```
PROC DATE
```

```
preproc { must immediately follow the "PROC" declaration }
  DATE = sysdate("DDMMYYYY");
{postproc would go here, if desired }
```

See also: Proc Statement, Postproc Statement, Onfocus Statement, Killfocus Statement, Order of Executing Data Entry Events, Order of Executing Batch Edit Events

Postproc Statement

Format:

```
postproc
```

Description:

The **postproc** statement declares that the following statements are executed at the end of a run, case, level, record, form, roster, or field. A postproc procedure can be coded in a proc for any run, case, level, record, form, roster, or field.

In data entry applications, statements in a postproc procedure are executed when you **complete** an object, that is flow off of it. Postproc statements are NOT executed when you click off a field, manually skip from a field, use Shift+tab to move backward from a field, or go to another field. If you want to execute the statements in these situations, code them in the killfocus procedure.

In batch edit applications, a postproc is used to execute logic at the end of a run, case, level, or record. For a data item there is no difference between placing all your logic in a preproc or postproc. Remember, if you don't code a preproc or postproc in a proc, all instructions in the proc are considered postproc statements by default.

Example:

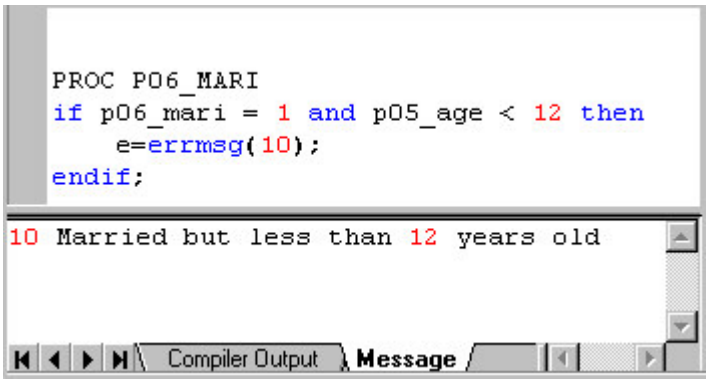
```
PROC SEX
 {preproc would go here, if desired }
postproc
 if ($ = 2 and AGE < 5) then
    reenter;
  endif;
```

See also: Proc Statement, Preproc Statement, Onfocus Statement, Killfocus Statement, Order of Executing Data Entry Events, Order of Executing Batch Edit Events

Logic

View Logic

To view the logic in a data entry application you can press icon; or in the **View** menu select "Logic"; or press **Ctrl+L**. To go back to the forms press ; or in the **View** menu select "Form"; or press **Ctrl+M**. From the Logic Window you can create or modify procedures that add logic to your application. The view is divided into two areas:



• Top: Text editor, where you write the logic statements

• Bottom: Message tab, where you create messages to be displayed during, or at the completion of, the execution of your application. This area is also used to display the results of the compiler after you compile a program. If you have errors in the logic, the compiler will display the error messages; otherwise it will display "Compile Successful".

Click on any element of the data entry or batch edit tree to see the logic which corresponds to that symbol. For example, if you click on a field, you see the logic for only that field. A group or level can also have logic associated with it. Click on the forms file node (usually the topmost node on form tree) to see the logic for the whole application. This is the way to see and enter logic for the global procedures.

See also: Create and Edit Logic, Order of Executing Data Entry Events, Order of Executing Batch Edit Events

Create and Edit Logic

You can use the CSPro language to write logic for virtually any part of your data entry or batch edit application—a level, roster, form, or field. In a Data Entry application you must make sure the screen has Logic View on the right and the data entry tree on the left, so you can click on the item for which you want to write logic. You can see the logic for the whole application by clicking on the form file (usually the topmost node) on the data entry or batch edit tree.

• **Example: Programming a message for the keyer**

Give a message for the keyer if there are married people under the age of 12. Click on the "Marital Status" field on the forms tree, P06_MARITAL_STATUS in our example. In the text editor, at the top of the Logic view, you will see:

```
PROC P06_MARITAL_STATUS
```

Now enter the following:

```
if P06_MARITAL_STATUS = 1 and P05_AGE < 12 then
  errmsg("Married but less than 12 years old");
endif;
```

Note that this particular verification can be done only after data have been entered in both fields. If for some reason "Age" is captured after "Marital Status," then these instructions would be placed in the "Age" field's logic.

• **Example: Programming a skip**

Program a skip after the marital status question to skip over age at first marriage if the person is never married. Click on the "Marital Status" field on the forms tree, P06_MARITAL_STATUS in our example. In the text editor, at the top of the Logic view, enter:

```
PROC P06_MARITAL_STATUS
if P06_MARITAL_STATUS = 1 then
  skip to P16_WORK_STATUS;
endif;
```

Find and Replace Logic

Press on the toolbar, or press **Ctrl+F**; or select **"Find"** from the **Edit** menu to open the **"Find"** dialog box. You can simply search for a text string by typing it in the dialog box and pressing Enter. You can then press the **F3** key to find the next occurrence of the text string. If you use the **"Mark All"** button, CSPro will show you each line that contains the text string by putting a yellow circle to the left of it.

Press **Ctrl+H**; or select **"Replace"** for the **Edit** menu to open the **"Replace"** dialog box. This is the standard Windows dialog box that allows you to replace one text string with another text string, either one at a time or all at once.

Set Compiler Defaults

From within an application, an option is available that determines if variables must be declared. By default, CSPro sets the mode of operation to explicit, but the user can change the behavior to implicit by going to the **Options** menu, and removing the check next to the **Set Explicit** option. Alternatively, you may include a **set explicit/implicit** command in your program to override the system setting. The following explains the impact of programmatically setting this switch, as opposed to using the system setting:

System Setting	Program Setting Result
✓ Set Explicit set explicit;	No effect, as program matches system setting
✓ Set Explicit set implicit;	Program overrides system setting, variables do **not** need to be declared
Set Explicit set explicit;	Program overrides system setting of implicit—variables must be declared
Set Explicit set implicit;	No effect, as program matches system setting

Compile Logic

When CSPro compiles your logic, it checks the logic you have written to see if there are any errors or warnings. Typical errors including spelling a command incorrectly, not using proper command syntax, and putting logic in the wrong place. Error messages appear in the panel at the bottom of the screen and a red dot appears to the left of the line that contains the error. Typical warning usually involve using commands in questionable ways. Warning messages also appear in the panel at the bottom of the screen and a yellow dot appears to the left of the line that contains the warning.
You can choose to compile code for a specific item, or for the entire application. To compile code for a specific item, simply select that item from the tree on the left side of the screen. The associated logic for that item will be displayed in the Logic View. Press on the toolbar; or from the **File** menu, select **Compile**; or press **Ctrl+K**. The results of your compile will be displayed in the "Compiler Output" area at the bottom of the screen. When you are ready to compile the entire application, select the topmost entry of the data entry tree or batch edits tree. This will display all logic written for the application in the Logic View. You can then press the compile button or press **Ctrl+K**.

CSPro always compiles your application when you run the data entry or batch edit application. If there are errors, you cannot proceed until the errors are corrected.

Language Elements

Delimiters

Delimiters separate elements in the CSPro language.

Delimiter	Symbol	Usage
Blank		Separate any language symbol
Comma	,	Separate parameters within functions
Quotation mark	" "	Specify the beginning and end of strings
Apostrophe	' '	Specify the beginning and end of strings
Semi-colon	;	Specify the end of statements
Colon	:	Separate the beginning and end of substrings
Parentheses	( )	Specify the beginning and end of function parameters
Brackets	[ ]	Specify substrings

Comments

Comments make applications easier to understand. They are used to explain the purpose of specific statements or to temporarily disable statements to help find errors. Any text enclosed by braces { } is a comment. The text within the brackets will be green. Comments can be placed anywhere in an application and are not checked for syntax errors. Comments can be nested, that is, comments within comments are allowed. Any text following double slashes // until the end of the line is also a comment.

The second line in the example below is a comment; a comment is also appended to the code in line four. It is highly recommended that the user document CSPro applications through the liberal use of comments.

```
PROC HHDAY
{Do not allow June to have more than 30 days}
  if HHMONTH = 6 and $ > 30 then
    X = errmsg (1, "June", 30, $);  // if error, then display message
    reenter;
  endif;
```

Variables and Constants

Data Items

Data items are defined in a data dictionary. You can assign a value to a data item, or get the value of a data item in any procedure. The following is an example of the use of data items:

```
PROC SEX
```

```
if AGE > 15 and NumOfKids <> notappl then
  $ = 2;
endif;
```

However, in developing a data entry or batch edit application, it will frequently be necessary to define variables that do not exist in the data file(s) attached to the application. These variables may be used throughout the application, but only exist during the execution of the application.

See also: Dictionary, Add or Modify Items, This Item ($)

This Item ($)

The dollar sign ($) is a short way of referring to a data item if used within that data item's procedure.

Example:

```
PROC AGE
  if MARITAL_STATUS > 1 then { ever married }
    if $ < 12 then { AGE < 14 }
      errmsg ("Person too young (%d) to be married", $);
    endif;
  endif;
```

Subscripts

Items with multiple occurrences or in multiple records have one name (the item name), but can occur multiple times. In order to indicate the specific occurrence of the item, you may need to use an index or subscript. The subscripts are integers and are numbered from 1.

Imagine that the SEX is an item in the multiple record CHILD.

The expressions

`SEX(1)` refers to the sex of the first child.

`SEX(3)` refers to the sex of the third child.

`SEX(i)` refers to the sex of the **i**th child.

Subscripts can be numeric expressions as well as numeric constants. For example, the expression

```
SEX(curocc(CHILD));
```

refers to the current occurrence of CHILD. (curocc is a function that returns the current occurrence of a multiple record). When referring to multiply-occurring items within the scope of their repetition, you do not need to use subscripts, as the current occurrence will be assumed. For example, suppose you have a population record that has multiply occurrences, and belonging to that record are the three variables SEX, AGE, and FERTILITY. If your code is contained within any of these variables' procedures, you do not need to use subscripts. To illustrate:

Example 1:

```
{this will check the sex and fertility values for each person in the
household}
 PROC SEX
```

```
  if $ = 1 then
    if fertility <> notappl then
      errmsg ("male found with fertility");
    endif;
  elseif $ = 2 then
    if age < 10 and fertility <> notappl then
      errmsg ("underage female found with fertility data");
    endif;
  else
    errmsg ("invalid sex code (sex=%d)", $);
 endif;
```

However, if you were to place the exact same logic elsewhere in your program, you would have to programmatically mimic the looping mechanism, and use subscripts. For example, if the above code were placed in the QUEST procedure, it would need to be adjusted as follows:

Example 2:

```
 PROC QUEST
   NumPeople=count (POP_RECS);
   do varying i=1 while i <= NumPeople
     if sex(i) = 1 then
       if fertility(i) <> notappl then
         errmsg ("male found with fertility");
       endif;
     elseif sex(i) = 2 then
       if age(i) < 10 and fertility(i) <> notappl then
         errmsg ("underage female found with fertility data");
       endif;
     else
       errmsg ("invalid sex code (sex=%d)", sex(i));
     endif;
   enddo;
```

Numbers

Numbers may be any positive or negative integer or decimal value. Negative numbers have a leading minus (-) sign. Positive numbers have no sign, but can have an optional leading plus [+] sign. Numbers can have up to 15 significant digits. Numbers must not have thousands separators. Decimal points can be either period (.) or comma (,) depending on the "Regional Options" setting of the computer.

Text Strings

A text string is any set of characters in the computer's character set enclosed between a pair of quotation marks (") or apostrophes ('). Any spaces enclosed within the quotation marks or apostrophes are considered part of the text string. Upper- and lower-case letters may be used. However, a text string 'a' is different from a text string 'A'. The maximum length of a text string is 250 characters. If you wish to have apostrophes (') embedded within your string, you **must** use the quotation marks (") to enclose it, and vice-versa. For example,

```
MyString='That's great!';
```

would set MyString to "that", and the trailing "s great!" would be considered outside the string,and would therefore provoke a compiler error. Thus, if you wanted to accomplish the above, you must write:

```
MyString="That's great!";
```

Similarly, if you wanted to embed quotation marks within your string, you must write the string as follows:

```
MyString='The chair is 23" high';
```

Strings that are surrounded by quotation marks will appear in pink. Strings that are surrounded by apostrophes will appear in black. We recommend using quotation marks ["], if the text of the string does not turn pink when you think you have completed entering it, it will be quickly apparent that you have not terminated your string properly.

Expressions

Expressions

An expression is a combination of operators and operands. Operands can be constants, items, variables, functions, or some combination thereof. Operators can be arithmetic (+, -, *, /), relational (=, <>, >, <, >=, <=) or logical (and, or, not). Every expression evaluates to a value and can therefore be used as a sub-expression of other expressions. There are three types of expressions: numeric, string, and logical.

• Numeric
They evaluate to numbers. The following are numeric expressions:

```
4
4 + 5
A / B
A*(B+C/D)
A + sqrt(B)
```

• String
They evaluate to strings. The following are string expressions:

```
answer="Yes";
concat(FIRST_NAME, " ", LAST_NAME);
edit("ZZZZ9", A + B);
```

• Logical
Logical expressions (conditions) evaluate to **true** (1) or **false** (0). The following are logical conditions:

```
KIDS > 5
SEX = 2 and AGE > 12
```

Substring Expressions

Format:
`String [start:length]`

Description:
A substring expression lets you extract a part (substring) of a string. The "**start**" gives the starting character position of the substring within the string, and "**length"** gives the number of characters to include in the substring, including the starting character. If "length" is not given, then it is assumed to be to the end of the originating string.

Example 1:
Suppose the variable STRING has the value "ABCDEF".

```
STRING[1] "ABCDEF"
STRING[3:1] "C"
STRING[3] "CDEF"
STRING[2:3] "BCD"
STRING[5] "EF"
STRING[4:7] "DEF"
```

Example 2:
Likewise, substring expressions can be performed on string arrays. Suppose the string array "crop" had the following definition:

```
PROC GLOBAL
  array alpha(10) crop (20); { 5 crop names, each up to 10 characters long }
PROC MY_PROGRAM
  preproc
    crop(1)= "maize";
    crop(2)= "wheat";
    crop(3)= "rice";
    crop(4)= "potatoes";
    crop(5)= "legumes";
```

The following substring expressions would yield the results as shown:

```
crop(1)[2] "aize"
crop(1)[3:1] "i"
crop(2)[3] "eat"
crop(3)[2] "ice"
crop(4)[5] "toes"
crop(5)[1:3] "leg"
```

Both "**start"** and "**length"** can be numeric expressions as well as constants. For example, to obtain the last 3 characters of STRING you could use the expression:

```
STRING[length(STRING) - 2:3]
```

In this example, if STRING is not at least two characters long, you may get unexpected results.

Special Values

There are three special values in the CSPro language: missing, notappl, and default. They have the following meaning and uses:

- **Missing**
  The value missing indicates that a data item was supposed to have a response and no response was given. Other terms for this are "not stated" and "non-response". To properly utilize this special value, you must create a value set for this item in the dictionary, setting one of the value set entries to the special value "missing." For example, you could set 8 (or 88, 888, etc.) or 9 (or 99, 999, etc.) to missing. Finally, although you must associate a number with the special value missing, you can only use the = or <> comparison operators against the special value `missing`—you can **not** refer to the numeric value you assigned it to in your dictionary value set.

- **Notappl**
  The value notappl indicates that a data item is blank. The item did not have a response because the question did not apply to this respondent. Fields that are skipped during data entry are assigned the value notappl.

- **Default**
  The value default indicates that a data item or variable has an undefined value. This can result from various circumstances. For example, a calculation that contains a special value as one of its operands returns the result default. Reading the value of a variable from a data file when the data type specified in the dictionary for the variable is numeric and the value in the data file contains non-numeric characters will cause the variable to become default. The same result occurs if the dictionary does not specify a decimal character for the variable being read but the value in the data file contains one. Additionally, if a numeric variable has numeric subitems, some of which are notappl (blank) then the value of the variable will sometimes be default. For example if the numeric variable DATE has 3 subitems DAY=1, MONTH= notappl and YEAR=2000, then the value of DATE will be default. The value default can also be written to the data file when value of the variable being written overflows the length specified for that variable in the dictionary (for example if the value of the variable is 999 but the dictionary specifies a length of 2 for the variable).

A particular value of a data item can be assigned one of these special values in the data dictionary.

Operators

Operators

- **Arithmetic Operators**

Operation	Symbol
Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulo (remainder)	%
Exponentiation	^

The addition operator can also be used to perform string concatenation.

- **Relational Operators**

Operation	Symbol
Equal to	=
Not equal to	<>
Less than	<
Less than or equal to	<=
Greater than	>
Greater than or equal to	>=
In range	in
Has range (for repeating items)	has

- **Logical Operators**

Operation	Symbol	Keyword
Negation	!	not
Conjunction	&	and
Disjunction	|	or
If and only if	<=>	

Either the symbol or keyword can be used. When more than one operator exists in an expression, the order in which the operators are evaluated is determined by their precedence.

In Operator

This operator is used in logical expressions to test whether an item or variable is within a set of values or ranges. The item or variable can be numeric or alphanumeric. A range of values is separated by a colon, for example 1:5. Elements of a list of values or ranges are separated by commas, for example 1, 3:5, 7.

Example 1:

```
if RELATIONSHIP in 1:5 then
```

is the same as

```
if RELATIONSHIP >= 1 and RELATIONSHIP <= 5 then
```

Example 2:

```
if WORK in 1,3,5 then
```

is the same as

```
if WORK = 1 or WORK = 3 or WORK = 5 then
```

Example 3:

```
if X in 1:4, missing, notappl then
```

is the same as

```
if (X >= 1 and X <= 4) or X = missing or X = notappl then
```

Example 4:

```
if NAME in "A":"MZZ" then
```

is the same as

```
if NAME >= "A" and NAME <= "NZZ" then
```

See also: Has Operator

Has Operator

This operator is used in logical expressions to test whether a repeating item is within a set of values or ranges. The item can be numeric or alphanumeric. A range of values is separated by a colon, for example 1:5. Elements of a list of values or ranges are separated by commas, for example 1, 3:5, 7.

This function is similar to the in operator except that it works on repeating items. It thus tests whether a group of items contains certain values.

Example:

```
    // suppose that there are five people listed on a record

    if SEX has 2 then // means: are any of the five people women?

    if AGE has 0:17 then // means: are any of the five people under the age
of 18?
```

See also: In Operator, Count Function, Seek Function

If and Only If Operator <=>

This operator has the following truth table.

if and only if [x **<=>** y]

	Y	
X	**true**	**false**
true	true	false
false	false	true

The following two sets of code give the same result:

```
if (SEX = 2 and AGE >= 10) <=> (CHILDREN_BORN <> notappl) then
  errmsg("Children ever born incorrect");
endif;
```

and

```
if (SEX = 2 and AGE >= 10) then
  if CHILDREN_BORN = notappl then
    errmsg("Children ever born incorrect");
  endif;
else
  if CHILDREN_BORN <> notappl then
    errmsg("Children ever born incorrect");
  endif;
endif;
```

See also Operators, Operator Precedence

Operator Precedence

The table below shows the order of precedence for operators. When operators of the same precedence are in an expression, they are evaluated from left to right. The order of precedence can be changed using parentheses. Operators in parentheses are evaluated first.

Order	Operator
1	^
2	* / %
3	+ -
4	= < > <= >= <> in has
5	not !
6	and &
7	or |
8	<=>

And/Or Truth Table

The truth table summarizes all possible evaluations when two expressions (X and Y) joined by an operator (**and** or **or**) are true, false, or undefined.

and [x **and** y]

	Y	
X	**true**	**false**
true	true	false
false	false	false

or [x **or** y]

	Y	
X	**true**	**false**
true	true	true
false	true	false

Files

External Files

An external file is a text file other than the primary data file that you can use in a data entry or batch application. You can read and/or write to external files, using CSPro logic. You must create a data dictionary that describes the format of any external file you want to use. An external file dictionary can contain only one level.

You can share external files across a network. If an external file is accessed only by read functions (loadcase, locate, find, key, retrieve), no special programming actions need to be taken to share the file. Multiple users can read the file at any time.

However, if an external file is accessed by any write functions (writecase or delcase) only one user at a time may use the file. For write functions, the external file is like a file in a filing cabinet. When one person has taken out the file for use, no one else can use the file until the person has returned it.

You can control when the file is in use by coding open and close functions. The file is in use between the execution of the **open** and the **close** function. This gives you complete control over when the file is in use. You should try to minimize the time the file is in use in order to allow other users to access the file.

If **open** and **close** functions are not coded for an external file used for writing, the following "open" and "close" rules apply:

• In batch processing, the file is opened at the beginning of the run and closed at the end.

• In data entry processing, the file is opened just before any external file function is executed and is closed immediately following the function, unless one of the following functions is used on the file:

- **loadcase** without a *var-list*
- **retrieve**
- **key**

When any of the above functions is used, the file is opened just before the first file function is executed, but is left open after the function is completed. These functions depend on remembering the current position of the file. If the file is closed, the current position is lost.

See also: Insert or Drop a File from an Application, Lookup Files

Lookup Files

A lookup file (external file) is a text file that can be used in a data entry or batch application from which you retrieve data to display on a form or to use in a calculation. It requires a CSPro data dictionary. Possibilities include:

• Geographic codes and names. Your application could show the name corresponding to the code the user keyed.

• Industry and occupation codes. Your application could ensure the user keys a valid code.

• Last year's data. Your application could look up a corresponding field from last year's data and calculate a percentage change.

• Generalized menu choices. Your application could read a lookup file and show the contents on the screen as a menu, then convert the user's choice to a code.

To use a lookup file (external file) in your application, do the following:

• Create the lookup file and its data dictionary

• Close the lookup file's data dictionary

• Create a data entry or batch edit application with a standard forms file and data dictionary.

• Insert the lookup file's data dictionary into the application.

- Add logic to the application to manipulate the lookup file. The Loadcase and Selcase functions are particularly useful

Note: The CSPro examples include an application that demonstrates the use of a lookup file. This is normally installed in the folder named "C:\Program Files\CSPro 6.0\Examples."

Working Storage File

"Working storage" contains alphanumeric variables and data items used in an application and which are not part of any data file. Definitions of working-storage variables and data items are contained in a data dictionary which is not connected to any data file. This data dictionary can have any number of records but can have only one level.

See also: Insert or Drop a File from an Application

Message File

The message file for a data entry application stores the error message text that is displayed during data entry. The message file for a batch application stores the error message text that will be included in the execution report. The message file has the filename <application-name>.msg. During data entry design, the message text is modified at the bottom of the logic screen.

- **Basic Messages**
  Each line in the message file contains one message. A message consists of a message number followed by text. The message text can be up to 240 characters long. It is displayed on the entry screen when an errmsg function with the message number is executed in a data entry application. It is written to the execution report when an errmsg function with the message number is executed in a batch application.

  For example, suppose a message file contains the following lines:

```
1 This is the first message
2 This is the second message
```

  When an errmsg(1) function is executed in a data entry application, the message "This is the first message" is displayed on screen. When an errmsg(2) function is executed, the message "This is the second message" will be displayed on screen.

- **Messages with Parameters**
  Parameters can be specified in the errmsg function. These parameters can be numeric expressions or string expressions. String parameters in the error message text are indicated by **%s**. Integer numeric parameters are indicated with **%d**. Decimal numeric parameters are indicated with **%f**. Any number of different messages can be included in the message file. The **errmsg** function can be used in any dictionary, form, group, or field procedure, or in a user-defined function. The maximum number of parameters in an **errmsg** function is 20.

  For example, an error message file might contain the following:

```
1 The month of %s has only %d days. You entered %d!
```

  The application could use this as follows:

```
x = errmsg (1,"June",30,31);
```

When the errmsg function is executed, it knows to use error message "1", and substitute the word "June" for **%s** in the message text, the number 30 for the first **%d**, and 31 for the second **%d**. The message "The month of June has only 30 days. You entered 31!" will be displayed on the screen.

Example:
The more general the parameters of the message, the more flexible the message. In the example below, the value of the variable HHDAY is used as a parameter. The error message will use the value of HHDAY if the errmsg function is executed.

```
   PROC HHDAY
if HHMONTH = 6 and HHDAY > 30 then
 x = errmsg (1, "June", 30, HHDAY);
  reenter;
    endif;
```

Program Information File

The program information files (.PFF) are used to run applications (data entry and batch edit) or tools (tabulate frequencies, sort data, export data, reformat data, compare data, and concatenate data) in production mode.

The .PFF file stores the name of the application or tool, the data file(s) to be used, and any run-time parameters specific to the application or tool.

You can use a .PFF file as a command line parameter for CSEntry, CSBatch, CSFreq, CSSort, CSExport, CSReFmt, CSDiff, or CSConcat.

See also: Run Production Data Entry, Run Production Batch Edits, Run Production Frequencies, Run Production Sorts, Run Production Exports, Run Production Reformats, Run Production Compares, Run Production Concatenates

Data Entry Module

Introduction to Data Entry

The Data Entry module allows you to create, using a single dictionary, one or more forms [screens] for data entry. You may also specify the data entry behavior and incorporate logic in a program to check for consistency between variables and to set up skip patterns. After you have developed the forms and the program to your satisfaction, use CSEntry to input the data.

You may enter the data in the office after the information is collected or you might want to use the Computer Assisted Personal Interviewing (CAPI) feature, in which the interviewer uses a laptop computer to enter responses in the field as they occur.
This section contains the following information:

General Data Entry Concepts
CSPro Data Entry Concepts
Create a Data Entry Application
Change Data Entry Characteristics

Data Entry Application

General Data Entry Concepts

Data Entry Philosophies

There are two different approaches to data keying:

- **Heads-Down Keying**
  This approach is most commonly used for keying of census forms because of the large volumes of data involved. While entering data, the operator generally does not look at the computer screen, but rather, looks down at the questionnaire on the table or work surface. The objective of heads-down keying is to transcribe to the computer, as quickly and accurately as possible, the data as they appear on the questionnaire. On-line checking is generally kept to a minimum and consistency errors are resolved in a later phase, generally through computer edit programs. Operators do not need to be familiar with the subject matter of the questionnaire. They make very few decisions to resolve data errors. The most important skill is speed and accuracy. CSPro provides Operator Statistics to help measure operator speed and accuracy.

- **Heads-Up Keying**
  This approach is most commonly used for entering data from surveys, due to the smaller number and greater complexity of the questionnaires (as compared with a census). While entering data, the operator often refers to the computer screen as well as to the questionnaire. The objective of heads-up keying is to catch and correct as many errors as possible as the data are being entered. As a result, there is generally more on-line checking programmed into the application. Operators need to be very familiar with the subject matter of the questionnaire. They will make decisions to resolve data errors, and must be properly trained to do so.

Skip Issues

- **To skip or not to skip.**
  "Skipping"—causing the cursor to jump over one or more fields during the data entry operation—is an issue that provokes discussion both pro and con. The decision of the application designer to use (or not use) skips will depend entirely on the type of application and the data entry staff.

  When the data capture operation is expected to be "heads-down," as it would be for a census or similar high-volume application, it will cause less confusion to the keyer if all skips are controlled by the operator rather than the application. There will then be no "surprises" for the keyer when the application logic forces the cursor to one field when the keyer, looking not at the screen but at the form, expects the cursor to be in a different field altogether. Thus, instead of speeding up the entry operation by anticipating probable cursor movement, this tactic eventually slows down the operation when there are inconsis tencies in the data or, simply, keying errors.

  When the data capture operation is of smaller volume, or with a more complex questionnaire or forms, it may make sense to use application-controlled skipping. The operator will be more likely to be aware of the cursor movement and less likely to be surprised by unexpected interruptions in the normal sequence. Of course, a combination of operator-controlled and application-controlled skipping may be used in any given application; the designer will have to weigh the keying environment and the forms to be entered to make the appropriate decision.

- **Manual Skips**
  During data entry, there may be times when you want the operator to be able to skip over certain fields that do not apply to the current case. For example, in a Population record, the fertility fields do not apply to males or to underage females, nor do questions on economic activity apply to children under a certain age. In CSEntry, the ‘+’ key on the numeric keypad is always active as a ‘skip’ key. Every field

has a skip field value associated with it. The default value is ‘next’, meaning that when the cursor is on that field and the skip key (‘+’) is pressed the cursor will move to the next field in sequence. CSEntry allows you to change this value to any later field in the sequence, or to the end of the screen or form. This approach corresponds to the "operator-controlled " option in CSPro.

- **Automatic skips**
  Automatic, or application-controlled, skips depend on information already keyed to direct the cursor movement. For example, in a household survey, if a female respondent states that she has at least one child living with her, the cursor can skip automatically to the form for capturing information about that child (and any others). Conversely, if the female indicates that no children are present, the cursor can be directed to skip over the information about children. It is clear that such skips depend entirely on the accuracy of the data keyed prior to the skip; if a "Yes" response is mistakenly entered as a "No", the cursor will be misdirected and the operator will find that the screen(s) presented for keying do not correspond to the information in the paper forms. This will cause loss of time as the operator seeks to uncover the error. Automatic skips, when used, must be well-documented so that the keyer is aware of the possibilities of non-sequential cursor movement. This approach corresponds to the "system-controlled" option in CSPro.

Errors at Data Entry

Errors are introduced into the data through miskeying. Verification (rekeying or double keying) can reduce these errors. A system called 'intelligent data entry' may be used to prevent invalid entries from ever getting into the system. An intelligent data entry system ensures that the value for each field or data item is within the permissible range of values for that item. Such a system increases the chance that the data entry operator will key in reasonable data and relieves some of the burden on later stages of the data preparation process.

At data entry time, CSEntry shows a message every time the keyer enters a value that is out of range according to the data dictionary. You may set the attribute to override the message and force the out-of-range value into the data file. If this attribute is not selected, the keyer cannot proceed until a valid value is entered.

See also: Data Entry Philosophies

Adding Logic

In most surveys, consistency errors are corrected manually as opposed to automatically. The correction process, as done traditionally, is often very lengthy, time-consuming and painful. In surveys of small volume and high complexity, such as Household Surveys or Income and Expenditures Surveys, it is often desirable to apply the edit specifications rules at data entry time and resolve any errors immediately while the questionnaire is still at hand. This approach is not recommended for a census.

You can use the CSPro language to write consistency checks for virtually any part of your data entry application—a level, form, roster, or field. The logic is executed as the data is being keyed. Any error messages are reported back on the screen, and the operator then has access to both the error messages and the questionnaire itself on the screen. The same logic can be run against the data after they are entered in either batch or interactive mode.

CSPro Data Entry Concepts

Operator vs. System Controlled

CSPro offers two distinct types of data entry applications. Your choice will determine certain behaviors at data entry time. Some special data entry keys will behave differently. For more detail about special data entry key behavior, please refer to the Data Entry User's Guide.

• Operator controlled
This is the default type of data entry application. This type generally allows more flexibility for the keyer during data entry. It is recommended for simple *ad-hoc* applications and for census applications. Operator-controlled applications have the following features:

- Some special data entry keys are active during data entry.
- CSEntry will **not** keep track of the path.
- 'Not applicable' values will be allowed.
- More appropriate to the heads-down methodology.
- Operator can bypass logic in the application using special keys.

• System controlled
These applications generally place more restrictions on the data entry operator. This type is sometimes used for complex survey applications. The behavior of these applications at data entry time is essentially the same as in ISSA. System controlled applications have the following features:

- Some special data entry keys are **not** active during data entry.
- CSEntry will keep track of the path.
- 'Not applicable' values will **not** be allowed.
- More appropriate to the heads-up methodology.
- Logic in the application is strictly enforced; operator cannot bypass or override.

You set the application type in the Change Data Entry Options dialog box; Options/Data Entry from the main menu toolbar.

Data Entry Path

CSPro supports a powerful feature called data entry path. The path can either be "turned on" or "turned off", depending on the data entry application type selected on the Data Entry Options dialog box. Operator controlled applications always have path turned off, while system controlled applications always have path turned on.

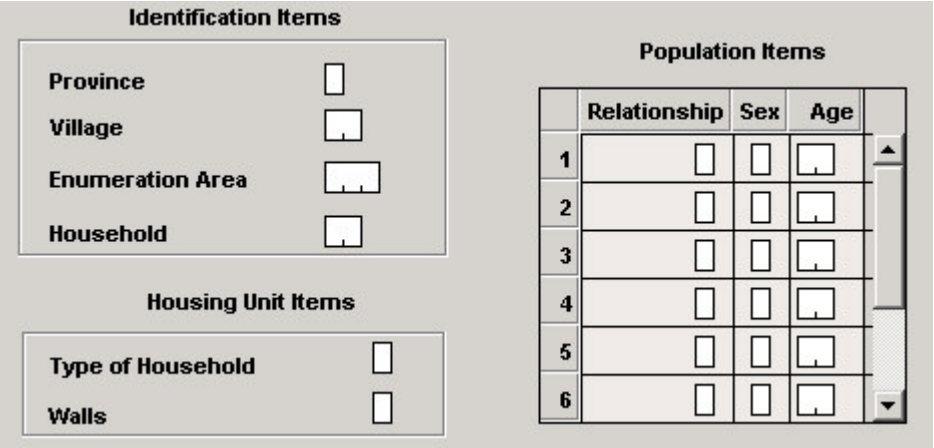
• Path on
CSEntry will keep track of the order in which the data entry operator entered all fields. If the operator goes backward, the cursor will go to the fields in the reverse order in which they were entered. For example, if the logic causes the cursor to skip over a set of fields, the cursor will also skip over these fields when the operator goes backwards. Fields that were skipped can never be entered, unless the operator goes backwards and chooses different values to avoid the skip. This helps ensure the integrity of the data file.

• Path off
CSEntry will not keep track of the order in which the data entry operator entered the fields. If the operator goes backward, the cursor will go to the preceding field even if it had originally been skipped.

Data Entry Elements

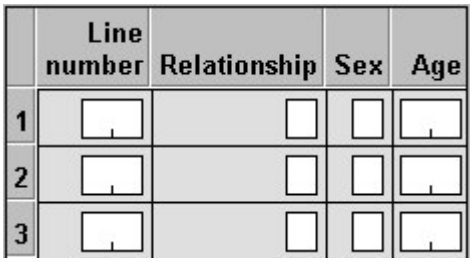
• Forms

A form is a collection of fields, text and/or rosters which appears on the screen at the same time during data entry. A form may be larger than the actual screen. When this is the case, the form will scroll as necessary during data entry so that the current point of entry is always visible to the keyer. A form may repeat if it contains fields from a dictionary record which has more than one occurrence.



• **Rosters**

A roster is a grid that shows multiple occurrences of a group at the same time. Many questionnaires have rosters printed on them. A typical example would show each person as a row and each column as a variable, as shown below. Rosters can also have a vertical orientation, in which case the rows and columns would be reversed.



In CSPro, you can show repeating groups as a roster on a single form or as individual fields on a form that repeats.

The darker gray area at the top of each column is called a column heading. In the example above, the column headings contain the text "Line number", "Relationship", "Sex", and "Age". The text in the darker gray area to the left of each row is called the "occurrence label." In the example above, the occurrence labels are "1", "2", "3". These are the default values.

In rosters with vertical orientation, column headings and occurrence labels are reversed.

• **Fields**

Fields are areas of a data entry form that may be keyed or may show values. Fields may be placed directly on the form or may be part of a roster on the form. Fields are always associated with dictionary items. Some properties of fields, such as length and type (numeric or alphanumeric), are defined in the data dictionary. Other properties are defined in the forms designer. In this example we have two fields:

See also: Add a Form, Add a Roster to a Form, Add Fields to a Form

Issues to Consider When Designing a Form

If you plan to key data from paper questionnaires you generally try to make the forms [screens] match the pages in the questionnaire. CSPro provides flexibility in the way you define forms, the number of forms to use, and the choice of fields to go on each form. There are, of course, limitations imposed by the structure of the data dictionary, some of which have to do with whether records and items are "multiple":

• A record is considered multiple if it is defined as "Max > 1" in the data dictionary.

• An item is considered multiple if it is defined as "Occ > 1" in the data dictionary.

• A sub-item is considered multiple if it has been defined as "Occ > 1" in the data dictionary or if the item it belongs to is defined as "Occ > 1".

Keep in mind the following rules when you design your data entry forms:

• You **can** mix items from different single records on the same form.

• You **can** mix ID items with items from single records on the same form.

• You **can** split items from the same record onto different forms.

• You **can** make more than one roster from a multiple record. The rosters can be on the same form or on different forms.

• You **can** mix items from a single and a multiple record on the same form, but the latter must be in a roster.

• You **cannot** mix items from different multiple records on the same form.

• You **cannot** mix items from different levels on the same form (applies to complex data dictionaries only)

If you have any multiple records, items, or sub-items in the data dictionary, you must decide whether you want to make them into a roster or use a form that repeats. You must take this into account when deciding which items to place on each form.

Cases and Levels
A **case** is the primary unit of data in the data file. A case usually corresponds to a questionnaire, although some complex applications may have a hierarchical set of questionnaires that comprise a single case. For example, the main questionnaire may consist of a household roster and other household information, and there may also be a separate questionnaire for each woman in the household.

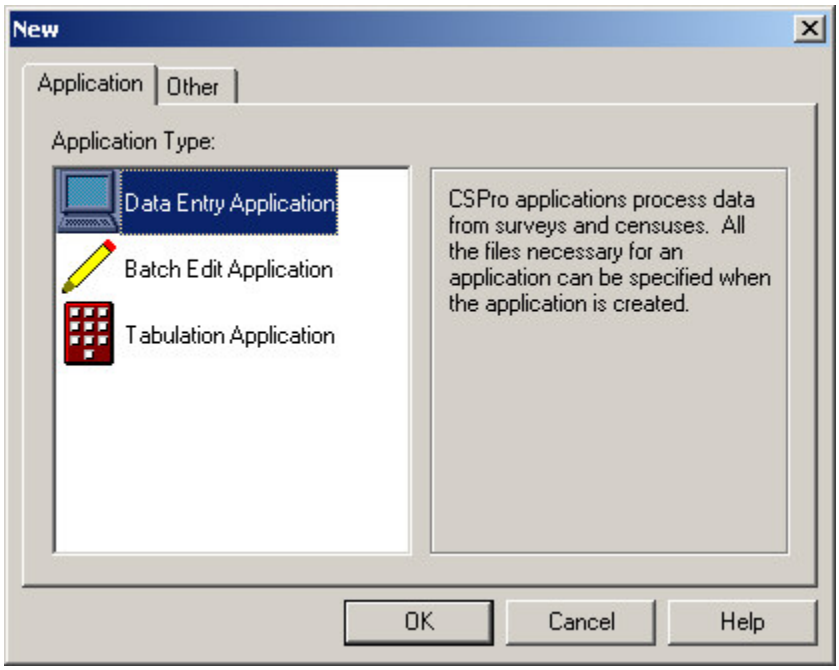
The data entry application may then contain two levels —one for the household and one for each woman in the household. The set of forms corresponding to the household make up level one. The set of forms corresponding to each woman make up level two. Each case would consist of a level one and a variable number of level **occurrences** for level two. Most applications consist of a single level.

Create a Data Entry Application

Create a New Data Entry Application

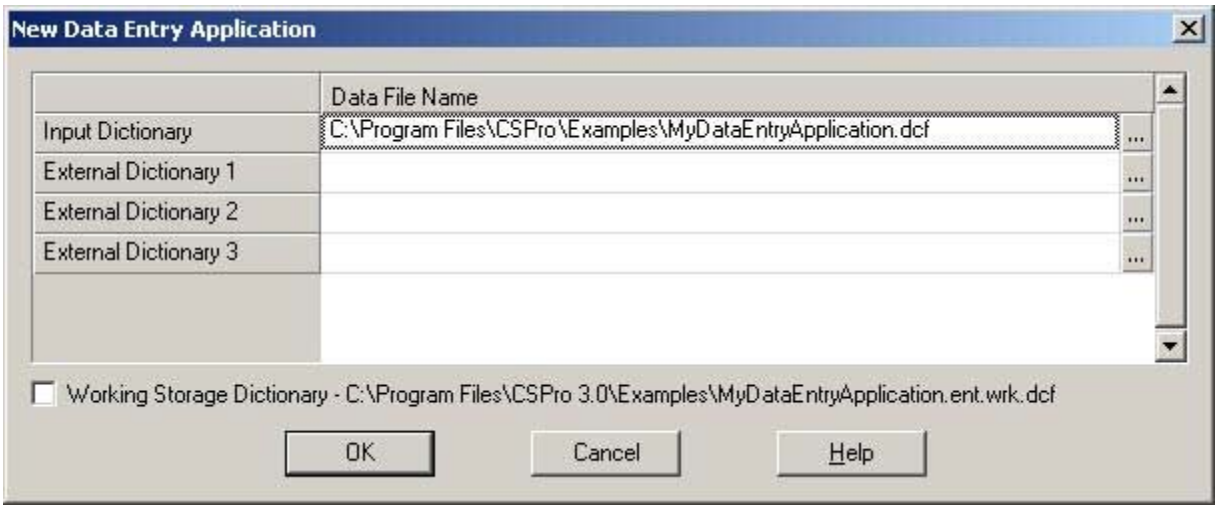
To create a new data entry application:

• Click on the toolbar, or from the **File** menu, select **New**. The following dialog box will appear.



• Select **Data Entry Application** and press **OK**.

• A file dialog box will appear. Enter the name of the application file. Make sure you are located in the folder where you want to place the application files. Then press **Create**. The following dialog box will appear.

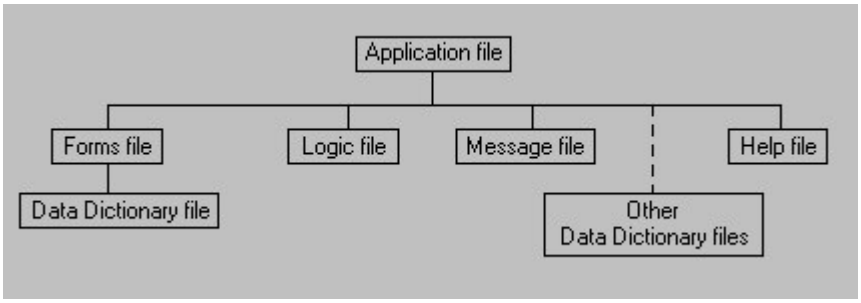


- A default name of the data dictionary describing the data enter file is given. You can use this name or change it. If you give the name of a dictionary file that already exists, that data dictionary will be used by the application. If you give the name of a dictionary that does not exist, a new data dictionary will be created.

- If you are using an existing CSPro data dictionary, then the system displays the question: "Would you like CSPro to create a set of forms for you, based on the input dictionary? If you answer "Yes" CSPro will automatically generate default data entry forms and will display the the drag option menu. If you answer "No" you may begin creating data entry forms.

- If you are creating a new CSPro data dictionary, you will need to enter information into the dictionary about records, items, and values before you can create forms.

Data entry applications consist of the following files:



- Data Entry Application File (.ent)
  This specifies all other files contained in the application and includes other application information.

- Form File (.fmf)
  There is usually one forms file per application, but there may be multiple forms files. Each forms file contains one Data Dictionary file (.dcf) which represents the primary data file that is being created or modified.

- Logic File (.apc)
  Contains CSPro language statements.

- Message File (.mgf)
  Optional file, it contains text for messages displayed during data entry.

- Question File (.qsf)
  Optional, contains text for CAPI text and help screens displayed during data entry.

- Other Data Dictionary Files (.dcf)
  Optional, it represents secondary data files (such as lookup files) that are read and/or written to during data entry.

Generate Default Data Entry Forms

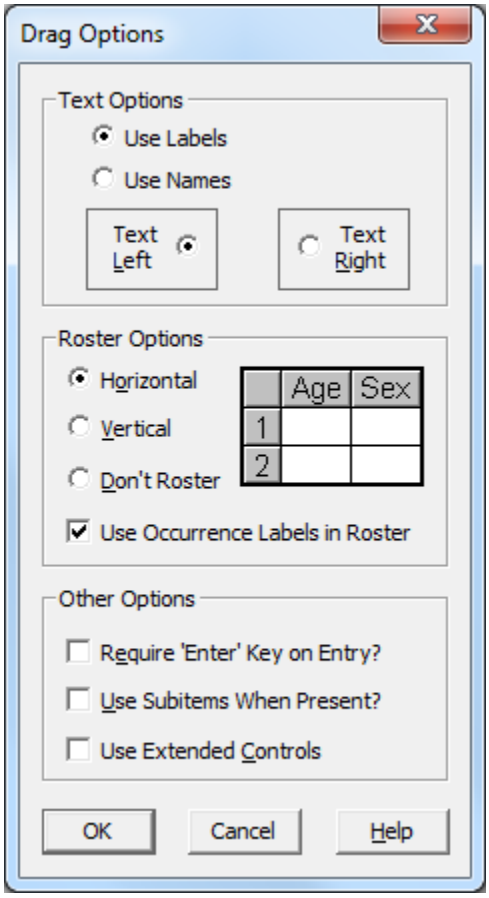
CSPro can automatically generate data entry forms that places all dictionary items onto forms. This can save time as it quickly builds up your form(s), allowing you to easily customize them to your specific needs. One form will be created for each dictionary record (identification items get their own form as well). Thus, for a one-level dictionary that contains at least one level ID and three other records, you will end up with four forms.

To generate new data entry forms, either press Ctrl+G; or, from the edit menu select "Generate Forms"; or from the dictionary tab on the left side of your screen, drag the dictionary book onto a form. As this action will destroy all existing forms, a warning message will appear, asking you to confirm that you wish to proceed.

If you choose to proceed, the "Drag Option" menu will appear. At this point you have the opportunity to decide text placement with respect to the data entry boxes; whether you want to roster items (when possible); whether you want sub-items dropped instead of the item, etc.

The Drag Option Menu

Whenever you automatically generate data entry forms, drag an entire dictionary, or drag a dictionary record onto a form, this dialog box will appear. When you drag an individual dictionary item to a form, this dialog will not appear, but the settings in effect will be used. [To access this dialog box without dragging, go to the Forms Designer toolbar and select **Options/Drag**.]



The following choices are available to customize your drag-and-drop operation:

- **Text Options**
  When fields are dragged onto a form from the dictionary, the dictionary text associated with the item is usually also included. You can select whether the item's label, the item's name, or neither of these (no text) is dragged onto the form.

You can also select whether the text is placed to the left or to the right of the data entry box. (This setting has no effect if the item is rostered.)

- **Roster Options**
  This affects dictionary records and items with more than one occurrence. To enter this type of data, you either need a form that repeats (to allow for the multiple occurrences of the data), or you need a roster.

  If you choose "Horizontal" CSPro will make rosters in which the occurrences are the rows and the fields are the columns. In CSEntry the cursor will move from left to right.

  If you choose "Vertical" CSPro will make rosters in which the occurrences are the columns and the fields are the rows. In CSEntry the cursor will move from top to bottom.

  If you choose "Don't Roster" CSPro will make forms that repeat.

  If you select "Use Occurrence Labels in Roster," the roster's rows, instead of being labeled 1, 2, 3, etc., will use the occurrence labels defined for the item or record in the dictionary.

- **Require Enter Key on Entry?**
  This option determines whether the **Enter** key must be pressed to advance an operator to the next data entry field.

  If left **unchecked**, the cursor will automatically advance to the next field as soon as the maximum number of characters are entered for the field (that is, if the field length is two, then after entering two characters the cursor will advance to the next field). An operator can always hit the **Enter** key to complete a field without having entered the full complement of digits.

  If this option is **checked**, the operator must always press the **Enter** key to advance to the next field.

- **Use Sub-items When Present?**
  If you have items with sub-items, you may **check** this box to place the sub-items, instead of the item, on the form. For example, if you have a Date item that contained the three sub-items Day, Month, and Year, the sub-items, rather than the item Date, would be placed on the form. However, if any of the sub-items overlap, the item will be used instead. (This setting has no effect if no sub-items are present.)

  If this box is left **unchecked**, items will always be used.

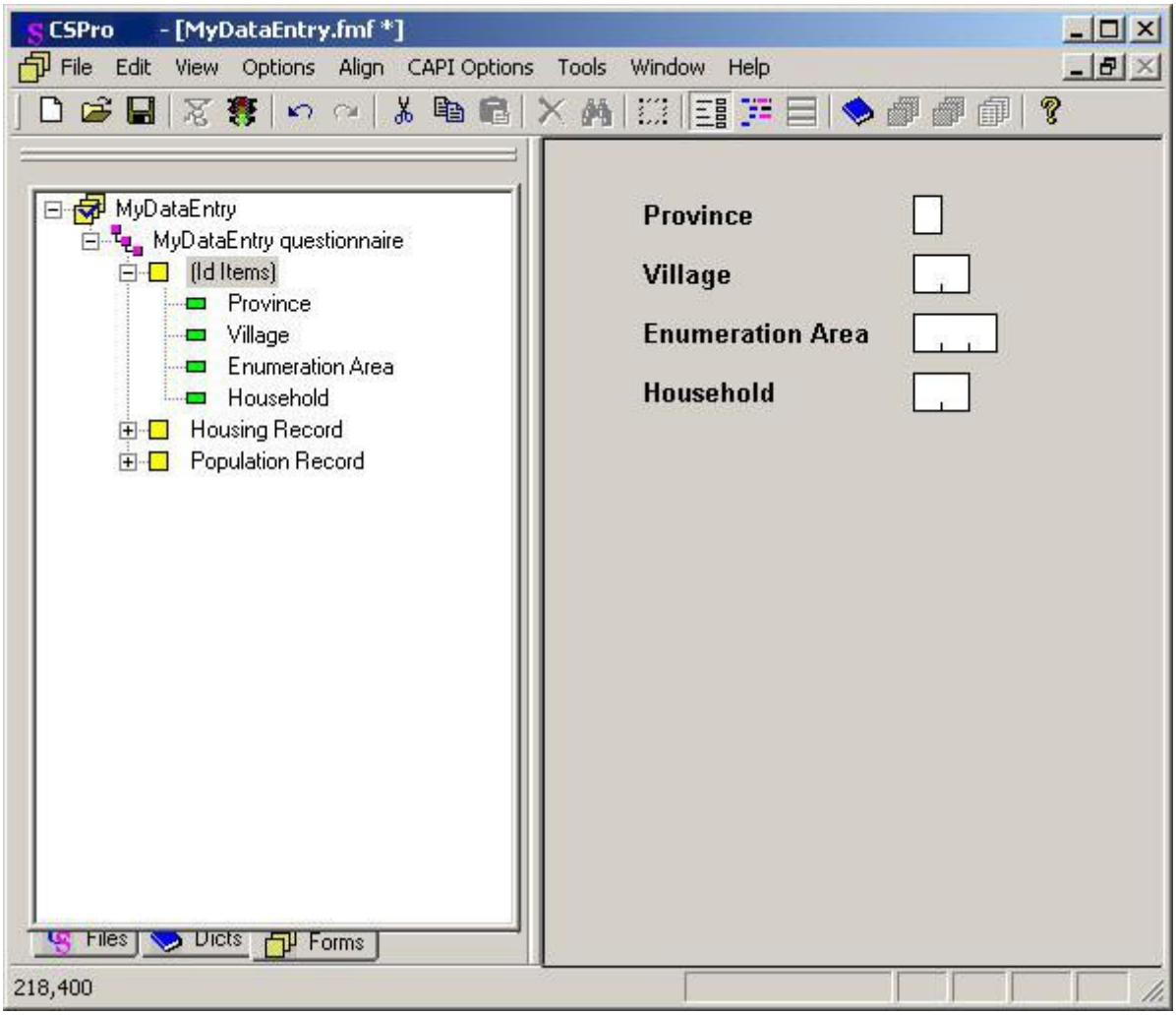
- **Use Extended Controls**
  If the item has a valid value set that can be can be represented by an extended control, the item will be added to the form with a preselected extended control. The control type selected depends on the type and length of the item but can be changed once the item is on the form.

  If this box is left **unchecked**, the control type will default to a text box (keyed entry).

Data Entry Forms Screen Layout

The CSPro window is split in half. The left side contains the data entry tree and three tabs at the bottom. The two tabs of interest when designing forms are the **Dict** [Dictionary] and **Form** tabs at the bottom. By pressing either one you will be able to view the dictionary tree or the data entry tree on the left side.

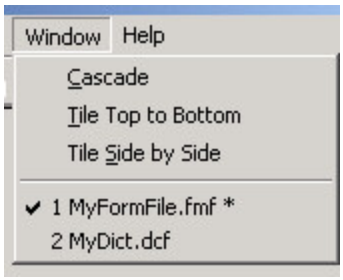
If you generated the screens by default, then the system will display the data entry tree and the Form tab on the left, and the first form created on the right window.



If you opted for creating the forms yourself then the system will display the data dictionary tree and the Dict tab on the left, and a blank form on the right. You are now ready to start dragging items and/or records from the dictionary to the form(s).

If you did not have a data dictionary, then the system will display the data dictionary tree and the Dict tab on the left, and the dictionary window on the right.

Use the **Window** menu to display a list of currently open files at the bottom of the **Window** menu. A check mark appears in front of the name of the file in the active window. Activate a window by choosing the name of its file from this list.

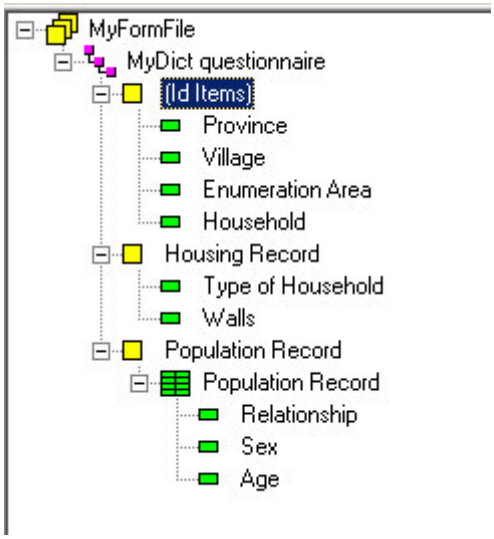


You can also switch between the forms and dictionary window by pressing either or in the Toolbar.

Data Entry Tree

If you generate default data entry screens, the system will create one screen for the identification items and a separate screen for each record type as defined in the dictionary. Press the "Forms" tab on the bottom left of the screen to see the **data entry tree**. The data entry tree will be identical to the dictionary tree; that is, the items will be listed as named and ordered in the dictionary.

However, if you design the forms yourself, you might decide to include more than one record in one screen or combine fields from different records in one screen. In that case the data entry tree will not be identical to the dictionary tree. In any case, the data entry tree has the following items:



- **Forms File:**
  This is the highest level node, i.e., the root node. It is the owner of all code, which is to say [1] level-, record-, and item-related code, [2] user-defined functions, and the [3] global routine.

- **Level:**
  This is the second-tier tree node, just below the root. It has a 1-to-1 correspondence with the same-named dictionary level.

- **Form:**
  This is the third-tier tree node, just below its level. It represents the form and all the items in that form.

- **Roster:**
  It represents the roster and all the items included in the roster.

- **Field:**
  This is the terminal or "leaf"-node; i.e., the lowest accessible level. It has a 1-to-1 correspondence with a dictionary item.

You are free to rename any of the above the unique names via the properties dialog box, but it is recommended that you retain the original name, so that it is easier for you to see which dictionary entity is being referenced. The data entry tree represents the order in which the data entry is keyed.

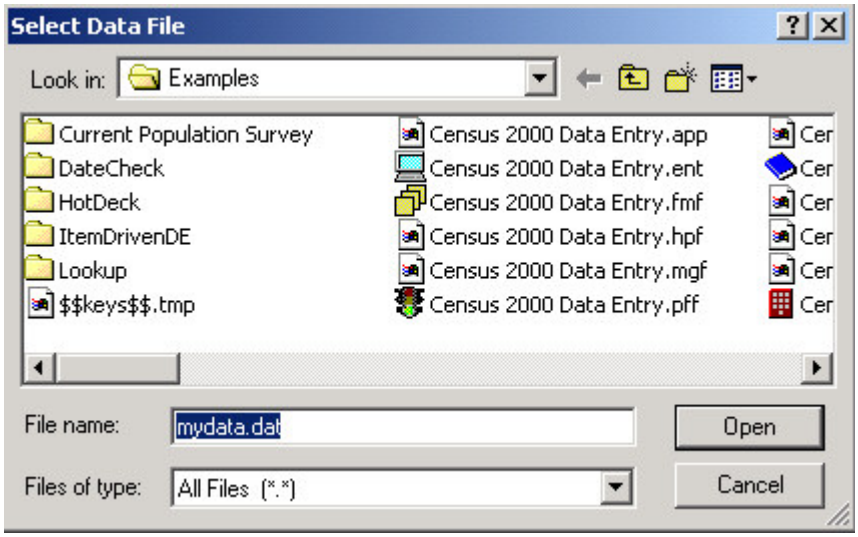
You can change the order of entry by reorganizing the forms or items within the forms by dragging them within the Data Entry tree view. When the operator enters the data, the cursor will follow the order in the tree not the order in the form. When selecting a new edit item, the contents of the logic view will change to display the logic for the selected entity.

Pressing **Ctrl+T** in the data entry tree will allow you to switch between the labels and the names of the items.

Run a Data Entry Application

After the forms are created you can run the application to start entering data in CSEntry.

Press ; or press **Ctrl+R**; or select **Run** from the **File** menu to launch CSEntry.



Enter the name of a file (any extension). If this is a new file, the system will ask if you want to create it and ask for the operator's ID. The system keeps track of the operator's keying record and display the operator's statistics in CSEntry. You may turn off this window in data entry options.

You will, of course, want to test the behavior of your data entry application before using it in a production environment, so this is just a quick way to launch CSEntry.

When your application is ready for production, you will want to launch CSEntry independently of CSPro as it will use less memory this way. To assistance with this, see Run Production Data Entry.

Run Production Data Entry

You can customize CSEntry's behavior for any data entry computer by creating a PFF file. You can then use the PFF file as a command line parameter for CSEntry.exe (the associated filename of this executable). For example, if you name your PFF file "MySurvey.pff", then you can launch CSEntry by invoking:

```
C:\Program Files\CSPro 6.0\CSEntry.exe MySurvey.pff
```

This assumes that CSEntry was installed in the default directory. Your PFF file must have a ".pff" extension.

You can create a PFF file in one of two ways:

• Create it yourself using a text editor (such as Notepad or Wordpad),

• Simply run CSEntry once, and a PFF file will be automatically created for you—it will be placed in the same folder as your data entry application, and it will have the same name as your application, but with a ".pff" extension instead of ".ent". For example, if your data entry application was named "MySurvey.ent", the system-generated PFF would be called "MySurvey.pff".

The following section shows the options available to you for a CSEntry PFF file. Please note that a PFF file is not case sensitive, so you may use any combination of upper and lower case text.

```
[Run Information]
Version=CSPro 6.0
AppType=Entry

[DataEntryInit]
OperatorID=John
StartMode=add
Lock=Verify,Stats
FullScreen=Yes
NoFileOpen=Yes
Interactive=ask
Language=Lingala

[Files]
Application=MyCensus.ent
InputData=.\Prov12\Dist05.dat

[ExternalFiles]
LOOKUP_DCF=.\Prov12.lup

[Parameters]
Parameter=your choice

[DataEntryIDs]
Province=12
District=05
```

The **[Run Information]** block is required and must appear exactly as shown in the example above.

The **[DataEntryInit]** block, as is all its possible entries, is optional. It gives you the opportunity to choose the following run-time characteristics:

OperatorID=John

"John" will be used as the operator ID for the purposes of logging operator statistics. If this line is not present but your data entry application has been set to ask for this, then CSEntry will prompt the operator for one at run time.

StartMode=add

CSEntry will drop immediately into add mode. If this line is not present, one of two things will occur: [1] if the data file does not exist, then the operator will be dropped into add mode; or [2] if the data file does exist, then CSEntry will wait for the operator to choose their desired mode. (Note that their choices may be constrained due to options indicated in "Lock," the next feature). The other two permissible entries are "modify" or "verify".

If start mode is modify, the word modify may be followed by a semicolon and the id of an existing case, for example: **StartMode=modify;0102003**. This will open CSEntry in modify mode and open the case indicated by the id. If start mode is add, the word add may be followed by a semicolon and the id of an existing case that has been partially added. This will open CSEntry in add mode and open the partially added case to continue adding.

Lock=Verify,Stats
This option tells CSEntry which modes an operator will not have access to. Therefore, in this example the operator can not enter Verify mode, nor see data file statistics. This parameter can be any combination of "Add", "Modify", "Verify", and "Stats", separated by commas.

FullScreen=Yes
CSEntry will open the application in full screen mode, with no case tree on the left. The parameter "NoMenus" will open CSEntry in a tablet-friendly mode whereby the case tree and the CSEntry menus and title bar are not shown.

NoFileOpen=Yes
From within CSEntry, the system will not permit the operator to open another data file if this is set to "Yes". However, if you fail to name the required files in the PFF file, the operator will, initially, have to supply them. But once they have been chosen, the operator can not open another file from within CSEntry.

Interactive=Both,Lock
CSEntry will display both out-of-range and errors generated from the errmsg command in interactive mode. This setting is locked, so the operator cannot change it. The parameter "Both" can be replaced with "Errmsg" error message only, "Range" out of range only, "Ask" operator will be asked what type of messages to display, or "Off" interactive mode disabled. The parameter "Lock" is optional. If it is not present, the operator can change the setting using the Options/Interactive Edit Options menu item. "Lock" is ignored for "Off" (always locked). If the Interactive line is not present, "Ask" is assumed.

Language=Lingala
If running a CAPI application, this option specifies the starting language of the question text. The parameter must match the name of the language specified in the question text editor.

The **[Files]** block is required, as is the "Application" entry within it. "Application=" names the data entry application you have developed,. "InputData" is optional, and names the data file you will be creating, modifying, or verifying via this data entry application. If you do not name the data file to work on, CSEntry will prompt the operator to supply one. If the operator fails to provide one, CSEntry will not run.

If the **[ExternalFiles]** block is present, it means that a second dictionary was linked to the data entry application. In the example above, "LOOKUP_DCF" is the internal (unique) dictionary name, and "Prov12.lup" is the name of the data file which contains the lookup codes. If there is a second dictionary linked to your application and you fail to name it in your PFF file, the operator will be prompted to provide it. If the operator fails to do so, CSEntry will not run.

If you would like to pass in a command-line parameter to your data entry program, you would do so via the [Parameters] block, using the Parameter command. The parameter can be any length, although the

alphanumeric variable that retrieves the value in your program (via the sysparm function) must be large enough to accommodate it. Once the parameter string is retrieved, it can be parsed for further usage.

The [**DataEntryIDs**] block is for use with any persistent IDs you have defined. CSEntry will assign the specified values to the indicated persistent fields when a new data file is created. This feature allows automatic definition of persistent fields, such as batch ids. However, if you provide values and run this on an already-existing data file, and the PFF file values do not match the values in the data entry file, the PFF values will be ignored. The syntax is as follows:

```
<unique-dict-name>=<numeric-value>
```

Generate Binary Data Entry Application

Binary data entry applications provide security during a production data entry operation. A binary data entry application consists of one single file with extension .PEN, which includes the same information as the set of text files that normally make up a data entry application.

This generated file cannot be changed in any way. It cannot be opened by the CSPro designer and it cannot be read in a text editor. If you make changes to the data entry application, you must generate the binary application again.

The binary application will run identically in CSEntry to a normal (text) application. An operator will not see any difference in behavior.

To generate a binary application, open the application in the normal way, then from the **File** menu select "Export Application" and then "Export to Portable Binary (.pen)."

Before CSPro 6.0, binary data entry applications had the extension .ENC. The 'P' in the new extension stands for "portable," which indicates that the file can be run not only on Windows machines, but also on Android devices. CSPro can no longer read .ENC files and if you would like to run an application with such a format, you will have to use an older version of CSPro.

Change Data Entry Characteristics

Change the Order of Entry

During data entry, the forms will be shown to the keyer in the order in which they appear in the forms tree. Within a form, the cursor will move among the fields in the order in which they appear in the forms tree.

To change the form order, simply drag and drop on the form tree. For example, suppose you currently have Form A, Form C, Form D and Form B in your level. The forms tree will now show the following:

- Form A
- Form C
- Form D
- Form B

If you drag the form icon for **Form B** and drop it on top of **Form C**, the forms tree will now show:

- Form A
- Form B
- Form C

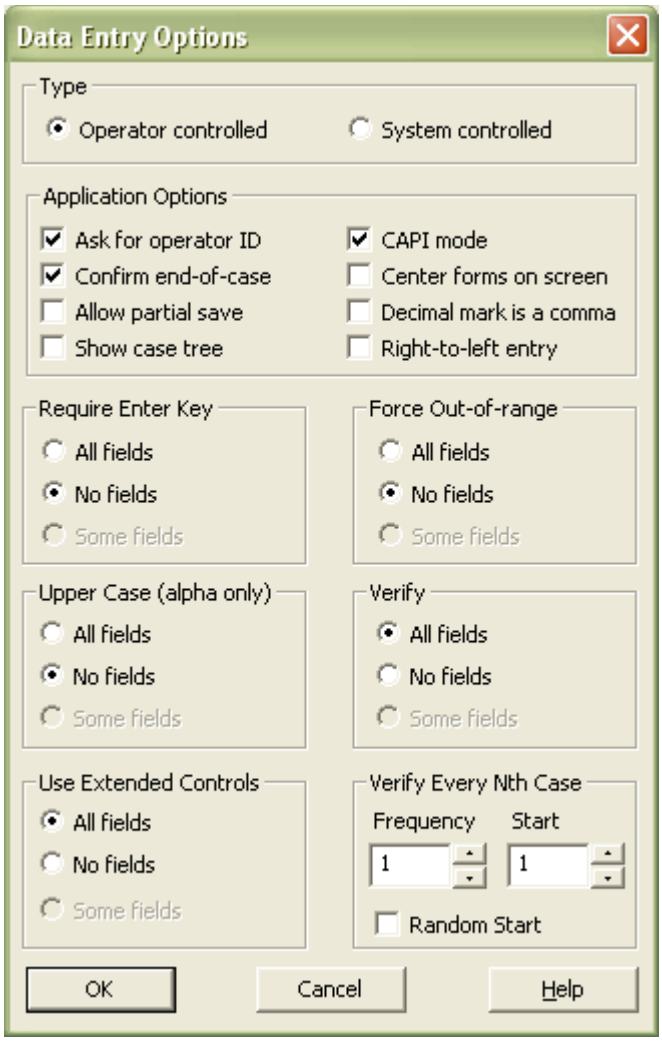
• Form D

Similarly, to change the order of fields within a form, use drag and drop on the forms tree.

To change the order of fields in a roster you must drag and drop within the roster rather than on the tree. You can change the default order in which the forms and fields will be keyed by using logic in the application.

Change Data Entry Options

Select "Data Entry" from the **Options** menu to change any of the following:



• **Type**
This choice is very important and will have a large effect at data entry time. See Operator vs System Controlled for more information.

• **Ask for operator ID**
If this box is checked, CSEntry will prompt the operator to enter an operator ID.

• **Confirm end-of-case**

If this box is checked, CSEntry will prompt the operator to accept the case at the end of each case entered.

• **Allow partial save**
If this box is checked, CSEntry will allow the operator, when in add, modify, or verify mode, to save a case which has not been completed.

• **Show case tree**
If this box is checked, CSEntry will allow the operator to view a tree on the left showing each item in the case currently being added, modified, or verified and its value.

• **CAPI mode**
If this box is checked, CSEntry will display the Computer-Assisted Personal Interviewing (CAPI) window. The top part is for question text (to be read during the interview); the bottom is for the normal form content.

• **Center forms on screen**
If this box is checked, CSEntry will center forms horizontally in the display window.

• **Decimal mark is a comma**
If this box is checked, CSEntry will use a comma instead of a period when showing numeric fields with a decimal component. The keyer must also type a comma instead of a period to enter the values after the decimal mark.

• **Right-to-left entry**
If this box is checked, CSEntry will orient rosters in a right-to-left manner, which is useful for languages like Arabic. The first column of a roster will be the furthest right in the roster and the roster will scroll from right to left. Unlike the above options, this is a form-specific option, so if you are using multiple forms, be sure to specify this option when editing each form.

• **Require Enter Key**
At data entry time, the operator may be required to press the **Enter** key to advance to the next field, or the system may advance automatically when the field is filled with the correct number of digits. In the latter case, the operator can still advance by pressing **Enter** if the digits are not all filled. You may set this attribute individually for each field.

- All fields
All fields in the forms file require Enter key to advance. This option adds keystrokes for the operator, but allow the operator to control the cursor movement. Thus it is more consistent with the "heads-up" methodology. Selecting this option will change the setting for **all** fields.

- No fields
All fields in the forms file advance automatically. This option means the operator will ultimately hit fewer keystrokes, but will have to be aware that the cursor will move automatically when the correct number of digits has been entered in a field. This option is more consistent with the "heads down" methodology. Selecting this option will change the setting for all fields in the form.

- Some fields
You cannot select this option; it is permanently deactivated. If this option is selected, it means there is a combination of enter options in effect—i.e., at least one field in the application has its "Use Enter Key" option checked, and at least one other field in the application has its "Use Enter Key" option unchecked. Select one of the other two options to force all fields to the same setting.

• **Force Out-of-range**
At data entry time, CSEntry shows a message every time the keyer enters a value that is out of range according to the data dictionary. You may set the attribute to override the message and force the out-

of-range value into the data file. If this attribute is not selected, the keyer cannot proceed until a valid value is entered.

- All fields
When this option is checked, all fields in the forms file can be forced. This option allows out-of-range values to be entered in the data file. To be sure of a file with only valid information, such values must be edited and corrected after keying is completed.

- No fields
No field in the form file may be forced. Checking this option prevents any out-of-range values from being entered in the data file. However, this forces the keyer to edit out-of-range values. For this reason, it is recommended that "No fields" be used only where the persons carrying out the data capture operation are qualified to make such on-the-spot editing decisions.

- Some fields
When this option is checked, it indicates that some fields in the forms file permit out-of-range values, and others do not. Checking one of the other two options will force all fields in the forms file to the same setting.

• Upper Case (alpha only)
At data entry time, alphabetic fields can either allow upper- and lower-case text, or they can force any letter entered to upper case. You may set the upper-case attribute for all or some of the alphanumeric fields.

- All fields
Checking this option forces all alphabetic text entered to upper case. This option permits operators to enter text in either case setting while being assured that the output will be upper-case. This is particularly useful for "yes/no" [Y/N] or letter [A/B/C/D] responses.

- No fields
Checking this option permits entry of case-sensitive alphabetic characters, which is particularly useful for names and addresses. All alphanumeric fields in the forms file will permit mixed upper- and lower-case characters.

- Some fields
When this option is checked, it indicates that different fields in the forms file have different settings. Selecting one of the other two options will force all alphanumeric fields in the forms file to the same setting.

• Use Extended Controls
CSPro 4.1 introduced the concept of extended controls. These are dialog boxes that appear during data entry that allow users to choose values from a list rather than key them. These controls are useful for applications run on tablet computers.

- All fields
Checking this option forces all fields with valid value sets to display a suitable extended control.

- No fields
Checking this turns off the extended controls for all fields, forcing each field to be keyed in.

- Some fields
When this option is checked, it indicates that different fields in the forms file have different settings. Selecting one of the other two options will force all fields in the forms file to the same setting.

• Verify

During verification, each item is either verified, that is, keyed again and compared with the value currently in the data file, or not verified, that is, displayed on the screen but not entered or changed. You may set the "verify" attribute for all or some of the fields.

- All fields
During verification, the cursor will pass through all fields in the same manner as during original keying. This option permits maximum verification of data.

- No fields
No data are verified. This option can be useful when only a few fields need to be verified: The "No fields" option can be set globally, and then the "verify" property can be set to "Yes" for those few fields which will require verification.

- Some fields
When this option is checked, it indicates that some fields in the forms file will be verified and others will not. Checking one of the other two options will force all fields to the same setting.

• **Verify Every Nth Case**
During verification, you may choose to verify only a subset of the cases in the data file, instead of verifying all the cases.

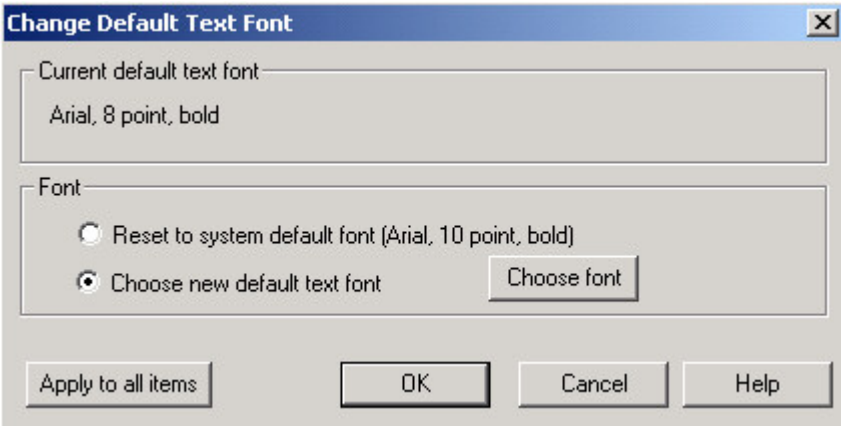
- Frequency
This is the interval between cases that CSEntry will use for verification. For example, if this value is 10, every 10th case will be verified.

- Start
This is the number of the first case in the data file to verify. For example, if this value is 5, and the "Frequency" is 10, cases number 5, 15, 25, etc. will be verified. The case number is determined by the physical order of the cases in the data file. The "Start" must be less than or equal to the "Frequency" value.

- Random Start
You may check this box instead of specifying a "Start" value. CSEntry will then choose a random number for the "Start" value. The random number will be between 1 and the "Frequency" value.

Change Default Text Font

Current default font is what CSPro will use whenever you add new text to any form. From the **Options** menu, select **Default Text Font.**



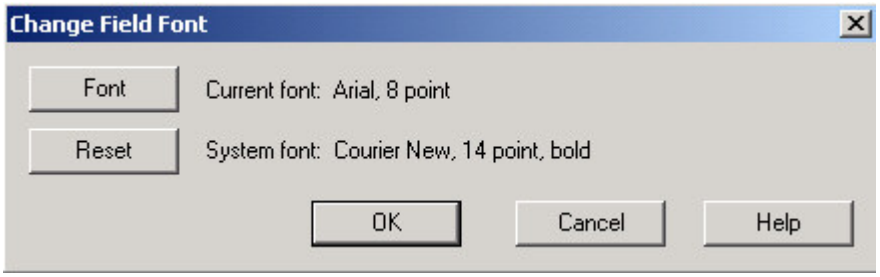
You can change this by using the **Font** radio buttons. If you select **Choose new default text font**, you can then click on the **Choose font** button to customize your own font

If you want to change the font for all text that is already on forms, you must press the **Apply to all items** button.

See also: Change Field Font, Change Text Properties

Change Field Font

From the **Options** menu, select "**Field Font**." If you change the field font you may want to also change the default text font and apply it to all items.



- Press the "**Font"** button to change the font in all the field boxes on all the forms. Changing the size of the font will change the size of the field boxes. You can change the language of the characters by choosing a different "Script" in the font dialog box.

- Press the "**Reset"** button to reset the font in all the field boxes on all the forms to the system default.

See also: Change Text Properties

Change Error Sound

When an error occurs during data entry, an error box is shown of the screen and sound (beep, tone or other sound) is generated.

In order for the sound to be heard:

- The computer must have sound card, with speaker connected and turned on.

- The volume on the sound system must be turned on and sufficiently loud to be heard.

- There must be a sound file associated with the Default Sound(Beep) under Control Panel/Sound.

To change the sound, go to Control Panel/Sound and change Default Sound(Beep) to a different sound file.

Forms Designer

Introduction to Forms Design

The previous screens show you how to create a data entry application using the forms generated by default. However, the CSPro allows you to create, using a single dictionary, one or more forms (screens) for data entry.

This is the design stage of the data entry process. This tool allows you to create new forms, add or modify text, enter lines and/or boxes, add color to the forms or text. If you have a printed questionnaire you will probably want to use it as a guide when deciding text and field placement, as well as the order of entry for the items.

After you have developed forms to your satisfaction, use CSEntry to input the data.

This section contains the following information:

Add Things to a Form
Modify Things in a Form
Change Form Properties

Add Things to a Form

Add a Form

There are three basic ways to add a new (blank) form. Each method will present you with the "Form Properties" dialog box.

• **From the Form Designer Tree tab**
Right-click over any of the tree entries (i.e., a Form File, Level, Form, or Item). A pop-up dialog box will appear. Select the Add Form option.

• **From the Form Designer's Menubar**
Select Edit, then the Add Form option.

• **From the Form itself**
Right-click anywhere over a form. A pop-up dialog box will appear. Select the "Add Form" option.

After you have pressed OK on the "Form Property" dialog box, you will notice on the form tree that the form was placed last in the current level. You can change the order of the forms by dragging forms on the tree.

Add Fields to a Form

• **Drag a Dictionary Item to your Form**
Expand the dictionary tree so that the desired item is visible. Holding down the left mouse button, select the item ( ) and drag it to the form, releasing the mouse button when the cursor is at the desired location on the form. Depending on the drag option settings, either your item or existing sub-items ( ) will be dropped onto the form. For example, if you have dragged an item from a record with multiple occurrences and you have chosen (in the "**Drag Options"** dialog) to roster items when possible, the item will appear as a one-column roster. Dragging additional items from this record and dropping them onto the roster will append the items to the roster.

• **Drag a Dictionary Record to your Form**

Expand the dictionary tree so that the desired record is visible. Holding down the left mouse button, select the record ( ) and drag it to the form, releasing the mouse button when the cursor is at the desired location. Depending on the record's properties and the Drag option settings, the item(s) within your record will either be dropped as individual fields or as a roster.

See also: Change Field Properties

Add a Roster to a Form

CSPro automatically creates a roster, under appropriate conditions, when you drag a dictionary item onto a form. In most cases where a roster is possible, CSPro obeys the **Roster Options** on the drag options dialog box. Make sure this option is Horizontal or Vertical before you begin. In some drag and drop operations a roster is not possible and will not be created. In other drag and drop operations a roster is the only alternative.

Common ways to create a roster include:

- Drag a multiple record ( ) from the data dictionary to a blank form. This will generate a roster containing all the items in the record.

- Drag one item ( ) from a multiple record in the data dictionary to a blank form. This will generate a roster containing only that item. You can then add more items to the roster one at a time.

- Drag an item from a multiple record, or the record itself, to a form that contains only items from another single record or ID items.

- Drag a multiple item or sub-item ( ) to a form. If you have a multiple item that has sub-items, and you want to create a roster of the sub-items, make sure you have the "**Use sub-items if present"** box checked in the Drag Options dialog box.

See also: Add Things to a Roster, Change Roster Properties

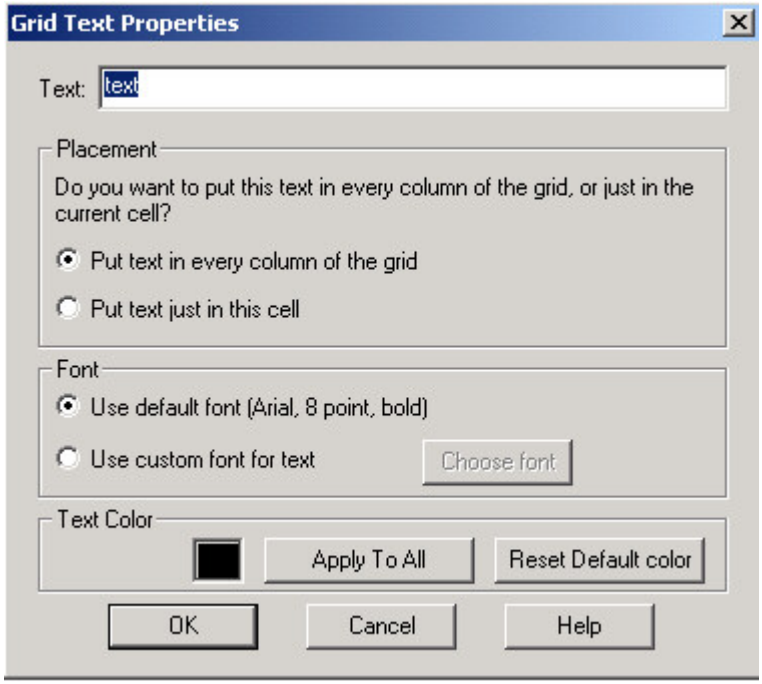
Add Things to a Roster

- **Add Items**

  Rosters can only include items from the same multiple record, or sub-items from the same multiple item. If you created the roster by dragging the entire multiple record or item onto the form (or by generating a set of forms), there are no more fields that can be added to the roster.

  Otherwise, you can drag an appropriate field from the data dictionary and drop it on the roster. CSPro will add a column to the end of the roster. If you don't want the field's column to be at the end, you can reposition the column after you add it by dragging and dropping the field in the desired position on the form, or in the list of fields on the Form tree. Be sure to drop the data dictionary item on top of the roster; otherwise, you will create a new roster.

- **Add Text**



Right click on the light gray space in the desired column of the roster and select "Add Text". You will see the Grid Text Properties window. Note that you can choose the text placement and select whether the text will go only in the cell in which you clicked, or if it will go at the same position in every cell in the column. You can change this attribute later if you want.

• **Add Boxes**
Right click on the gray space in the roster and select "Add Boxes". Note that you can choose whether the boxes will go only in the cell in which you clicked, or if they will go at the same position in every cell in the column. Drawing boxes in a roster is essentially the same as drawing boxes on a form.

Add Text to a Form

When you add a field to a form by dragging it from the dictionary tree, the dictionary item's label is automatically placed on the form. You may also add other text to the form (a heading across the top, for example) by doing the following:

• Right-click on the form at the point where you want the text to start.

• Select **Add Text** from the pop-up menu.

• Type in the text, and press **Enter**.

See also: Change Text Properties

Add Lines or Boxes to a Form

CSPro also allows the user to draw boxes, as a means to both help visually organize your data and make the layout of your form look more professional. For example, if you wish to place fertility data on one

portion of your form and then indicate to the viewer that these data are related, you could draw a box around the related items.

When you select multiple items with the mouse, you'll notice during the selection process a box that drags with you to show what you're including. To draw a box on a form, it seemed logical to have that same mechanism at work, so we've introduced the **Select Items/Boxes** button. Click on to toggle between the two states. When you first click on this button it will appear depressed, and a floating toolbar will appear with the following buttons:

Click	To
	Allows you to toggle states between selecting items and drawing boxes without having to close down the toolbar
	Draw a box with an etched edge
	Draw a box with a raised edge
	Draw a box with a thin edge
	Draw a box with a thick edge

When you have finished drawing boxes and no longer need the **Box-Draw** toolbar, close it down by either toggling the button, or close the Box-Draw toolbar.

Modify Things in a Form

Selecting Items

When the Forms Designer first opens, the mouse is in selection mode. That is, if you click on a field, roster, or text item, the item becomes selected. Similarly, if you press the left mouse button and hold it down while dragging over a group of fields, rosters, and/or text items, all of those items will be selected. You can then choose to do operations on the selected item(s), such as moving or deleting them. If one item is selected, you can also review its individual (field/roster/text) properties. You can also hold down the Ctrl key while individually clicking on each item to be selected with your left mouse button.

To quickly select several fields in their entirety, just grab their data entry boxes. This will cause automatic selection of any accompanying text, as well. To quickly select just the text portion of several fields, be sure that the selection field visible on the screen does not touch any of the data entry boxes.

Methods of selection:

- To select several items, hold down the left mouse button while dragging a selection box around the desired items. Or you can also hold down the Ctrl key while individually clicking on each item (box or text) to be selected with your left mouse button.

- To select both a data entry box and its associated description text, hold down the Shift key while clicking on either the entry box or its associated text.

- To select several data entry boxes and their associated descriptions, hold down both the Shift and Ctrl keys while individually clicking on either the edit box or its text for each different field.

Field Properties

Right-click on the field and select Properties to get to the Field Properties Dialog Box. In CSPro you may define the following special types of fields:

- **Persistent fields**
  Persistent fields are ID fields that take the value from the previous case in the data file as their default. Persistent fields are typically used for geographic IDs that change very seldom from one case to another. These fields are shown as light gray boxes on the form. In CSEntry, the operator must press a special key (F7, or Ctrl-P on the Pocket PC) to change the value of a persistent field. You can make any ID field (except for mirror fields) persistent, as long as it is already on a form.

- **Sequential fields**
  Sequential fields automatically increment at data entry time. They are commonly used as occurrence-number fields in multiple groups. A sequential field takes the value `1` on the first occurrence. For subsequent occurrences, CSEntry will use the value of the previous occurrence and add `1`. If the field is not also marked as "protected", the operator may change the sequence at any time by simply keying a new value, and from that point, CSEntry will use this new value to continue the sequential incrementation. You can make any field (except for mirror fields) sequential, as long as it is already on a form. You can define your own kinds of sequential behavior for fields by writing pre-processing logic. In this case, do not use the sequential field attribute.

- **Protected fields**
  Protected fields are not keyed during data entry. Protected fields are commonly used to display a value that is calculated elsewhere (for example, the sum of other keyed fields). You must write logic to set the value of a protected field. You can make any field protected, as long as it is already on a form.

- **Upper Case fields**
  Alphanumeric fields can be upper case. This means that every alphabetic character that is keyed will be forced to upper case.

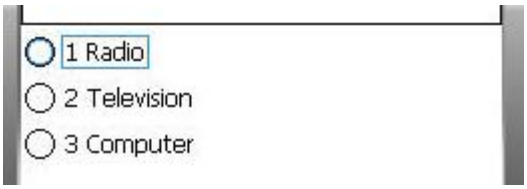
- **Mirror fields**
  Mirror fields show the value of a previously-entered field on the screen. The cursor never goes to a mirror field during data entry. Mirror fields are useful to display values from one screen on another screen. Any field from a single-occurrence group can be a mirror field. A common use of mirror fields is to show the geographic IDs on all screens. The first form might contain the geographic (level) ID fields which the operator keys in, and subsequent forms might contain the geographic ID mirror fields, which will show the operator the ID values even when the ID form is not on screen. The first time you drag a dictionary item onto a form you create the normal entry field. On each subsequent occasion that you drag the same dictionary item onto a form, you create a mirror field.

See also: Change Field Properties, Field Properties (for Multiple Fields)

Field Capture Type

This page describes choices available to applications that are prepared for Pocket PC (PPC) data entry. To view choices for non-PPC applications, view the Extended Controls page. On a Pocket PC (PDA), every field will be one of the following types:

- **Radio Button**
  The field must have discrete values defined in the first value set in the data dictionary. Each value will be displayed as a separate radio button on the Pocket PC. For example:



• **Check Box**

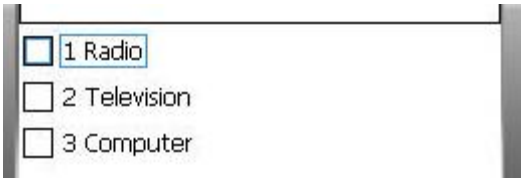
Check Boxes are used to capture a "multiple response" variable. To do this, you must first set up the item and its value set in the data dictionary in the following way. The item must be of type "Alpha", and must have a length equal to the number of possible responses. The value set must have that same number of discrete values. Each value corresponds to one of the responses and will be displayed as a separate check box on the Pocket PC. When the boxes are checked, the corresponding values will be put into the alpha field, from left to right.

For example, consider a multiple response question, "Which of the following do you have in the household". The possible answers are "Radio", "Television", "Computer".

In this example you would define an "alpha" item in the data dictionary with length three and make a value set with three values:

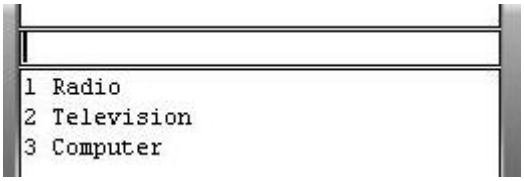
Value Label	From	To
Radio	1	
Television	2	
Computer	3	

This would appear on the Pocket PC as:



• **Drop Down**

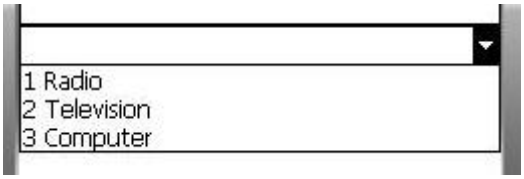
The field must have a value set in the data dictionary. Each value in the first value set will be displayed as a separate line in the response area. The user can tap on the appropriate response, or enter the numerical value.



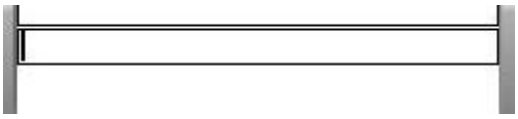
• **Combo Box**

The field must have a value set in the data dictionary. Each value in the first value set will be displayed as a separate line in the response area. The user can tap on the appropriate response, or enter the

numerical value. This is very similar to the Drop Down, except that the user can tap on the small arrow on the right to open and close the list.



• **Text Box**
The is the capture type when the field has no values defined in the data dictionary. The user must enter a response via the virtual keyboard.



• **Date**
The field must have a length of 4, 6, or 8. The choice of "Date format" will determine how the data will be stored in the data file. This capture type gives the user a nice visual interface from which to choose the date.

Move Things

When you drag a dictionary item onto a form, it will be placed on the form at the point where you released the mouse button. The dictionary label will be used as identifying text for the field, and it will be placed on the form according to the Drag Options in effect, which may mean the item becomes rostered. Once the field is on a form, you can fine-tune its placement.

• **Move a Field**
To move a field, select the box and drag it to the desired location. Each field has a text item associated with it. You can see the text by holding down the Shift key and clicking on the field. This will select both the field and its text. You can now move both of them together by dragging and dropping. You can move a field's associated text separately by simply dragging and dropping only the selected text.

- **Move a Roster**
  To move a roster, select it by clicking on the gray space in any cell or in the small box in the top left corner of the roster. Drag it to the desired location. You can also resize a roster.

- **Move Text**
  To move any text, simply select and drag it to the desired location.

- **Move a Block of Items**
  First select a block of items. Then, move the mouse over one of the tracker regions selected. When you see the mouse cursor change from to , you are ready to move the block. Press down with the left mouse button and drag it to its new location.

  A tracker (or tracker region) refers to the item(s) that has(have) been selected with the mouse. Visually, you will see a heavy dashed line drawn around the item(s).

Align Things

If you have developed your form by dragging individual items from the dictionary to the form, it is probable that the fields are not precisely aligned to the left and/or right margins of the form. To correct this problem, you need only select the items you wish to align, and choose one of the alignment schemes below.

- **Left**
  This will take the left-most item (whether the text of a field, the data entry box of a field, a roster, etc.) and use it as the basis for aligning all other selected elements. This alignment method works best for fields that have been placed on the form in a top-to-bottom manner, with text either to the left or right of the respective data entry box.

- **Center**
  This will take the [horizontal] mid-point between the left-most and right-most items (whether the text of a field, the data entry box of a field, a roster, etc.) and use it as the basis for centering all the selected elements. This alignment method works best to center text items that have been placed in a top-to-bottom manner, or to center the text of a field over the data entry box.

- **Right**
  This will take the right-most item (whether the text of a field, the data entry box of a field, a roster, etc.) and use it as the basis for aligning all other selected elements. This alignment method works best for fields that have been placed on the form in a top-to-bottom manner, with text either to the left or right of the respective data entry box.

- **Top**
  This will take the top-most item (whether the text of a field, the data entry box of a field, a roster, etc.) and use it as the basis for aligning all other selected elements. This alignment method works best for fields that are spread out across the form in a left-to-right manner, with text either above or below the respective data entry box.

- **Mid**
  This will take the [vertical] mid-point between the top-most and bottom-most items (whether the text of a field, the data entry box of a field, a roster, etc.) and use it as the basis for aligning all the selected elements on the mid-point. This alignment method works best to center text items that have been placed in a left-to-right manner, or to center the text of a field next to the data entry box.

- **Bottom**

This will take the bottom-most item (whether the text of a field, the data entry box of a field, a roster, etc.) and use it as a basis for aligning all other selected elements. This alignment method works best for fields that are spread out across the form in a left-to-right manner, with text either above or below the respective data entry box.

- **Evenly Horizontal**
  This will evenly space three or more items (whether the text of a field, the data entry box of a field, a roster, etc.) horizontally. The left-most and right-most items will not move. This alignment works best to evenly space data entry boxes across the screen.

- **Evenly Vertical**
  This will evenly space three or more items (whether the text of a field, the data entry box of a field, a roster, etc.) vertically. The top-most and bottom-most items will not move. This alignment works best to evenly space data entry boxes one above the other.
  Please note that aligning items could have unintended results. For example, if your fields are spread across the form from left to right and you choose to left- or right-align them, they will end up superimposed, one field on top of another. Similarly, if your fields are displayed in a list-type fashion down the page and you choose to top or bottom align them, they will again end up superimposed, one field on another. If this happens, you should press **Ctrl+Z** to undo the change and restore your previous layout.

Undo and Redo Changes

CSPro keeps track of the last dozen changes you have made to your forms on an "undo stack". Beware that not all changes can be undone.

If you have made a mistake and want to undo it, press on the toolbar; or from the **Edit** menu, select **Undo**; or press **Ctrl+Z**. CSPro will try to restore your forms to the state previous to last change you made. To undo the next-to-last change, press the **Undo** button again.

Sometimes you may undo several changes and realize you have gone too far back. Press on the toolbar; or from the **Edit** menu, select **Redo**; or press **Ctrl+Y**. Redo is an "undo" of an undo.

Cut, Copy, or Paste Things

To move things around in the Dictionary use **cut, copy**, and **paste**. **Cut** will delete the material from the dictionary and place it on the clipboard. **Copy** will just place a copy of the material on the clipboard. **Paste** will place a copy of the material on the clipboard into the dictionary.

- **To cut things**

  Select the material you want to cut, then click on the toolbar; or from the **Edit** menu, select **Cut**; or press **Ctrl+X**.

- **To copy things**

  Select the material you want to copy, then click on the toolbar; or from the **Edit** menu, select **Copy**; or press **Ctrl+C**.

- **To paste things**

Select the place where you want the records, items, or values to be pasted, then click on the toolbar; or from the **Edit** menu, select **Paste**; or press **Ctrl+V**.
You can paste cut or copied material to more than one location. Use undo if you paste to the wrong place.

See also: Undo

Resize and Reposition Things in a Roster

• **Change roster size**
Select the roster by clicking on the gray space in any cell. You will see eight small black squares [the resize handles] around the edges at the corners and sides of the roster. Move the mouse pointer on top of any resize handle until the mouse pointer changes to a double-headed arrow. Click and drag to the desired size. CSPro will automatically create or remove scrollbars as needed.

• **Change column width**
Move the mouse pointer over the right edge of the column you wish to resize until the mouse cursor changes to a double-headed arrow. Click and drag to the desired width.

• **Change row height**
Move the mouse pointer over the bottom edge of the row you wish to resize until the mouse cursor changes to a double-headed arrow. Click and drag to the desired height.

• **Change order of columns**
At data entry time, fields are keyed in the same order in which they appear in the roster columns, left to right (or top to bottom if the roster orientation is vertical). To change the order of columns, click on the column heading and drag to the desired position. A gray separator line will tell you where you are about to drop the selected column.

• **Move fields, text, or boxes**
Select the object by clicking on it. Move the mouse pointer over the object until the mouse pointer changes to a four-headed arrow. Click and drag to the desired position.

Join and Split Roster Columns

By default, CSPro puts one field in each column. You can put two or more fields in a column by using the **Join** facility.

• **To join two or more columns**
Select two or more adjacent columns. You can do this by holding down the Ctrl key and clicking on each column. Right-click and choose **Join** and type in the text for the joined column.

• **To split a column**
Click on the column heading to select it, then right-click and choose **Split.** If a column already has more than one field in it (from a previous join), you can **Split** the column so that there is one column for each field.

Delete Form Elements

- **To delete a field from a form:**
  Click on the field on the form, or on the forms tree, then press **Delete**, or right-click and choose "**Delete**."

- **To delete a form:**
  Click on the form on the forms tree and press **Delete,** then choose "**Delete form"** from the main menu at the top

Matching the Application to the Data Dictionary

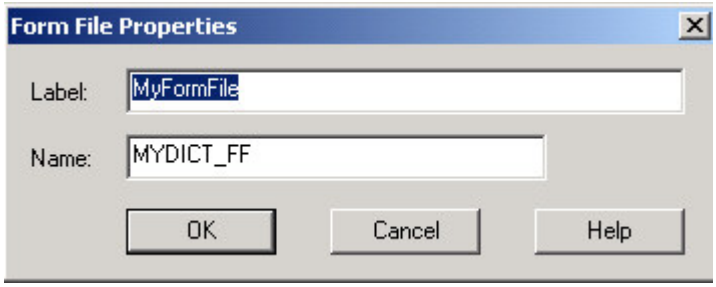
Whenever an existing application is specified, the system automatically checks to make sure the application matches the data dictionary. This is to alert you in case a change was made to the data dictionary after the application was last saved, or in case you typed in the wrong data dictionary name.

If the application does not match, the system will indicate that discrepancies exist and will ask you to permit the updating of the application. If you prefer not to accept the update or if you wish to first investigate the cause of the discrepancy, you may answer "No" and the system will not open or update the application. You can then verify that you have the correct dictionary and review any changes that might have been made since the last time the application was opened.

Change Form Properties

Change Forms File Properties

Right click on the form file ( ) on the tree (top most entry) and choose Properties.



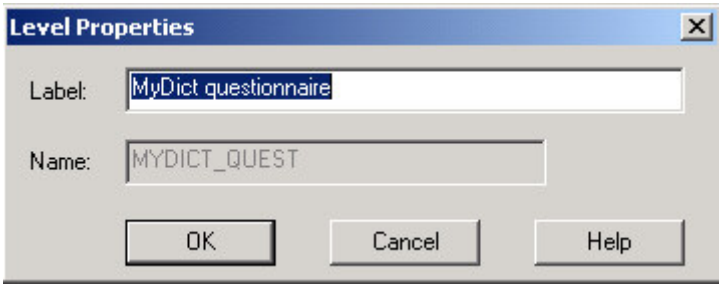
- **Label**
  This is descriptive text that helps you identify the current forms file. It may contain any characters (including blanks) and be up to 120 characters long.

- **Name**
  This is the name of the forms file which you would use when writing programming logic. It may be up to 32 characters long, and must consist of letters, digits and the underscore ('_') character. It must not begin or end with underscore.

  You can see either labels or names on the forms tree. Press **Ctrl+T** to switch back and forth between them.

Change Level Properties

Right click on the level ( ) on the tree and choose "Properties."



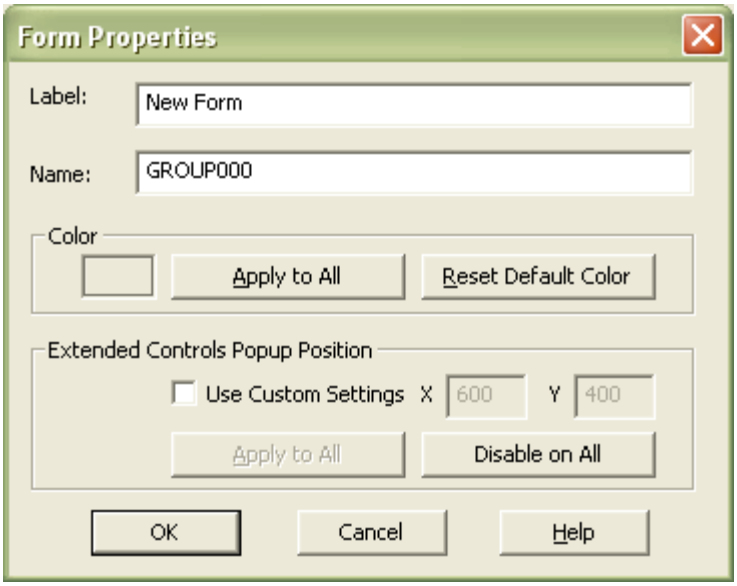
- **Label**
  This is descriptive text that helps you identify the current level. It may contain any characters (including blanks) and be up to 120 characters long.

- **Name**
  This is the name of the level which you would use when writing programming logic. It may be up to 32 characters long, and must consist of letters, digits and the underscore ('_') character. It must not begin or end with underscore.

  You can see either labels or names on the forms tree. Press **Ctrl+T** to switch back and forth between them.

Change Form Properties

Right-click on the form ( ) on the tree and choose "Properties," or right click on empty space on the form itself and choose "Form Properties."



- **Label**
  This is descriptive text that helps you identify the current forms file. It may contain any characters (including blanks) and be up to 120 characters long.

- **Name**
  This is the name of the forms file which you would use when writing programming logic. It may be up to 32 characters long and must consist of letters, digits, and the underscore ('_') character. It must not begin or end with underscore.

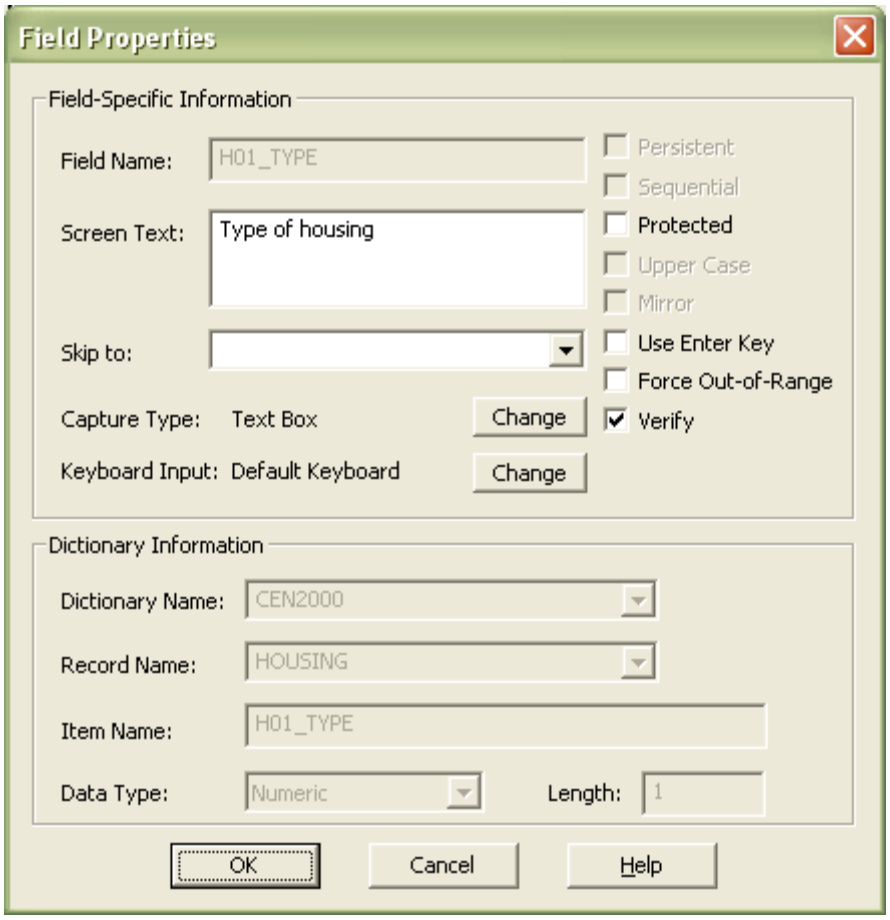
  You can see either labels or names on the forms tree. Press **Ctrl+T** to switch back and forth between them.

- **Color**
  The button shows the color of the form. To change the form color, click on this button, select a new color and click **OK**. You can change the form color back to what it was originally (usually gray) by clicking on the "**Reset Default Color**" button. You can make all forms the same color by clicking on the "**Apply to All"** button.

- **Extended Controls Popup Position**
  If your program uses extended controls, you can specify where on the form you want the popup boxes to appear. Enable this option by clicking on the "**Use Custom Settings**" checkbox and then specify the X (horizontal) and Y (vertical) coordinates for the boxes. The (0,0) origin point of the form is located at the form's top left corner. When editing a form, the status bar will show the X and Y coordinates of your cursor, which is one way to determine your desired coordinates. Clicking "**Apply to All**" will set the capture position for all forms. Clicking "**Disable on All**" will turn off this functionality for all forms and the popup windows will appear in their default positions, as close to the field as possible. The capture position can also be set in logic using the setcapturepos function.

Change Field Properties

Right click on the field in the tree or on the form and select "Properties." You can also press Alt+Enter to bring up the properties window.



Field-Specific Information

- **Field Name**
  This is the name of the dictionary item associated with this field. It is the name you use to refer to this field when writing logic. Mirror fields will show the dictionary name with three digits appended to it. You cannot change this property.

- **Screen Text**
  This is the text that is associated with the data entry box on the form. You can hold the **Shift** key and click on a data entry box to see its associated text.

- **Skip to**
  This is the name of the field that will be skipped to if the operator presses the plus (+) key on the numeric keypad. If the "skip to" field it is blank and the plus key is pressed, CSPro skips to the next field in sequence. "Skip to" is available only in operator-controlled data entry mode.

- **Capture Type**
  This attribute allows you to specify an extended control (a popup window that shows entries in the value set) for the field, or to change the appearance of an alpha field.

- **Keyboard Input**
  This attribute allows you to choose a keyboard ID that specifies what keyboard input to use for the field, which may be useful in multiple language environments.

- **Field type**

Check on the box to activate the field, either "Persistent," "Sequential," "Protected," "Upper Case," or "Mirror."

- **Use Enter Key**
  Check this box if you want to force the data entry operator to press the Enter key to advance to the next field. If left unchecked, the cursor automatically advances to the next field (after the maximum number of characters have been entered).

- **Force Out-of-range**
  Check this box if you want to allow the operator to enter an out-of-range value, that is, a value which is not defined in the dictionary for this field. If left unchecked, the operator can only enter values defined in the dictionary. If checked, the operator can force the acceptance of a non-valid value during data capture.

- **Verify**
  Check this box if you want to verify the field when the operator is in verification mode. If left unchecked, verification is skipped. If checked, verification will occur as follows: after each field is keyed, the value entered is compared with the value currently in the data file. If there is a difference, an error message is displayed, and the field must be reentered.

Dictionary Information

A form field must be based on an existing dictionary item. The properties listed below give you some information about this dictionary item.

- **Dictionary Name**
  This is the internal name of the dictionary to which this dictionary item belongs.

- **Record Name**
  This is the internal name of the record to which this dictionary item belongs.

- **Item Name**
  This is the internal name of the dictionary item itself. Note that if this is a non-mirror field, the "**Field Name"** and "**Item Name"** will be identical. If this is a mirror field, the "**Field Name"** will be based on the "**Item Name**."

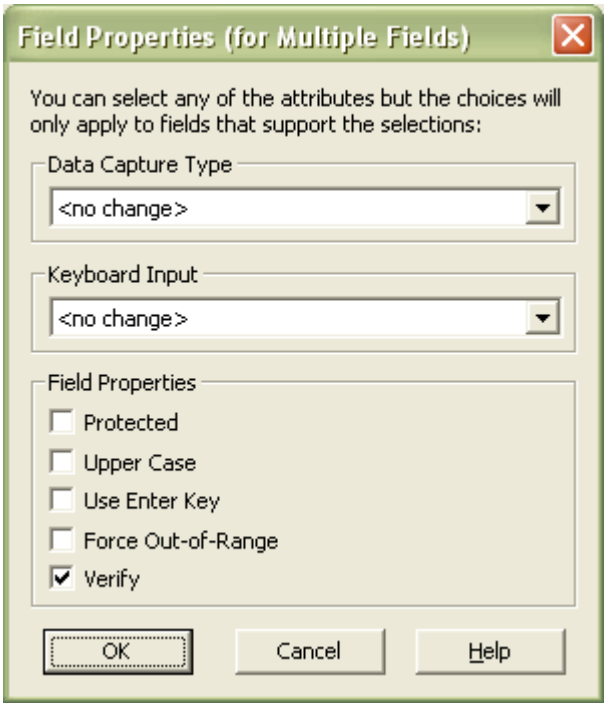
- **Data Type**
  This is the type of data expected during data entry. It is usually "**Numeric**."

- **Length**
  This is the maximum number of characters that will be allowed during data entry.

See also: Field Properties, Field Properties (for Multiple Fields)

Change Field Properties (for Multiple Fields)

Select multiple fields on a form and then right-click on a field and select "Properties," or press Alt+Enter.



You can select options that you would like to apply to many fields at one time. The attributes will only apply to fields that support the options. For example, if "Upper Case" is selected, that option will be ignored for numeric fields and will be applied only to alpha fields.

The Data Capture Type options allow you to change the type of extended control used for the field. It will list the union of valid possibilities for the selected fields, but if a change is requested it will apply the change only to fields that can support the specified type. For example, selecting "Radio Button" will only apply that option to fields that have a value set with discrete values. The capture type of fields without such value sets will remain unchanged.

See also: Change Field Properties, Field Properties

Keyboard Input

To manually change the keyboard input for a field, view the field's properties and select Keyboard Input -> Change.

You can specify a specific keyboard input method for a field if you would like to avoid making keyers change input methods manually. The list is populated by currently activate input methods. If an input method is not listed, activate it first, typically by modifying the Regional and Language Options in the Control Panel.

The keyboard input method will only apply to the selected field. After the keyer moves away from the field, the input method will return to the default keyboard setting. For example, suppose that there are four fields:

Field Name	Keyboard ID
Field1	0
Field2	0
Field3	1067
Field4	0

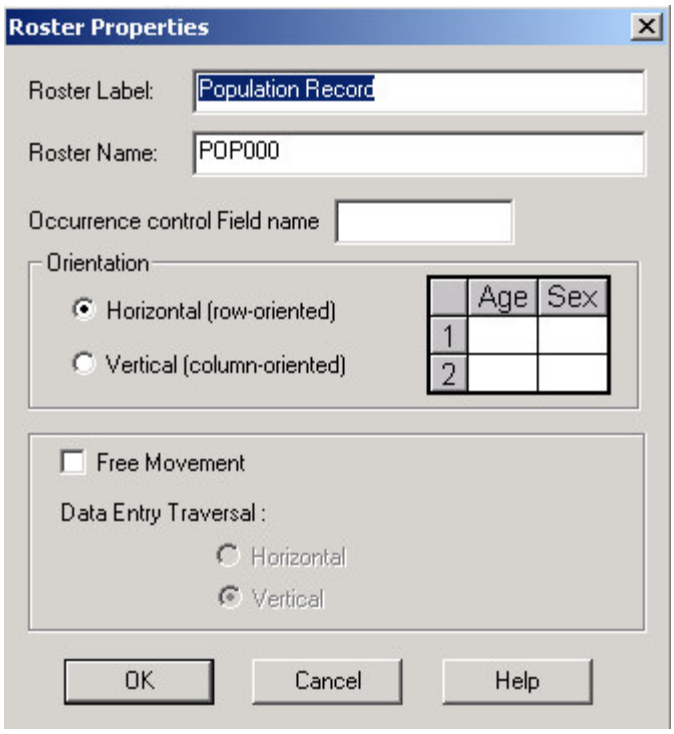
In the above scenario, the keyboard input method will only be modified for Field3, which will be entered in Armenian. However, suppose that the keyer manually changes the keyboard from the default keyboard to Arabic while on the Field2. In that case, Field3 will be entered in Armenian, and then the input method will be changed back to Arabic for Field4. In other words, the Default Keyboard input method resets the keyboard back to whatever input method was last active on another field with a keyboard ID of 0.

When CSEntry loads the data entry program, it also loads any keyboard input methods that are not currently active on the machine. The input methods will be unloaded when CSEntry terminates. However, some input methods require additional files that do not come in a standard Windows installation. CSEntry can only use input methods that contain all the required files. If the files do not exist, the Default Keyboard will be used for those fields. Before deploying your application you will want to ensure that the required input methods are installed on each keyer's machine.

See also: Changekeyboard Function, Field Properties (for Multiple Fields)

Change Roster Properties

Right click on the gray space in any roster cell and choose "Properties."



- **Label**
  This is descriptive text that helps you identify the current roster. It may contain any characters (including blanks) and be up to 120 characters long.

- **Name**
  This is the name of the roster which you would use when writing programming logic. It may be up to 32 characters long, and must consist of letters, digits and the underscore ('_') character. It must not begin or end with underscore.

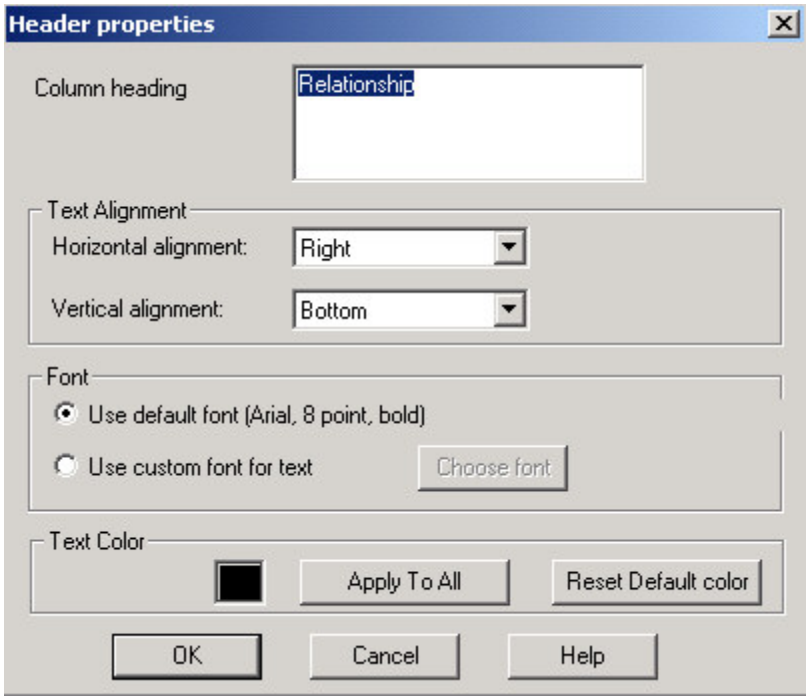
- **Orientation**
  This defines whether the cursor will move from left to right or from top to bottom during data entry.

- **Free Movement**
  This defines the order in which to enter a record with multiple occurrences: all the items in a record n times or each item n times before continuing with the next item

  You can see either labels or names on the forms tree. Press **Ctrl+T** to switch back and forth between them.

Change Column Heading Properties

Right click on a column heading and choose **Properties**.



- **Column Heading**
  This is the text that shows in the heading. CSEntry may automatically wrap text to make two or more lines, if the column width is small. You can force your own multi-line text by using **Ctrl+Enter** at the end of each line. For example if you type "Age of", then **Ctrl+Enter**, then "Mother", you will have two lines of text no matter how wide the column is.

- **Horizontal alignment**
  This allows you to force the text to be left-aligned, right-aligned, or centered within the column heading area.

- **Vertical alignment**
  This allows you to force the text to be aligned at the top, middle, or bottom of the column heading area.

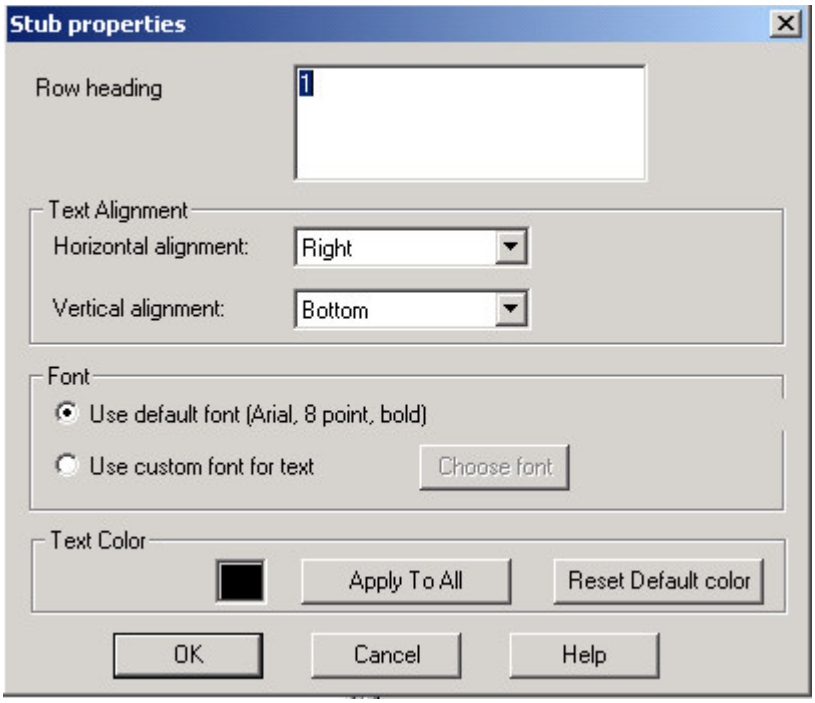
- **Font**
  To change the font of the column heading text, choose the "Use custom font for text" radio button then click on the "Choose font" button.

- **Text Color**
  The button shows the color of the selected text. To change the text color, click on this button, select a new color and click "OK." You can change the text color back to what it was originally (usually black) by clicking on the "Reset Defaut Color" button. You can make all text on all forms the same color by clicking on the "Apply to All" button.

See also: Change Row Heading Properties

Change Row Heading Properties

Right click on a row heading (occurrence label) and choose "Properties."

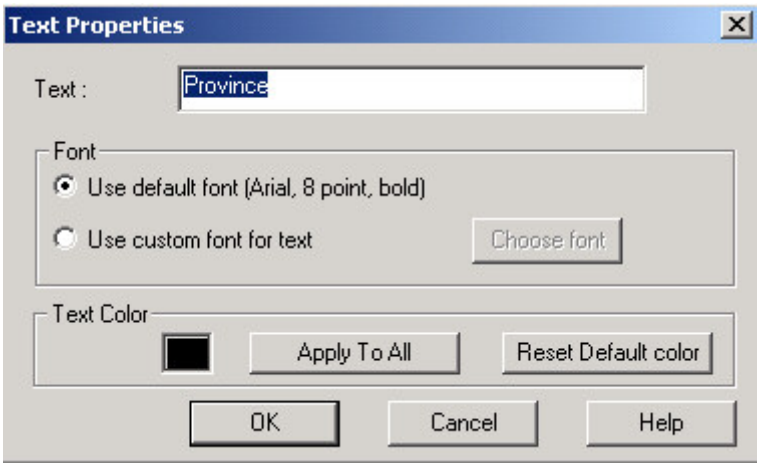


- **Row Heading**
  This is the text that shows next to the row. CSEntry may automatically wrap text to make two or more lines, if the width of the occurrence label area is small. You can force your own multi-line text by using **Ctrl+Enter** at the end of each line. For example if you type "College or", then **Ctrl+Enter**, then "University", you will have two lines of text no matter how wide the occurrence label area is.

See "Change Column Heading Properties" for a description of the rest of the properties.

Change Text Properties

Select any text item to change the text itself. Select any text item or group of text items to change their font. Once selected, right click and choose "Properties." You can change the font for all text fields on all forms by choosing "Option/Global font" from the main menu.



- **Text**
  Enter the new text here. It may contain any characters (including blanks) and be up to 120 characters long.

- **Font**
  This shows what font is in effect for the selected text. To change the font, select the "**Use custom font for text"** radio button then click on the "**Choose font"** button.

- **Text Color**
  The button shows the color of the selected text. To change the text color, click on this button, select a new color and click "**OK**." You can change the text color back to what it was originally (usually black) by clicking on the "**Reset Default Color"** button. You can make all text on all forms the same color by clicking on the "**Apply to All"** button.

Data Entry Editing

Introduction to Data Entry Editing

As we saw in previous chapters, a data entry application contains a set of forms (screens) which a data entry operator uses to key data to a disk file. Data entry applications can be used to add new data to a file and to modify existing data. You can also include logic to perform consistency checks, generate complex skip patterns, look up information in one or more external files, and display messages after any field is entered. A number of statements and functions in the CSPro language set may be used only within data entry applications. They are generally related to the creation and/or modification of a case, and may require operator interaction.

You use the Forms Designer module of CSPro to develop the data entry application. You use CSEntry to run the data entry application, and use the CSPro language to develop validation procedures to be carried out at data entry time.

This section contains the following information:

Editing Concepts
Writing Logic

Editing Concepts

Type of Edits in Data Entry

- **Validate individual data items:**
  These checks are designed to determine whether a response has a value that is inside or outside the valid limits for that response as defined in the Data Dictionary. The data entry application can be designed to allow forcing out-of-range values, to force the operator to reenter a valid value, or not to allow any input on the item.

- **Perform structure or consistency edits**
  You can write code to check the structure of the case or test the consistency within related items. These instructions can be written for any object such as a case, level, form, record, roster, or field. The instructions can be executed before the cursor moves into the object (Onfocus Statement); at the beginning of an object (Preproc Statement); after cursor moves off the object (Killfocus Statement); or at the end of the object (Postproc Statement). You can also perform Interactive Editing after you finish entering a case.

- **Display error messages**
  The system displays automatic error messages if the item is out-of-range, but you can also use the Errmsg Function to write customized messages to be displayed in the screen during data entry.

- **Control the data entry flow**
  You can use skip or advance statements to control the data entry flow in a case or end data entry at any time if a particular condition occurs. See Endlevel, Endgroup statements

- **Control stopping data entry**
  The onstop function can be used to keep the operator from stopping data entry or to allow stopping only under certain conditions. You can also stop the application for the current case or terminate the operation. You can stop the application at any time and the system will save the partial case. See also Ispartial Function

- **Write notes**
  Write notes on a field and saved on a separate file with extension .NOT. The notes can be viewed by the operator and/or edited. Can be used to display instructions to the keyer or elaborate on a particular value. See Putnote, Getnote, and Editnote Functions.

- **Generate reports**
  You can write customize reports to a file. Ex: Create a report to summarize the demographic characteristics in the household; or to produce a reorganized version of some items on the data file.

- **Use secondary forms**
  You can use a secondary form to enter data under certain conditions.

- **Display option menus**
  The system displays a window with the valid values and the operator may select the correct one. See Accept Function.

- **Use external files**
  The most common functions are Loadcase Function and Retrieve Function. The Selcase Function can only be used in data entry applications. See "External File Functions" for a complete list.

- **Examine content of item**
  You can examine the content of a numeric item (Visualvalue Function) or alphanumeric item (Highlighted Function) before the item has been keyed.

Structure Edits

Structure edits can be used to check coverage and to establish that each case has the correct number and type of records. The tests that make up a structure edit ensure that the questionnaires are com plete before beginning the consistency edit. In a typical Housing and Population census, the structure edits will check that:

- For all collective and conventional households within an enumeration area, all required records are present and are in the proper order.

- Each enumeration area (EA) batch has the correct geographic codes.

- Where required, each household has one and only one housing record.

- Each occupied housing unit has at least one person record, and vacant housing units (where permitted) have no person records.

- Within a household (conventional or collective), there must be one and only one person record for each member of the household (i.e., no duplicate or missing records) such that the total number of person records is equal to the recorded count of persons in the household.

- Within each conventional household, there is one and only one qualified head of household; within each collective household, there is no head of household.

Consistency Edits

Consistency edits look at the individual data items within a questionnaire or case, and examine the validity of each item and the consistency of each item with respect to other related items. For example, the degree of educational attainment of an individual should have some predictable relationship with the person's age. This means that it is not expected that a 9-year-old child will have progressed much beyond the fourth or fifth year of elementary school, depending on local standards. If, in the unedited Census data, a 9-year-old indicates educational attainment above that level, it is probable that either the age or the educational level have been incorrectly recorded. Whether the age or the educational level is modified to produce data consistency is a decision left to the subject-matter specialists, who should know whether age data or education data are more correctly reported. It is also important to take into account the method of data capture; some, like scanning, are prone to systemic error (that is, where a "0" is consistently read as a "9," for example). If such error is not recognized and taken into consideration when assessing the reliability of reported information, it can lead to even greater error in the final data.

At a minimum, consistency checks should seek to resolve all errors which might eventually lead to doubts about the quality of the data. That is, the subject-matter specialists must consider the types of tabulations to be produced and the uses to which the data may be put in the future. For example, if the plan includes tabulations of educational achievement by age and sex, the edit specifications must include a means of detecting and correcting the data for individuals whose declared educational achievement is not in line with their declared age (as in the example of the preceding paragraph). If a published table shows even one 9-year-old having completed secondary school, the quality of the data will be called into question, simply because (in that particular culture) it is not the norm that a child so young could be so advanced educationally. It makes no difference if, in fact, the child is a prodigy and did indeed complete secondary school; when it is a case of one or two "outliers" and the weight of probability is against them, the values must be changed and brought in line with reasonable expectations. It is not recommended that such changes be made only in the logic which generates the tables; if the data (even a subset) are to be given out to researchers or other users outside the statistical office, they must be clean and consistent throughout before distribution.

Checking errors

You can check for errors at the time you enter the data or perform interactive editing after the data have already been entered.

You can use the same programmed logic that was in effect during data entry to find problems that were left unresolved by the original keyer, or you can use different logic to check for other conditions.

Improperly identifying errors can waste precious personnel resources, so a precise set of rules should be developed with input from subject-matter specialists.

During data entry the system automatically displays a message if value for item is out-of-range. However, you might want to write your own message for missing or out-of-range values. For example, if enumerators had been instructed to record all persons older than 110 years of age as '110,' the first-pass check for errors might include the following code:

```
PROC AGE
if AGE in 0:110 then
  exit; { the age range is OK, nothing else to do }
else
  errmsg ("Person %d, has incorrect age: %d", $);
endif;
```

So what does this code do? If a person's age is in the range from 0 to 110 ('0' is for infants born less than 12 months before the Census reference date), then the value is accepted as valid and program flow exits the procedure. If not, then the value is either outside this range or missing, in which case the subsequent errmsg statement will be executed, showing the reported age for Person N.

More involved edits may be needed for other variables. For example, fertility information is only asked of a female of a certain age. So if fertility information is present, you may wish to confirm the values of other variables. A possible test could be as follows:

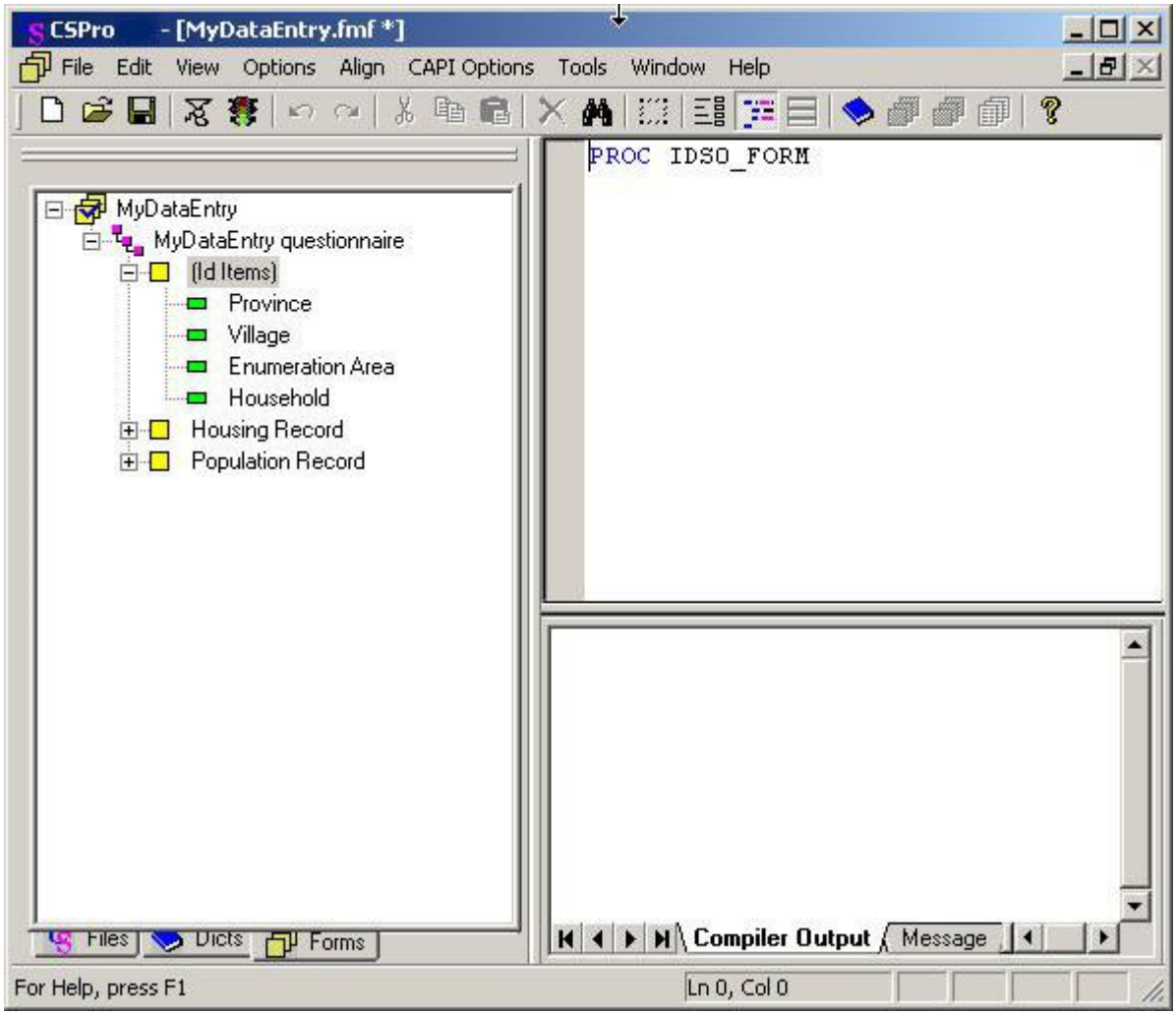
```
PROC FERTILITY
if FERTILITY in 0:20 then { there are 0-20 children }
  if sex = 1 then {if sex = male}
    errmsg ("male has fertility info present"); {message displayed}
    reenter; {operator must enter valid value}
  else {sex is not male}
    if sex = 2 then {if sex = female, check woman's age }
      if age < 15 then { 15 = minimum age for fertility }
        errmsg ("woman is too young (%d) to have children", age);
      endif;
    endif;
  endif;
else { Fertility is blank }
  if sex = 2 then { a problem if the woman is "of age" }
    if age >= 15 then
      errmsg ("woman aged %d should have fertility info", age);
    endif;
  endif;
endif;
```

As you see, the complexity of the logic used to find (and soon, correct!) errors will depend on the specifications provided. In the case where specifications are minimal, it is important that the programmer consider all situations/paths in developing the logic.

Writing Logic

Data Entry Logic Screen Layout

Logic for data entry screens is written when developing the application. Forms Designer permits viewing either the forms or the logic associated with the application on the right-hand side of the split screen. To initiate the process of creating logic for an application, click on ; or from the View menu select "Logic"; or press **Ctrl+L.** To go back to the forms, click on ; or from the View menu select "Form"; or press **Ctrl+M**.



The screen is divided into three main work areas: the Tree View, the Logic View and the Message View.

- **Tree View**
  The window on the left half of the screen displays the data entry tree with the first form ( ) selected.

- **Logic View**
  This is the window block in the upper portion of the right half of the screen. It is the programmer's "clean slate, to which may be added logic for any part of the data file: any item, any section, any record, even the file as a whole. It is up to the programmer to determine the correct placement and sequence of execution for each logical element. The initial screen (`PROC IDSO_FORM`) represents the Identification form or initial form. If you move to the top of the tree ( ), then the logic view will display:

```
{Application 'MYDATAENTRY' logic file generated by CSPro }
PROC MYDICT_FF

PROC GLOBAL
```

  This is the beginning of your program. This two lines of code will always be in your application file. You write the declaration statements under `PROC GLOBAL`, then the procedures for the event.

If code has been written for a given edit level, form, roster, or field, a check mark will appear superimposed on the icon for that entity. This is how, at a quick glance, you can see where you have placed programming logic. Once one line of code has been written anywhere in the program, a check mark will appear on the Forms File icon.

- **Message View**
  This is the window block in the lower portion of the right half of the screen. It is devoted to messages (user-created and system-generated). As with the Tree View, tabs are available to the programmer; clicking on one of them will make the contents of that view active. The **Compiler Output** tab displays errors found during compilation of your program; if the code compiled successfully, it will state "Compile Successful." The **Message** tab is used to type in error messages that will be used in the execution of the program.

If you wish to modify the size of any of these three work areas, just place the mouse over one of the separating bars, grab it, and drag to resize.

Moving Around a Logic Application

To move around your edit application, select individual items from the **Forms** (in a Data Entry application) or **Edit** (in a Batch application) tree tab. The Logic View will update to display the programming logic (if any) for that item. If you select the root of the tree, all logic written for the entire edit application is displayed.

For example, suppose you select "Age" from the tree and there is no associated programming logic; you will see:

```
PROC AGE
```

in the Logic View. Since there is no logic, "`PROC` Age" is generated "on-the-fly" and will not be saved in the .apc file. If there was associated logic, you might see something like this:

```
PROC AGE
  POSTPROC
  if not (AGE in 0:99) then
    errmsg ("Invalid age found");
  endif;
```

Note the code above, by default, would have been placed in the **postproc** section, so it is not necessary to explicitly state "postproc."

Order of Executing Data Entry Events

An "event" is a unit of logic associated with an element of the edit tree, such as a form, a field, etc. The order in which events are executed during data entry is dependent on two factors: (1) the inherent order imposed by the CSPro software design, and (2) the order in which data items are entered, which is determined by the designer of the application.

CSPro is based on the concept of a "case" containing one or more types of "collections of information". The "case" will usually correspond to the questionnaire used in the survey or census, and the "collections of information" [or groups of data items] will usually correspond to one or more record types that make up the case/questionnaire. These elements constitute a hierarchy, and in applying logic, CSPro follows this hierarchy—that is, it begins with any logic that pertains to the file itself, and works "down the tree" through

the various levels. CSPro executes application events one case at a time. During data entry, preproc, onfocus, killfocus and postproc statements are executed in the order in which they are encountered.

By default, if not otherwise specified, logic will always be considered to be in the **postproc** portion of the edit entity. The other three don't need to be present.

The following diagram illustrates the default order of events for a two-level data entry application that has no skip or advance statements that might otherwise alter the program's natural flow. Level 1 has two forms (1 and 2) and level 2 has one form (3). This description applies to both system- and operator-controlled applications.

```
   Form File preproc
 Level 1 preproc
  Form 1 preproc
  Form 1 onfocus
   Field 11 preproc
   Field 11 onfocus
  <entry of Field 11>
   Field 11 killfocus
   Field 11 postproc
     :
   Field 14 preproc
   Field 14 onfocus
    <entry of Field 14>
   Field 14 killfocus
   Field 14 postproc
  Form 1 killfocus
  Form 1 postproc
  Form 2 preproc
  Form 2 onfocus
   Field 21 preproc
   Field 21 onfocus
    <entry of Field 21>
   Field 21 killfocus
   Field 21 postproc
    :
   Field 26 preproc
   Field 26 onfocus
    <entry of Field 26>
   Field 26 killfocus
   Field 26 postproc
  Form 2 killfocus
  Form 2 postproc
  Level 2 preproc
   Form 3 preproc
   Form 3 onfocus
    Field 31 preproc
    Field 31 onfocus
     <entry of Field 31>
    Field 31 killfocus
    Field 31 postproc
     :
    Field 35 preproc
```

Field 35 `onfocus`
<entry of Field 35>
Field 35 `killfocus`
Field 35 `postproc`
Form 3 `killfocus`
Form 3 `postproc`
Level 2 `postproc`
Level 1 `postproc`
Form File `postproc`

Please note that the natural flow through the fields can also be altered by the use of the cursor or mouse. For example:

- **Scenario 1**
  Given: three fields [A, B, and C]. From Field A, click on Field C. Assume all fields have **preproc**, **onfocus**, **killfocus**, and **postproc** events.

  Result: Field A's **killfocus** would execute, but its **postproc** would not. Nothing would execute for Field B. Finally, Field C's **preproc**, global **onfocus**, and local **onfocus** would all execute.

- **Scenario 2**
  Given: Form 1, which contains one field, Field 1. Form 2, which also contains one field, Field 2. After keying Field 1, you are automatically advanced to Form 2, Field 2. You then decide to use the up/left arrow to move back to Form 1. Assume both forms and both fields have **preproc**, **onfocus**, **killfocus**, and **postproc** events.

  Result: Field 2's **killfocus** would execute, but its **postproc** would not. Next, Form 1's **onfocus** would execute. Finally, Field 1's global **onfocus** and local **onfocus** would execute—but its **preproc** would not. Please note that it does not matter how many fields are on Form 1; the **onfocus** for Form 1 would always execute.

Essentially, if the programmer uses logic, or if the data entry operator moves backwards or forwards with the mouse or arrow keys, the natural flow of the program will be altered. If exiting a form, field, or roster prematurely, the **killfocus** event will execute but the **postproc** event will not. Similarly, if entering a form, field, or roster by backing up into it, the **onfocus** event will execute but the **preproc** event for it will not.

Sequence dictated by designer

In the course of creating the application, the designer may choose to modify cursor movement under certain conditions. In the example of a population census, if data are being captured for a male, or for a female under the age of 15, the application designer may wish to skip over the items relating to fertility because they are not applicable to these population subgroups. If the skip occurs (whether programmed or through operator control), the edit sequence will be modified accordingly. In addition, if the operator moves backward and forward through the forms, the edit sequence may also be affected, depending on the type of control (operator or system).

Compile an Application

When CSPro compiles your application, it checks the logic you have written to see if there are any errors or warnings. Errors typically include spelling a command incorrectly, not using proper command syntax, and putting logic in the wrong place. Error messages appear in the panel at the bottom of the right-hand screen and a red dot appears to the left of the line that contains the error. Warnings usually involve using

commands in questionable ways. Warning messages also appear in the panel at the bottom of the right-hand screen and a yellow dot appears to the left of the line that contains the warning.

You can choose to compile code for a specific item, or for the entire application. To compile code for a specific item, simply select that item from the edit tree. The associated logic for that item will be displayed in the Logic View. Press on the toolbar; or from the **File** menu, select **Compile**; or press **Ctrl+K**. The results of your compile will be displayed in the **Compiler Output** area at the bottom of the right-hand screen. When you are ready to compile the entire application, select the root node from the data entry or batch edit tree. This will display all logic written for the application in the **Logic View**. You can then press the "Compile" button or press **Ctrl+K**.

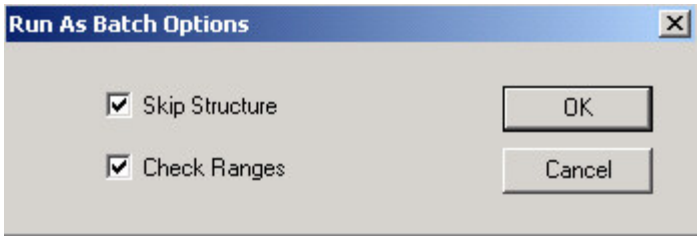
CSPro always compiles your application when you run the application. If there are errors, you cannot proceed until the errors are corrected. During code development, you should compile only the logic for an individual data entry or batch edit entity. This saves time, because the system does not have to recompile the entire logic module. Furthermore, your entire file may not be ready for compilation (i.e., there are unfinished parts awaiting someone's input), so it will be more efficient to compile only untested code.

If your logic compiles with no problem the system will display "Compile Successful" in the Message View. Now you are ready to run the data entry or batch edit application.

See also: Run as Batch

Run as Batch

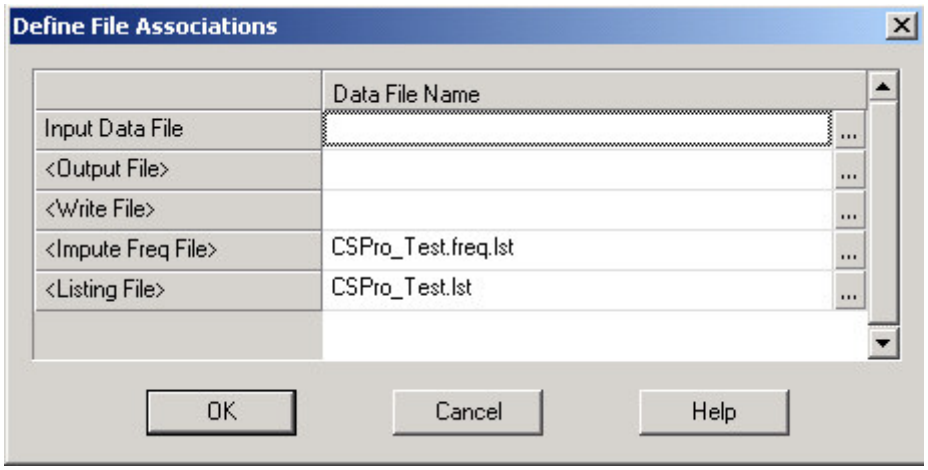
You might also run the data entry application after you finish entering data for a file. If you select "**Run as Batch**" under the File menu, the system will display:



- **Skip Structure**
  If this option is checked, the application will check for entries in items that should be blank when they are skipped under certain conditions, and will list those items.

- **Check Ranges**
  If this option is checked, the application will list the items with out-of-range values.

After you press "**OK**" the system will display the following dialog box:



- **Input Data File**
  This line is required, and asks for the name of the data file against which you wish to run your batch application. This data file will not be modified in any way; it will only be opened, read, and closed.

- **Output File**
  The output file is where the results of correcting your data will be written. If you are **not** making any corrections in your program, then the generated file will be an exact copy of the original data file. If you **are** making corrections to your data file, then this will be the corrected data file. The default file extension is .out, but you can use whatever you'd like. This field is optional; therefore, if you are making corrections to your data, but forget to specify an output file, no corrected file will be generated.

- **Write File**
  If you have any write functions in your program, they will write information to this file. The default file extension is .wrt, but you can use whatever you'd like. This field is optional; therefore, if your program contains write statements, but you forget to specify a write file, no file will be generated. Similarly, if you indicate a write file but your program does not contain any write statements, no file will be generated.

- **Impute Freq File**
  If your program contains any impute statements, the results of this command will be written to this file. The default file extension is .frq, but you can use whatever you'd like. This field is optional; therefore, if your program contains **impute** commands, but you forget to specify a frequency file, no file will be generated. Similarly, if you indicate a frequency file but your program does not contain any impute commands, no file will be generated.

- **Listing File**
  This line is required, and asks for the name of the file to which you want to write the results of the run. Results from errmsg functions will be written here. This file will always be generated, regardless of whether or not your program includes errmsg commands.

CAPI Data Entry

Introduction to CAPI

CSPro supports Computer-Assisted Personal Interviewing (CAPI) data entry. In CAPI the interviewer uses a laptop, tablet, or mobile device to enter responses as they occur. The interview is conducted face-to-face, and the entry application (sometimes called the "instrument") determines the question order, and also performs editing of responses.

CAPI offers a flexible approach to collecting data, and can result in improved data quality and more efficient interviewing and processing. Creating good instruments, however, requires training and experience on the same level as creating good batch edit programs.

CATI (computer assisted telephone interviewing) is similar to CAPI, but interviews are conducted over the phone instead of in person.

This section contains the following information:

CAPI Features
CAPI Strategies
Extended Controls
How to …

CAPI Features

A CAPI application is similar to a regular data entry application, but involves making use of some additional features. CSPro offers the following features suited to implementing CAPI surveys:

Question text	Customized text for each question can be displayed at the top of the entry form.
Help text	A help hot-key (F2) can bring up customized help information for each field.
Multiple languages	The CAPI entry application can be developed with question and help text in several languages; the interviewer can switch among languages as needed.
Response boxes	During entry, a pick-list of valid values for each question can be optionally displayed; these can make the interview go more smoothly.
Notes	Interviewers can enter field-level notes when needed.
Support for partial save	CAPI applications can be developed so that they support partial save. This lets the interviewer save an incomplete case, then return later to complete it.
Customized branching	The entry application can determine on the fly which forms or questions to present based on the user’s previous responses. This is accomplished through program logic
Immediate editing	Responses can be edited, and errors or possible mistakes signaled to the interviewer. This can help improve data quality.

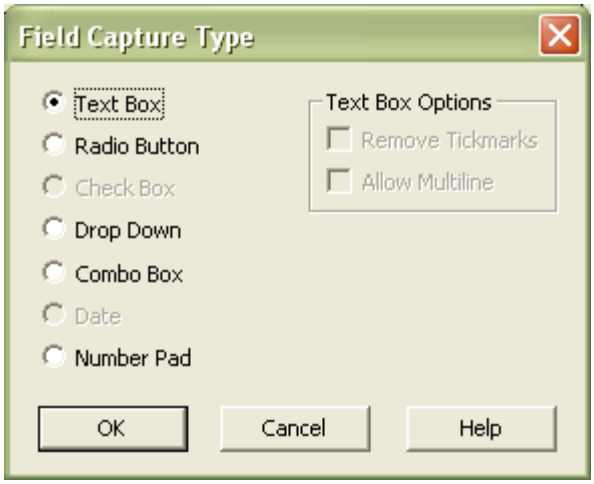
Extended Controls

Occasionally it may be useful to display to an enumerator or keyer the possible values for a given field in a data entry application. For example, if a survey is being conducting on a tablet computer, it may be easier for an enumerator to use a finger or stylus to select a value rather than to key the value using the screen's touch keyboard. CSPro's extended controls allow for such flexibility by allowing the program to specify the capture type associated with a field.

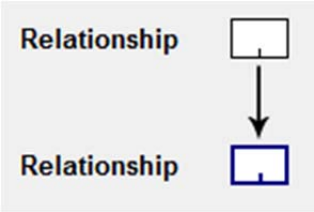
Selecting Extended Controls

You can indicate that you want to automatically use extended controls by selecting the Use Extended Controls option in Drag Options. If this option is selected, CSPro will automatically choose a capture type for the field based on the field's primary value set.

To manually change the capture type for a field, view the field's properties and select Capture Type -> Change.



CSPro only allows you to select a capture type that is suitable for the item's first value set. For more programmatic control, see the setcapturetype function. The CSPro Designer shows the fields that use extended controls by coloring the border of the field in blue. The blue border does not exist when the data entry application is run in CSEntry. The field border will be a lighter blue color when the Number Pad is selected.

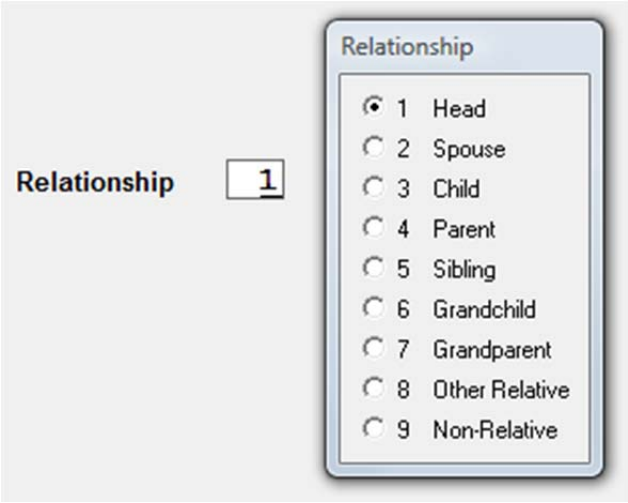


Capture Types

Though there are seven listed capture types, there are essentially only five extended controls. The Text Box attribute is the default option for a field and indicates that no extended control will be displayed. The Text Box Options, enabled only for alpha fields, allow the user to customize the size and appearance of the Text Boxes.

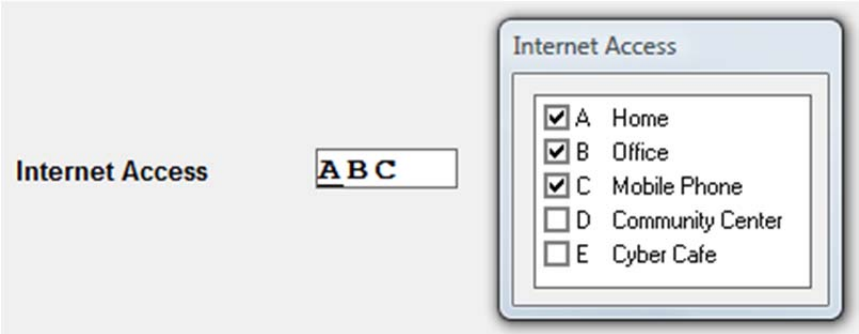
Radio Button

The radio button capture type can be selected if an item is either numeric or alpha and if the primary value set has only discrete values, meaning that there are no "to values" defined.



Check Box

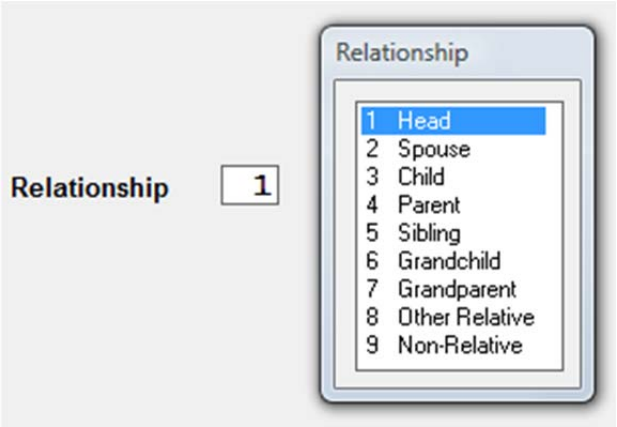
Check boxes are used to capture multiple response variables. Items using check boxes must be alpha, and the number of values in the primary value set must match the length of the item. Each value in the value set corresponds to one of the responses and will be displayed as a separate check box. When the boxes are checked, the corresponding values will be put into the alpha field, from left to right.



Drop Down / Combo Box

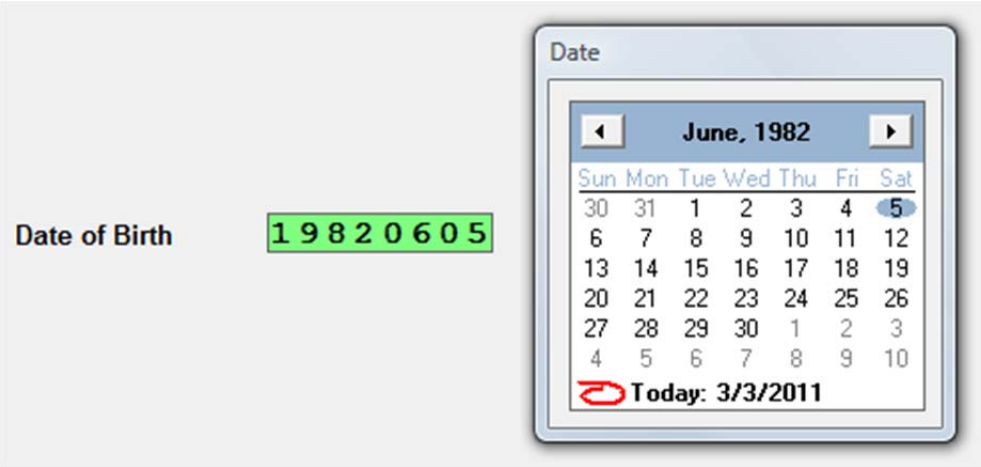
This capture type displays all values in the primary value set of a numeric or alpha item. Unlike the radio button capture type, the combo box capture type allows for the displaying of ranges of values (that is, value sets with "to values"). If the enumerator clicks on a value with a range, the first value in the range is placed in the field.

While the drop down and combo box are visually identical on CSEntry for Windows, in the Android environment there is a difference between the two options. Drop down fields, which must contain only discrete values, will be displayed as radio buttons. Combo box fields, which can include ranges, will appear as text boxes with a corresponding button that displays the entire value set, allowing users to see the ranges as well as to select from discrete values (such as a Missing code).



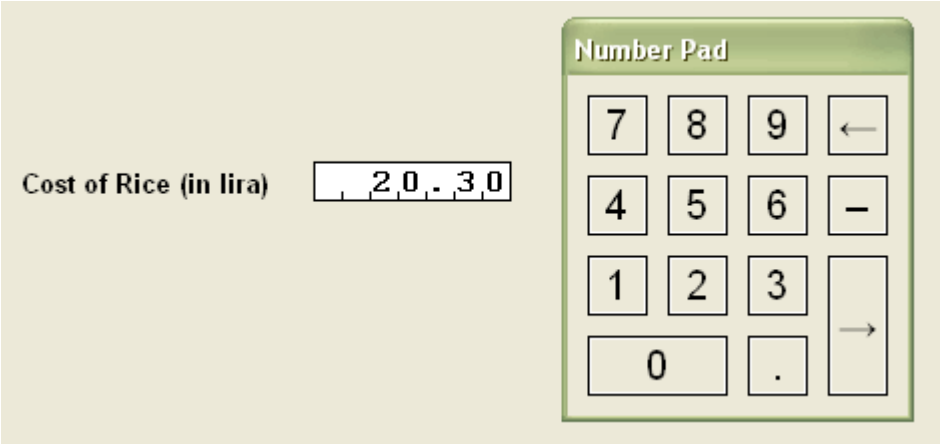
Date

The date capture type is valid for alpha and numeric items of length 4, 6, or 8. The item does not need to have a value set. The choice of date format affects how the date is stored in the field.



Number Pad

Some users find it cumbersome to use the on-screen keyboard while entering data on a tablet. This control displays a number pad for entering numeric values with a finger or mouse.



Hiding the Title of an Extended Control Window

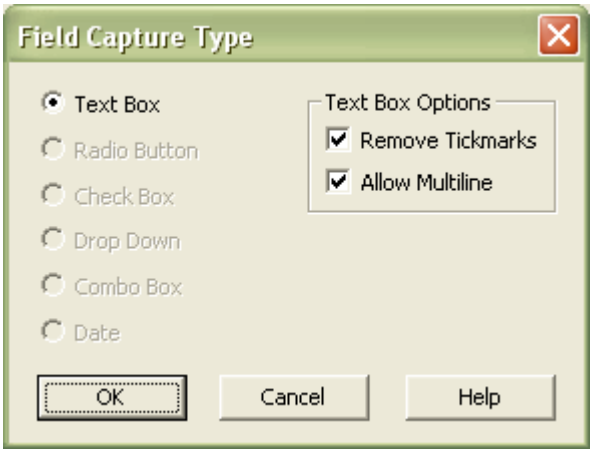
By default CSEntry displays the label of the value set in the window of the extended control, but this title can be turned off using programmatic logic.

```
set attributes(dictionary-symbol) assisted off(variable(title));
```

See also: Getcapturetype Function, Setcapturetype Function, Setcapturepos Function, Text Box Options

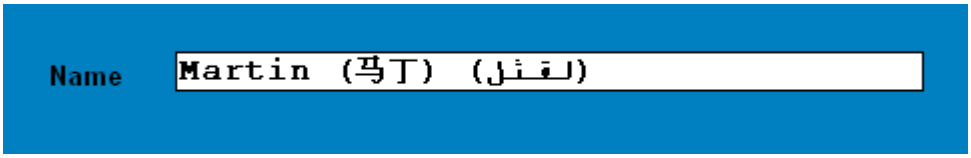
Text Box Options

To facilitate the handling of non-Latin characters, CSPro offers a couple options for alpha fields. To change these options, first modify a field's capture type. There are two options: **Remove Tickmarks** and **Allow Multiline**.

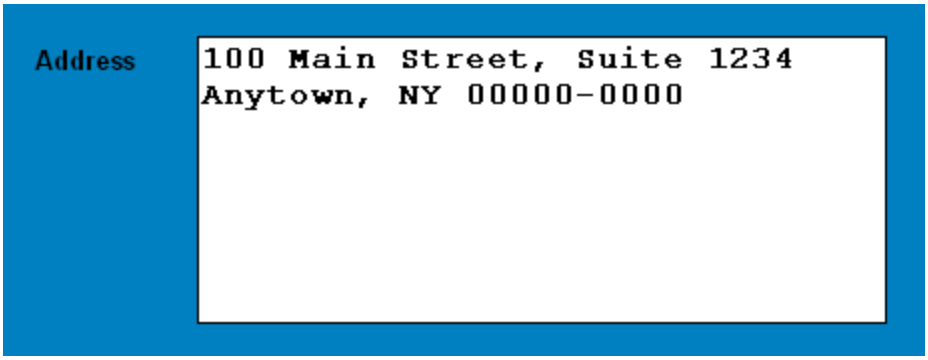


By default, with no options selected, CSEntry displays alpha fields in the same way that numeric fields are displayed, with tickmarks dividing each characters:

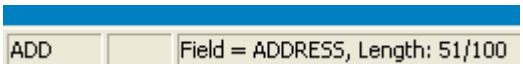
After selecting **Remove Tickmarks**, CSEntry will not display any tickmarks and characters will be displayed with any connecting features. This is useful for some scripts, like Arabic, which have elements that connect, or other languages that require multiple characters to create a composite character. In the CSPro Designer you can click on the field and resize it horizontally to set an appropriate size..



To select the **Allow Multiline** option, the **Remove Tickmarks** option must first be selected. A multiline control provides a convenient way to collect multiple lines of information in one alpha field. This may be useful for information like an address or a memo field. In the CSPro Designer you can click on the field and resize it horizontally and vertically to set an appropriate size. In CSEntry, the data entry operator can move up and down in the field using the arrow keys, but as with other fields, pressing Enter will move to the next field. To add a line break, the operator must hold down the Control key before pressing Enter.



Due to the lack of tickmarks, a data entry operator will not immediately know many characters they have entered into the field and how many characters are remaining. CSEntry shows, on the status bar at the bottom of the screen, the number of characters entered, and the total size of the field.



For multiline controls, the characters **\n** will be placed in the field text for every line break. CSEntry will automatically translate **\n** characters to line breaks, but if you use your data with another software package, you may need to adjust for this behavior. In the above screenshot, the text would appear in the data file as:

```
100 Main Street, Suite 1234\nAnytown, NY 00000-0000
```

Both of these options can also be specified for alpha fields in rosters. To resize the field within a roster you may need to first adjust the roster's row heights or column widths.

CAPI Strategies

Forms

Form layout should be easy for the interviewer to see and understand. For this reason, it's usually best to place fewer items on each form that you might with a non-CAPI application. Also, remember that screen real estate is limited in a CAPI application, since question text will occupy a large portion (sometimes up to half) of the top of the entry window.

Consider breaking your application into a series of sections, each of which deals with a topic (earnings, fertility, etc.). One or more forms are then used for each section's questions.

Avoid scrolling in CAPI applications, since this prevents the interviewer from seeing the entire form (or roster). Also, the interview environment – laptop, maybe no mouse, possibly poor lighting – often makes scrolling cumbersome.

Also, it is good practice to gather the names of all household members at the start of the interview. Then the interview can cycle through each member. Doing this minimizes the risk of accidentally omitting household members. Take a look at the sample CAPI application for more information.

Try to limit the size of your forms so that the interviewer won't have to scroll them. When developing your application, be sure to test your application using the screen resolution that will be on the laptops used in the field.

Fields

In CAPI data entry, consideration needs to be given to the properties of fields used in the application. The following lists field certain field properties and how they might be used:

Use enter key	This is usually selected, because otherwise the application will automatically advance to the next field, which might be confusing for the interviewer
Mirror fields	These are very useful in CAPI applications, as they show the contents of fields in other forms. This can assist the interviewer.
Protected fields	CAPI applications use protected fields to show calculated or derived values. These provide useful feedback to the interviewer.
Force out-of-range	This is almost always tuned off, since the interviewer should only be allowed to enter valid responses. However, you should consider including "not applicable", "don't know", and "refused" response categories.

CAPI applications frequently make greater use of alphanumeric responses than to traditional key-entry systems. This is because of the interactive nature of CAPI, and it also makes the interview more interactive (by capturing names, for example).

Questions

CSPro shows customized question text in the top part of the entry window. To help the interviewer distinguish between text for the interviewer and the respondent, consider using different color fonts. For example, things the interviewer should say to the respondent (for example "How many children do you have?") might be shown in black. Instructions to interviewer (for example "Ask of each household member") might be shown in blue. CSPro includes two user defined font, CAPI font #1 and CAPI font #2, which can facilitate this.

Question text can be customized using fills, or text substitutions. These fills reference the contents of variables or dictionary items that can be embedded into the question text. They are identified by name of the variable or item surrounding by percent characters (%). During the interview, the value of the variable or data item is substituted into the question text. For example, "How old was %FIRST_NAME% on January 1?" might be transformed into "How old was Edward on January 1?" Fills help customize the question text and frame it for specific respondents.

Your CAPI application can also present different sets of question text based on conditions. For example, the text for certain questions might differ based on the number of persons in the household. Conditions can also let your application support multiple kinds of questions, depending on criteria you decide.

Values

A CAPI application can optionally show a pop-up box containing an item's valid values. This makes it easier for the interviewer to read and select response categories. The pop-up box appears to the side of the item, so a response could also be typed in. The list of responses that appears comes from the item's first value set.

If you decide to use this feature, be sure to create descriptive value sets for your questions, since these will be displayed as during the interview.

Organization of the Instrument

CAPI entry applications are almost always created in system controlled mode. because they take advantage of CSPro's ability to use logic to determine the question order.

To get started, it is a good idea to define sections containing sets of questions related to one topic. The number of sections will vary from survey to survey. You might need a section to begin the interview and a section to end the interview. You also might need some help sections with FAQ or roster of persons in household.

Next you need to develop an order of the sections, that is the order of the interview. Some sections like FAQ or person roster may be independent of the interview sequence, and may best be placed after the last subject matter section or after the end of the interview section.

Now with a section you need to define the order of questions. Place one item or several items which are related to one another on separate forms. The forms will be in the sequence the questions are asked.

Using Multiple Languages

CAPI respondents often speak different languages. In fact, it is not uncommon to find several languages spoken among the households included in a survey. It is advantageous for your entry application to accommodate the different languages that respondents might speak, since doing so will improve data quality and response rates.

CSPro supports multiple languages, and lets you add languages and define question text for each. By default, just one language (English) is available.

A CAPI application can contain question text in more than one language. You can define additional languages (the default is just English), then enter question text for each as needed.

During the interview, the interviewer can switch among the application's languages on-the-fly. Each question's text will be displayed at the top of the screen, in the chosen language. So, if an interviewer arrives at a household and finds that the respondents prefer to conduct the interview in another language, this can be easily done.

When you develop a multi-language application, it is probably easiest to finalize and test all the question texts in one language. Then, once this is done, language specialists can translate them into the other languages your application will support. This is easy to do, since the question text editor can display question text for two languages at the same time. The translator can copy and paste text or bitmaps, if needed. Finalizing all the question text in one language first also helps avoid version control problems that might arise if things were translated then later modified.

Note that the optional pop-up box which shows each item's response categories will only show the values from the item's first value set. It is not possible to switch among multiple sets of response categories.

Breaking Off the Interview

Sometimes an interview cannot be completed in one session. CSPro gives the developer control over how an interview can be terminated. You might want to include a form that asks when the respondent would prefer to do the follow-up visit.

CSPro fully supports partial save, which causes a case to be saved to disk even if it is not yet complete. The case can be continued at a later time. Partial save is a very useful feature in CAPI applications, and should be used if any of the interviews might not be completed in a single sitting. Very small applications, like listing operations or a short-form census, probably would not require partial save ability. To enable partial save, select the box in the data entry options dialog.

The user function OnStop can be used to trap the stop button ( ) or attempts to exit the interview (for example, by pressing Alt+F4). The developer might want to include a form that manages the exit process and saves the partial case to disk.

You could also use OnStop() to intervene and prevent an interview from stopping, or to jump to a callback form before exiting. Your application also could store the field being entered, so that it could jump to that section when the interview is later continued.

If it is necessary to return later to finish a CAPI interview, you might want to include a form that asks when the respondent would prefer to do the follow-up visit. In a production CAPI system, a separate case management system could take advantage of this information and assist the interviewer in tracking and scheduling appointments.

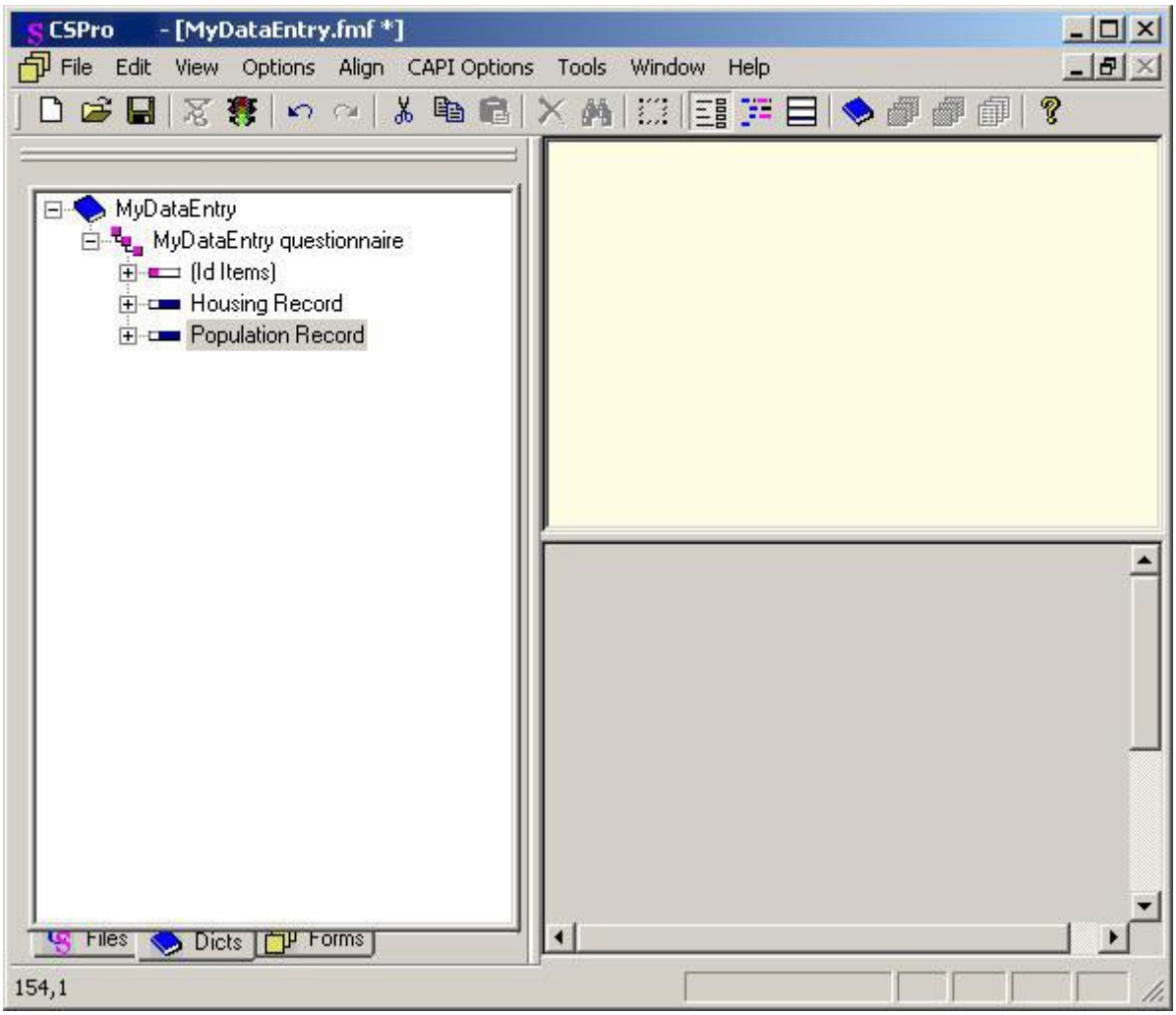
Coming Back Later

You need to think about what behavior will be needed if an interview is continued at a later time. That is, should they start again from the beginning of the instrument, the place where the last interview left off, or something else. Sometimes instruments will always go through a front section to confirm household member names and other vital information and then jump to the section where the earlier interview ended.

How to ...

Create a New CAPI Application

1. Create a New Data Entry Application
2. Don't generate forms
3. From the options menu, select Data Entry Options.
4. Check the box "CAPI Mode"
5. Check the box "Partial Save", if you want to use partial save.



Notes:

- After enabling CAPI mode, the forms screen will be divided horizontally. The top part is for question text (to be read during the interview), the bottom is for the normal form contents.

Define Languages

A CAPI application can contain question text in multiple languages. The interviewer can switch among available languages as needed.

1. From the CAPI Options menu, select Change CAPI Languages. The CAPI Languages dialog box will be displayed, showing the languages currently defined for the application.
2. You can add, remove, or modify the your application's CAPI languages

Notes:

- Language names follow the same rules as names; that is, they must be unique and cannot contain spaces. Try using abbreviations like ENG (English), SPA (Spanish), FRA (French), or POR (Portuguese).
- Language labels can contain any text to describe the language.
- During data entry, the interviewer can easily switch among languages

See also: Create customized helps

Organize Forms

Divide your CAPI questionnaire into sections, one for each topic area. Add any special sections, such a begin and end section or global help sections, like an FAQ. These sections organizing units for defining forms and movement within the application.

Break up the questions within a section into forms containing one or several responses. Remember, you only have about ½ of the screen at the bottom to use for a form, because the question text may take up the top half of the screen.

If the form contains a roster, you should make sure that only the roster scrolls, not the form. Scrolling forms are confusing to interviewers.

Enter Question Text

CAPI applications include a question text window (at the top of the form). You can customize the text that appears in this window for each question in the application.

1. From the View menu, press CAPI Questions (or press Ctrl+Q). The question editor will be displayed.
2. Select an item from the tree on the left. Any text already defined for this item will be displayed in the editing window on the right.
3. Enter text into the editing window. You can format the text using the toolbar, or use the formatting choices shown in the CAPI Options menu.
4. When you have finished entering the question's text, you can select a different item or switch to the form view or logic view.

Notes:

- The question text editor supports copy and paste operations, including pasting bitmap images.

Create Fills In Questions

Fills are used to customize question text based on respondents' specific characteristics. This is done by embedding variables or dictionary items into the text and surrounding them with percent characters ("%").

Enter some question text that needs to be customized. For words or phrases that need to be filled, enter a dictionary item or variable with % characters before and after. For example:

Can I speak with %FNAME% now?

where FNAME is a dictionary item that contains a person's first name.

When the entry system comes to this text, it will insert the person's FNAME value into the question text:

Can I speak with Marjorie now?
Can I speak with Allyson now?

You can also customize the question text by inserting an item or group's occurrence labels into the text. For example:

> How much did you spend on %getocclabel% in the last month?

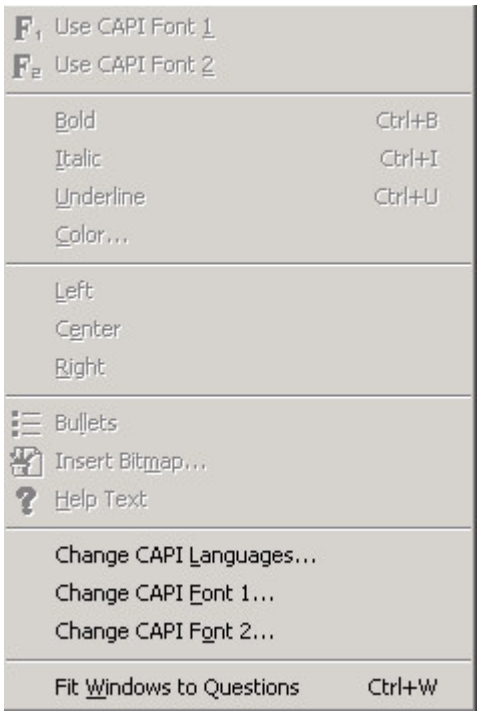
Create Standard Forms

Frequently, CAPI applications format question text using one or two specific fonts. CSPro has two dedicated font selections (CAPI font 1 and CAPI font 2) that can be defined and used for this purpose.

1. Select Change CAPI Font 1 or Change CAPI Font 2 from the CAPI Options menu.
2. A font dialog will appear, where you can change the font type and size. You can also change the font formatting (underline, boldface, italic) and color.

Change Formatting

Question text formatting options include:



use CAPI font 1	use the font specifications from CAPI font 1
use CAPI font 2	use the font specifications from CAPI font 2
bold	make text **bold**
italic	*italicize text*
underline	<u>underline text</u>
color	change text color
left	Align text left
center	Align text center
right	Align text right

bullets	convert text into a bulleted list
insert bitmap	insert a bitmap file (.BMP extension) into the question text
change CAPI font 1	change the font characteristics for CAPI font 1
change CAPI font 2	change the font characteristics for CAPI font 2

Add Pictures

Pictures and other graphics can be added to your question text. The picture or graphic must in a bitmap (*.bmp) file. There are two ways to do this:

- Copy and paste them directly into the question window, or
- Select Insert Bitmap from the CAPI Options menu.

Pictures are limited to 512K bytes in size. You can determine a picture's size by right-clicking on it in the Windows Explorer and selecting Properties. The size, in bytes, will be displayed in the properties window. Bitmaps with 24 bit format take up the most space. Bitmaps with only 16 colors take up the least space.

Notes:
You can also add bitmaps by right-clicking inside the question window, and choosing the Insert Bitmap option.

Use Multiple Language

If your application supports multiple languages, then two CAPI question text windows will be shown. Use the drop down list in the toolbar above each question text window to select a language. Each question can have text for each language, although this is not required.

Tips:

- If your CAPI application supports multiple languages, consider starting by entering text for all the questions in the first language. Then use can translate each text into the other languages by putting the first language in the top question window, and translating in the bottom question window. You can also copy and paste between question text windows.

Create Conditional Questions

The bottom window of the question editor is used to add conditions.

1. Select an item for which you want to add a condition (from the left-hand tree).
2. If the item's record type is multiply occurring, then columns for "Min Occ" and "Max Occ" (minimum occurrences and maximum occurrences) will be shown in the condition editor window. A "condition" column is also shown.
3. Right-click on the blue highlight bar in the condition window
4. Choose Add Condition
5. Enter a condition.
6. Enter text for this question in the window above.
7. Add more conditions, and then question text for each.

When the entry application comes to this question, it will evaluate the first condition. If it is true, then the question text for that condition will be shown to the user. If not, the next condition will be checked, and so on. If no conditions are satisfied (i.e., they are all false), then no question text will be displayed. It is not necessary to add an actual condition; a blank condition will always evaluate true.

The "Min Occ" and "Max Occ" let you easily base conditions on the number of occurrences of the record type. For example, you might have an item AGE in a person record, with conditions:

Min Occ	Max Occ	Condition	Effect
1	1		this question text will be shown if there is 1 person record in the case
2	5		this question text will be shown if there are 2-5 person records
6	15		this question text will be shown if there are 6-15 person records
			this question text will be shown if the three previous conditions are all false (that is, if there are 16+ person records)

Note: Conditions can include dictionary items or variables, and can even call functions. In fact, they can be any expression that could be evaluated in a logical "if…then" statement in the application's program logic.

Display Questions Without Scrolling

CSPro can resize all the question text windows in your CAPI application so that they will not scroll. The horizontal separator bar will be positioned appropriately during entry.

1. Select Resize Question Windows from the CAPI Options menu.
2. Note that all of the application's question texts are now sized so that they can be seen without scrolling.

Structure Movement

Decide what movements within the case, an interviewer will be allowed to make. For example, will the interviewer only be allowed to move backwards or forward through the interview questions or will they be allowed to jump around. If they can jump around, what jumping movements will they be allowed to make. Are there sections that must be completed first? Are there optional sections that don't have to be filled in? Are there sections that can be completed in any order?

Decide what happened when the interviewer tries to stop the interview before it is completed? Will this be allowed? If it is, how will the interview be ended? Through an end section, which may collect when the respondent will be free to continue the interview and where interview notes may be inspected? Or by just remembering the current question being asked?

If an interview can be continued at a later time, how will the interview be restarted? By just advancing to the last question completed? Or by having the interviewer ask a few questions to reorient the interviewer and respondent.

Create Helps for Fields

In addition to question text, a CAPI application can have help information for each question. During entry, the F2 key is used to display a question's help text.

1. From the View menu, press CAPI Questions (or press Ctrl+Q). The question editor will be displayed.
2. Select an item from the tree on the left. Any text already defined for this item will be displayed in the editing window on the right.
3. Switch from editing questions to editing helps by pressing the button, or by choosing Help Text from the CAPI Options menu.
4. Enter help text into the editing window. You can format the text using the toolbar, or use the formatting choices shown in the CAPI Options menu.

When you have finished entering the question's text, you can select a different item or switch to the form view or logic view.

Show Values for Selection

You can display a list of response values for a field, so that the interviewer can select a response. The list of response values comes from the first value set of the field in the data dictionary.

To turn on the display of values for selection during the interview, use the following statement.

```
set attributes(field-1[, field-2, ...]) assisted on;
```

The interviewer can turn the responses on or off during the interview by pressing Ctrl+C or going to the options menu and selecting Show Responses (this field).

Show Images and Values for Selection

You can display a list of response values for a field with an image associated with each value, so that the interviewer can select a response based on the picture. This can be helpful in CAPI situations to select units of measure or other things that may be best described visually. The list of response values and images comes from the first value set of the field in the data dictionary.

To turn on the display of values and images for selection during the interview, use the following statement.

```
set attributes(field-1[, field-2, ...]) assisted on;
```

The interviewer can turn the responses on or off during the interview by pressing Ctrl+C or going to the options menu and selecting Show Responses (this field).

To use images in value sets, you must first use a text editor (such as Notepad) to insert the following line in the Data Dictionary file (.DCF), in the first section marked `[Dictionary]`:

```
ValueSetImages=Yes
```

To associate an image with a particular value in a value set, you must create a bitmap file (.BMP) for that value. Then, using the Data Dictionary interface for value sets, which shows the values in the right-hand pane, you click on the gray box in the column to the far right. A dialog box will then come up from which you select the .BMP file. The name of the .BMP file will then appear in the second column from the right.

Show Full Screen Value Sets

For some data capture devices, such as an Ultra-Mobile PC (UMPC) or other small tablet-like computer, due to limited screen size, you may want to force CSPro to show only one question at a time, with more screen space dedicated to the response values. You can do this with any CSPro data entry application, no matter how the forms are arranged and no matter how many fields are on any given form.

To do this, you must use the FullScreenValueSet command line parameter. For example,

```
csentry test.pff /FullScreenValueSet
```

When you use this parameter in a CAPI application, there will be three panes (windows), one above the other. The topmost pane will display the question text, as usual. The middle pane will contain a field for the response and the bottom pane will show the values in the value set.

There are two optional settings for this command line parameter which control the height of the middle (response) and bottom (value set) panes. For example,

```
csentry test.pff /FullScreenValueSet(20:400)
```

These settings are useful to fine-tune the display to the particular device you are using for data capture.

Note: This option is not recommended for desktop or laptop computers.

Handle Don't Know and Refused

The best way to allow don't know and refused as values to a numeric data item is to make the item alpha numeric. It is not easy to specify a value set for alpha values, so postproc code needs to be written to perform the range check.

The procedure below provides an example of how to test the range of item that allow don't know and refused. The item is defined as alpha. The allowed alpha values of "D" for don't know and "R" for refused are tested first and accepts. If blank is entered, the value must be reentered. Convert the value to a number using the tonumber function. Test the value for default, meaning so non-numeric value and force a reenter. Finally convert back to a string, right justifying the result.

```
Proc AGE
If $ in "D", "R" then
  exit;
elseif $ = " " then
  reenter;
endif;
X = tonumber($);
If x = default then
  reenter;
else
  $ = edit("Z9",x);
endif;
```

Handle Multiple Answers

Some questions allow for multiple responses. For example a such as "Which of the following items do you have in your household?", allows for multiple responses.

There are two methods that could be used to collect this data in a CAPI application. Both involve defining in the Data Dictionary a data item with multiple occurrences. For example if there are 10 possible responses, then the item, usually a single character, would occur 10 items. On the data entry form, the item would be displayed as a roster with 10 occurrences. Response labels can be given on the form by changing the occurrence labels from numbers from 1 to 10 to the text of the responses, so the enumerator will know, and can read the possible responses.

One way to collect the responses is to create a roster with free movement, or spreadsheet like behavior, so that enumerator can move the cursor between occurrences to mark the ones to which the respondent gives positive answers.

Another way to collect the data is to protect the roster and have a separate data item to collect the number of the response. When the number is entered, the corresponding occurrence of the item is marked. If the number is entered again the corresponding occurrence is unmarked.

Choose Topic Sections

Often in a CAPI questionnaire, the enumerator needs to be able to move from one section of the questionnaire to another.

To make this happen, a global navigation key needs to be established and handled by the onkey function. The accept function can be used to collect the enumerators choice of what section to move to and define the first field in the section to move to. Movement, however, must be made from a location, a field, before any of the sections are encountered in order maintain the path structure. This field is protected so that no data need to be entered into it.

Below is code for the onkey function to support using F9 to move to another section in the application. The NAVIAGE data item provides the location of movement.

```
function Onkey(x);
if x = 120 then {F9 – To navigate among sections }
  n = accept("What Section?",
             "Section A",
             "Section B",
             "Section C",
             "Section D",
             "Section E");
  if n = 1 then {Section A}
    JumpField = "Q_A1";
  elseif n = 2 then {Section B}
    JumpField = "Q_B1";
  elseif n = 3 then {Section C}
     JumpField = "Q_C1";
  elseif n = 4 then {Section D}
     JumpField = "Q_D1";
  elseif n = 5 then {Section E}
     JumpField = "Q_E1";
  else {esc}
    onkey = 0;
    exit;
  endif;
  JumpFlag = 1;
  move to NAVIGATE;
endif;
end;

Proc NAVIGATE
OnFocus
$ = "Z"; {need to have a value in the field}
PostProc
if JumpFlag = 1 then
  JumpFlag = 0;
```

```
  move to JumpField;
endif;
```

Create General Helps

Often a specialized general help is needed for a CAPI application. Such a help can be established in the following manner.

1 Put form with one alpha item at end of all forms.
2 Make question text for the item be the help text.
3 Code in the onkey function a key that will jump to help form.
4 Code alpha item on the help form that will to jump back.

In the code below, Ctrl+H is used to jump to the general help. The data item X_HELP is on the form with the help text.

```
Proc Global
alpha (32) xlast, hlast;

function OnKey(x);
xlast = getsymbol(); {current field location}
if x = 2072 and xlast <> "X_HELP" then {ctrl+h}
  hlast = xlast;
  move X_HELP; {move to help form}
endif;
onkey = x;
end;

Proc X_HELP
reenter hlast;
```

Test Application

CAPI applications require even more testing than data entry applications from paper forms because of the enumerator's need to be able to move easily and quickly around the questionnaire.

Here are some suggestions on how to test an application.

- Go through the case in intended sequence to test skip patterns, range checks, and edit checks.
- If breaking off and restarting the interview are allowed, try doing that from different locations. Can you get back to where you left off easily?
- Test moving around within the application, moving backward and forward.
- Test moving between sections, if that is allowed.
- Test field helps and general helps.

Android Data Entry

Android Development

Overview

CSEntry, CSPro's data collection tool, is available as an Android application from the Google Play store. This application can be used to run data entry applications on devices such as phones or tablets running

Android version 4.0 (Ice Cream Sandwich) or higher. CSPro data entry applications can be designed on Windows machines and then run on either Windows or Android devices. This is a "build once, deploy many" system, whereby the running behavior across the two operating systems is very similar, so you have the freedom to decide what device you want to use to conduct CAPI surveys using CSEntry.

In the Android environment you will only be able to run binary data entry applications (.pen files). You will not be able to execute .ent files or any other kind of CSPro application (concatenation, batch processing, etc.). This .pen file, along with the .pff runtime script, constitute the two files needed to run a data entry application on an Android device.

All applications on the Android device must be stored in the <external storage>/csentry/ folder on your device. Connect your Android device to your computer and, via USB transfer, copy your .pen and .pff files to the Android device. Generally the device's external storage is its SD card. If you have already run the CSEntry application, the csentry folder will have been automatically created for you. Otherwise, you must create it manually before copying over any .pen and .pff files. The application searches subdirectories located in the csentry folder, so it is possible to add multiple applications while maintaining a clean directory structure.

You can also transfer applications to your device by using the data synchronization features for Android.

Data Security

As in the Windows environment, the Android application does not encrypt data files, but simply stores all data in plain UTF-8 encoded text files. Fortunately, it is easy to encrypt an Android device so that all files stored on it are encrypted. If you do not want your data to be stored in plaintext, on your Android device:

1. Go to Settings.
2. Under the Personal group of options, select Security.
3. Select the Encrypt Phone option and follow the subsequent instructions.

Encrypting the phone will require that you supply a PIN or password to protect the data on the phone.

Differences Between Windows and Android

As of CSPro 6.0.1, the Android application only displays one question per screen. A future release will allow the displaying of rosters and multiple fields on a single screen. In addition, though most functions operate identically on the Windows and Android platforms, there are a few functions that behave differently, such as execsystem. Notes about differing functionality are noted in the help documents about each function.

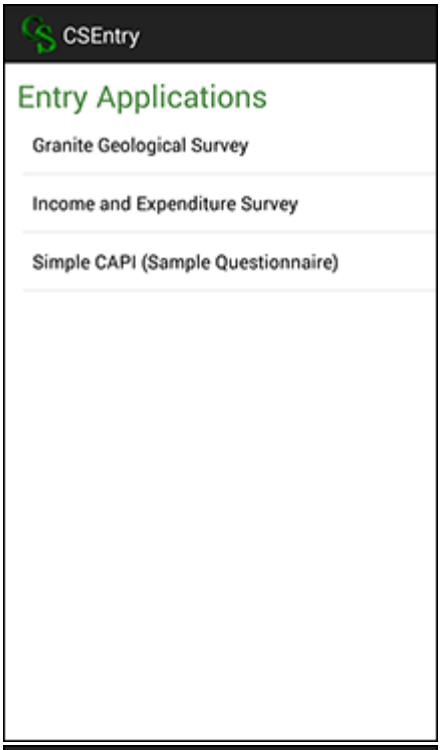
Android Navigation

Overview

The CSEntry for Android application is designed to be straightforward to use, with features and options that interviewers can easily control.

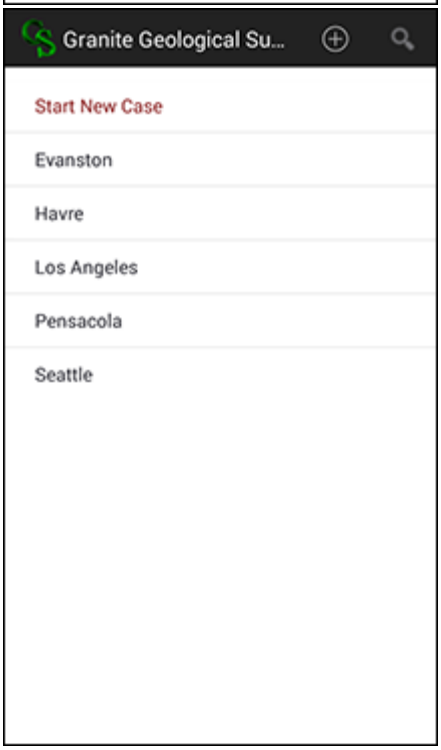
Entry Applications Screen

Upon opening CSEntry, all data entry applications on the device are displayed. Click on an application to open it. If only one application is on the device, it will open automatically.

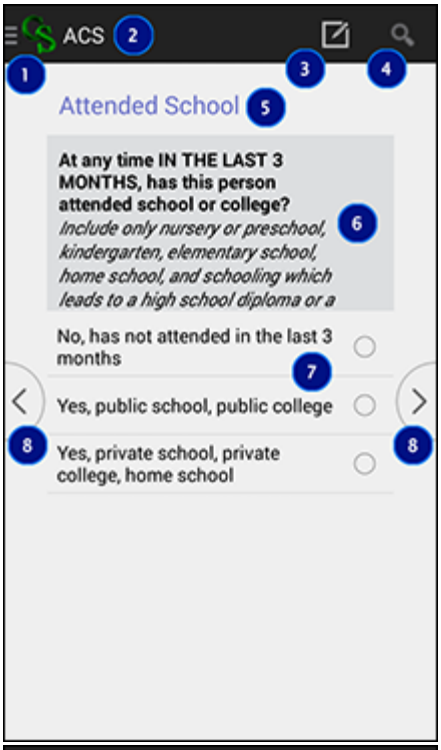


Case Listing Screen

The case listing displays all cases that have already been added to the data file. Clicking on a case will open it for modification. Clicking on "Start New Case," or the icon of a cross in a circle, will add a new case. To delete a case, hard-press on the case until a deletion menu appears. The menu on this screen has options to sort the cases alphabetically, or to display only incomplete cases.



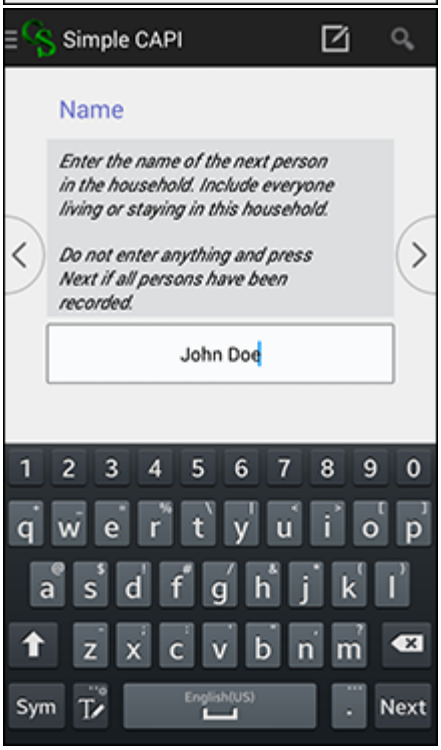
Screen Elements



This illustration showcases the features of the data entry screen.

1. Clicking on the CS icon brings up the case tree.
2. This is the name of the survey.
3. Clicking on the pencil and paper icon brings up a screen to type a field-specific note.
4. Clicking on the search icon and typing in a query allows you to filter the response listing.
5. The field label is a short description of the field that you are currently entering.
6. This is the question text for the field that you are currently entering. This will be more descriptive than the field label.
7. These are the response labels. CSEntry can display text boxes, radio buttons, check boxes, combo boxes, or date selectors.
8. These are the left and right navigation buttons, which correspond to moving backwards or forwards in the data entry application. Clicking on the Android back button will close out of adding or modifying a case.

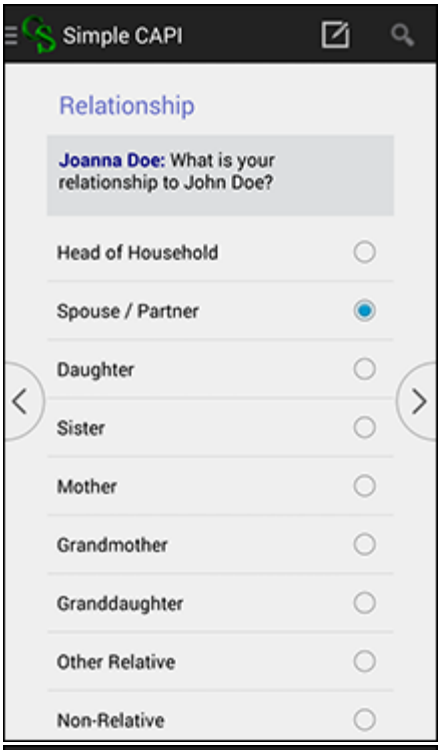
User Inputs: Text Box



To enter data in a text box field, simply type the response using the keyboard. If a keyboard does not appear automatically, click on the field and a keyboard will appear. If your keyboard has a Next key, you can click on it to move to the next field.

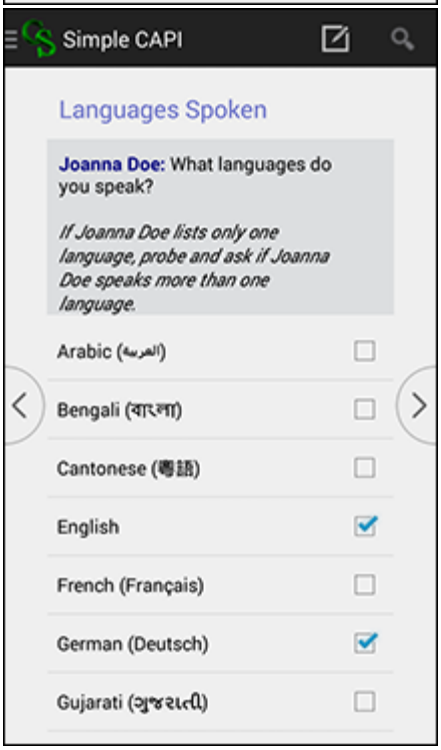
User Inputs: Radio Buttons

When presented with a list of radio buttons, you must select one, and only one, response. Click on the response label, or the corresponding radio button, to make your selection.



User Inputs: Check Boxes

When shown checkboxes, you can select none, one, or multiple responses. Click on the response label, or the corresponding checkbox, to make a selection. Once you are finished making all selections, you can proceed to the next field.

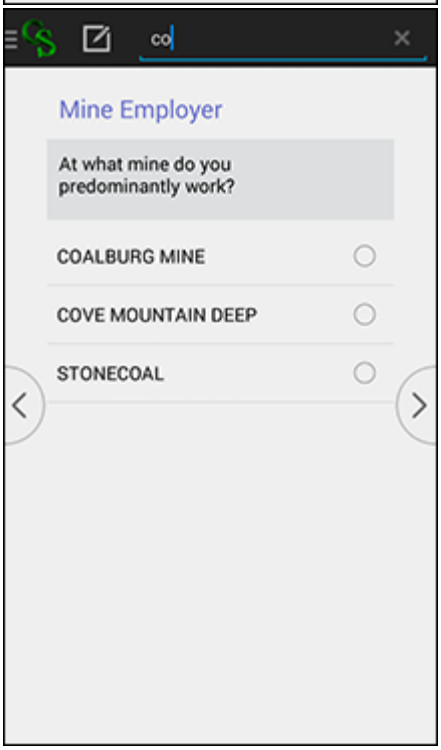


User Inputs: Combo Box

With a combo box, you can either enter a value within the defined range (as in a text box field), or choose a discrete value (as with radio buttons). To bring up the discrete values, click on the button to the right of the text field.

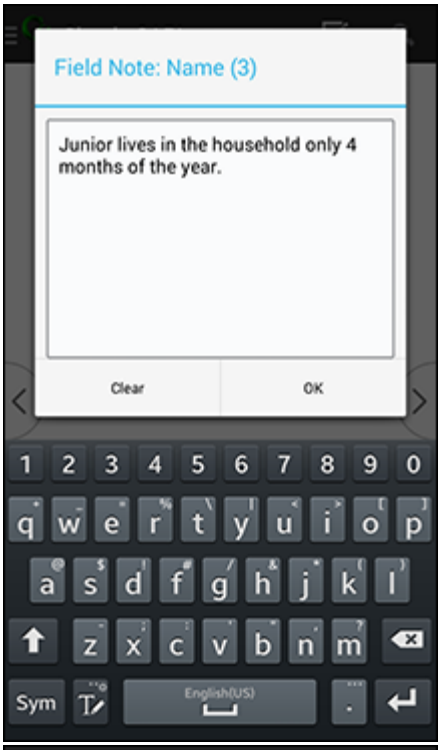
Search Filter

After clicking on the search icon, a space will appear where you can type a search query. The list of responses will automatically filter based on your search query. The full text of each response is searched, not necessarily starting from the first letter of the response.



Field Notes

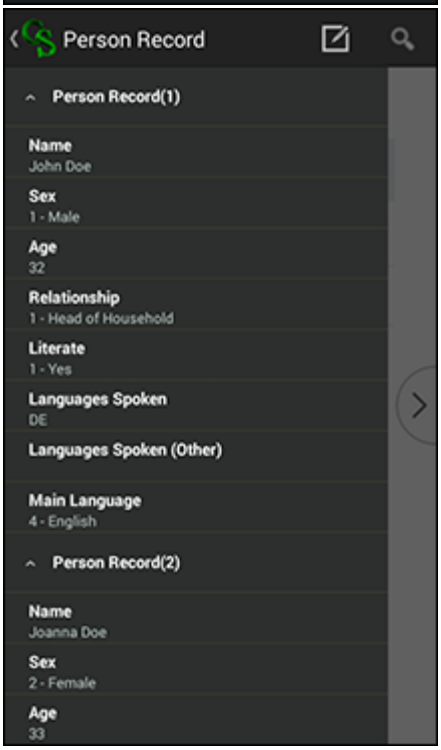
After clicking on the pencil and paper icon, you will have the opportunity to type a field-specific note. If you do not want to add a new note, you can press the Android back button to cancel.



Case Tree

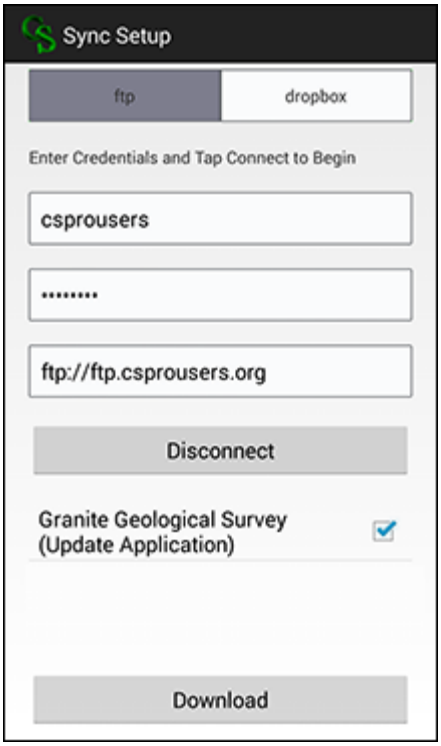
Clicking on the CS icon brings up the case tree. The case tree displays all fields that have been entered in the data entry application, showing the field label as well as the field response. If you click on a field, you will be taken to that field. This allows you to quickly move from one part of the questionnaire to another.

Once the case tree is shown, clicking on the CS icon again will take you up the hierarchy of the data entry application. For example, you might go from seeing details about a person to seeing details about the household. To insert or delete a repeating occurrence, hard-press on the occurrence label until you are presented with a menu that allows for insertions and deletions. You can dismiss the case tree by using the Android back button.



Sync Setup Screen

From the Entry Applications screen, clicking "Sync Setup," one of the menu options, brings you to a screen where you can download data entry applications from a Dropbox account or an FTP server. For specific instructions on how to use this screen, talk with your survey data processing manager.



Pocket PC (PPC) Data Entry

Introduction to Pocket PC (PPC) Data Entry

CSPro supports data entry using handheld computers running the Windows Mobile operating system, also known as Pocket PCs, or PDAs. Developing CSPro data entry programs for Pocket PCs is very similar to creating CSPro data entry applications for standard desktop computers. In fact, most CSPro desktop applications can run on a Pocket PC with little or no changes. Applications for handheld computers are developed on a desktop or laptop computer using the standard CSPro data entry designer. They are then compiled and copied onto the handheld device.

The basic process for creating CSPro applications for the Pocket PC is the following:

1. Develop and test the application on the desktop (or laptop) using CSPro.
2. Generate a binary (compiled) data entry application.
3. Copy the binary application to the Pocket PC for further testing and eventual use in the field.
4. Run the application on the Pocket PC to collect data.
5. Copy the data collected back to the desktop (or laptop) computer for editing and analysis.

This section contains the following information:

PPC Requirements
How to …

PPC Requirements

In order to develop and run CSPro applications for the Pocket PC you will need the following hardware and software:

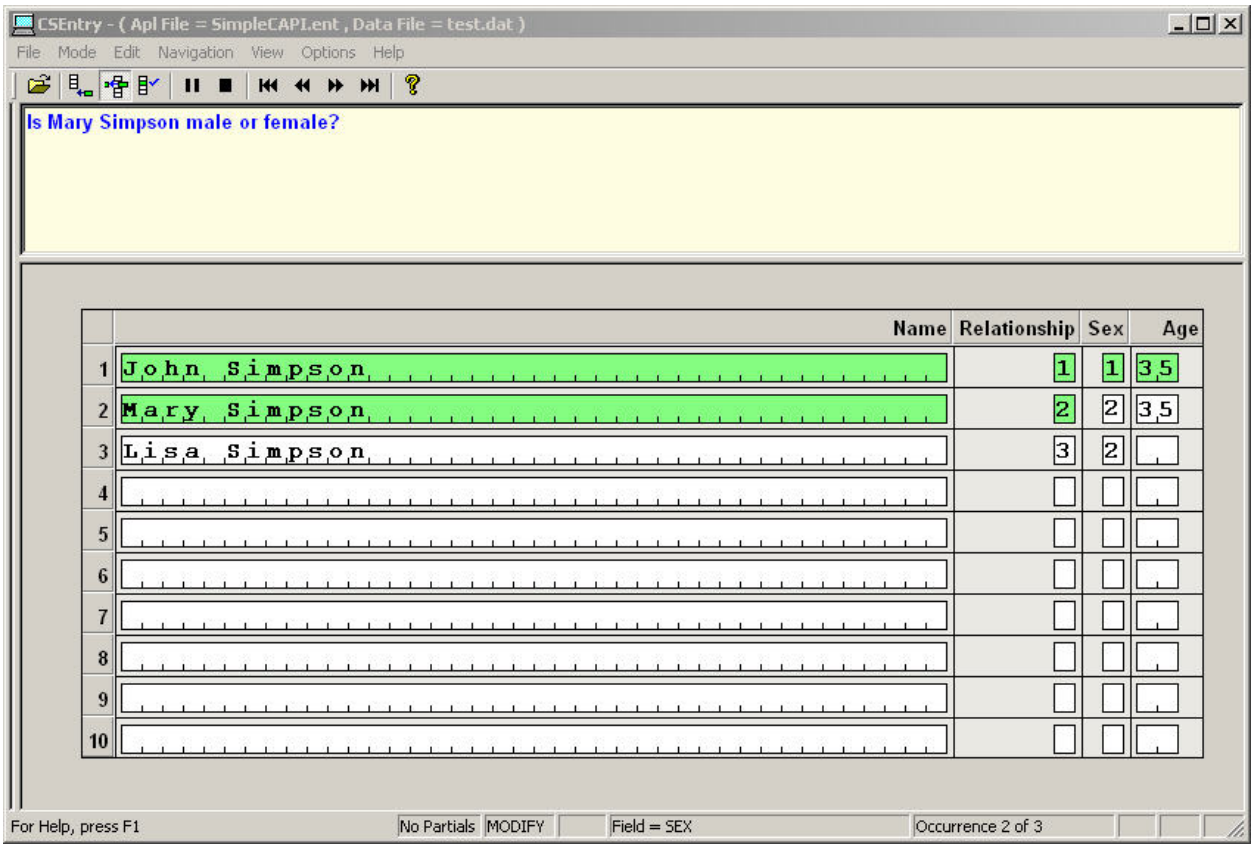
- Pocket PC running Windows Mobile 5.0 or later. Note that earlier versions of Windows Mobile, such as Windows Mobile 2003 and Pocket PC 2002 are not supported. In addition, the SmartPhone versions of Windows Mobile are not supported.
- Desktop or laptop PC capable of running CSPro 4.1.
- USB cable or cradle to connect the Pocket PC to the desktop or laptop computer.
- Microsoft ActiveSync 4.2 or later. This software should come with your Pocket PC. If not, it may be freely downloaded from the Microsoft web site.
- CSPro 4.1.
- CSProMobile. This is the installer for the Pocket PC. It is separate from the installer for the "standard version" of CSPro on a desktop computer. It can be downloaded from the CSPro website.

How To ...

Write PPC Data Entry Application

Writing applications for the Pocket PC is really no different than writing regular CSPro data entry applications. In fact, in most cases, the same application can be run seamlessly on both platforms.

The major difference between CSPro data entry on the desktop PC and the Pocket PC is that on the Pocket PC, only one question at a time is displayed on the screen while on the desktop PC, the entire form is visible. However, this difference is handled automatically by CSPro and takes no extra effort on the part of the application designer. You can build your application using forms and rosters as you normally would and when you run it on the Pocket PC, CSPro will only display the current field rather than the whole form. The following images illustrate this difference by showing the same application running on a desktop computer and on a Pocket PC.



Entire form visible when running on desktop

Only current field visible when running on Pocket PC

Note that labels, additional text and boxes on a form that are visible when running on a desktop computer are not shown on the Pocket PC. We recommend that you place information such as labels and interviewer instructions in the CAPI question text rather than on the form. For this reason, most applications for the Pocket PC will be CAPI data entry applications.

Usually the first step, once the application is created, is to go to the "Data Entry Options" dialog box (from the CSPro menu at the top select "Options" / "Data Entry"), then check the box labeled "CAPI Mode". Make sure the box labeled "Use PPC Controls" is also checked.

It is also important to pay attention to the value sets in your dictionary that you use for your variables. An additional difference between standard data entry and the Pocket PC is that the values from the first value set for each variable are displayed underneath the field so that the user may choose an option rather than use the virtual keyboard to enter a value. In the image above, the user can simply click on Male or Female. In order for this feature to work, it is important to specify value sets with clear labels for each possible value rather than simply specifying valid ranges. Of course, this will not be possible for all variables.

On the Pocket PC, every variable has a field capture type. Depending on the variable's value set, this will be either: radio button, check box, drop down, combo box, text box, or date. You can see or change a variable's capture type, by right-clicking on the field on the form and selecting "Properties".

There are a couple of potentially useful commands that have been added to CSPro data entry to better support the Pocket PC. These are (1) the Show function which displays items from a roster in a grid view; and (2) the select clause added to the Errmsg function which adds buttons to the error message dialog that allows the user to choose a field to move to in order to correct the error. Both of these new features are fully documented in the online help and are also used in the Simple CAPI example which can be found in the CSPro examples directory.

Some advanced users may want to write a CSPro data entry application that calls another CSPro data entry application. Applications that run on a desktop or laptop would include an Execsystem function to invoke CSEntry.exe. The executable on the Pocket PC that corresponds to CSEntry.exe is named

CapiESD.exe. However, due to limitations of the Windows Mobile operating system, you cannot have two instances of CapiESD.exe running at the same time. CSPro therefore installs a file named CapiESD2.exe, which is an exact copy of CapiESD.exe. You should use "CapiESD2.exe" in the Execsystem function of the calling application when you intend the application to run on Pocket PC.

Once you have created your data entry application on a desktop or laptop, you are ready to deploy the application on the Pocket PC.

Install CSPro on a Pocket PC

Before you can run an application created with CSPro on the Pocket PC, you first need to install CSPro on it.

Before you start the installation, first connect your Pocket PC to the desktop or laptop computer using the cradle or USB cable.

The installer for the Pocket PC is separate from the installer for the "standard version" of CSPro on a desktop or laptop computer. The Pocket PC install is called CSProMobile.msi and can be found on the CSPro CD or downloaded from the CSPro website.

Start by double-clicking on this file.

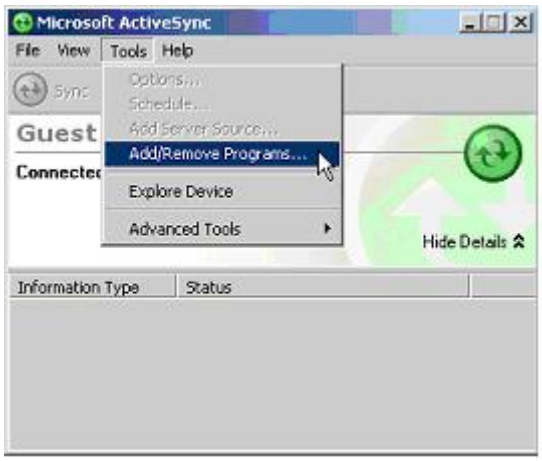
Follow the instructions in the installer. If your Pocket PC is correctly connected to the computer, it will automatically install CSPro on the Pocket PC. When it is complete, it will ask you to check the screen of your mobile device.

At this point you should see a message on your Pocket PC indicating that the installation was successful.

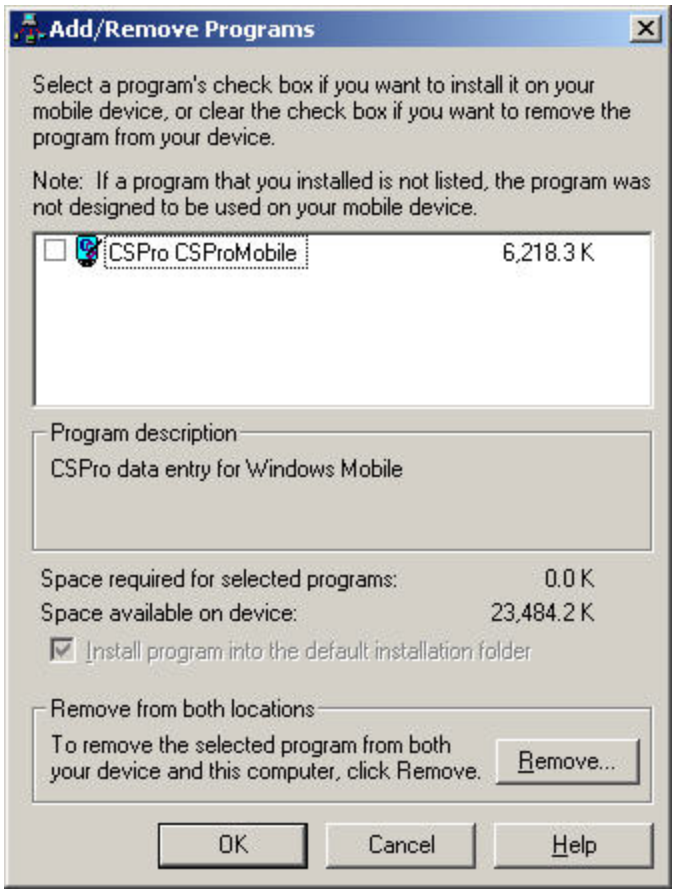
Once the installation has completed, you should test it by running CSPro data entry on Pocket PC. You can access it via Start Menu / Programs / CSPro Data Entry.

Installing CSPro Mobile on multiple Pocket PCs

Once you have run the CSPro mobile installation once on your desktop or laptop computer, you can install it on multiple Pocket PCs from the desktop computer by using the Add/Remove Programs from the Tools menu in ActiveSync. Simply connect the Pocket PC you wish to install to, launch ActiveSync and choose Add/Remove Programs to bring up the Add/Remove Programs dialog.



In the Add/Remove Programs dialog, check the box next to CSPro Mobile and click OK.



Once you click OK, the installation process will begin and lead you through the steps described earlier in this section.

If you do not want to connect each Pocket PC to a desktop or laptop computer, you can use a storage card instead to install CSPro Mobile to multiple devices. Once you have installed CSPro Mobile using the procedures described above, there will be a file on the desktop or laptop computer named "CSPro CSProMobile.PPC500.PPC500.CAB". This is usually installed in the "C:\Program Files\CSPro 4.1 Mobile\CSPro 4.1 Mobile" folder. Do the following:

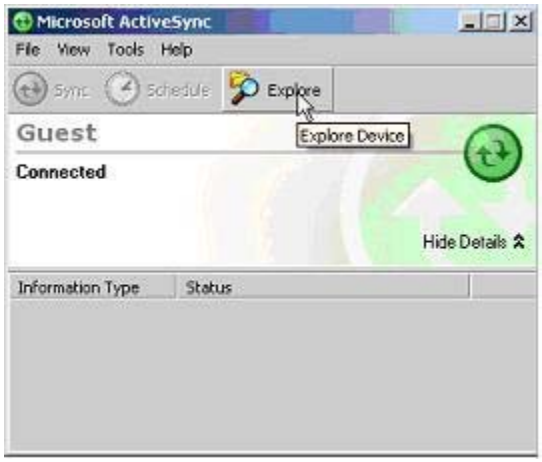
- Copy "CSPro CSProMobile.PPC500.PPC500.CAB" to the storage card.
- Insert the storage card into a Pocket PC.
- Use the Pocket PC's File Explorer to locate this file.
- Tap on the file to begin the installation.

Deploy PPC Application to the Pocket PC

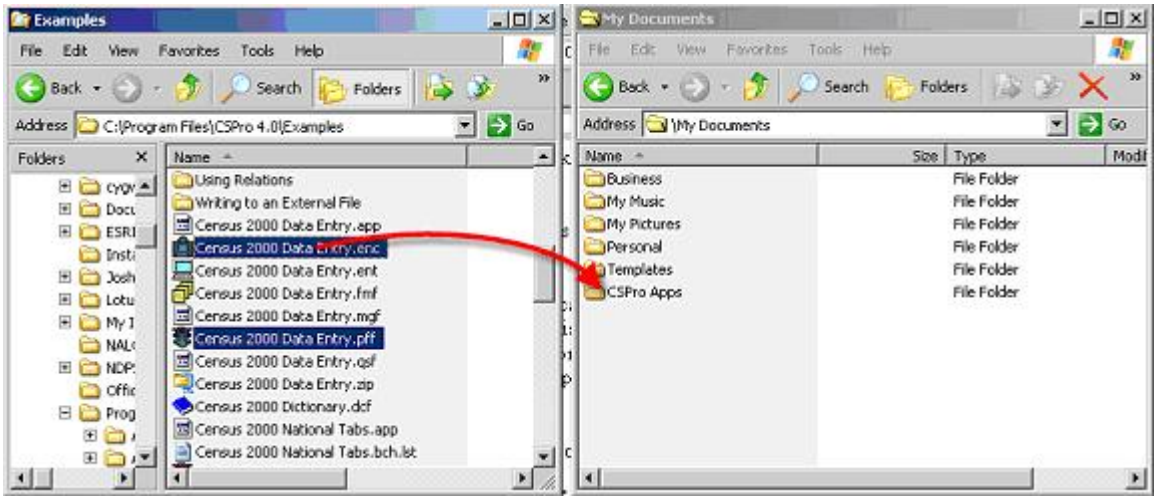
(CSPro no longer supports pocket PC applications. This information is kept as a reference.)

To deploy your data entry application to the Pocket PC you must do the following.

- Generate a binary (compiled) version of the application. To do this, open your application in CSPro and choose "Generate Binary" from the file menu. The binary application will have .ENC file extension.
- Prepare a .PFF file. This file tells CSPro which data file(s) to use with your application. When you run your application on the desktop computer, CSPro automatically generates a .PFF file for you. It will have the same name as your application only with a .PFF file extension and a traffic light for an icon. It will be placed in the same directory as your application. In most cases, you can use the .PFF file generated on the desktop computer on the Pocket PC. The only case where this will not work is if you place your data file in a different directory relative to your application on the desktop than you do on the Pocket PC. For example, if your data file is on a different drive from your application (D: instead of C:), this will not work on the Pocket PC since there are no drive letters in Windows Mobile. If this is the case, then you can open the .PFF file in a text editor such as notepad and modify the path to the data file to use a directory that will be valid on the Pocket PC. If your data file and your application are in the same directory on both the desktop and the Pocket PC then you will not need to make any changes. For more information on .PFF files, see "Run Production Data Entry" in the CSPro online help.
- Copy the .ENC and .PFF files to the Pocket PC. The easiest way to do this is to use ActiveSync. Click on the "Explore" button in ActiveSync to browse the files on your handheld.



You can then drag and drop the .PFF and the .ENC from the desktop computer onto the Pocket PC.



- You should place the .PFF and the .ENC files into either the My Documents folder, a sub-folder of My Documents, or a folder on the storage card if there is one. This is because standard Windows Mobile file dialogs used in CSPro do not look in folders other than these.

Once the files have been copied to the Pocket PC, you are ready to run the application on the Pocket PC.

Run PPC Application on the Pocket PC

To launch your data entry application on the Pocket PC, tap "Start", then "Programs", then "CSPro Data Entry". If you have already run the application you may see "CSPro Data Entry" right after you tap "Start". You will then see a list of .PFF files that have been moved to the Pocket PC. Simply tap on the line corresponding to the .PFF file to get started.

Launching CSPro Data Entry from the programs folder and choosing a .PFF.

Note that you can also launch the application by finding the correct .PFF file through the Pocket PC's File Explorer and tapping on it.

If the data file (specified in the .PFF file) has any cases in it, you will first see a list of them, like this:

You have the following options:

About	See general information about CSPro.
Add	Add a case to the data file.
Mod	Modify the selected case in the data file.
Del	Delete the selected case from the data file.
Language	Change the language of the question text.
Exit	Exit CSPro.

If you add or modify a case you will see the first question on the screen, like this:

In this example, the user tapped on **More** simply to show other options. The options are:

Prv	Move to the previous field.
Nxt	Move to the next field.
Adv	Advance to the end of the case.
Note	Write a note for this field.
Save	Save this case.
Tree	Show a tree with all the data in the case. This is very useful to see where you are in a long interview. You can tap anywhere on the tree to move to that field.
More / Stop	Stop adding or modifying the current case.
More / Language	Change the language of the question text.
More / Help	Show the "help" text for this field as defined in the application.
More / About	See general information about CSPro.

Note that the icon to the right of **More** brings up the virtual keyboard, which allows you to enter text when needed.

Also note that there are two "keys" you can use from the virtual keyboard with operator-controlled applications:

Ctrl + / Exit the current roster or group. Same as the desktop/laptop version.

Ctrl + P Move to previous persistent field. Equivalent to F7 in desktop/laptop version.

A convenient feature to be aware of is that you can minimize the question text area, and therefore maximize the response area, by tapping on the question text area. The first tap will minimize the screen area that contains the question text and the next tap will restore it. This is useful in cases where the question text is long and the list of responses is also long. Normally the interviewer would have a small window that shows very few of the responses, and requires a lot of scrolling. After reading the question text, they can tap on that window to be able to see more of the responses.

Retrieve Data Files from the Pocket PC

After the data are captured, you must move them to a desktop or laptop computer for further processing. CSPro does not provide tools for this. You must use methods and tools that come with the Pocket PC or that you set up on your own. There are many ways to do this. Examples include:

- ActiveSync, using the File Explorer to drag and drop.
- By way of a physical memory card.
- Bluetooth or other wireless connection.

Batch Editing Applications

Introduction to Batch Editing

Over the years, the questions of whether or not to edit data, and if so, how, have generated a great deal of discussion and even controversy. Today, it is generally recognized that data from any survey or census will require at least minimal editing [and correction] in order to be usable. The following sections present a more detailed discussion of these questions, and the tools available in CSPro to carry out these tasks.

The Batch Edit Designer module allows you to create and modify batch edit applications. A batch edit application is used to clean (via editing and imputation) your data files.

For examples and methodology on how to develop your edit routines, refer to the recently revised United Nations Handbook on Population and Housing Census Edits.

This section contains the following information:

Create a Batch Edit Application
Order of Editing
Correcting Errors
How to …
Steps in Developing a Batch Editing Program

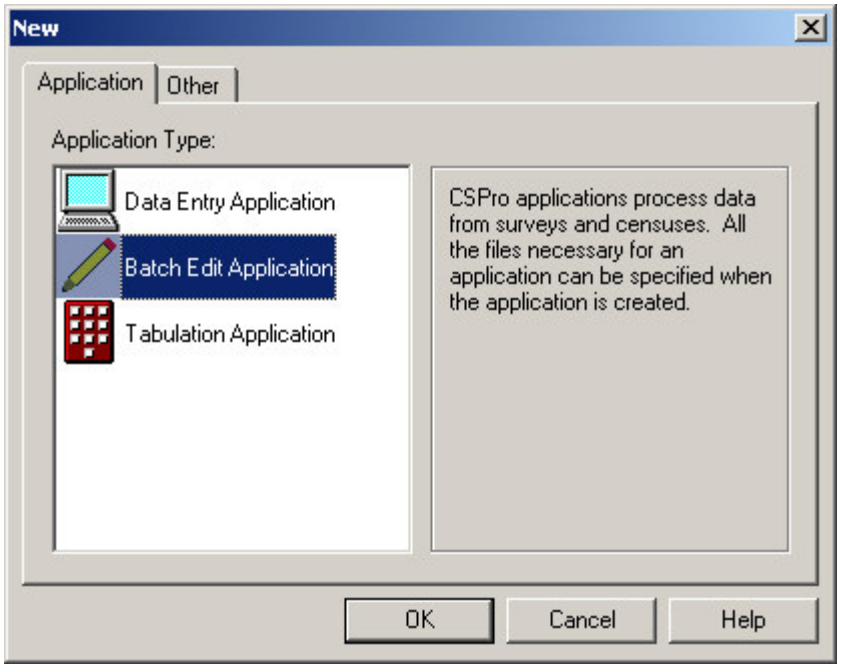
Create a Batch Edit Application

Create a New Batch Edit Application

To perform batch editing, you will need a data dictionary to describe the data file you are editing. If you already have a data dictionary for the file, you can specify this dictionary when you create a new batch edit application.

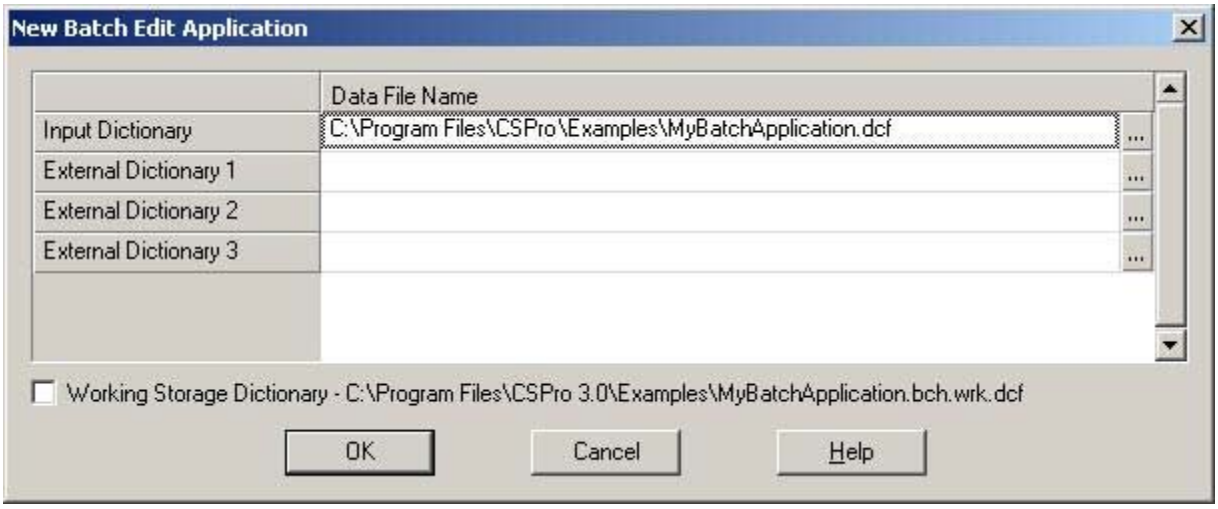
To create a new batch edit application:

• Click on the toolbar, or from the **File** menu, select **New**. The following dialog box will appear.



• Select **Batch Edit Application** and press **OK**.

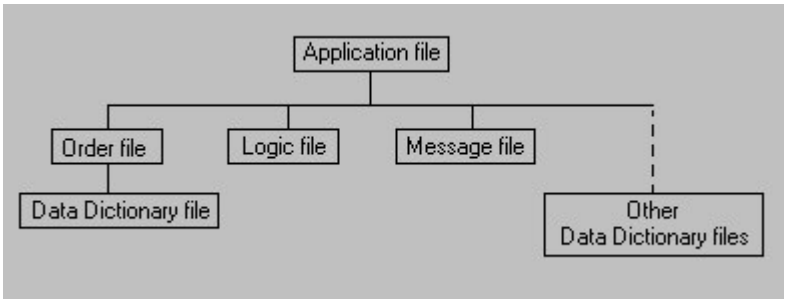
• A file dialog box will appear. Enter the name of the application file. Make sure you are located in the folder where you want to place the application files. Then press **Create**. The following dialog box will appear.



- A default name of the data dictionary describing the data file is given. You can use this name or change it. If you give the name of a dictionary file that already exists, that data dictionary will be used by the application. If you give the name of a dictionary that does not exist, a new data dictionary will be created.

If you are using an existing CSPro data dictionary, you may begin creating batch edit procedures. If you are creating a new CSPro data dictionary, you will need to enter information into the dictionary about records, items, and values before you can create edits.

Batch edit applications consist of the following files:



- Batch Edit Application File (.bch)
  Specifies all other files contained in the application and includes other application information.

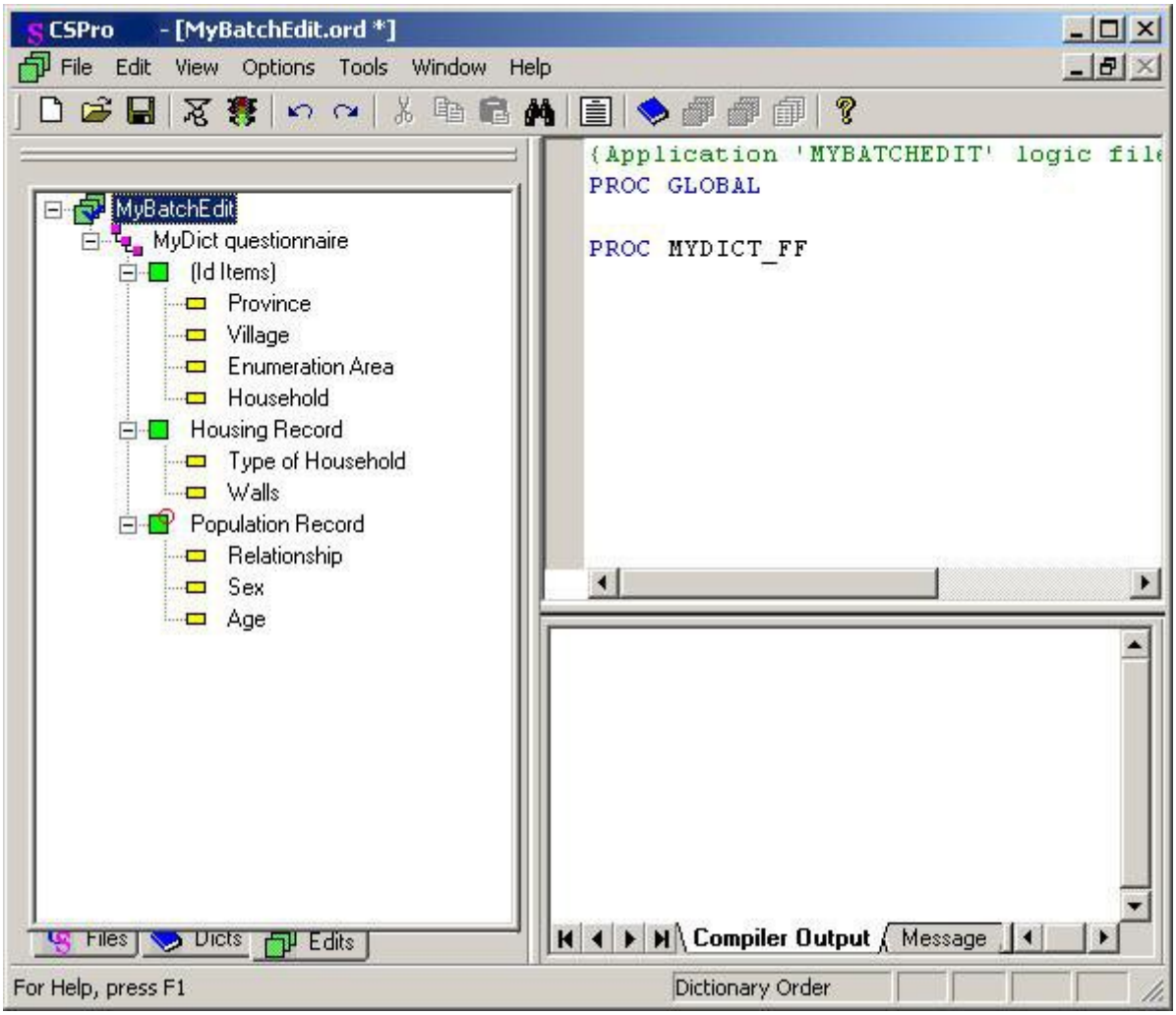
- Edit Order File (.ord)
  Specifies the order in which logic in the application is executed. There is usually one order file per application, but there may be multiple order files. Each order file contains one Data Dictionary file (.dcf) that represents the primary data file that is being read and/or written.

- Logic File (.apc)
  Contains CSPro language statements

- Message file (.mgf)
  Optional file, it contains text for messages displayed on the output listing

- Other Data Dictionary Files (.dcf)
  Optional, it represents secondary data files (such as lookup files) that are read and/or written to during the batch run.

Batch Application Screen Layout



The screen is divided into three main work areas: the Tree View, the Logic View and the Message View.

- **Tree View**

  The window on the left half of the screen displays the batch edit tree with the root node ( ) selected.

- **Logic View**

  This is the window block in the upper portion of the right half of the screen. It is the programmer's "clean slate, to which may be added logic for any part of the data file: any item, any section, any record, even the file as a whole. It is up to the programmer to determine the correct placement and sequence of execution for each logical element. The initial screen will display:

```
{Application 'MYBATCHEDIT' logic file generated by CSPro }
PROC GLOBAL

PROC MYDICT_FF
```

  These two lines of code will **always** be in your application file. You can delete them, but they will always be regenerated and placed in your file on **open**, **save**, or **exit**. This is the beginning of your program. You write the declaration statements under `PROC GLOBAL`, then the procedures for the event.

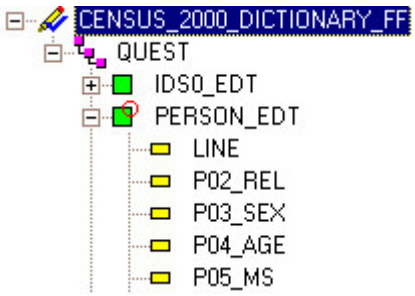
- **Message View**
  This is the window block in the lower portion of the right half of the screen. It is devoted to messages (user-created and system-generated). As with the Tree View, tabs are available to the programmer; clicking on one of them will make the contents of that view active. The **Compiler Output** tab displays errors found during compilation of your program; if the code compiled successfully, it will state "Compile Successful." The **Message** tab is used to type in error messages that will be used in the execution of the program.

If you wish to modify the size of any of these three work areas, just place the mouse over one of the separating bars, grab it, and drag to resize.

See also: Moving Around a Logic Application

Batch Edit Tree

When you create a batch edit application, the edit tree will be identical to the dictionary tree; that is, edit items will be listed as named and ordered in the dictionary. However, there are a few distinctions to make, as follows:



- **BatchEdit File:**
  This is the highest level node, i.e., the root node. It is the owner of all code, which is to say [1] level-, record-, and item-related code, [2] user-defined functions, and the [3] global routine.

- **BatchEdit Level:**
This is the second-tier tree node, just below the root. It has a 1-to-1 correspondence with the same-named dictionary level.

- **BatchEdit Record:**
This is the third-tier tree node, just below its level. It has a 1-to-1 correspondence with the same-named dictionary record.

- **BatchEdit Item:**
This is the terminal or "leaf"-node; i.e., the lowest accessible level. It has a 1-to-1 correspondence with a dictionary item.

You are free to rename any of the above the unique names via the properties dialog box, but it is recommended that you retain the original name, so that it is easier for you to see which dictionary entity is being referenced. The batch edit tree represents the order in which the logic associated with each edit item is executed.

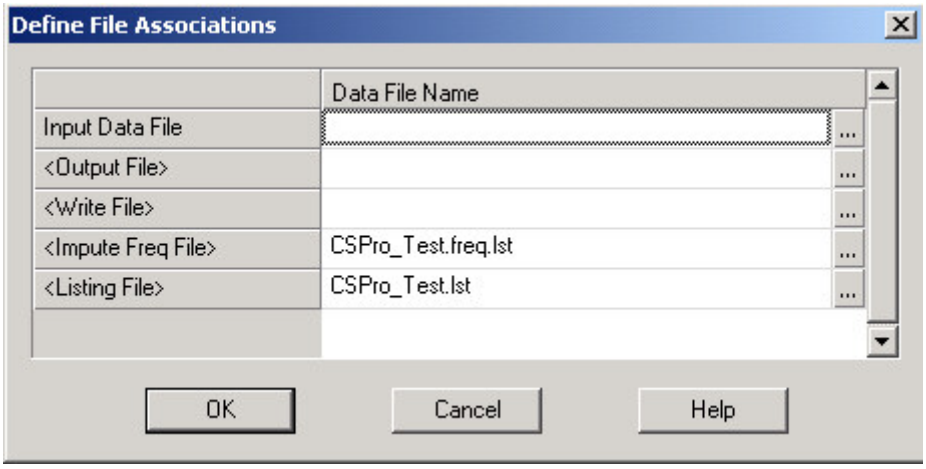
If code has been written for a given edit level, record, or item, a check mark will appear superimposed on the icon for that entity. This is how, at a quick glance, you can see where you have placed programming logic. Once one line of code has been written anywhere in the program, a check mark will appear on the root node.

You can never delete edit levels, records, or items (i.e., the entries shown on the **edit** tab). However, you can change the order of the logic execution by dragging the items within the Batch Edit tree view. When selecting a new edit item, the contents of the logic view will change to display the logic for the selected entity.

Pressing **Ctrl+T** in the batch edit tree will allow you to switch between the labels and the names of the items.

Run a Batch Edit Application

Press on the toolbar; from the **File** menu, select **Run**; press **Ctrl+R**. If you've made changes since you last compiled, CSPro will first compile your application. If your program compiles successfully, you will receive the following dialog box:



- **Input Data File**
  This line is required, and asks for the name of the data file against which you wish to run your batch application. This data file will not be modified in any way; it will only be opened, read, and closed.

- **Output File**
  The output file is where the results of correcting your data will be written. If you are **not** making any corrections in your program, then the generated file will be an exact copy of the original data file. If you **are** making corrections to your data file, then this will be the corrected data file. The default file extension is .out, but you can use whatever you'd like. This field is optional; therefore, if you are making corrections to your data, but forget to specify an output file, no corrected file will be generated.

- **Write File**
  If you have any write functions in your program, they will write information to this file. The default file extension is .wrt, but you can use whatever you like. This field is optional; if no write file is specified at run time, the write file lines are placed in the default data entry or batch error report.

- **Impute Freq File**
  If your program contains any impute statements, the results of this command will be written to this file. The default file extension is .frq, but you can use whatever you'd like. This field is optional; therefore, if your program contains **impute** commands, but you forget to specify a frequency file, no file will be generated. Similarly, if you indicate a frequency file but your program does not contain any impute commands, no file will be generated.

- **Listing File**

This line is required, and asks for the name of the file to which you want to write the results of the run. Results from errmsg functions will be written here. This file will always be generated, regardless of whether or not your program includes errmsg commands.

Order of Editing

Order of Executing Batch Edit Events

CsPro executes the procedures in an application one case at a time. There are two types of procedures for each element: a preproc and a postproc. If the type of procedure is not named, **postproc** is assumed. The following diagram illustrates the order of operations for a two-level batch edit application. Level 1 has two records (1 and 2) and level 2 has one form (3).

```
Application File preproc
 Level 1 (case) preproc
  Record 1 preproc
   Item 11 preproc
   Item 11 postproc
     :
   Item 14 preproc
   Item 14 postproc
  Record 1 postproc
  Record 2 preproc
   Item 21 preproc
   Item 21 postproc
      :
   Item 26 preproc
   Item 26 postproc
  Record 2 postproc

  Level 2 preproc
   Record 3 preproc
    Item 31 preproc
    Item 31 postproc
      :
    Item 34 preproc
    Item 34 postproc
   Record 3 postproc
  Level 2 postproc
   … (repeat level 2 procs for other instances of level 2
 Level 1 (case) postproc
Application File postproc
```

See also: Batch Edit Order

Batch Edit Order

Eventually you'll reach the point where you have written edits for many variables and you will begin to wonder, just how do you control the order of execution? It's in the batch edit tree. The order of the items listed in the BatchEdits tree tab shows you the order of logic execution. (If there is no associated logic for an edit item, then the order is of course not important.)

What if you don't like the order that's given? Change it. As mentioned above in "The Batch Edit Tree" you can re-order items and records (but not levels) on the batch edit tree. When developing edit specifications, the edit of one variable might depend on another edit having already been completed (say,

relationship before sex). If the dictionary wasn't designed in the order you need, then when a batch edit application is generated, the order will be incorrect.

Having said all this, there are a few nuances to the editing process that you may wish to note, specifically with regard to the execution of **preproc**s and **postproc**s execution:

- For BatchEdit items, there is no benefit to writing and maintaining **preproc** and **postproc** code blocks. Because a BatchEdit item is at the lowest level in the order tree, no other code would be executed in the interval between a **preproc** and **postproc** code block. Therefore it is suggested that only one code block exist for each item. If you do not preface the instructions with either "**preproc**" or "**postproc**", the code by default will be in the **postproc** block. This is the recommended approach; that is, accept the default.

- If a BatchEdit item is within a record that repeats, the logic will be repeated for each occurrence of the record. For example, if you have a population record that allows 30 occurrences, the logic for each of its member items can repeat up to 30 times, depending on the actual number of persons in the household. Suppose you have a household with three members: the head, the spouse, and a child. The logic for each data item (such as "relationship," "sex," and "age") will be executed three times.

- If a record repeats, the associated logic for that record will NOT repeat; instead, it will be executed once and only once for questionnaire. For example, take that population record again that allows 30 occurrences. Whether there are 1 or 5 or 30 people in the household, the associated logic for the BatchEdit record will execute only once. Therefore, if you have logic that must occur for each person in the household, we suggest you place that code under the first BatchEdit item in the record.

- Logic written for Level 1 will only execute once for a questionnaire/case. Logic written for Levels 2 or 3 will execute for each node, i.e., for each set of records contained in that level.

- Finally, logic written at the BatchEdit File node will execute only once for a data file. Therefore, if you have global variables that you need to initialize, etc., this is the place where that should take place.

Change Edit Order

By default, a new batch edit application fixes the order of editing to the order of items in the dictionary. If new items are added to or rearranged in the dictionary, the editing order is determined by the new dictionary arrangement.

To make your own custom order of the editing items within records, you need to do two things: from the **Options** menu, select "Custom Order"; then drag and drop items in the batch edit tree into the order you wish the edits to be performed. If you rearrange items within a record in the dictionary, the custom order will not change. If you add new items to a record, the new items will be placed at the end of the record for purposes of editing. If you unselect "Custom Order", the edit order will return to the order of items in the dictionary.

See also: Batch Edit Order

Correcting Errors

Methods of Correcting Data

Before you can correct your errors, you need to know what kind of errors you have. You have two methods of finding these errors: manually or automatically. Manually checking large quantities of data is an extremely time-consuming and error-prone task and not recommended. By contrast, automated searches for your errors is quick and, if done properly, a reliable method to use. If you wish to use CSPro to automate the search for errors, you must create a Batch Edit application.

Using a Batch Edit application to identify errors is a relatively easy task, though care must be taken to do so correctly. Improperly identifying errors can waste precious personnel resources, so a precise set of rules should be developed with input from subject-matter specialists.

Just as you have two methods available to you when searching for errors, you have two methods available to you for correcting errors: manually or automatically.

- **Manual Correction**
  Manual correction of a census could take years, and the possibility of human error is great. When large volumes of data are collected in censuses and surveys, it is not always practical to refer to the original document in order to correct an error. Often the data recorded on the original questionnaires are wrong or inconsistent

- **Automatic Correction**
  With computer editing, both time and the possibility of human error is reduced significantly (just how much depends on how well your logic is written!). The high degree of accuracy and uniformity in computer editing cannot be obtained in manual editing. In computer editing, range checks and within-record consistency checks can easily be made, between-record edits can be done, and unknown information can be allocated (imputed) automatically. If an allocation method is used, you should strive to retain as much of the original information as possible. With a computer editing and imputation system like CSBatch, erroneous data can be corrected immediately and reports can be generated of all errors found and all changes made.

  The programmer should plan and design computer edits to inspect the data and have the computer correct them according to specifications supplied by a subject-matter specialist. It would most likely be an extension of the original program written to find the errors—when you reach the point where there is an error, instead of (or in addition to) printing out an error message, you should now correct it with an appropriate value.

Actual methods of making corrections vary depending upon the item. In most instances, data items can be assigned valid codes with reasonable assurance that they are correct by using responses for other data items within the record, or in other records in the questionnaire. When recorded responses are missing, impossible, inconsistent, or unreasonable and cannot be determined from other responses in the same questionnaire, the hot deck technique can be used to assign entries.

Guidelines for Correcting Data

The purpose of editing is to make the data as representative of the real life situation as possible; do this by eliminating omissions and invalid entries, and by changing inconsistent entries. Below are some important principles that should be followed:

- The fewest number of changes should be made to the originally recorded data. You are only trying to make a record or questionnaire acceptable, not make it conform to what you think should be acceptable.

- If you must change a data value, do so only once. If you change a person’s age, then later find this age doesn’t work for another edit, then you didn’t write the original edit correctly. Go back and review the first edit.

- For certain items it may be acceptable to have a "not reported [NR]" or not stated [NS]" category. Thus, in case of an omission or an inconsistent, impossible, or unreasonable entry, a code for "NR" or "NS" can be assigned.

- Obvious inconsistencies among the entries should be eliminated.

- Providing corrected values for erroneous or missing items should be supplied by using other values as a guide; (for example, entries for the housing unit, person, or other persons in the household or comparable group), and always in accordance with specified procedures.

- Specifications for editing the questionnaire data should be developed at the same time as the questionnaire itself. Information to the computer programmer concerning what checks and imputations should be made in the data are provided through editing instructions or specifications. A subject-matter specialist, in collaboration with the computer specialist, should write editing instructions. The instructions describe the action to be taken on each data item. The editing instructions should be clear, concise, and unambiguous, since they serve as the basis for the CSPro editing program. Specifications for machine editing may be written instructions, decision tables, flowcharts, or pseudo-code. Pseudo-code (IF/ELSE logic) is recommended because it can be easily translated to CSPro code. Pseudo-code can be prepared using basic word-processing or text-editing software and can be easily modified.

  If the hot deck method of imputation is used, it is important that the edit specifications indicate where, during the processing, hot decks are to be updated, that is, at which points in the logic the data items can be considered valid.

Imputation

Imputation refers to the process of providing values for missing, erroneous, or inconsistent responses. For example, if a person’s sex code is invalid (i.e., out of range or otherwise unacceptable) or missing; then an appropriate response should be substituted.

You may decide that for missing data, you’d rather just keep it "missing" and publish your tables with an extra column (or row) for unknown values. This is a very cumbersome method, however, as the number of missing values will vary for each data item, and so the number of missing entries will vary from table to table, making the data difficult to analyze.

Inconsistent responses occur when a response yields an impossible situation with respect to another response. For example, if a 5-year-old female reports having children, either her age is wrong, or her fertility data are wrong (i.e., that section should be blank). This type of error must be corrected, as your users will place very little faith in the quality of your data if this type of condition becomes evident in the tabulations. Many users also do not look kindly on "missing" or unreported data. Of course, nothing can correct for bad data, and if you find that a significant amount of your data are bad (poorly designed questionnaire, inadequate field procedures, inattentive coders and keyers, etc), you may want to reconsider whether the data should be released at all.

Procedures have been developed to provide the missing information, thereby avoiding discrepancies and the need to determine percentages twice (with and without unknowns). For a detailed discussion on using imputation and the methods available to you, please refer to the United Nations Handbook on Population and Housing Census Edits.

Essentially, two methods of imputation are available: static and dynamic.

Static Imputation

Static imputation means providing a value from a pre-determined set of values. Suppose a person’s sex is missing or invalid (out of range). If we decide to change the response using static imputation, there are two basic methods to use: hard-coded or from a "cold-deck."

- **Hard-coded**
  Using our example above, we would programmatically set SEX to the value we think it should be. For example,

```
PROC GLOBAL
  toggle_sex = 1;

PROC SEX
if $ = notappl or not ($ in 1:2) then
  $ = toggle_sex;
  toggle_sex = 3 - toggle_sex;
endif;
```

What we've done above is a very primitive form of imputation. Essentially, when we encounter a bad value for sex, 50 percent of the time the variable SEX will be assigned the value "male", and 50 percent of the time the value "female". Note that no accommodation was made for other responses; for example, if fertility data were present, you might not wish to make this person "male." Or if this were an enumeration of a prison where the entire population is male, you would probably not want to be adding females to this group! So while this method can be used, you need to take into account other responses. We attempt to do this in our next method of static imputation, where we use a "cold-deck."

- **Cold deck**

With the "cold deck" approach, known information about individuals with similar characteristics (sex, age, relationship to household head, economic status, etc) is used to determine the "most appropriate" response to be used when some piece (or pieces) of related information is invalid.

For example, suppose a person's age is missing or invalid. We might have a table as follows, where the row indices represent the person's sex (1=male, 2=female), and the column indices refers to the person's relationship codes (1-5) (this table assumes that the relationship and sex codes have already been corrected):

		(head)	(spouse)	(child)	(other rel)	(non-rel)
		1	2	3	4	5
(M)	1	35	50	10	41	65
(F)	2	32	48	10	37	68

In the event a female child was found to have a missing age, she would be given the age of 10. If a female head of the household had a missing age, then her age would be given as 32. This method is acceptable if you do not need to use it often; that is, if very little of your data are missing or invalid. Also, if your population is fairly homogeneous (for example if you were correcting for religion and 90% of the population is Muslim), then this will not result in an unrealistic portrayal of your country.

However, if you find yourself referring to this table often, or you have a very diverse population where a few static values do not give an accurate portrayal, then your data will end up skewed. For these reasons (and others), dynamic imputation is the preferred method.

Dynamic Imputation (Hot Deck)

Dynamic imputation refers to the concept of using constantly changing values for your allocation routines. It is similar to static (cold-deck) imputation, except that instead of creating a table and assigning allocation values that remain fixed, the tables are continually updated with valid and consistent values taken from the population being edited.

For example, assume, for a given person, that the age, relationship, and sex codes appear correct and that consistency checks validate these items. You can use the values from this person to update your "'cold' deck", making it a "'hot' deck". Below is the table given as the "cold deck" example:

		(head)	(spouse)	(child)	(other rel)	(non-rel)

		1	2	3	4	5
(M)	1	35	50	10	41	65
(F)	2	32	48	10	37	68

If the person in question is a male 6-year-old child, the table can be updated with new information, giving the following:

		(head)	(spouse)	(child)	(other rel)	(non-rel)
		1	2	3	4	5
(M)	1	35	50	6	41	65
(F)	2	32	48	10	37	68

You would proceed in this way for every person in the household who had correct age, sex, and relationship values. Then, when you encounter a person with an invalid or missing age, you can extract from the table, using the sex and relationship of the person, a value for age. This value is more likely to be appropriate for the person than would be a purely random value. (The preceding example is clearly a simplification of the "hot deck" technique, which requires great care in constructing and updating the tables used for allocation.)

For a more detailed explanation of how to use hot-decks in your program, view "Use Hot Decks", refer to the CSPro hotdeck example folder or to the United Nations Handbook on Population and Housing Census Edits.

DeckArrays

Using "DeckArrays" in CSPro is a simplified way of working with hotdecks. It is important to understand the hotdeck process well before using DeckArrays because the functions getdeck and putdeck automatically handle recodes and thus hide much of the behavior of hotdecks from the programmer. DeckArrays eliminate the need for the programmer to recode variables to fit within the boundaries of an array as the recodes are processed automatically based on the values in a value set.

If using DeckArrays it is very easy to change the parameters of a hotdeck. For instance, if a subject matter specialist decides that a variable should be hotdecked based on five-year intervals of age but later changes the specification to be ten-year intervals, programming such a change is very quick if using DeckArrays, whereas with standard hotdecks reprogramming the recodes may be time-consuming.

Declaring a DeckArray

Declare a DeckArray in PROC GLOBAL as you would a normal array but instead of using numeric dimensions, specify the name of a value set. The size of the dimension will match the size of the value set. If you make changes to the value set, the size of the array will automatically reflect the changes.

```
array Age_HD_from_Sex(SEX_VS); // same as array Age_HD_From_Sex(2);
array Age_HD_from_Sex_Relationship(SEX_VS,RELATIONSHIP_VS);
```

To create a DeckArray, at least one of the dimensions must be a value set, but it is permissible that the other dimensions are declared with a numeric size. Because the program can only recode values for the dimensions based on value sets, it is necessary to specify the indices of any numeric dimensions when calling getdeck or putdeck.

```
array Age_HD_from_EA_Sex(200,SEX_VS); // same as array
Age_HD_From_EA_Sex(200,2);
```

Accounting for "Else" Values

Occasionally a programmer wants to create a hotdeck with a "leftover" row for any values not contained in the value set. Sometimes these are invalid values, other times the value set can be created so that this leftover, or "else," row consists of values valid for the census or survey. For instance, if a user wants to hotdeck a variable based on Christian sects, the programmer would create a value set with all the Christian sects, and then would create a DeckArray with a leftover row for all people who do not belong to the Christian sects identified. This is indicated by adding a (+) after the value set name.

```
array CEB_HD_Religion(RELIGION_CHRISTIAN_SECTS_VS(+));
```

Example 1

Simplified edit specification: If the head of household's age is missing or invalid, the value should be imputed with a hotdeck based on the head's sex and the spouse's age. If there is no spouse or no valid age for the spouse, hotdeck based on the head's sex and the household size, which will be assumed to have a maximum value of 10+.

```
PROC GLOBAL

array ageFromSpouseHotdeck(SEX_VS,AGE_VS);

numeric maxHHSize = 10;

array ageFromHHSizeHotdeck(10,SEX_VS); // household size is from 1 - 10+


PROC AGE

    universe RELATIONSHIP = 1; // we're only editing head of households here

    // assumption: by now RELATIONSHIP and SEX have been edited for the head
of household

    numeric ptrSpouse = seek(RELATIONSHIP = 2);
    numeric minSpouseAge = 12;

    if SEX = 2 then
        minSpouseAge = 15; // males cannot marry until 15
    endif;

    if AGE in 15:95 then  // the head of household has a valid age

        if ptrSpouse <> 0 and AGE(ptrSpouse) in minSpouseAge:95 then  // the
spouse has a valid age

            putdeck(ageFromSpouseHotdeck,AGE,SEX,AGE(ptrSpouse)); // update
the hotdeck

            // because SEX belongs to the current record being examined (the
head of household)
            // it can be left blank and CSPro will pick the correct SEX; the
age of the spouse must always
            // be passed because we are updating the hotdeck based on the age
of the spouse, not the age
```

```
            // of the head (the age of the head is invalid after all)

            putdeck(ageFromSpouseHotdeck,AGE, ,AGE(ptrSpouse)); // this is
the same as above

        endif;

        putdeck(ageFromHHSizeHotdeck,AGE,low(totocc(),maxHHSize),SEX);  // up
date the hotdeck based on HH size
        putdeck(ageFromHHSizeHotdeck,AGE,low(totocc(),maxHHSize));  // this
is the same as above


    else  // the head of household's age is invalid so we must impute it

        if ptrSpouse <> 0 and AGE(ptrSpouse) in minSpouseAge:95 then  // the
spouse has a valid age
            impute(AGE,getdeck(ageFromSpouseHotdeck, ,AGE(ptrSpouse)));

        else
            impute(AGE,getdeck(ageFromHHSizeHotdeck,low(totocc(),maxHHSize)))
;

        endif;


    endif;
```

Example 2

VS_SEX has two values: 1 (Male) and 2 (Female). VS_EDUC has three levels: 1 (No schooling), 2 (Primary schooling), 3 (Secondary schooling).

```
// Command:                                        // The hotdeck looks like
                                                   // this after the command:

array ageHotdeck(VS_SEX,VS_EDUC(+));               //  00   00   00   00
                                                   //  00   00   00   00

SEX = 1;   EDUC = 1;   AGE = 20;                   //  20   00   00   00
putdeck(ageHotdeck,AGE);                           //  00   00   00   00

SEX = 2;   EDUC = 3;   AGE = 50;                   //  20   00   00   00
putdeck(ageHotdeck,AGE);                           //  00   00   50   00

SEX = 1;   EDUC = 8;   AGE = 64;                   //  20   00   00   64
putdeck(ageHotdeck,AGE);                           //  00   00   50   00

SEX = 2;   EDUC = 3;   AGE = notappl;              //  20   00   00   64
AGE = getdeck(ageHotdeck); // AGE = 50             //  00   00   50   00

SEX = 2;   EDUC = 0;   AGE = 10;                   //  20   00   00   64
putdeck(ageHotdeck,AGE,,);                         //  00   00   50   10
```

```
SEX = 2;   EDUC = 0;   AGE = 11;                  //  20   00   00   64
putdeck(ageHotdeck,AGE,SEX,);                     //  00   00   50   11

SEX = 2;   EDUC = 0;   AGE = 12;                  //  20   00   00   64
putdeck(ageHotdeck,AGE,SEX);                      //  00   00   50   12

SEX = 2;   EDUC = 0;   AGE = 13;                  //  20   00   00   64
putdeck(ageHotdeck,AGE,,EDUC);                    //  00   00   50   13

SEX = 2;   EDUC = 0;   AGE = 14;                  //  20   00   00   64
putdeck(ageHotdeck,AGE,SEX,EDUC);                 //  00   00   50   14

                                                  //  20   00   28   64
putdeck(ageHotdeck,28,1,3);                       //  00   00   50   14

putdeck(ageHotdeck,28,0,3); // returns            //  20   00   28   64
// DEFAULT because 0 is not in VS_SEX             //  00   00   50   14
// and no (+) specified
                                                  //  20   00   28   64
AGE = getdeck(ageHotdeck,2,300); // AGE = 14       //  00   00   50   14
```

See also: Getdeck Function, Putdeck Function, DeckArray Leftover Rows

DeckArray Leftover Rows

When using unedited variables as parameters of your DeckArray, it may be useful to include a "leftover" (spillover) row for cases in which the variable is not valid. This functionality allows the user to simulate the use of many hotdecks using a single array. An example will best explain this functionality.

This following is a hotdeck for housing type, based on the source of drinking water (H8), the type of toilet (H9), and whether or not the household has electricity (H10).

```
array housingTypeHD(H8_VS1(+),H9_VS1(+),H10_VS1(+)) save;
```

At the point that housing type, H6, is edited, none of the dependent variables in the hotdeck have been edited, so the values can be blank or invalid. That is why, in the array definition, a (+) has been added after the value set names. The (+) gives an extra row for any value that is not in the value set.

When updating the hotdeck in cases where the housing type value is valid, adding a (+) after the DeckArray name will update the "leftover" rows.

```
putdeck(housingTypeHD(+),H6);
```

This means that when the hotdeck is called upon for an edit or imputation, having valid values for H8, H9, or H10 is not necessary to return a valid H6 value. This fundamentally makes the housing type DeckArray eight hotdecks combined into one. The table below explains what happens when executing a getdeck function call:

```
impute(H6,getdeck(housingTypeHD));
```

H8	H9	H10	Take the H6 value from the nearest neighbor with the same values for...
Valid	Valid	Valid	H8, H9, and H10

Valid	Valid	Invalid	H8 and H9
Valid	Invalid	Valid	H8 and H10
Valid	Invalid	Invalid	H8 only
Invalid	Valid	Valid	H9 and H10
Invalid	Valid	Invalid	H9 only
Invalid	Invalid	Valid	H10 only
Invalid	Invalid	Invalid	The previous valid household's value

Or in words:

- If electricity is blank, use a hotdeck based on water and toilet.
- If electricity and water are blank, use a hotdeck based on toilet.
- If water is blank, use a hotdeck based on electricity and toilet.
- If all three are blank, use a hotdeck based on the previous valid household.
- Etc.

See also: DeckArrays, Getdeck Function, Putdeck Function

Types of Edits in Batch Editing

Errors in data can be (roughly) categorized as problems in either structure or consistency. Problems with structure may require a different approach to correction than will problems with validity or consistency. For this reason, many users choose to implement a two-stage edit, where the data are first rendered structurally correct, and then passed through a second edit to ensure that all items are valid and consistent. However, it is expected that while the subject-matter specialists will have established rules for validity and consistency, the computer specialist may have equally important input into the definition of structural validity.

After keying or scanning your data, there will be errors in the data file. This is unavoidable, and will be a combination of human and computer error. It will therefore be necessary to correct the data by writing a series of edit routines (procedures) to systematically and consistently clean your data files.

The Batch Edit Designer module allows you to create and modify batch edit applications. A batch editing application contains logic which you can apply against one set of files to produce another set of files and reports. Batch editing applications can be used to gather information about a data file.

To create these edit routines, you will use CSPro to develop the batch editing application based on the dictionary (.dcf) that describes your data file(s). If you received this datafile from someone else and do not have a dictionary that describes it, you will need to create a dictionary before you are ready to develop programming logic for it. You use CSBatch to run the application. For small surveys and for testing applications, you can run CSBatch directly from CSPro, on the same computer. For large surveys and censuses, which require a production environment, you can transfer the application files to other computers and run CSBatch on them.

You can have the following run-time features in your batch editing application:

• **Check case/questionnaire structure:**
These checks ascertain that all records that should be present for a particular questionnaire [case] are supplied, and that no extra records are included.

• **Validate individual data items:**

These checks are designed to determine whether a response has a value that is inside or outside the valid limits for that response. Although these checks are normally performed during data entry, you can also perform them with CSBatch.

- **Test consistency between items:**
  In these checks, two or more responses in a questionnaire are compared for consistency. For example a male person reporting having any children is an inconsistency between the response to "sex" and the response to "children born". (Only females report "children born".) The responses being compared may be in the same record or in different records within a questionnaire.

- **Automatic modification:**
  When a census or other large survey is being processed, it would be unduly cumbersome to make most data corrections by visually examining the errors. CSPro provides the facility for not only finding incorrect or inconsistent data, but also making modification to the data. Needless to say, modifications are not always corrections. Moreover, any modifications to data that have been collected must be carefully thought out and monitored through CSPro's edit statistics reports.

- **Generate edit reports automatically:**
  CSPro generates comprehensive statistics about the edit tests carried out and the number of changes made to the data. The user may also create a customized report including or excluding any of the information generated by CSPro during the editing process.

- **Generate reports**
  You can write customize reports to a file.

- **Match files:**
  CSPro allow for matching two files and gathering information from both. The feature is useful, for example, when a file must be created which has a combination of data from two other files.

- **Modify data values:**
  During the editing process, values that are invalid, inconsistent, or otherwise unacceptable will need to be replaced with correct values.

- **Use arrays for imputations**
  Whether you choose to correct data using "hot-" or "cold-deck" methodology, the arrays are easily defined, accessed, and updated.

- **Generate imputation statistics**
  CSBatch will automatically keep track of changes, but users my choose to format reports to their specifications. By specifying a denominator variable, users may obtain rates of imputation, including rates for individual values [e.g., "male" or "female" when imputing the variable "Sex"].

- **Create additional variables**
  Recoded or composite variables may be created during the editing process. The only requirement is that space be allocated [via the data dictionary for the file] in the output record. The updated output file is automatically created when a file name is specified at the time of execution.

- **Reference multiple lookup files**
  An essentially unlimited number of secondary or auxiliary files may be attached to an application and used as reference or look-up files. The application may read from and write to any of these files.

How to ...

Manipulate Automatic Reports

During the testing and debugging stages of developing your application, you'll want to write out a lot of error messages to help find problem areas, and keep statistics on the number of times certain code blocks are being executed (or values are being imputed). You may begin to notice that you're using the same error message in several places. Rather than write out the message every time it's needed, you can "define" it once, and refer to it whenever needed. For example, suppose you have the following error message scattered throughout your code:

```
errmsg ("Current age after imputation is %d", age);
```

Why bother retyping it each time? You can simply define it once and reference it over and over. Simply do the following:

- In the Message View, select the **Message** tab. You will see one line that has been generated for you; it reads: `{Message code file generated by CSPro}.` Beneath this simply add your error message (we'll give it number "10"):

10 "Current age after imputation is %d"

- Then, whenever you want to use this message in your code, simply write (where '$' is a shorthand notation for the current **PROC**'s variable):

```
errmsg (10, $);
```

Besides simplifying your work, after you run your program, a convenient summary statistic will be generated for each user-defined error message, indicating how often it was used. A sample listing is shown below:

```
Number Freq Pct. Message text Denom
------ ---- ---- ------------ -----
10 24271 - "sex imputed to %d" -
```

See the errmsg function for a detailed explanation of all the options available to you.

Create a Specialized Report

Manipulate Automatic Reports showed how to add messages to the default report which is automatically generated after a run. This type of report may be useful for debugging, but in general, the subject-matter staff responsible for edit review will usually prefer a more "user-friendly" report - that is, one that presents the information in a format dictated by the users.

Such a "custom" report can be generated by using the write function. This command will put the information you want where you want it in your report. For example, for each questionnaire, you'll want to know the identifying ID values. If this were a population census, the case ID would likely be composed of levels of geography [Province, District, Village, EA, etc.] attached to a household identification. The "errmsg " command could display the Ids as follows:

```
Case [010117100110870031] has 12 messages (0 E / 0 W / 12U)
```

As you can see, this may be difficult for the non-programmer to decipher. But by using the **write** command, you can more clearly display this. One way would be to put the following **write** statements in the preproc of the first level (in this way it would only get written out once per questionnaire):

```
PROC QUEST
preproc
  write ("***************");
  write (" Province: %3d", PROVINCE);
```

```
write (" District: %3d", DISTRICT);
write (" Village : %3d", VILLAGE);
write (" EA : %3d", EA);
write ("***************");
write (" "); { blank line }
```

After the execution of the program, the .wrt [report] file would show the following (of course, actual values will vary depending on the questionnaire IDs):

```
***************
Province: 1
District: 7
Village : 30
EA : 4
***************
```

Additional **write** statements can be included in the batch edit program to generate a customized report.

Use Hot Decks

Hot decks in CSPro are implemented using arrays. First, you must declare the array under the PROC GLOBAL section. Then you need to set the initial values for the hot deck array. There are two ways to initialize and maintain arrays in CSPro: inline (in program logic), or using saved arrays.

Once the array has been declared and initialized, you need to add logic to check each occurrence of the variable you wish to impute using the hot deck. If the value of the variable is valid, update the hot deck by assigning this value to the appropriate cell in the array. If the value is invalid, set the value of the variable based on the corresponding cell in the hot deck. For example:

```
PROC AGE
```

`if AGE = notappl then`
`  impute(AGE, AgeSRDeck(SEX, RELATIONSHIP));`
 	
`else`
`  AgeSRDeck(SEX, RELATIONSHIP)= AGE`
`endif;`
	`{if the value for age is not valid}`
	`{assign the value from the hot deck based on sex and relationship}`
	
	`{update the value of the hot deck}`

When an age is missing during the data file's processing, we will use a value from the array AgeSRDeck; if the age is present, we will "refresh" the age for the person using the current sex and relationship codes as indices into the array.

If you wish to use hot decks in your application, refer to examples directory. For a more detailed explanation of what hot decks are, refer to the United Nations Handbook on Population and Housing Census Edits.

Initialize Hot Decks In Program Logic

In this technique, the hot deck is initialized in the body of the program before use. Hot deck values are updated during the program's execution but the values are not saved externally. Because the hot deck is initialized once, the instructions must be included in the preproc for the file so that they will be executed before reading of the data file begins.

The following is an example instructions that will initialize age values based on sex and relationship:

```
PROC GLOBAL
array AgeSRDeck (2,6); {declare array}

PROC HOTDECK01
preproc
  AgeSRDeck(1,1) = 27; {male head of HH} {initialize values}
  AgeSRDeck(1,2) = 30; {male spouse of head}
  AgeSRDeck(1,3) = 6; {male child of head}
  AgeSRDeck(1,4) = 58; {father of head}
  AgeSRDeck(1,5) = 11; {male other relative of head}
  AgeSRDeck(1,6) = 24; {male non-relative of head}
  AgeSRDeck(2,1) = 32; {female head of HH}
  AgeSRDeck(2,2) = 25; {female spouse of head}
  AgeSRDeck(2,3) = 8; {female child of head}
  AgeSRDeck(2,4) = 60; {mother of head}
  AgeSRDeck(2,5) = 10; {female other relative of head}
  AgeSRDeck(2,6) = 27; {female non-relative of head}

PROC AGE
```

`if AGE = notappl then`	`{if the value for age is not valid}`
`  impute(AGE, AGeSRDeck(SEX, RELATIONSHIP));`	`{assign the value from the hot deck based on sex and relationship}`
`else`	
`  AgeSRDeck(SEX, RELATIONSHIP)= AGE`	`{update the value of the hot deck}`
`endif;`	

For a complete example using use hot decks, refer to the examples provided in the *Examples\HotDeck* directory.

See also: Dynamic Imputation (Hot Deck), Use Hot Decks, Initialize Hot Decks Using Saved Arrays

Initialize Hot Decks Using Saved Arrays

In this technique, the hot deck is initialized by running the program twice and using the results of the first run to initialize the hot deck for the second run. The array is declared using the save keyword. No initialization of the array is done in the program logic. The program is run twice. During the first run of the program, the array has not been initialized so it is likely that some imputations from the hot deck will use these uninitialized values. The results (the output file) from the first run of the program are therefore discarded. However, during the first run of the program, the hot deck is filled with valid values. Since the array has been declared as a saved array, these values are written to the saved array file. When the program is run a second time, these values are read from the saved array file and used as initial values from the array. The advantage of this approach over initializing the array in the program logic is that all values used in the hot deck are taken from the data file.

The following example uses a saved array to initialize a hot deck.

```
PROC GLOBAL
  array AgeSRDeck (2,6) save; {declare array, initialize from file}

PROC AGE
```

`if AGE = notappl then`	`{if the value for age is not valid}`
`  impute(AGE, AgeSRDeck(SEX,`	

RELATIONSHIP));	{assign the value from the hot deck based on sex and relationship}

else
  AgeSRDeck(SEX, RELATIONSHIP)= AGE
endif;
	{update the value of the hot deck}

For a complete example using use hot decks, refer to the examples directory.

See also: Dynamic Imputation (Hot Deck), Use Hot Decks, Initialize Hot Decks In Program Logic, Array Statement

Interpret Reports

After specifying your file(s), a progress dialog bar will be displayed as CSBatch works its way through your data file, then TextViewer is launched and generate the following listing (.lst) report:

```
Application      C:\Program Files\CSPro 2.5\Examples\CSPro_Test.bch
Type             BATCH
Input Data       C:\Program Files\CSPro 2.5\Examples\Popstan Census 2000.dat
<Output>         C:\Program Files\CSPro 2.5\Examples\Popstan Census 2000.out

Date             May 04,2004
Start Time       10:24:48
End Time         10:24:51

CSPRO Process Summary
          +-------------------------------------------------------+
          |   29143 Records Read ( 100% of input file)            |
          |       0 Ignored (        0 unknown,        0 erased) |
          |   47063 Messages (      36 U,    9004 W,   38023 E) |
          +---------+--------------+--------------+-------------+
          |  Level  |  Input Case  |  Bad Struct  |  Level Post |
          +---------+--------------+--------------+-------------+
          |     1   |        4872  |           0  |        4872 |
          +---------+--------------+--------------+-------------+



Process Messages

*** Case [010705802820460191] has 12 messages (8 E / 2 W / 2U)
    W 88870 Value '01' out of range - check P16_IND(1)
    W 88870 Value '01' out of range - check P16_IND(5)
    U   -69 Household tenure is vacant, but there are 5 person records.
    E 88203 6 inconsistent fields detected following a 'skip to H13_PERSONS'
          command in Var H06_TENURE PostProc
    E 88212 ... H07_RENT should be blank (currently '000')
    E 88212 ... H08_TOILET should be blank (currently '9')
    E 88212 ... H09_BATH should be blank (currently '5')
    E 88212 ... H10_WATER should be blank (currently '6')
    E 88212 ... H11_LIGHT should be blank (currently '6')
    E 88212 ... H12_FUEL should be blank (currently '7')
    U   -84 Number of persons is 0, number of person records is 5.
    E 88230 Unexpected 'reenter' command reached in Var H13_PERSONS PostProc
```

The "CSPro Process Summary" gives you information about the records and cases. In this example:

• 29143 records were read from the input file, which represent 100 % of the input file

• "Ignored" is the sum of "Unknown" and "Erased" cases.

• "Unknown" represents number of cases with invalid record type code.

• "Erased" represents number of records with a "~" character

• The list report contains 47063 messages of which: 36 messages were defined by the user; 9004 are warning messages generated by the system; and 38023 are error messages generated by the system

• **"**Level 1" indicates that the following statistics are for Level 1 only. If the application had more than one level, the statistics will be displayed for each level.

• The input data file included 4872 cases (questionnaires) of which: there were "0" case with bad structure, meaning that no required records were missing; and 4872 cases were written to the output file ("Level Post").

The "Process Messages" gives you information on each case:

• *** Case (010705802820460191) has 12 messages (8 E/ 2 W/ 2 U)
Indicates that questionnaire 010705802820460191 (codes correspond to the ID items in the Dictionary) has 12 messages: 8 error messages generated by the system; 2 warning messages generated by the system; and 2 error messages defined by the user.

• W 88870 Value ‘01’ out of range – check P16_IND(1)
Indicates that field P16_IND in first person (1) has a value "01" which is out of range because it has not been declared in the Dictionary. This is Warning 88870, generated by the system.

• U -69 Household tenure is vacant, but there are 5 person records.
Message defined by the user in line 69 of the program

• E 88212 … H07_RENT should be blank (currently ‘000’)
Indicates that field H07_RENT has a value ‘000’ but it should be blank. (This is a vacant unit)
This is Error 88212, generated by the system.

If a write (.wrt) or frequency (.frq) file was generated, then they will also be loaded into TextViewer for display; to rotate between the various files, select the "Window" option from TextViewer, and choose from among the files listed at the bottom. If TextViewer was already running when you launched your application, it will be refreshed with the latest run results.

See the explanation given under Run an Application

Run Production Batch Edits

You can customize CSBatch’s behavior by creating a PFF file. You can then use the PFF file as a command line parameter for CSBatch.exe. For example, if you name your PFF file "MyEdits.pff", then you can launch CSBatch with this application by invoking:

```
C:\Program Files\CSPro 6.0\CSBatch.exe MyEdits.pff
```

This assumes that CSBatch was installed in the default directory. Your PFF file **must** have a ".pff" extension.

You can create a PFF file in one of two ways:

- Create it yourself using a text editor (such as Notepad or Wordpad)

- Simply run CSBatch once, and a PFF file will be automatically created for you—it will be placed in the same folder as your batch application, and it will have the same name as your application, but with a ".pff" extension instead of ".bch". For example, if your batch application was named "MyEdits.bch", the system-generated PFF would be called "MyEdits.pff".

The following section shows the options available to you in a CSBatch PFF file. Please note that a PFF file is not case sensitive, so you may use any combination of upper and lower case text.

```
[Run Information]
Version=CSPro 6.0
AppType=Batch

[Files]
Application=.\MyEdit.bch
InputData=.\p12d05.dat
OutputData=.\p12d05e.dat
Listing=.\MyEdit.lst
WriteData=.\ViewMe.dat
ImputeFreqs=.\MyEdit.freq.lst

[ExternalFiles]
LOOKUP_DCF=.\Prov12.lup

[Parameters]
ViewListing=Always
ViewResults=Yes
ListingWidth=80
MessageWrap=No
ErrmsgOverride=No
Parameter=your choice
```

The [Run Information] block is required and must appear exactly as shown in the example above.

The [Files] block is required and defines all files used in the batch run. A description of the files is as follows:

- Application = the batch edit application you created

- InputData = the data file against which the batch edit program will run; this file will not be modified during the run

- OutputData = the revised/corrected input data file will be saved as this file

- Listing = a report of the batch operation

- WriteData = if there is one or more write function in your batch program, the text of these statements will be written here

- ImputeFreqs = if you have any impute statements in your batch program, the results of these statements will be written here

If the [**ExternalFiles**] block is present, it means that a second [or more] dictionary was linked to the data entry application. In the example above, "LOOKUP_DCF" is the internal (unique) dictionary name, and "Prov12.lup" is the name of the data file that contains the lookup codes. If there is a second dictionary linked to your application and you fail to name it in your PFF file, the operator will be prompted to provide it.

The [**Parameters**] block is optional. This section defines parameters for the batch run.

ViewListing determines whether you see the batch run report. If this entry is missing or set to "ViewListing=Always", then you will see the generated report. Other available options are "OnError", in which case you will see the report listing only if an error occurred during the run, or "Never", in which case you will never be shown the generated report.

ViewResults determines whether or not the write or impute file(s) are displayed with TextViewer at the end of the run. The available choices are "Yes" or "No." If the "ViewResults=" entry is missing, the resultant data file(s) will be displayed by default. For more information on these files, see Run a Batch Edit Application.

ListingWidth allows you to control the number of characters outputted to the listing file before forcing the start of a new line. This is set to 80 characters by default. If your screen resolution and/or printer permits, it may be useful to increase the width of the listing file.

MessageWrap determines whether or not summary messages displayed at the end of a listing file will be outputted on several lines when they are too long to fit on one line. The default option is no, in which case the messages are truncated so that they fit on one line.

ErrmsgOverride allows you to override the default behavior of the errmsg function. The override only affects errmsg functions used in the code without a case or summary specifier. "No" maintains the default behavior (displaying the messages for each case, as well as in the summary). "Summary" only displays the message in the summary. "Case" displays the message for each case but not in the summary.

Parameter allows you to pass in an alpha-numeric string to your program. The parameter can be any length, although the alphanumeric string that retrieves the value in your program (via the sysparm function) must be large enough to accommodate it. Once the parameter string is retrieved, it can be parsed for further usage.

Steps in Developing a Batch Editing Program

General Issues

This guide has described the CSPro language and has presented examples of various types of edits. However, the job of writing and testing a CSPro program to perform many edits is still a large one. Because of the structure of the CSPro language, this job can be distributed among several people if it is well coordinated.

Before coding begins, a complete set of edit specifications and a formatted listing of the data diction ary should be available. A team leader should be appointed; this person should have a thorough understanding of the CSPro language. The leader should develop the naming conventions and standards that are to be followed by other members of the team. During the course of program development, questions will arise about the edit specifications, and it is important to involve subject-matter specialists to resolve these issues.

This section outlines a step-by-step approach to developing a CSPro program. It assumes that a data dictionary has been developed. If this is not the case, then the first step is to build the data dictionary using the Data Dictionary module and to check the dictionary carefully against original specifications.

An editing system may contain more than one CSPro program. It is not unusual to have at least two CSPro programs in the edit system: the first is used to find and report structure errors which must be resolved before the data moves on to the next stage (see Section 7.2.2). These include missing records, duplicate records, or other incompatibilities with requirements which are considered essential to the processing. The second program is the more traditional value-validation and consistency edits.

Review Edit Specifications

In order to determine the total editing task, the user should understand what edits are to be performed. Are the edits fairly simple or mostly complex? Are the edits merely to find and report errors in the data file, or will the program correct the errors as well? Is a non-response value being allowed for any or all data items? Is a look-up file going to be needed? Are there any auxiliary files to be produced? What types of reports will be needed and how will they be organized?

The three types of edit statistics reports are:

- **Case**
  Output listing shows case by case

- **Summary**
  Output listing shows (1) a summary of the number of times each error message was generated, and (2) the number or errors as a percentage of total cases in which the edit test was invoked.

- **Frequency**
  This report is generated when at least one **impute** statement is coded in the program. It assigns a value to a data item and logs the frequency of assignments.

Typically, during the testing phase, the report by case is very useful, because it permits detailed examination of the effects of the logic coded. After testing, it is usually not used because of the volume of reports generated.

Define Coding Standards

Without a good set of coding standards, it will be difficult to incorporate the various subroutines into one program. Consistent naming standards and coding techniques must be considered; specific suggestions include the following:

- Names for variables to be defined in the program should have unique prefixes. The standard could be that variables needed for editing LINE-NUMBER be N01 through N10; those needed for editing AGE would be N11 through N20, and so forth.

- Names for hot-deck arrays should also be standardized. For example, an array for hot decking age by marital status, relationship, and esx would be defined as HD-AGE-SC-RL-SX. The variable whose values are in the array (in this case, "age") should be named immediately after the prefix (in this case, "HD").

- It may be necessary for a procedure to call a function, and standards should be defined for naming them.

- Standards for coding **impute** statements should be set. If it is desirable to relate an imputation to the original edit specifications (and this is usually helpful), then case numbers should be assigned to every imputation case in the edit specifications document. The case number could be included in the **message** phrase of the **impute** statements as follows:

Impute Relationship = HD-REL-EDUC-AGE (educ,age) **msg** "Relationship imputed - Case 2A"

- Standards for code indentation should be established so the program is easily readable. All the examples in this manual are indented consistently and can be used as guides for establishing a standard.

- Column labels as comments appear in this manual with the discussion of hot decking.

- File naming conventions also should be established.

Code Edits of Individual Data Items

Edit specifications need not be assigned to team members in any specific order. It is recommended that the simpler edits be coded first, especially if the coding is being done by inexperienced program mers. In this manner, they will gain experience and will later be able to handle more complex edits.

Team members should code the edit specifications and review the code with the team leader. The team leader should then incorporate the new code into the CSPro program. All syntax errors should be corrected before more code is added to the main program. It is important to maintain earlier versions of the program as insurance against problems that may creep in.

Develop Comprehensive Test File

A comprehensive test file should be created; it can be made at any time after the edit specifications are finalized, and it can be done by anyone familiar with the data file. Each member of the team should create records to test all paths of his/her subroutines. These records can be combined into one test file in a logical sequence.

Test CSPro Program

The CSPro program should be thoroughly tested using the file with "invented" data. Earlier in this guide is a discussion of the **errmsg** statement and a technique for listing records before and after imputation. This technique (or a similar one) should be used to verify that imputations are being done properly and under the right circumstances. Each record that contains imputed items should be examined closely. Records containing no imputed items also should be checked to ensure that no imputations were needed.

When hot-deck imputations are used to correct invalid or inconsistent items, it is particularly important to ensure that the hot deck arrays are being updated under the appropriate circumstances and with valid values. This is the most common error in program construction, and one way to avoid it is by requiring that the edit specifications explicitly indicate where, in the sequence of actions, hot decks should be updated.

The output file produced by the **Run** program step should be "clean", assuming imputations were performed to correct all invalid and inconsistent data. To verify that the CSPro program does not contain contradictory logic, that it corrects errors properly, and that it does not introduce any new errors, the **Run** program step should be rerun with the "clean" [output] data file as input. In this rerun, no errors should be found and the edit statistics should reflect this. If errors are found, then the logic of the CSPro program must be corrected.

Re-Test with Live Data

The CSPro program should be tested using real data with perhaps 2,000 to 3,000 records. The edit statistics should be examined closely. The number of imputations should be a very small percentage of records. If a large number of imputations are being done for any particular edit test, the code should be examined, and the imputations accounted for before proceeding. The edit statistics should be examined

by subject-matter specialists also, so they can determine whether or not the rate of imputation is too high. If appropriate coding techniques are used with the **impute** statements, the imputations can easily be related back to the edit specification document.

Remember: Just because the CSPro program is free of syntax and logic errors is no guarantee that all the required editing is being done and is being done correctly! An error-free CSPro program cannot compensate for incomplete edit specifications. Among the test most commonly omitted tests when creating edit speci fi ca tions are: making sure there is one and only one head of household, and making sure the head of household is of an acceptable age.

Begin Production Editing

If no changes to the CSPro program are needed, only the **Run** program step needs to be executed for each batch of data to be edited. If the data dictionary is changed, then you must re-compile the program before running it again. During production processing, edit statistics for each run should be checked to ensure that the number of errors and imputations are within the normal range.

As soon as enough batches have been edited to form a larger geographic area (perhaps all enumeration areas within a district), tabulations should be run at the higher geographic level (in this case, district) to see if any inconsistencies are apparent. If so, it will be necessary to pause and discover the reason for the inconsistencies. This may involve modifying and re-testing the CSPro program until the data are producing satisfactory results. Each time the CSPro program is modified, all previously-edited batches must be re-edited [always starting with the same original data files!] so that final data will all have been edited and corrected following the same logic.

Tabulation Applications

Introduction to Tabulation

Tabulation applications in CSPro allow you to tabulate data quickly and easily, producing basic frequency distributions and complex cross-tabulations from one or more data files. The only requirements are that:

• The data to be tabulated is in text format, with all items in fixed positions

• All the data files to be used in a single run share the same format

• There be a data dictionary describing the data file.

Each of these requirements is easily met: if the data are currently in a proprietary format (that is, within a database, spreadsheet, or other package), they must be exported to a text format (sometimes called "printer" format); if the data dictionary does not yet exist, CSPro will ask you to create the dictionary while you are creating the tabulation application.

When you create your application, you can use an existing data dictionary or you may create one as you create the application. You can produce most tables by way of the menu-driven, point-and-click, drag-and-drop interface, i.e. without using the CSPro programming language. The procedural language is available for the most complex kinds of tables.

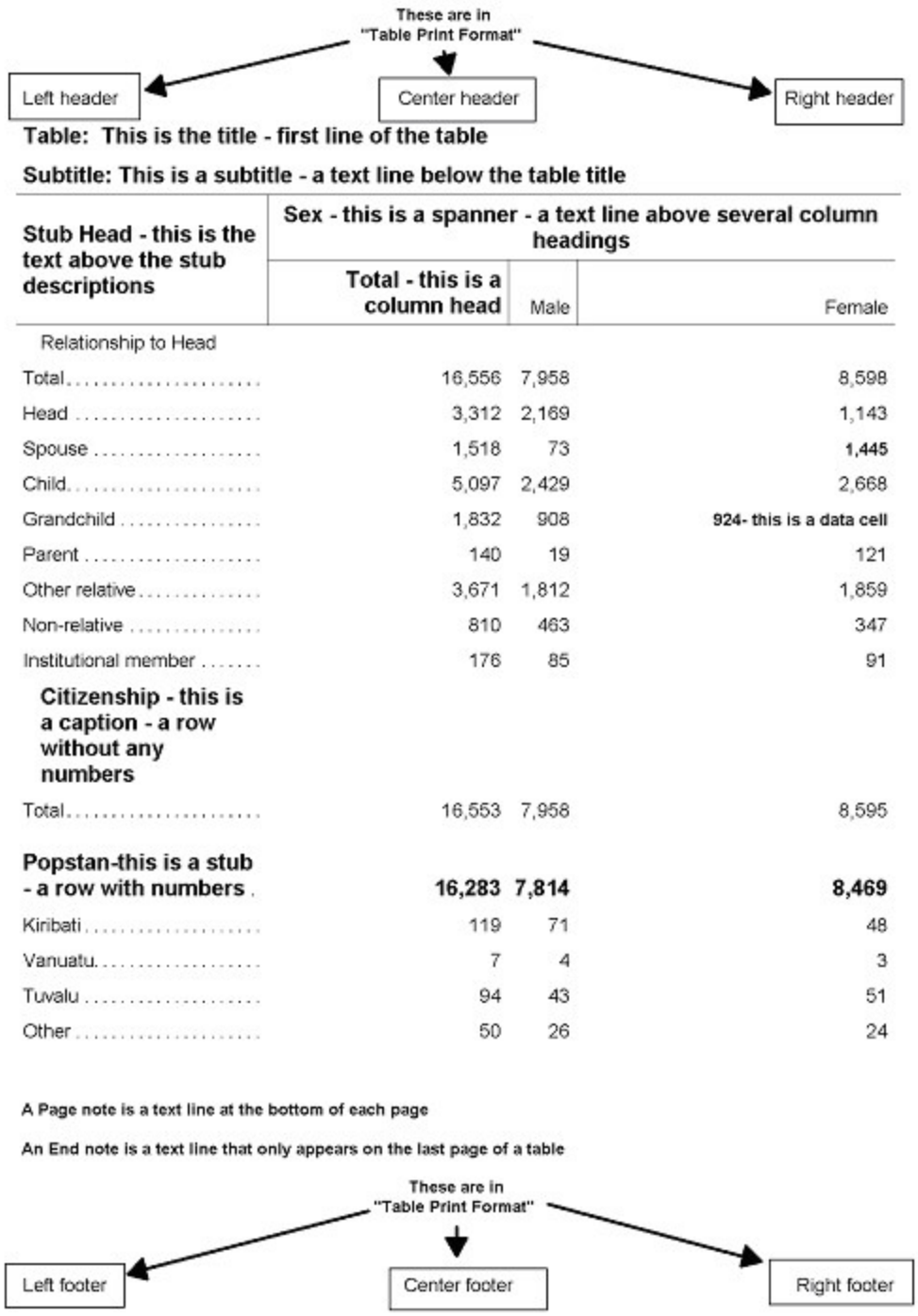
This section contains the following information:

Parts of a Table
Common Uses of Tabulation
Capabilities of Tabulation
Creating Tables
Formatting Tables

Creating Tables by Geographic Area
Printing Tables
Tabulation Preferences
Saving and Copying Table Data
Table Post Calculation
Run Production Tablulations
Advanced Table Topics
Table Tips and Tricks

Parts of a Table

Shown below are the terms used by the CSPro software to reference the parts of a table.



Design/Data View Contents

- **Title:** Top line of table that usually describes its contents.
- **Subtitle:** An optional informational title below the (main) title.
- **Spanner:** An informational text box ‘spanning’ several columns.
- **Column Head:** Text box describing the contents of a column of data.
- **Stub Head:** Text box describing the general contents of the stubs (rows).
- **Stub:** Text line describing the contents of a row of data.
- **Caption:** Text lines interspersed with Stub lines but which do not have data associated with them.
- **Page Note:** A text line that appears at the bottom of each page of the table [a footnote on each page]
- **End Note:** A text line that appears at the bottom of the LAST page of a table [a footnote at end of table].
- **Area Caption [Not shown]:** When the area option is used for tabulations, this is a caption line that will be replaced by the text in the ‘area name file’.
- **Boxhead:** This is the union of the Stub Heads, the Spanners, and the Column Heads

Additional Contents for Print View

- **Left, Center, Right Header:** Optional text printed at the top left, center, and/or right of each printed page (above the title).
- **Left, Center, Right Footer:** Optional text printed at the bottom left, center, and/or right of each printed page (below any footnotes).
- **Secondary Stub Head [Not shown]:** When stub text is displayed on both sides [left and right] of the print table, this text is displayed.

Additional Table Terms

- **Tally:** Characteristics associated with the numbers that appear in a table. These include percents, averages, medians, etc.
- **Subtable:** When more than one set of categories is crossed with another characteristic, then each is considered a subtable. A table with only one set of stubs and column headings can also be considered as a subtable. (See below: a comma in the table title separates subtable item labels.). A subtable is a portion of a larger table that can have its own settings for universe, weights, value tallied and unit of tabulation different from other subtables in the same table. When hidden parts in the view menu is enabled, a colored box will be drawn around each subtable.

Table 1. Sex, Citizenship by Birthplace

	Birthplace		
	Total	Popstan Provinces	Countries
Sex		SUBTABLE	
Total	16,528	16,204	324
Male	7,946	7,796	150
Female	8,582	8,408	174
Citizenship		SUBTABLE	
Total	16,528	16,204	324
Popstan	16,259	16,079	180
Kiribati	118	117	1
Vanuatu	7	5	2
Tuvalu	94	3	91
Other	50	-	50

Common Uses of Tabulation Applications

Tabulation applications may be used for a number of different purposes. The statistical organization may generate tables for dissemination to outside users of the data, and the same organization may also generate tables to be used internally, as a tool for control and analysis of the data themselves. Some other common uses of tabulation applications are:

- **Design Table Stub Groups**
  In the initial stages of designing the tabulations that will eventually be generated from the final edited data, it is important to optimize the category breakdowns. Certain variables present few problems in this respect, either because the number of categories is inherently limited (e.g., sex) or the number of categories is limited to those pre-coded on the questionnaire (e.g., relationship). Other items, how ever, may require that a choice be made (e.g, age, occupation, educational levels, etc.) in grouping the available values. Tabulation applications may be used to examine the effects of various groupings before the final design is approved.

- **Examine Quality of Data and Design Edits**
  The design of proper consistency and validity edits is a difficult task, but the use of tabulation applications can help in determining the accuracy and completeness of the edit program. In fact, effective quality control requires that users verify and document the results of the edits. By using tabulation applications to generate the same tables from the pre- and post-edit data, the user can compare the cell values to ensure that all anomalies have been corrected and that the edits have not distorted the distribution of data values across categories.

- **Test Edits**
  Although CSBatch automatically produces edit statistics which report the extent of the editing it performed on a file, the subject-matter specialist and/or computer programmer has the option of using tabulation applications to analyze a data file before and after editing to examine the extent of the changes to a file, and to review relationships between data items in the edited file. Tabulation applications can be used to determine whether the edits did the job they were intended to do.

- **Test Tables**
  Tabulations that are destined for publication and for distribution to outside users must be carefully verified to ensure that the values presented are without error. Tabulation applications are currently designed for use as a custom tabulation package but it can also be useful as a means of comparing the results produced by other tabulation software. In this case, the relative rigidity and transparency of tabulation specifications means that there will be virtually no errors attributable to programming faults; if differences are encountered between the tables produced by CSPro and tables produced by other software, the tables produced by CSPro are likely to be closer to the "true" counts. In any case, tabulation applications can be an effective tool to create publication-quality tabulations or for cross-checking numbers produced by other software.

Capabilities of Tabulation

Cross Tabulations

Cross tabulations can display relationships between two or more data items. You may have both dependent and independent variables in each dimension [row and column].

Table 1. Mother living and Literacy by Urban/Rural and Sex

Mother Alive and Literacy	Urban/Rural								
	Total			Urban			Rural		
	Sex			Sex			Sex		
	Total	Male	Female	Total	Male	Female	Total	Male	Female
All persons									
Literacy									
Total	16,528	7,946	8,582	9,707	4,595	5,112	6,821	3,351	3,470
Literate	13,890	6,836	7,054	8,830	4,229	4,601	5,060	2,607	2,453
Illiterate	2,638	1,110	1,528	877	366	511	1,761	744	1,017
Mother living									
Literacy									
Total	12,064	5,834	6,230	7,202	3,435	3,767	4,862	2,399	2,463
Literate	10,481	5,106	5,375	6,690	3,189	3,501	3,791	1,917	1,874
Illiterate	1,583	728	855	512	246	266	1,071	482	589
Mother deceased									
Literacy									
Total	4,464	2,112	2,352	2,505	1,160	1,345	1,959	952	1,007
Literate	3,409	1,730	1,679	2,140	1,040	1,100	1,269	690	579
Illiterate	1,055	382	673	365	120	245	690	262	428

For example, this table has Urban/Rural and Mother Alive as independent variables; Sex and Literacy as dependent variables. In addition, this table could be produced by ‘area’; usually this is a geographic item but it could be any item that meets certain conditions. The ‘area’ item is basically an extra tabulation variable.

The user can elect to display tabulations in terms of actual counts or as percentages of the total, and has the option of counting special and/or undefined values. The counts in tabulations may be unweighted or weighted. Users can produce a table on a subset of a file using the universe option, or select an item or numeric value to be used as a weighting factor during tabulation.

The selection of an item as a row or column variable will affect the table layout.

See also: Create a Table

Tabulate Counts or Percents

In addition to the quantitative numbers generated in your tabulations, you can choose to show the distribution of values for an item as a percentage of either row or column totals, or as a percentage of the table total. You can also choose to show values only as percentages and suppress the actual numbers. This is useful when you have a small data set and prefer not to display values which might identify individual cases.

See also: Include Percents

Tabulate Values and/or Weights

Tabulation allows the use of a data item or a constant value as a weighting factor during tabulation. This is particularly useful in the case of a survey, where the weight assigned to each case or observation in the sample must be taken into account in order to produce numbers representative of the whole. If no weight value is specified, weight is assumed to be "1".

In the same fashion, values (rather than counts) may be tabulated; like weights, values are numeric data items or constants with or without decimal positions. When a value is speci fied for tabulation, the effect is that of cumulative addition of the specified value into the cell at the intersection of the row and column coordinates. If you leave the value item blank, a value of 1 will be added into the appropriate cell during tabulation. Tabulating values can be used, for example, when a variable such as "number of children born" has been collected for each woman in the population and you wish to tally the total number of children born rather than tallying the number of women with a particular number of children. In this case you would use the "number of children" variable as the value to tally so that for each woman, the number of children born to that woman would be added to the appropriate cell.

If both value and weight are specified for a given table, the specified value is first multiplied by the specified weight and the **product** of this multiplication is then added to the cell in question. This feature is useful in the case where the unit of observation has a weight, and one of the data items to be tabulated is a quantitative value. An example might be a fertility survey, where each female of child-bearing age carries a sampling weight, and where some of the data items might be "Number of children ever born," "Number of children surviving," etc. Tabulation applications easily permits the accumulation of weighted values to provide the total number of children ever born for all females in the sample.

If the weight or value includes decimals, the decimal may be explicit or implicit, but the dictionary definition should reflect the correct situation. The weight may be assigned to just the current subtable, the entire table or all the tables currently defined in the application.
See also: Add Weights to a Table, Tabulate Values Instead of Frequencies

Produce Summary Statistics

Tabulation applications allow the inclusion of summary statistics in the tables. Available statistics include Mean, Mode, Median, Minimum value, Maximum value, Standard Deviation, and Variance, as well as Percentiles (N-tiles). Most of these give information about the distribution of values for the data item.

These do not make sense for some items so you should only use them if you have some understanding of their meaning and purpose. For example, none of these measures have any meaning for an item such as sex.

Mean, Mode and **Median** are measures of central tendency. These are measures of the location of the middle or the center of a distribution. The definition of "middle" or "center" is purposely left somewhat vague so that the term "central tendency" can refer to a wide variety of measures. The mean is the most commonly used measure of central tendency. Mean is the average value of the item, mode is the most common value occurring in the data, and median is the middle value (half of the values occur above this value and half below, e.g., 50th percentile).

Minimum and **Maximum** are the lowest values and highest values, respectively, found for the data item.

Standard deviation is a statistic that tells you how tightly all the various values are clustered around the mean in a set of data. This measure is the square root of the variance.

Variance is a measure of how spread out a distribution is. It is computed as the average squared deviation of each number from its mean.

N-tiles are values that divide the data values into N groups, each of which contain 1/N of the values. If N is two, then this measure is equivalent to a median.

See Also: Tally Attributes for a Variable

Restrict a Universe

When you define a universe, CSPro will only tabulate data records in the questionnaires that meet the conditions stipulated by you. The "universe" specification acts as a filter, as the tables produced use only a subset of the data file's records. Therefore, values in the table may be lower than they would be with no universe specified, since the universe restricts the data available for tabulation. Note that each table has its own universe definition, but that a given universe specification may be extended to all tables in an application.

See also: Define a Universe

Format Tables for Printing

The layout and presentation of the text and numbers in a table is generally referred to as the "Table Format". CSPro, through various menus, gives the user almost complete control over all aspects of the Table Format. These include the text font, its colors, its indentation, its alignment, its borders and the text itself. Other formats control the presentation of numbers, headers and footers on the printed page, and margins.

See Also: Formats for Parts of a Table, Formatting Row, Column, or Cell Data, Formats for a Table, Formats for a Tabulation Application, Formats for Printing

Load and Save Formatting Preferences

CSPro has a predefined set of formatting standards built into the system; these would be considered ‘Defaults’. These include the text font, its colors, its indentation, its alignment, etc. Users have the option of creating their own defaults called ‘Preferences’. These Preferences can be saved and reused for other applications. They may also be shared among multiple users working on the same publications in order to keep consistent formatting for all tables produced by that team.

See Also: Preferences and Default Formats, Modifying Preferences, Loading and Saving Preferences

Produce Tables by Area

The "Area Processing" feature groups table data according to areas that you define. For example, if you are producing tables for industries, you may want to output the data according to industry type (Agriculture, Mining, Fishing, etc.).

One of the most common uses of area processing is for geography. Whatever items you choose to use for your area, they must be part of the questionnaire identification or items on a singly- occurring record. Area processing is in addition to any row/column items selected.

To use this feature, you must first create an area names file, which defines the levels of the area and assigns text names to the numeric code. The Area Processing applies to all tables in your application.

See also: Area Processing, Create an Area Names File, Area IDs Dialog Box, Area Names File

Save Tabulations in Different Formats

CSPro lets you select an entire table or parts of a table. The tables can be saved in several formats:

- **CSPro Tables (.tbw):**
  Lets you save tables so they can be used later by the CSPro Table Viewer.

- **Rich Text Format (.rtf):**
  Lets you save your tables so they can be used later by word processors such as Word or WordPerfect. You can open the (*.rtf) in your word processor, and the table will appear in the word processor's table format.

- **HTML files (.htm):**
  Lets you save your tables so they can be later incorporated into Internet applications in table format.

- **Tab-delimited (.other):**
  Lets you save your tables so they can be used later by a **spreadsheet** such as *Excel*, *Quattro Pro*, or *Lotus 1 2 3*. You can open the file in your spreadsheet, and the table will appear as a matrix of cells with columns lined up as you would expect.

See also: Save Tables, Copy Table to Other Formats

Copy Table to Other Formats

In addition to saving a table file in a particular format (as described in the preceding section), the user can use standard Windows copy-and-paste functions to copy all or part of a table directly into a word processor or spreadsheet. If only a portion of a table [i.e, not all rows and/or columns] is highlighted, CSPro will automatically include stub and column header text for only the affected rows/columns.

See also: Save Tabulations in Different Formats, Select and Copy Table Data to Other Applications

Copy and Paste Table Specification

CSPro lets you copy the entire table specification (before you run it against a data file) to the clipboard. The table specification includes the variables in the table, tally attributes and formatting information. You can then paste the table into the current tabulation application or into a different tabulation application.

This feature is very useful for moving tables from one application to another. If you have many tables to produce, you can divide them into several applications each with a small number of tables, then use copy

and paste table specification to assemble them into a single application at the end. If you do this, you must be careful to use the same data dictionary in each application.

This feature is also useful if you need to add a table in the same application which is very similar to an existing table.

To copy or paste a table specification:

- **From the Tabulation Tree tab**
  Right-click over any of the tree entries. A pop-up dialog box will appear. Select "Copy Table Spec" or "Paste Table Spec".

- **From the Tabulation Menu**
  Select Edit, then "Copy Table Spec" or "Paste Table Spec".

- **From the table itself**
  Right-click anywhere over the cells of a table. A pop-up dialog box will appear. Select "Copy Table Spec" or "Paste Table Spec".

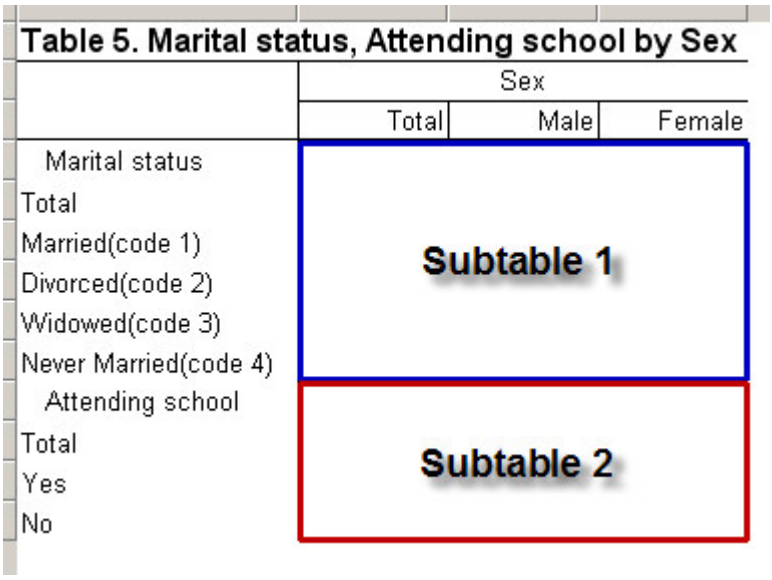
Map Results by Geographic Area

If a digitized map of the country is available to the user, CSPro can be used to produce thematic maps from one or more values in a tabulation. In many cases, such maps are more effective in communicating information, particularly to the lay public, than are tables. For example, if the table shows the geographic distribution of a particular resource [for example, persons holding post-secondary degrees by sex], these values can be imported into the Map Viewer; the thematic map generated can often dramatically illustrate existing conditions, particularly where the distribution of the resource is uneven.

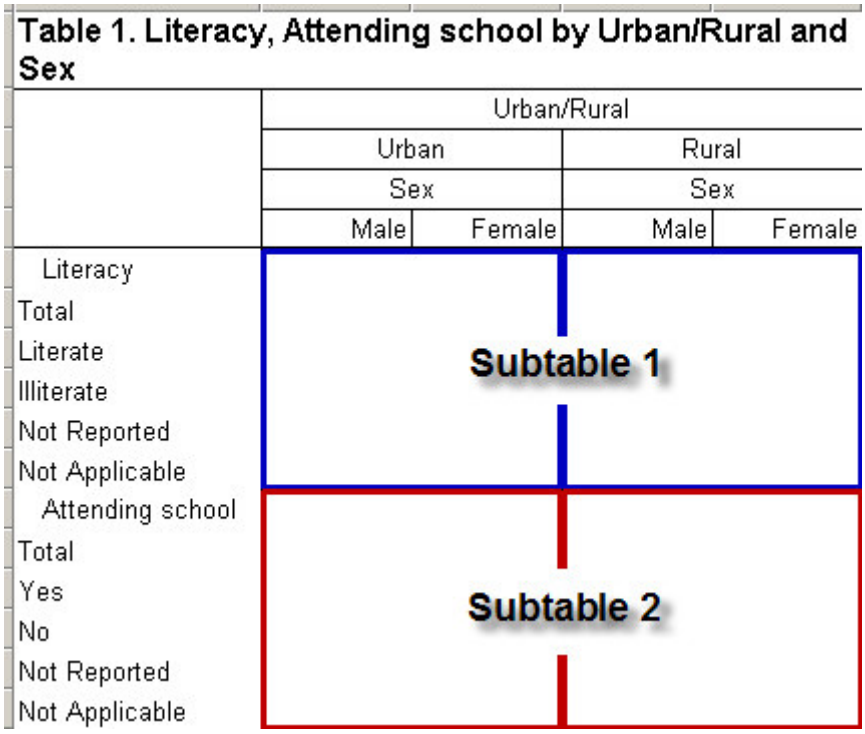
See also: Create a Thematic Map of Results

Create Multiple Subtables

Subtables are themselves tabulations nested within a larger tabulation -- hence the prefix ‘sub’, meaning part of a larger object. Each subtable can be made up of one to four variables (two for row and two for column). Each subtable has one independent variable in the rows and one independent variable in the columns. It may optionally have one dependent variable in the rows and one dependent variable in the columns. In the first example, each subtable has only two variables, one row and one column.



In the second example, there are three variables in each subtable, one row variable plus two column variables (the independent variable Urban/Rural and the dependent variable Sex).



Subtables are created automatically as the user builds the table. Every time an independent variable is dropped onto the rows or columns of the table, new subtables are created as appropriate.

Subtables are designated as part of the larger tabulation using colored outlines when viewing hidden parts. Each subtable has the same properties available to the entire tabulation, i.e., universe, weight, value tallied and unit of tabulation. These properties may be set differently for each subtable within a table.

See Also: Add a Variable to a Tabulation, Using Subtables

Change Unit of Tabulation

For most tabulations the "unit of tabulation" is easily determined. For example, the unit of tabulation for a table of Age by Sex would be the person record. A table of Householder Tenure and Number of Rooms would be tallied for each House record. In cases where all items in the table are contained on the same type of record, the unit of tabulation could only be that record itself. At times, tables require inputs from two different record types (or two different groups of data). The choice of the unit of tabulation will affect the numbers in the table. CSPro allows this selection if there is any "cross-group" tallying in the table.

See Also: Tally Attributes for a Table, Changing the Unit of Tabulation

Tally Items from Related Records

Data items on different records are often linked or related via codes on each of the records. For example, in a labor force survey there may be an initial roster of information about each household member followed by an additional set of questions for persons 15 years old and over. If a table involves age and sex from the initial roster and an economically active recode from the supplemental form, the data must be somehow linked so that the tally is done correctly. CSPro allows the definition of 'Relations' in the data dictionary, which will link these two related records for the purpose of creating cross tabulations.

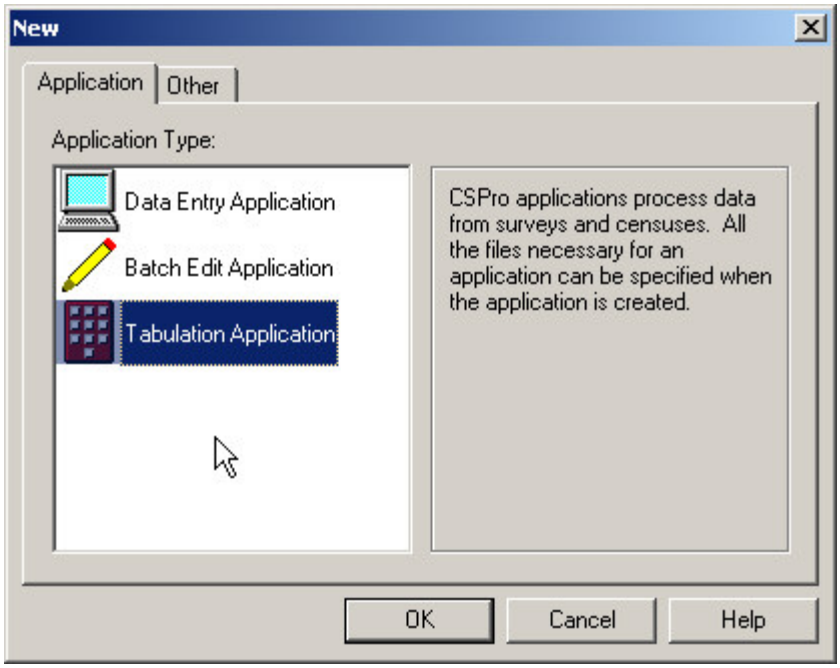
See Also: Relation Description, Tabulations Using Relations

Creating Tables

Create a New Tabulation Application

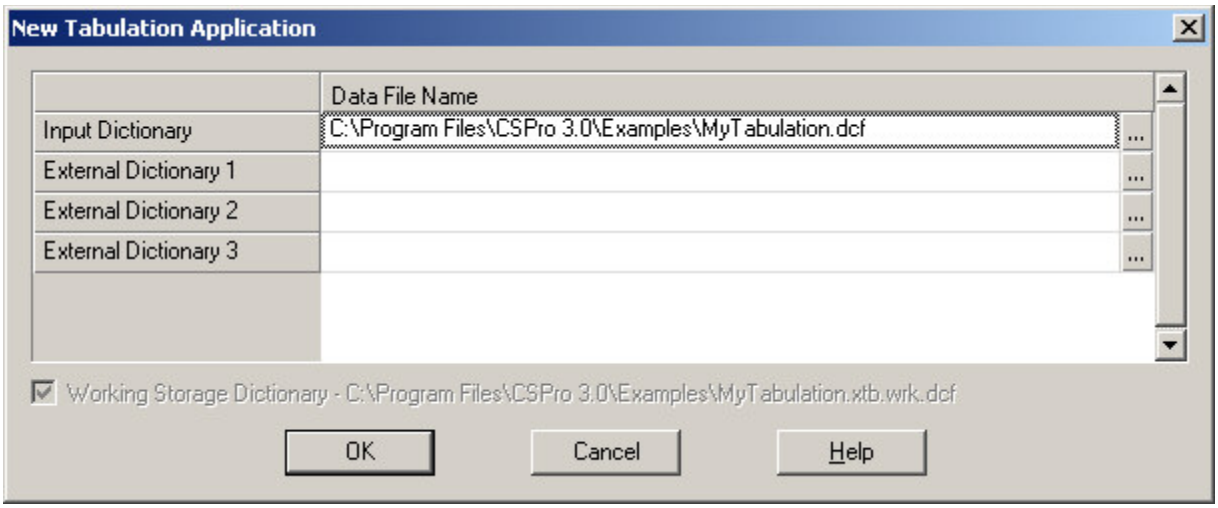
To perform tabulations, you will need a data file, and a data dictionary to describe the file. If you do not have a data dictionary, you will need a written description of the data file structure and organization. You can create the CSPro data dictionary as you create the new Tabulation application.

• Click on the toolbar, or from the **File** menu, select **New**. The following dialog box will appear.



• Select **Tabulation Application** and press **OK**.

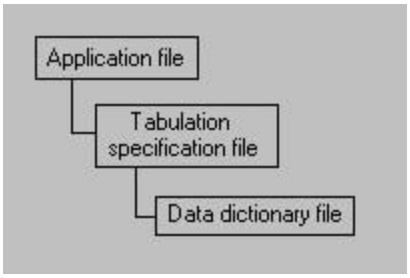
• A file dialog box will appear. Enter the name of the application file. Make sure you are located in the folder where you want to place the application files. Then press **Create**. The following dialog box will appear.



• A default name of the data dictionary describing the data file is given. You can use this name or change it. If you give the name of a dictionary file that already exists, that data dictionary will be used by the application. If you give the name of a dictionary that does not exist, a new data dictionary will be created.

• If you are using an existing CSPro data dictionary, you may now start creating tables. If you are creating a new CSPro data dictionary, you will need to enter information into the dictionary about records, items, and values before you can create tables.

Tabulation applications consist of the following files:



• Tabulation Application file (.XTB)
Specifies all other files contained in the application and includes other application information.

• Table Specifications file (.XTS)
Contains variable names and other parameters which define the tables in the application.

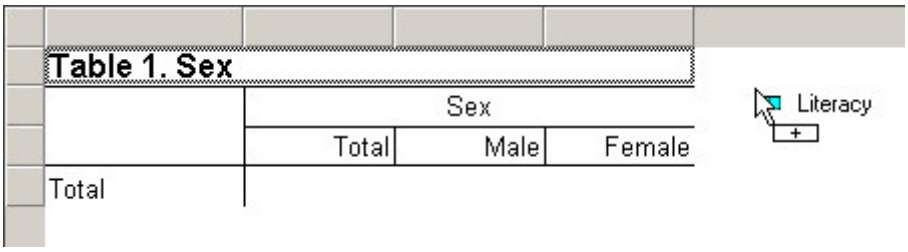
• Data dictionary file (.DCF)
Contains the physical format of the data file(s) to tabulate.

See also: CSPro Initial Screen Layout, Introduction to Data Dictionary

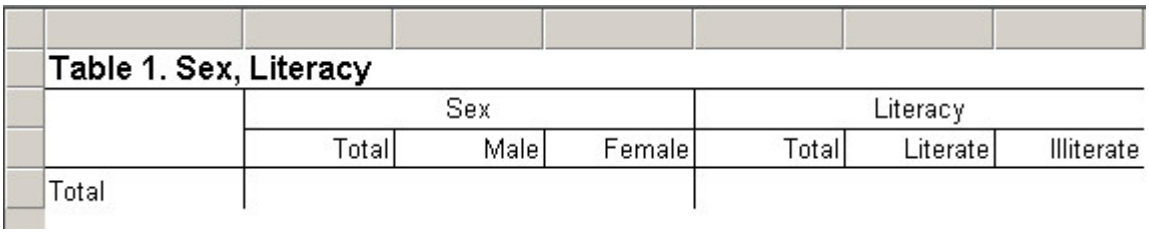
Create a Table

To create a table, do the following:

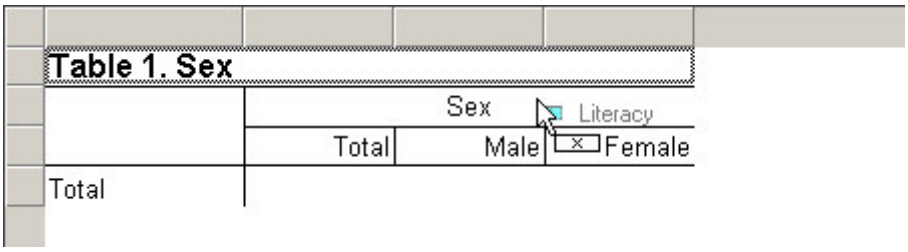
- Select the Dictionary [Dicts] tab to make the dictionary file structure visible.
- Expand the tree until the item(s) you wish to use for row or column variable(s) appears on the tree.
- Drag the desired dictionary item and drop it in the table window. Where you drop it on the table will determine whether it is used as a row or column variable. For a row item drop it on the left side below the generated "Table nn". For a column drop it on the top but a little below the generated "Table nn". If you imagine a diagonal line connecting the top-left corner of the table window to the bottom-right corner, dictionary items dropped below and to the left of the line will become row variables and items dropped above and to the right of the line will become row variables.
- This can be repeated with additional items. The key to placement for additional items is whether they are dropped next to or on top of existing items. This is indicated by the "+" or "x" displayed with the item while it is being dragged.



An item dropped next to an existing item is indicated by the "+" while dragging.



The dropped item will be appended to the existing item, e.g., Total, Male, Female followed by Total, Literate, Illiterate.



An item dropped on top of an existing item is indicated by the "x" while dragging.

Table 1. Sex and Literacy									
	Sex								
	Total			Male			Female		
	Literacy			Literacy			Literacy		
	Total	Literate	Illiterate	Total	Literate	Illiterate	Total	Literate	Illiterate
Total									

The dropped item will be crossed with the existing item to make sub-groupings, e.g., Total, Literate, Illiterate under Total, Male and Female.

- To delete a row/column variable, left-click on one of its value names, then drag and drop it back on the dictionary tree. In the ‘x’ example above, if "Sex" is deleted, then "Literacy" will also be deleted. However, "Literacy" can be deleted without affecting "Sex".

The rows/columns that are created when you add an item to a table come directly from the value set defined in the dictionary for that item. If you would like to have different rows/columns created for a particular item, you can either modify the value set in the dictionary or create an additional value set for the item and use the new value set instead. To use a different value set, rather than dragging the item itself onto the table, drag the value set instead. For items with more than one value set, the value sets appear underneath the item in the dictionary tree. For more information see Implications of Data Dictionary Value Sets.

Optional Definitions

• Define the universe for the table.

• Define the area for the table (required if you plan to create a thematic map of results from your table).

• Select the value Item or constant you want to tabulate.

• Select the weight Item or constant

• Add percentages.

When two variables are selected for the same dimension (row or column), the first one selected becomes the independent variable and the second one becomes the dependent variable. (See the discussion on row/column variables for more details.)

See also: Tabulations, Add a Variable to a Tabulation, Remove a Variable from a Tabulation

Create Tables with Multiple Variables

Row variables are those dictionary items or value sets which appear in the rows of the table. Column variables are those dictionary items or value sets which appear in the columns of the table. Every table must have at least one row or column variable specified; any given subtable can have a maximum of two row variables and two column variables. The number and disposition of row and column variables will affect the type of table generated.

• **1 Row/0 Column Variables**
If a table consists of one row variable and no column variables, the system will produce a table with the variable’s categories as rows and frequency counts as the only column.

- **0 Row/1 Column Variable**
  If a table consists of no row variables and one column variable, the system will produce a table with the tabulation categories in the columns and totals in the single row of the table.

- **1 Row/1 Column Variable**
  If a table or subtable consists of one row variable and one column variable, the system will produce a normal cross-tabulation, with the tabulation categories of each variable in row or column, as appropriate. By default, totals will appear in the left-most column and in the top-most row (it is possible, however, to change the position of the total).

- **2 Row/Column Variables**
  When a table or subtable is designed with two variables or value sets in the row and/or column, one of each pair is considered to be the independent [major] variable, and the other is considered to be the dependent [minor] variable. The tabulation categories of the dependent variable appear nested within the categories of the independent variable. Totals for a dependent variable appear as the topmost row or left-most column within each tabulation category of its independent variable.

Because there are effectively no limits on the number of rows and columns in a table, the combination of two variables/value sets can produce tabulations which will be extremely difficult to view and to understand. Users should give careful thought to the placement of variables and value sets in rows and columns, particularly when one or more of the items has a large number of tabulation categories. It is almost always easier to manipulate tables with large numbers of rows than those with the same number of columns.

Whenever the area processing function is invoked for a table set, the area levels are included as additional row categories within which the other row variables are displayed.

Subtables are created by adding variables to the bottom of the existing rows or to the right of the existing columns. When a subtable is created it is usually outlined in color to designate this fact. This is an example.

Table 1. Marital status by Sex, Literacy

Marital status	Sex			Literacy				
	Total	Male	Female	Total	Literate	Illiterate	Not Reported	Not Applicable
Total								
Married								
Divorced								
Separated								
Widowed								
Never Married								

Notice these are like two separate tables side-by-side; 1) Marital Status by Sex and 2) Marital Status by Literacy. There is a total of 6 columns (3 for Sex **plus** 3 for Literacy). Note also the wording in the generated title.

Compare the above to the following:

Table 1. Marital status by Sex and Literacy									
Marital status	Sex								
	Total			Male			Female		
	Literacy			Literacy			Literacy		
	Total	Literate	Illiterate	Total	Literate	Illiterate	Total	Literate	Illiterate
Total									
Married									
Divorced									
Separated									
Widowed									
Never Married									

This is a single table (1 row variable and two column variables) with a total of 9 columns (3 for Sex **times** 3 for Literacy). Note the wording in the generated title.

Implications of Data Dictionary Value Sets

Within the data dictionary, the user can define multiple value sets for each data item. For use in Tabulation applications, these sets may have value names for both individual values and value ranges. If one table requires five-year age groups but another uses single years of age, both requirements can be met by having two value set definitions for age.

If an item in the dictionary has more than one value set, you can choose which of the value sets to use when adding that item to a table. If you drag the item itself from the dictionary, CSPro will add the first value set for that item to the table. However, if you drag one of the value sets underneath the item in the dictionary tree window, CSPro will use the value set that you dragged.
When creating a value set, you should take into account the following:

- Value labels must be given for the value set since these appear as text (row or column) in the tables. If no label is given, the corresponding row/column label will be blank.
- Counts associated with the value labels are based on the item values defined for the label. This might be a single value, e.g., Male (sex=1); a range of values, e.g., Under 5 years old (age from 0 to 4); disjoint values, e.g., Teacher (occupation=23, 42, or 67); or some combination of these. That is, each value in the data file that falls into a particular range/value in the value set will cause the count to be incremented for the corresponding cell in the table.
- Counts are ONLY made in the table for item values defined in the value set. Values in the data file that are not defined in the value set are not counted. In the example below, the ‘Marital Status’ value set is missing codes 2 and 3 from its value set and the total reflects only the values listed in the value set. The exception to this rule is when Special Values are included in the Tally Attributes for the table or subtable.

N	Value Set Label	Value Set Name	Value Label	From
	Marital status	P05_MS_VS1		
			Married	1
			Widowed	4
			Never Married	5

N	Value Set Label	Value Set Name	Value Label	From
	Marital status	P05_MS_VS1		
			Married	1
			Divorced	2
			Separated	3
			Widowed	4
			Never Married	5

Table 5. Marital status by Sex

Marital status	Sex		
	Total	Male	Female
Total	23,472	11,587	11,885
Married	6,985	3,439	3,546
Widowed	788	149	639
Never Married	15,699	7,999	7,700

Table 6. Marital status by Sex

Marital status	Sex		
	Total	Male	Female
Total	24,271	11,787	12,484
Married	6,985	3,439	3,546
Divorced	799	200	599
Separated	-	-	-
Widowed	788	149	639
Never Married	15,699	7,999	7,700

- In the current version of the software, alphanumeric items CANNOT be tallied.
- By default, certain summary statistics (median, n-tiles and mode) are calculated using the counts in the values/ranges in the value set. The accuracy of these summary statistics is highly dependent on the number and size of values/ranges in the value set. If the range for a particular variable is large and there are a small number of categories in the value set, these statistics will be inaccurate. For best results, when calculating median, mode and n-tiles, use either single values for discrete variables or small, uniformly sized ranges for continuous variables. Note that by setting the appropriate options for n-tiles and median, you can use different categories for the n-tile/median calculation than those in the value set. See Tally Attributes for a Variable for details.

See Also: Include/Exclude Special Values in a Variable, Tally Attributes for a Table, Value Sets Description, Create a Table, Add Summary Statistics to a Table, Tabulate categories with disjoint values, Debug Table Totals

Tabulate Items with Multiple Occurrences

If an item used as a row or column variable is defined as having two or more occurrences, the following conditions apply:

• If the parent item [i.e., the one described as occurring `n` times] is dragged to the table as a row or column variable, the variable name will be shown **without parentheses**.

• If one of the occurrences of an item is dragged to the table as a row or column variable, the variable name will be shown with a number **in parentheses**.

For example, `MyItem` contains two occurrences, `MyItem(1)` and `MyItem(2)`. If `MyItem` (the parent item) is dragged from the dictionary tree to the table as a row or column variable, no parentheses will appear with the name `MyItem` in the table heading because no specific occurrence of `MyItem` was requested. However, if `MyItem(1)` is dragged from the dictionary tree to the table, then the item name `MyItem` will be shown with the identifying number in parentheses—`MyItem(1)`—because the user selected a specific instance [occurrence] of the variable.

If you choose an item with occurrences as a row or column variable, **all** occurrences of that item will be tabulated across **all** corresponding records in the data file. For example, if you choose `MyItem`, both `MyItem(1)` and `MyItem(2)` would be tabulated across all records.

If you choose a specific occurrence of an item as a row or column variable (e.g., `MyItem(2)`), only that occurrence will be tabulated across all corresponding records.

You may also use items with occurrences as the value or weight item. Note that when the unit of tabulation is not the repeating item, you will get errors when running your table unless you specify the occurrence number to use (e.g., `MyItem(2)`) or change the unit (e.g., to `MyItem`).

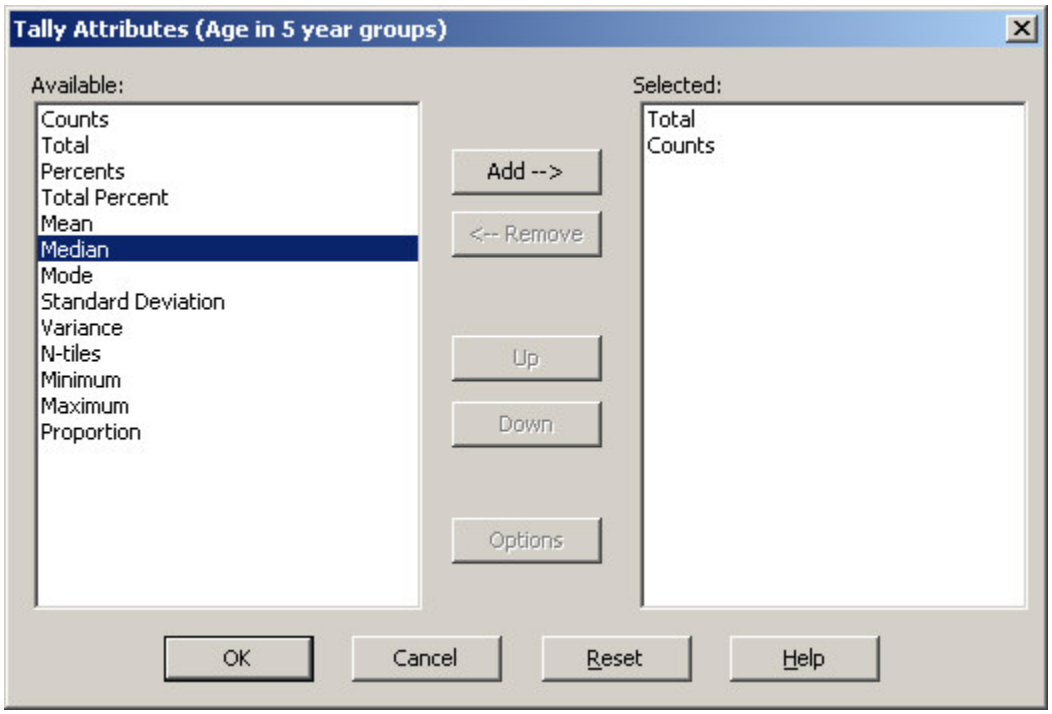
See Also: Create a Table

Tally Attributes for a Variable

The Tally Attributes (Variable) Dialog Box allows you to choose which additional calculations should be displayed for a variable in a table. Using Tally Attributes (Variable), you can display totals, percents and summary statistics for each variable in the table.

To bring up the Tally Attributes (Variable) Dialog Box, right-click on a stub or column header for the variable whose attributes you wish to modify and choose "Tally Attributes (<Variable Name>)".

Unlike the Tally Attributes (Table/Subtable) Dialog Box, the settings made in the Tally Attributes (Variable) Dialog Box apply to the chosen variable and not to the rest of the table or subtable.



The Tally Attributes (Variable) Dialog Box contains two lists: on the left is the list of available calculations and on the right is the list of calculations selected for the current variable.

To add a new calculation, select the type of calculation in the list on the left by clicking on it and then click the "Add" button. The selected calculation will appear in the list on the right.

Each calculation has its own set of options such as percent type for percents, number of tiles for n-tiles, etc. To modify the options for a calculation, first select the calculation by clicking on it in the selected list, and then click the "Options" button to bring up the options dialog box.

To remove an existing calculation, click on the calculation in the list of selected calculations on the right and click the "Remove" button. The calculation will be removed from the list of selected calculations.

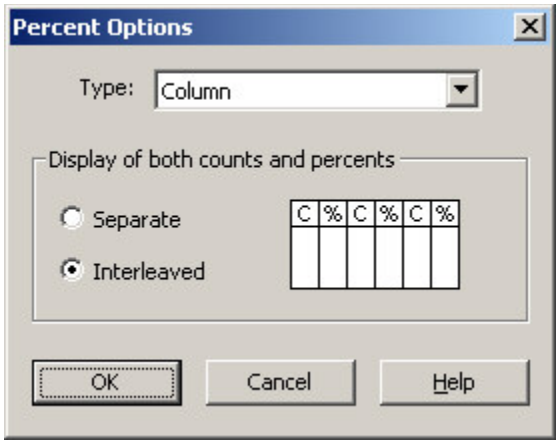
The order that the calculations appear in the selected list will be the same order in which they appear in the table. That is, if you have ‘Total’ before ‘Median’ in the selected list, then in your table, you will have a "Total" row or column before the row/column containing the median. To change the order of the calculations, choose a calculation in the selected list by clicking on it and click either the "Up" or "Down" button to change its position in the list.

The following calculations are available:

Counts: Displays the counts (frequencies) for each value in the value set. Adding counts to the variable will add a row or column to the table corresponding to each value in the value set for the variable.

Total: Displays a single row or column with the total for all values in the value set.

Percents: Displays the counts for the variable as percentages of the row total, column total or subtable total. Adding percents to the variable will add a row or column to the table for each value in the value set for the variable, but unlike ‘Counts’, the results are displayed as percentages. Percents have the following options:



Type: Selects how to calculate percents for the selected variable. Specifically, this gives the ‘base’ (100%) or denominator for the percent. One of three options can be selected:

- **Total** - use the total count for the subtable as a base for each percent.
- **Row** - use the total count for the subtable row as the base for the percents in each row. If there is a total column, it will contain ‘100 %’s.
- **Column** - use the total count for the subtable column as the base for percents in each column. If there is a total row, it will contain ‘100 %’s.

Interleaved/Separate: Selects whether or not the percents are alternated with the counts. For percents to be interleaved, counts must be present directly before or directly after the percents in the selected list. When percents and counts are interleaved, the order in which the counts and percents appear in the selected list determines whether the first column/row will be a percent or a count -- i.e., if counts appear first in the selected list, then the first row or column will be a count followed by a percent.

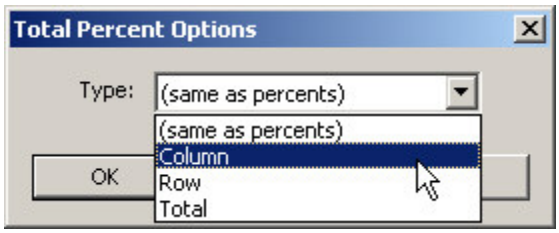
Sex					
Total	Percent	Male	Percent	Female	Percent

Interleaved percents

Sex					
Total	Male	Female	% Total	% Male	% Female

Separate percents

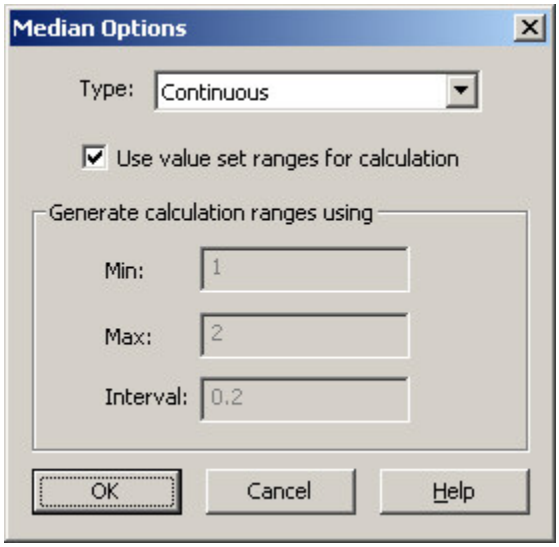
Total Percent: Displays a single row or column with the total for all values in the value set expressed as a percentage of the row, column or subtable total. Total percent has the following options:



- **Type:** The type of percent (row, column or total as for percents above). You may also choose '(same as percents)' to force the total percent to use the setting from the percents.

Mean: Displays the arithmetic mean (average) for each row or column.

Median: Displays the median for each row or column. Median has the following options:



- **Type:** Specifies the type of variable – either discrete or continuous. This should be set to "Discrete" for discrete variables (such as number of rooms) and to "Continuous" for continuous variables (such as age or income). The median is approximated using linear interpolation based on a frequency distribution for the variable. For discrete variables, 0.5 is subtracted from the result to reflect the fact that values in a category are all equal to the lower limit of the category whereas for continuous variables values in a category are in between the lower and upper limits of the category.

- **Calculation ranges:** Controls which frequency distribution is used to calculate the median. If "Use value set ranges for calculation" is checked, then the median will be approximated using a frequency distribution for the value set that was dropped onto the table. Otherwise,

you can specify the ranges to use for the distribution by entering the min, max and interval. CSPro will generate ranges of width ‘interval’ starting at ‘min’ and going up to ‘max’. For example, a "min" of 0, "max" of 20 and "interval" of 5 will generate the following four ranges:

0 up to but not including 5
5 up to but not including 10
10 up to but not including 15
15 up to but not including 20

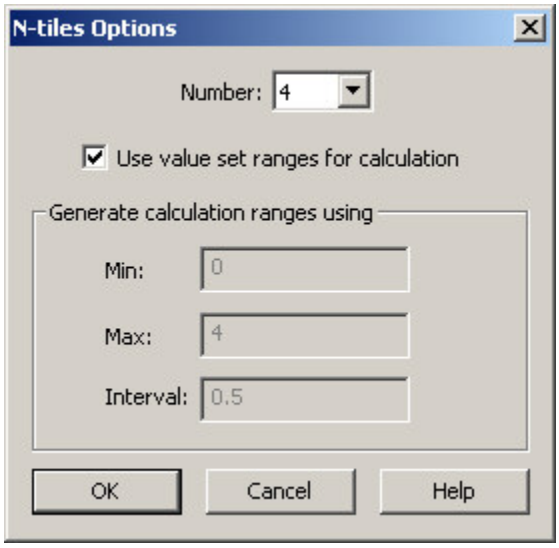
If the value set you use does not have very many ranges, it is better to generate your own ranges in order to get a closer approximation of the median. For example, using a value set with only three categories for age such as 0-14, 15-49 and 49+ will generally result in a very poor approximation of the median, while using 5-year groupings will generally result in a good approximation. For an even better approximation, turn off "Use value set ranges for calculation" and specify a min of 0, a max of the largest age in the population and an interval of 1. For variables such as "income" or "price of a house", you may need to experiment with different sets of ranges to come up with a good approximation for the median.

Mode: Displays the mode (value with the greatest number of occurrences in the data set) for each row or column.

Standard deviation: Displays the standard deviation for each row or column.

Variance: Displays the variance for each row or column.

N-tiles: Displays percentiles for each row or column. N-tiles has the following options:



- **Number:** Sets the percentiles to show. Set it to 4 for quartiles, 5 for quintiles, etc.

- **Calculation ranges:** Controls which frequency distribution is used to calculate the n-tiles. If "Use value set ranges for calculation" is checked, then the n-tiles will be approximated using a frequency distribution for the value set that was dropped onto the table. Otherwise, you can specify the ranges to use for the distribution by entering the min, max and interval. CSPro will generate ranges of width "interval" starting at "min" and going up to "max". For example, a "min" of 0, "max" of 20 and "interval" of 5 will generate the following four ranges:

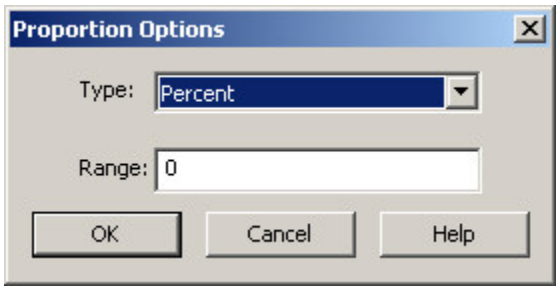
  0 up to but not including 5
  5 up to but not including 10
  10 up to but not including 15
  15 up to but not including 20

If the value set you use does not have very many ranges, it is better to generate your own ranges in order to get a closer approximation of the n-tiles. For example, using a value set with only three categories for age such as 0-14, 15-49 and 49+ will generally result in a very poor approximation, while using 5-year groupings will generally result in a good approximation. For an even better approximation, turn off "Use value set ranges for calculation" and specify a min of 0, a max of the largest age in the population, and an interval of 1. For variables such as ‘income’ or ‘price of a house’, you may need to experiment with different sets of ranges to come up with a good approximation for the n-tiles.

Minimum: Displays the minimum value for each row or column.
Maximum: Displays the maximum value for each row or column.

Proportion: Displays the proportion of a specified subset of the counts for the variable as a percent or ratio of the total for the variable. You could also use proportion, for example, on the age variable, to display the percentage or fraction of the population with age between 15 and 49. You could also proportion on a yes/no variable such as ‘owns a television’ to show the percentage or fraction of the population that owns a television. The proportion is calculated by taking a subset of the values in the value set for a variable, computing the total for that subset and then dividing that total by the total for all values in the value set. The result may be displayed as a percent or as a ratio. The numerator used to calculate the proportion is the total number of values that match the specified subset of the value set and the denominator is the total number of counts for all values in the value set. For example, in the case of the proportion of people between 15 and 49, the numerator would be the total number of people between 15 and 49 and the denominator would be the total number of people of any age.

Proportion has the following options:



- **Type:** The type of proportion to display. It may be one of the following values:
    - Percent: Adds a single row or column to the table showing the counts for the values selected in the range as a percentage of the total.
    - Percent & Total: Adds two rows/columns to the total: the first showing the percent as above and the second showing the total number of counts for the values selected in the range (i.e., the numerator used to calculate the percent).
    - Ratio: Adds a single row or column to the table showing the counts for the values selected in the range as a fraction of the total.
    - Ratio & Total: Adds two rows/columns to the total: the first showing the ratio as above and the second showing the total number of counts for the values selected in the range (i.e., the numerator used to calculate the fraction).

- **Range:** Specifies the set of values whose counts will be used in the numerator of the percent or ratio and as the total. Range can be a single value (e.g., 3), multiple values separated by commas (e.g., 1, 2, 5, 17) and it may also include value ranges using colons (e.g., 1, 2, 5:10, 25:30).

To show the percentage of the population between 15 and 49 you would add "proportion" to the age variable, set the type to "Percent" and set the range to 15:49. To show the fraction of the population that

owns a television, add "proportion" to the variable "owns a television", set the type to "Ratio" and the range to 1 (assuming 1 means "yes").

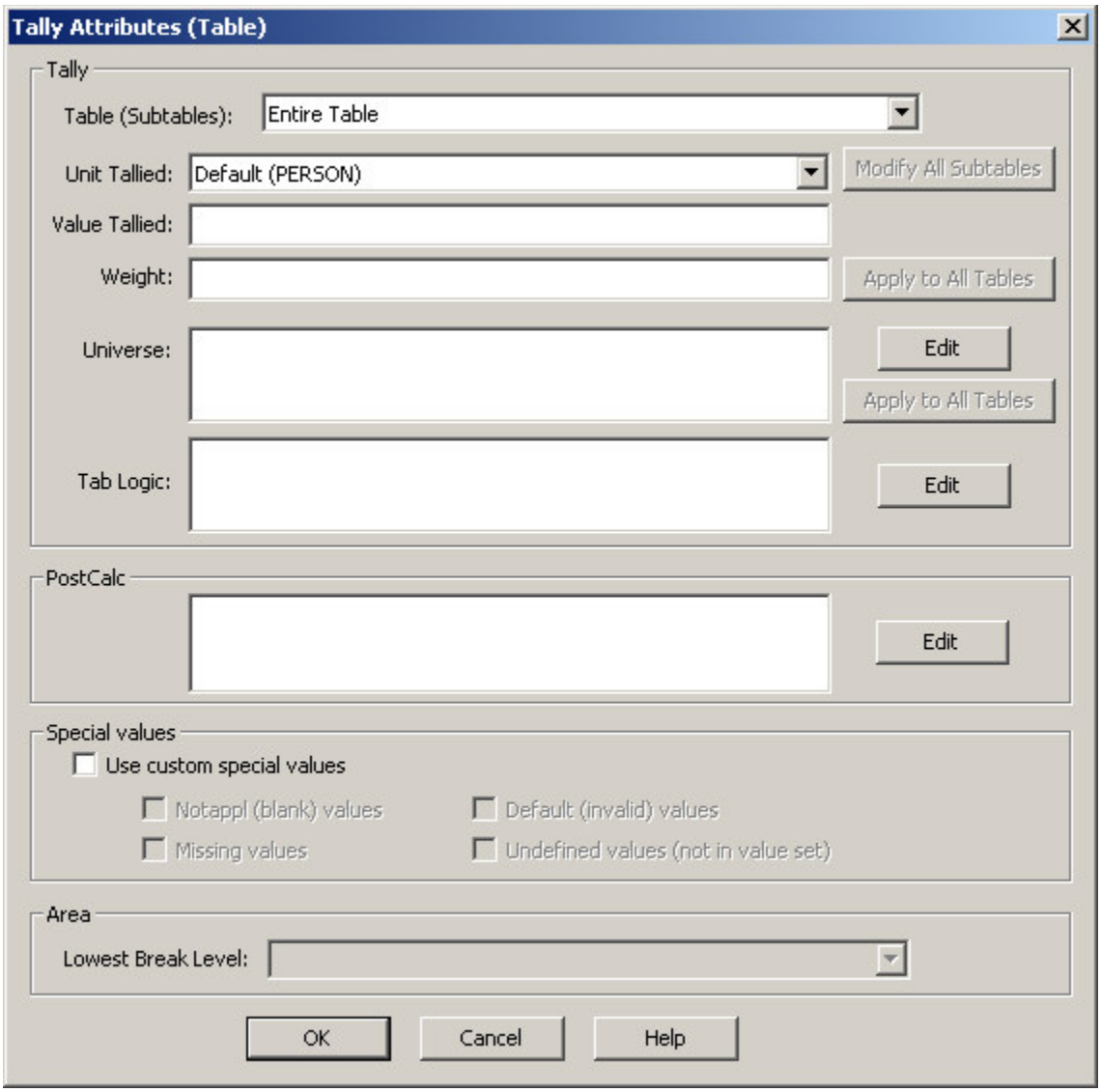
Note that the values calculated for median, mode and n-tiles are dependent on the value set used. For more information see Implications of Data Dictionary Value Sets.

See Also: Hide or Change the Position of the Total, Add Percents to a Table, Add Summary Statistics to a Table

Tally Attributes for a Table

Click on in the toolbar or right-click on a table or subtable and choose Tally Attributes (Table) or Tally Attributes (Subtable) to bring up the Tally Attributes (Table/Subtable) Dialog Box. This dialog allows you to change the results of the tabulation of a table or subtable by adding weights, restricting the universe of values, changing the values tallied, changing the unit of computation and changing the way special values and areas are considered.

By default, CSPro counts or tallies once each time it finds an occurrence of a variable in your data set. That is, each time an occurrence of a variable is found, it looks at the value of the variable found, determines which cell of the table this occurrence falls in and increments the value in that cell by one. By changing the tally attributes, you can have CSPro change the amount that it increments the cell by (using value tallied or weight). You can also make CSPro ignore certain occurrences of a variable and not count them (using unit tallied or universe).



The Tally Attributes (Table/Subtable) Dialog Box contains the following settings:

Table (Subtables): Allows you to choose to modify the tally attributes of either the entire table or one of the subtables. Any changes made to Unit Tallied, Value Tallied, Weight, Universe, and Tab Logic will only apply to the selected table or subtable. Changes made to Special Values, PostCalc and Lowest Break Level always apply to the entire table.

Each subtable may have its own unique settings for Unit Tallied, Value Tallied, Weight, Tab Logic and Universe.

- If a weight or value tallied is specified for both the entire table, and for a subtable, then the two weights or values (table and subtable) will be multiplied together to compute the final weight or value for the subtable.
- If a universe is specified for both a subtable and for the entire table, then the two universes will be combined (using the *and* operator) to compute the final universe for the subtable.
- If a table has more than one subtable, then you cannot set the unit tallied for the entire table. In this case, there is no unit for the entire table. You must set the unit for each subtable. However, you can set the unit for all subtables in one step by clicking on the Modify All Subtables button.

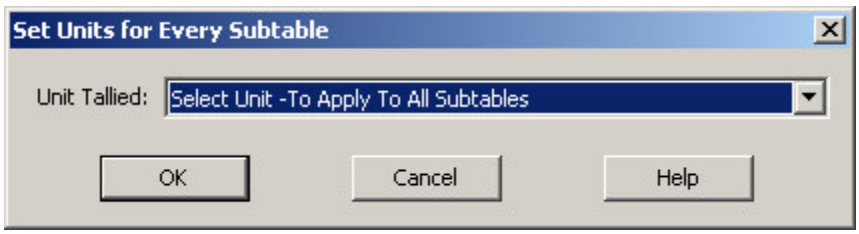
Note that Modify All Subtables only changes the unit for existing subtables. Newly-added subtables will be given the default unit unless you modify them explicitly after they are created.

- If Tab Logic is specified for both a subtable and for the entire table, the logic for the entire table will be executed for each subtable in addition to any logic for the subtable. The logic for the entire table will be executed before the logic for the subtable.

Note that if you choose Tally Attributes (Subtable), the subtable that you clicked on will automatically be selected in this list.

Unit Tallied: Allows you to change the unit of computation for the selected subtable. The unit of computation is the level, record or item in the dictionary that is counted for the tally. For example, when tallying the rent variable of the housing record, by default, the unit would be the housing record so each household would be counted once. However, you could tally the rent variable on a per person basis by setting the unit to the person record. This would count the number of *people* paying a given rent rather than counting the number of *households* paying a given rent. In most cases, the unit is left as the default.

You can modify the unit for all subtables in a table at once by clicking on the "Modify All Subtables" button. This brings up the following dialog box:



This dialog displays all units that can be applied to all of the subtables. Choose the unit from the drop down and click "OK". Potential units will not be included if they cannot legitimately be applied to one or more of the subtables. For example, if one subtable contains an item from a multiple record (e.g., "sex" on the person record), then only the multiply occurring record, and multiply occurring items on that record, are considered legitimate units to be applied to all subtables.

Value Tallied: Optional name of a numeric item in the dictionary whose value would be tallied into the data cells instead of the unit (one) tally. For example, setting the value tallied to ‘children ever born’ would count the actual number of children instead of counting the number of cases with a given number of children. This can also be a numeric constant.

Weight: Optional name of a numeric item in the dictionary that contains the inflation factor (weight) for a survey case. For each tally, this value will be tallied instead of the unit (one) tally. This can also be a numeric constant.

Universe: An expression that restricts the data used for tabulation. When you define a universe, CSPro will tabulate only those data records in the questionnaires that meet the conditions stipulated by you. The "universe" specification acts as a filter, as the tables produced use only a subset of the data file's records. Therefore, values in the table may be lower than they would be with no universe specified, since the universe restricts the data available for tabulation.

Tab Logic: Optional CSPro logic statements that may be used to modify the values of variables in the case during tabulation. This is used mainly for recodes of existing main dictionary variables into new variables used in the table. For information see Table Logic (tablogic). Logic may be entered directly in the associated edit box or you can click on the Edit button to bring up a larger window in which to modify the logic.

PostCalc: Optional CSPro logic statements that may be used to modify values in the table after the tabulation is complete. This logic can be used to add values to the table calculated from the tabulated data such as ratios. For more information see Introduction to Table Post Calculation. Logic may be entered directly in the associated edit box or you can click on the Edit button to bring up a larger window in which to modify the logic.

Special Values: Allows you to customize which special values are counted in the selected table. By default, only special values included in the value sets being tallied are considered. By checking "use custom special values" you can specify which, if any special values to include in the tabulation regardless of the special values contained in the value sets. For example, if the value set used in a subtable contains the special values *notappl* and *missing*, by default *undefined* and *default* values will not be included in the tabulation. Enabling custom special values and checking *notappl* while leaving the others unchecked will cause only *notappl* values to be included in the tabulation.

Lowest Break Level: Allows you to limit the geographical areas used for area processing for just the selected table. The lowest break level is the lowest geographical area that will be used in the table. For example, if your file includes data at the province, district and village level but you set the lowest break level to district, then only province and district will appear in the selected table while all three levels will appear in other tables in the file.

See Also: Create Multiple Subtables, Define a Universe for a Table, Add Weights to a Table, Tabulate Values Instead of Frequencies, Include/Exclude Special Values in a Variable

Add, Insert, and Delete Tables

Add Table

Any table added to an existing set of tables will always be placed **after** the last existing table. If you want a new table to appear in any other position in the table set, you must insert the table.

To add a new table simply press the **Add** button on the toolbar. You'll notice a new table is created with the name "`Table #`" (where # represents the number of the table—if this is the 5th table in your table set, it will initially be named "`Table 5`"). Finish the definition of the added table by adding dictionary items and specifying any universe definitions or other tabulation parameters desired.

You can also add a table by right-clicking anywhere in a table and selecting **Add Table** from the pop-up menu. Alternatively, you can add a table from the **Edit** menu: select **Add Table**.

Insert Table

You can insert a table before another table by right-clicking anywhere in a table or on the table name in the table tree, then selecting **Insert Table** from the pop-up menu. Alternately, you can insert a table by clicking on the icon, or selecting **Insert Table** from the **Edit** menu.

Any table inserted into the existing table set will always be placed before the currently displayed table, which we will call "Table N". You'll notice a new tab is created with the name "Table #A" (where # is N-1). If you insert another table before this one, it will be created as "Table #AA". If you insert another table before "Table N", it will be created as "Table #B". In both cases # is N-1. Once created, finish the definition of the inserted table by adding dictionary items and specifying any universe definitions or other tabulation parameters desired.

Delete Table

The table that is currently on view in the table window is always the one affected when you choose to delete. So if the table you want to delete is not displayed in the table window, you must first make it visible. Either select it from the table tree on the left or by using the table paging arrows in the Tabulation toolbar.

You can then delete the table by right-clicking anywhere in the table or on the table name in the table tree, then selecting **Delete Table** from the pop-up menu. Alternately you can use the icon on the toolbar or select **Delete Table** from **Edit** menu. You will need to confirm that you actually want to delete the table.

Move Between Tables

To move through the tables press the or buttons on the toolbar.

To go to a specific table select the table title from the Table Tree on the left. If the Table Tree is not showing in the left panel, click on the **Tables** tab at the bottom of the panel.

Run a Tabulation Application

Once you have finished defining your table(s), you are ready to run them against your data files to produce the tables.

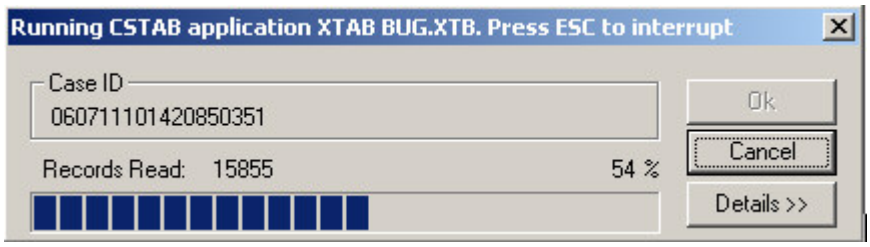
Press the Run Tabulation button or press **Ctrl+R**. If the application has been modified, you will need to save the changes in order to run.

The run will request the following files as appropriate:
Input Data: The data file(s) being tabulated. There is NO required extension for CSPro data files. Multiple input data files can be selected using the browse button.

Area Names: Only used for applications with area processing. An Area Names File is used to associate the area codes in the data file(s) with their descriptive text. The .ANM extension is required.

Output TBW: The Table Viewer file created by the run. The .TBW extension is required, but does not to be specified by the user.

During the run, a bar will be displayed showing progress in processing the data file (See below).

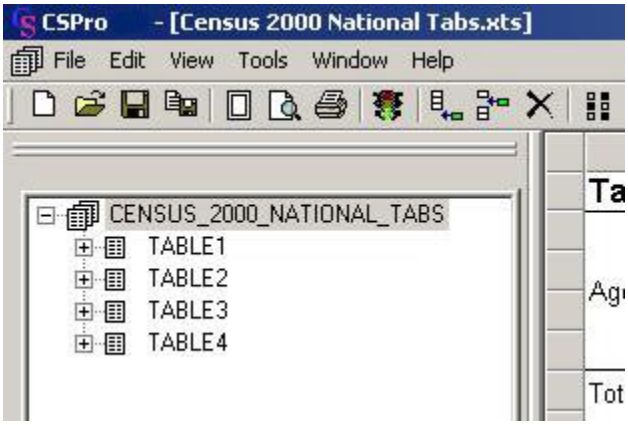


Note that you can also stop the processing by using the "Cancel" button.

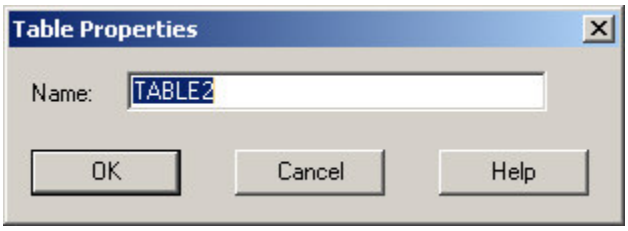
If there are no warnings or errors, the application will display the resulting tables. If your application has generated messages, these will be displayed via a listing in Text Viewer but the results will still be shown in your tables.

Renaming Tables and Table Applications

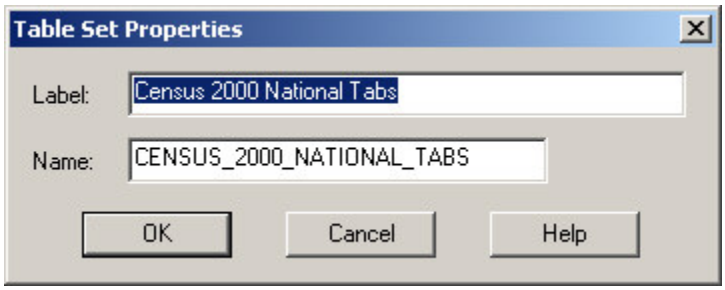
CSPro automatically assigns a name to each table in the application, such as TABLE1, TABLE2, etc. These names are displayed in the table tree (when "names in tree" is checked on the view menu), shown in certain error messages, and used in table post calculation.



You can change the name of a table using the table properties dialog. Right-click on the table in the "Tables" tree and choose "Properties". Edit the name of the table and click "OK". Note that the system will not allow you to have two tables with the same name.



You can also change the name and label of the table application by right-clicking on the table application in the "Tables" tree and choosing "Properties". The tables application is always located at the top of the tree. Edit the name and label of the application and click "OK."

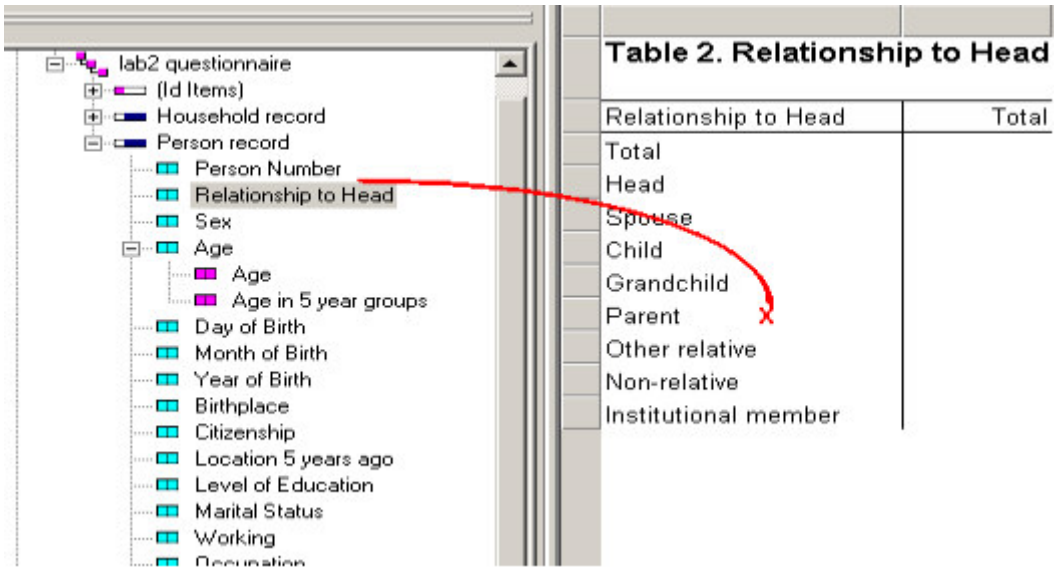


How To ...

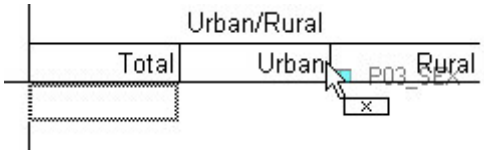
Add a Variable to a Tabulation

A table (or subtable) is created by selecting an item OR value set [if multiple value sets are available] from the data dictionary and then drag and dropping it into the table window on the right. Dropped in the upper right it becomes a column and in the lower left it becomes a row.

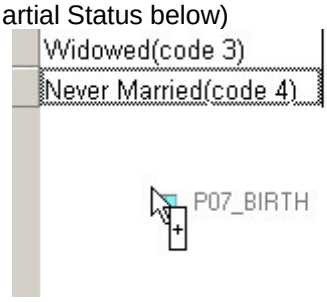
Here is an example of dropping a row.



In order to create subgroupings of categories already defined in the table, e.g., by sex, drag and drop the subgroup item (sex) anywhere on the existing value set. (See 'Sex' in column below)



In order to create an additional set of row/column categories, drag and drop the item or value set below the existing rows [for additional set of rows] or to the right of the existing columns [for additional set of columns]. (See Place of Birth and Martial Status below)



The resulting table would be as follows:

Table 1. Marital status, Place of birth by Urban/Rural and Sex

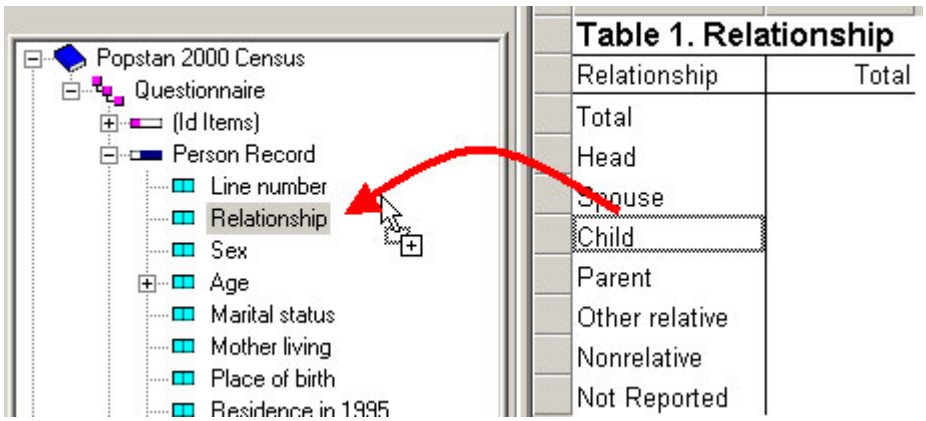
	Urban/Rural								
	Total			Urban			Rural		
	Sex			Sex			Sex		
	Total	Male	Female	Total	Male	Female	Total	Male	Female
Marital status									
Total									
Married(code 1)									
Divorced(code 2)									
Widowed(code 3)									
Never Married(code 4)									
Place of birth									
Total									
Popstan									
Endar									
Victoria									
Africa									
Middle East									
Asia									
Europe									
Americas									

The title of the table is generated by combining the LABELS from the dropped items/values. The word "**and**" indicates that the second is dependent on the first. The word "**by**" separates the row labels from the column labels. A comma separates the subtable items in a row or column.

In example above: "Marital Status" and "Place of Birth" are two row items (separated by a ","). These two row items are separated from the column items using the word "by". The column item, "Urban/Rural", is subdivided by "Sex" (as indicated with "and").

Remove a Variable from a Tabulation

To remove a value set from a table, left-click on any of its value labels, then drag and drop it back in the tree panel. See below.



If an independent item is removed then the associated dependent item will also be removed.

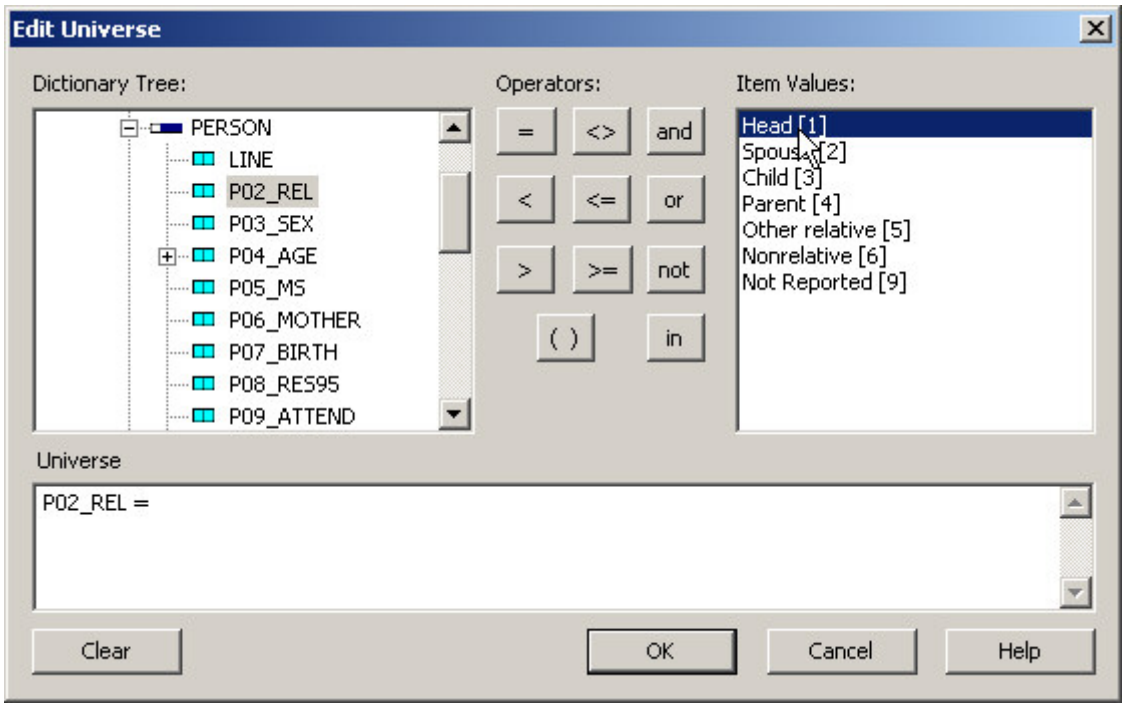
Define a Universe for a Table

The definition of a universe is available from the Tally Attributes for a Table dialog box. Universes are available for the entire table and/or individual subtables.

Experienced users, always remembering to use item names, can type the universe conditional directly in the box provided by the **Tally Attributes** menu.

Use the **Apply All** button to the right of the box if the universe is for the entire table and you want to propagate it to all tables currently defined. It will replace any existing Universe in other tables.

Less experienced users can access the universe "wizard" from the "Edit" button to the right of the universe box. This brings up the following:



You can type a universe condition statement, as CSPro logic, in the Universe box. Or, you can double click on an item name, press a relation button, and double click on a value.

- Double click on the item in the left-hand box that you want to use in the universe condition. The item name will appear appended to the text in the universe condition below.
- Click a relationship button (`=`, `<>`, `>`, `>=`, `<`, `<=`). The relation will appear appended to the text in the universe condition below.
- Double click on a value in the right-hand box. The value, not the text label, will appear appended to the text in the universe condition below.
- You may enter several conditions using the **and / or**. You can also add parentheses to modify the order of evaluation of the conditions.
- To delete the universe simply erase the contents of the universe condition.
- Press OK when you have completed the universe condition.
- You can apply a universe to all the tables, by pressing the **Apply All** button to the right of the Universe box in the **Tally Attributes** dialog box

Examples:

- To restrict your table to females of reproductive age, you might state:

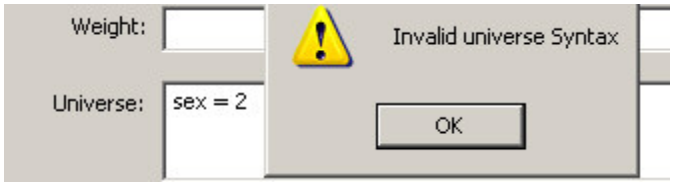
```
P03_SEX = 2 and P04_AGE in 12:49
```

- To restrict your table to heads of households who are economically active, you might state:

```
P02_RELATION = 1 and ECON_ACTIVE = 1
```

Notes:
If there is a universe condition for the entire table and another for a subtable, the two conditionals will be combined by "and" to determine tallies for the subtable.

If there is some type of error either in the typed or generated conditional, the system will display an "Invalid Universe Syntax" message and the problem must be corrected before the universe can be accepted. (In the example below, "Sex" is not the CSPro name of a variable.)



See also: Restrict a Universe, Tally Attributes for a Table

Add Weights to a Table

The declaration of weights is available from the Tally Attributes for a Table dialog box. Weights are available for the entire table and/or individual subtables.

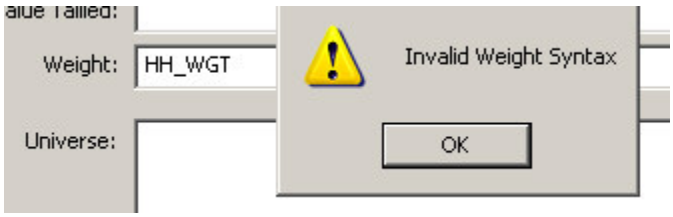
Weights can be a data item name, a numeric constant, or any arithmetic expression.

To add a weight to a table or subtable, simply type the expression the item name, numeric constant or expression directly into the box provided for weight.

Use the **Apply All** button to the right of the box if the weight is for the entire table and you want to propagate it to all tables currently defined. It will replace any existing weight in other tables.

Notes:
If the value of the weight data item or expression is not numeric when a tally is to be made, the tally is NOT done. This is equivalent to a tally of zero.

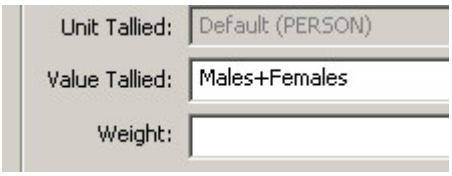
If the weight expression is not valid it must be corrected before the Tally Attributes dialog box can be closed.



See also: Tabulate Values and/or Weights, Tally Attributes for a Table

Tabulate Values Instead of Frequencies

The declaration of values to be tallied is available from the Tally Attributes for a Table dialog box. Tally values are available for the entire table and/or individual subtables.



A value to be tallied can be an individual data item, a numeric constant, or any valid expression that can be resolved into a single value.

Note:
If the value to be tallied is not numeric when a tally is to be made then the tally is NOT done. This is equivalent to a tally of zero.

If the tally value expression is not valid it must be corrected before the Tally Attributes dialog box can be closed. A message is issued if the expression is invalid.

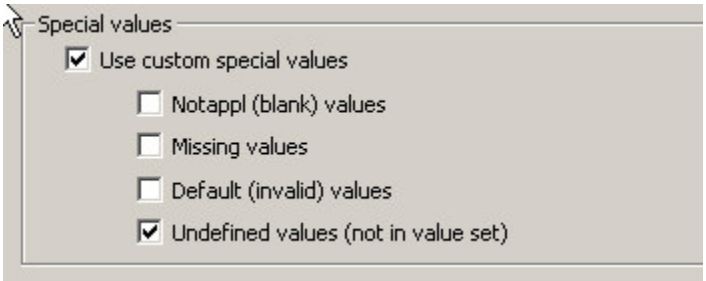
See also: Tabulate Values and/or Weights, Tally Attributes for a Table

Include/Exclude Special Values in a Variable

The addition of special values to be tallied is available from the Tally Attributes for a Table dialog box. It allows you to add rows and columns to your table to tally special values that are not contained in the value sets for your variables or to remove the rows and columns containing special values that are included in the value sets for your variables.

Use custom special values always applies to all variables in a table. You cannot change the special values for a single variable or single subtable.

In the Tally Attributes dialog box users have the following special value options:



To activate the Special Values portion of then menu, check the **Use custom special values** box (as shown).

After that select the types of special values you want to add to/remove from the value sets of each item in the table by checking the appropriate box.

- **Notappl** values counts blanks occurring in the data item.
- **Missing** values counts special codes defined as "missing" in the data dictionary.
- **Default** values counts the invalid values (shown as asterisks "*" in the data).
- **Undefined** values counts any values for the item which is not otherwise counted.

Once **Use custom special values** is checked, this setting overrides any special values in the value sets for items in the table. If **Use custom special values** is checked, any of the special values that are checked in the tally attributes dialog will be displayed for every item in the table regardless of whether or not the value set for that item contains the special value. In addition, only the special values that are checked in the dialog box will be displayed in the table even if they appear in the value set for the an item in the table. For example, if your value set contains **Notappl** and you check **Use custom special values** but leave **Notappl** unchecked then the columns/rows corresponding to the **Notappl** value in your value set will be removed from the table. If your value set contains no special values, and you check **Use custom special values** and **Notappl**, then a row or column for **Notappl** will be added for each variable in your table.

Note: Undefined values count includes blanks if **Notappl** values is not checked. If it is checked then those counts are taken out of the "Undefined values" count and tallied separately. Similar statements apply to other special values.

Hide or Change the Position of the Total

By default, a total count is automatically associated with each value set placed in a table. The usual placement for rows is the top of the value labels; for columns it is to the left of the value labels. To modify this use the Tally Attributes for a Variable dialog. To bring up the Tally Attributes for a Variable Dialog, right-click on the variable in the table and choose Tally Attributes (<variable>).

To remove the total row or column, select the total by clicking on it in the list of selected calculations on the right hand side of the dialog, and then click on the "Remove" button to delete the total.

To move the total after the counts, select the total by clicking on it in the list of selected calculations on the right hand side of the dialog, and then click on the "Down" button to move it until it is after the counts.

The Default is set in the Preferences and Default Formats.

See Also: Hide or Show a Row or Column

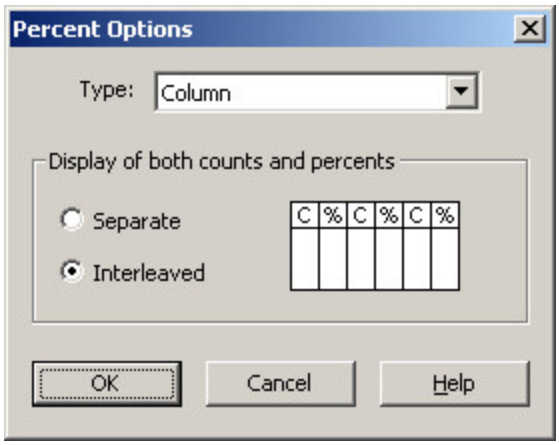
Add Percents to a Table

There are optional percentages available for each value set in a table. To create percentages, use the Tally Attributes for a Variable dialog.

To bring up the Tally Attributes for a Variable Dialog, right-click on the variable in the table and choose Tally Attributes (<variable>).

To add percents to the variable, select "Percents" in the list of available calculations on the left-hand side of the dialog and click the "Add" button.

There are two options for "Percents" which can be set by clicking the "Options" button while the "Percents" are selected:



Type

- **Total** – Percents are based on total cell for the table (% cell = (associated cell value * 100) / total of all the cells in the table).
- **Row** – Percents are based on total for the row (%cell = (associated cell value * 100) / total of all the cells in the row).
- **Column** – Percents are based on total for the column (%cell = (associated cell value * 100) / total of all the cells in column).

Interleaved/Separate
By default, percents and counts are interleaved; that is, for each value in the value set, the frequency for that value is listed in the column/row directly preceding or following the percent for that value. Percents may also be separate, in which case all the frequencies are grouped together and all the percents are grouped together.

Sex					
Total	Percent	Male	Percent	Female	Percent

Interleaved Percents

Sex					
Total	Male	Female	% Total	% Male	% Female

Separate Percents

For percents to be interleaved, the percents must directly follow or directly precede the counts in the list of selected calculations in the tally attributes dialog.

You can change the position of the percents relative to the other selected calculations (including the counts) by selecting the percents in the list on the right-hand side of the dialog and clicking the "Up" or "Down" buttons.

To show percents only (percents without counts), add the percents as described above and set the options to "Separate". Then select the counts in the list of selected calculations on the right-hand side of the dialog and click the "Remove" button to delete the counts. Finally, do the same to remove the "Total".

To show percents only with a number row, add the percents and remove the counts as above, but do not remove the "Total".

The Defaults are set in the Preferences and Default Formats.

By default, data in rows and columns containing percents will be displayed with one decimal place (e.g., 12.3). To change the number of decimal places see Change the Number of Decimal Places Displayed.

You can also show the percentage of the counts for a subset of the values in a value set (e.g., percent of people with age between 15 and 49) using proportions. See the section on proportions under Tally Attributes for Variable for more information.

Add Summary Statistics to a Table

There are optional summary statistics available for each value set in a table. To add summary statistics to a table use the Tally Attributes for a Variable dialog.

The choices for statistical measures are:

- **Mean** – average value of observations.
- **Median** - middle value (half of the observations are above this value and half below).
- **Mode** – for categorical (discrete) variables this is the most frequently observed value. For grouped variables, this is the lower limit of the modal class or the range in the value set in which the most observations lie.
- **Std Deviation** – Standard Deviation, a measure of how clustered the observations are around the mean (square root of the variance).
- **Variance** – another measure of dispersion.
- **N-tiles** - values that divide the values into N groups each of which contain 1/N of the total observations (N = 2 is equivalent to median).
- **Minimum** – smallest value found in all observations.
- **Maximum** – largest value found in all observations.
- **Proportion** – show counts for certain values in the value set as a fraction or percentage of the total for the variable.

It is assumed that users know the meaning and relevance of any statistics that are selected. They are only meaningful for data that represents a true numeric value, e.g., age, income, hectares, etc.).

Many of the statistics above have additional parameters that can set by selecting them and clicking on the "Options" button. These options are documented in Tally Attributes for a Variable.

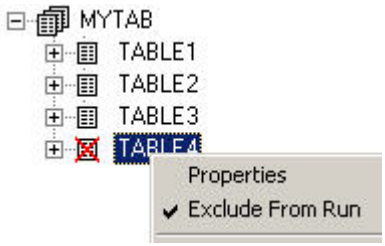
Note that in the case of grouped or continuous variables, the **median**, **n-tiles** and **mode** depend highly on the value set used. This is because these statistics are calculated using the frequency distribution for the value set rather than on the raw data itself. The **mode** will be the category in the value set that contains the most observations. The **median** is calculated by interpolation using the limits of the category in the value set within which the cumulative frequency reaches 50%. This means that the **mean**, **mode** and **n-tiles** will be more accurate for value sets with more smaller categories rather than fewer large categories. For best results, use value sets with a large number of uniform categories when calculating **median**, **mode** and **n-tiles**. For example, using single year of age or 5 year age groups rather than 15 or 20 year age groups will result in a more accurate **median** age. Using a single grouping for age (for example one category for 0-115 years) will result in a highly inaccurate result. Note that for **median** and **n-tiles**, it is possible to set groupings for the median/n-tile calculation which are different from the categories in the value set in order to get a more accurate result. See Tally Attributes for a Variable for details.

See Also: Implications of Data Dictionary Value Sets

Include/Exclude Tables from Run

By default, all the tables you define in a Tabulation Application will be produced when you run the application. However, there are times when you may not want all the tables to be produced. For example, if you already have fully tested many tables and you add another table that you wish to test, you may want to run only that table in order to save time. In this case you would exclude all the other tables from the run.

To exclude a table from the run, first make sure the table is selected and appears on the screen. From the **Edit** menu, select **Exclude Table from Run**, or right-click on the table name on the table tree on the left and select **Exclude from Run**. The icon next to the table in the tree on the left will show a red X when the table is excluded.



If a table is already excluded, follow the same steps to change the setting back to included in the run. The red X will disappear.

To exclude all the tables but the one currently selected, select the **Exclude All But This** option. This is a useful during testing when you want to run only one table at a time.

Debug Table Totals

At times, the totals in a table may differ from totals calculated using other methods. There are some easy ways to figure out the cause of such discrepancies.

Some values that exist in your data file may not be in the value set and thus these values will not be counted in the totals. For example, a value set might include the numbers 1 through 6. The data file might have a value of 7 or 8 for the variable or in some cases might contain blank (notappl) values for the variable. To see if this is the case, try including custom special values in your table. Make sure to check the boxes for the special values **notappl**, **undefined**, and **default**. Run the table again and check to see if any of the rows/columns for special values have counts in them.

Table 1. Place of birth by Sex

Place of birth	Sex					
	Total	Male	Female	Default	NotApplicable	Undefined
Total	24,271	11,787	12,484	-	-	-
Popstan	23,350	11,325	12,025	-	-	-
Endar	547	273	274	-	-	-
Victoria	122	60	62	-	-	-
Middle East	-	-	-	-	-	-
Asia	56	28	28	-	-	-
Europe	78	39	39	-	-	-
Default	-	-	-	-	-	-
NotApplicable	-	-	-	-	-	-
Undefined	118	62	56	-	-	-

If you have counts for the special value **undefined** then there are values in your data file that are not in the value set that you are tabulating. You may either edit the data file to eliminate such values or change your value set to include them. To find the actual values that are not in the value set, use the Tabulate Frequencies tool with the Each value found option.

If you have counts for the special value **notappl** then there are blanks in your data file for the variable in question. You will either need to impute the blanks or add an entry for notappl to your value set.

If you have counts for the special value **default** then there are values in your data file that cannot be read by CSPro. This can be because the properties of the variable in the dictionary do not match the data in the file. For example if the data type in the dictionary is numeric but the value in the data file contains non-numeric characters or if there is a decimal character in the data file but none specified for the item in the dictionary. It can also be because when the data file was written the value of the variable overflowed the length specified for the variable in the dictionary.

You should examine the data file and the dictionary file for discrepancies and also check any programs that have written to the data file to see if they could be writing **default** values.

There may be some values in your value set that are repeated and are being counted in more than one row or column of the table. For example, in the following value set, age 5 is in both the 0-5 and the 5-10 ranges and therefore each person with and age of 5 will be counted twice: once in the 0-5 category and once in the 5-10 category. The total will be the correct count of the number of persons, however, the sum of the age categories will not match the total.

Value Set Labe	Value Set Name	Value Label	From	To	Special
Age	P04_AGE_VS1				
		0 to 5 years	0	5	
		5 to 10 years	5	10	
		10 to 15 years	10	15	

Check your value set to make sure that no single value is repeated and that no ranges overlap. Of course, in some cases, you may want to have overlapping ranges in your value set, for example if you want to show subtotals.

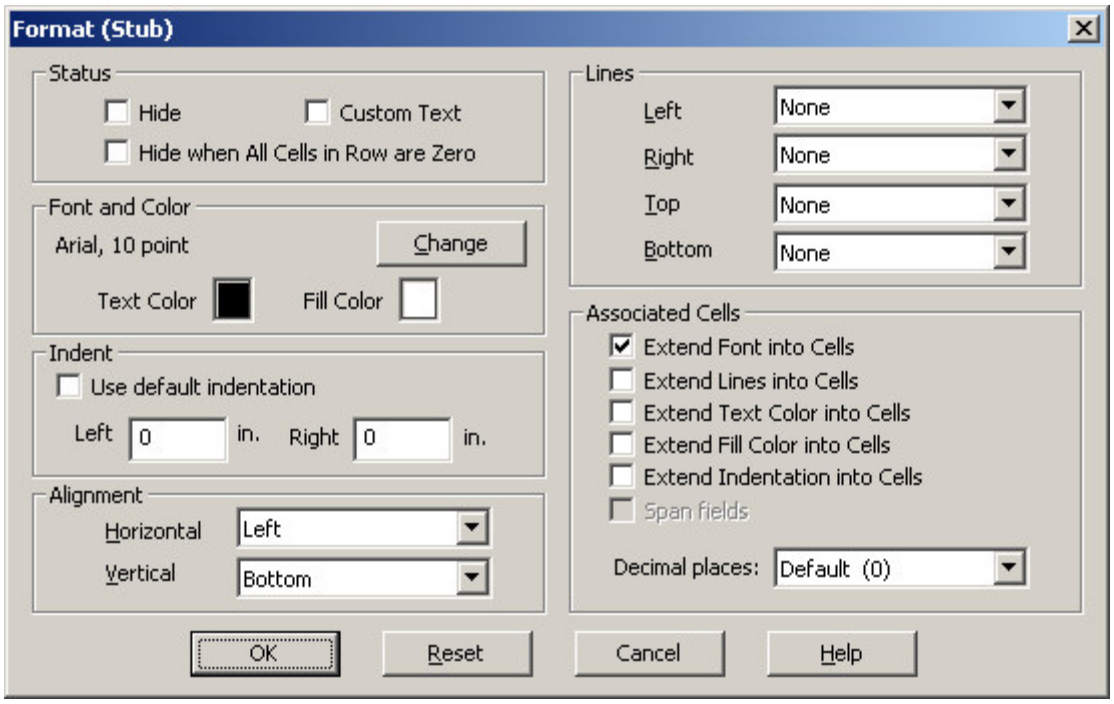
See Also: Implications of Data Dictionary Value Sets, Special Values

Formatting Tables

Formats for a Part of a Table

The Format (Table Element) Dialog Box allows you to change the appearance of individual elements of a table such as stubs, column heads, data cells, etc.

To bring up the Format (Table Element) Dialog Box, right-click on the element in the table that you wish to modify and choose Format (<element name>) where <element name> is the type of table element that you clicked on (title, stub, column head, etc…).



The Format (Table Element) Dialog Box has the following settings:

Hide: When checked, the selected element will not be displayed in the table. If a column head or row stub is selected, then the associated column or row will also be hidden. Note that stub heads, titles, and data cells may not be hidden.

Hide when All Cells in Row are Zero: Available only for stubs. When checked, if all cells in the row contain the value zero, then the row will be hidden.

Custom Text: When checked, the default text generated by CSPro for the selected element will not be displayed. You may add your own text for the selected element by double-clicking on the element and typing in the new text. Note that if you double-click on a table element and add your own text without first checking the custom text box, CSPro will automatically check it for you. You can uncheck this box to replace your new text with the default CSPro-generated text.

Font: Displays the current font and font size for the selected element. You may switch to a different font and/or font size by clicking on the change button.

Text Color: Shows the current color used to display the text of the selected element. To change the text color, click on the square displaying the current color.

Fill Color: Shows the current color used to display the cell background of the selected element. To change the fill color, click on the square displaying the current color.

Indent: Allows you to set the indentation of the selected element from the right and left edges of the cell. To set the indentation for the selected element, first uncheck "use default indentation" and then modify the numbers for "Left" and "Right" indentation appropriately. Note that columns in the table will automatically be made wider to accommodate indentation.

Alignment: Allows you to set the horizontal and vertical justification of the selected element within its cell. Using the horizontal alignment control, elements can be right-justified, left-justified or centered. Using the vertical alignment control, elements can be top-justified, bottom-justified or centered (middle).

Lines: Allows you to set the borders on the top, left, bottom and right sides of the selected element. Each side of the element may have no border, a thick border or a thin border. Note that lines may not be modified for an individual data cell. Instead you should set the lines for a whole row or column by setting the lines for the associated stub or column head and checking the "extend lines into cells" checkbox (see below).

Extend Font into Cells: When this is checked for a column head or stub, the font and font size for the selected element are applied to all cells in the associated column or row. For a stub, all cells in the row of the stub will share the font of the stub. For a column head, all cells in the column underneath the head will share the font of the column head.

Extend Lines into Cells: When this is checked for a column head, spanner, stub or caption, the lines (borders) for the selected element are applied to all cells in the associated column(s) or row. For a stub or caption, all cells in the row of the stub or caption will share the top and bottom line settings of the caption. For a column head, all cells in the column will share the left and right line settings of the column head. For a spanner, the cells in the rightmost column underneath the spanner will share the right line setting of the spanner and the cells in the leftmost column underneath the spanner will share the left line setting of the spanner.

Table 1. Age in 10 year groups by Marital status

Age in 10 year groups	Marital status					
	Total	Married	Divorced	Separated	Widowed	Never Married
Total						
0 to 9 years						
10 to 19 years						
20 to 29 years						
30 to 39 years						
40 to 49 years						
50 to 59 years						
60 to 69 years						
70 to 79 years						
80 to 89 years						
90 years and over						

(Thick lines extended into cells from a stub ('30 to 39 years') and a column head ('Separated'))

Note that the lines between rows and columns are shared between multiple elements. For example, the vertical line between two columns is both the left edge of one column and the right edge of the other

column. It may also be the edge of one or more spanners. At times, you may need to change the line setting in more than one of these elements in order to change a line. For example, in order to make the vertical line between two columns disappear, you may have to set the left line of one column to ‘none’ and the right line of the other column to ‘none’. You may also have to set the left or right line of one or more spanners to ‘none’.

Extend Text Color into Cells: When this is checked for a column head or stub, the text color for the selected element is applied to all cells in the associated column or row. For a stub, all cells in the row of the stub will share the text color of the stub. For a column head, all cells in the column underneath the head will share the text color of the column head.

Extend Fill Color into Cells: When this is checked for a column head or stub, the background color for the selected element is applied to all cells in the associated column or row. For a stub, all cells in the row of the stub will share the fill color of the stub. For a column head, all cells in the column underneath the head will share the fill color of the column head.

Extend Indentation into Cells: When this is checked for a column head or stub, the indentation setting of the selected element is applied to all cells in the associated column or row. For a stub, all cells in the row of the stub will share the indentation of the stub. For a column head, all cells in the column underneath the head will share the indentation of the column head.

Span Cells: This setting applies only to captions. When this is checked for a caption, the caption spans the entire row. By default, captions are placed in the first column of the table along with the stubs, and the cells containing them are the same size as the cells containing the stubs. This means that there are empty cells in all other columns of the caption row. When "span cells" is checked, the empty cells in the caption row are removed and the caption is placed in one large cell that spans the entire row. This allows you, for example, to center or right-justify the caption in the entire row (by setting the alignment).

Decimal Places: Sets the number of decimal places used to display numeric values for the selected element. If applied to a data cell, this sets the number of decimal places used for that individual cell. If applied to a column head or stub it sets the number of decimal places for the entire row or column. This may only be applied to data cells, column heads and stubs.

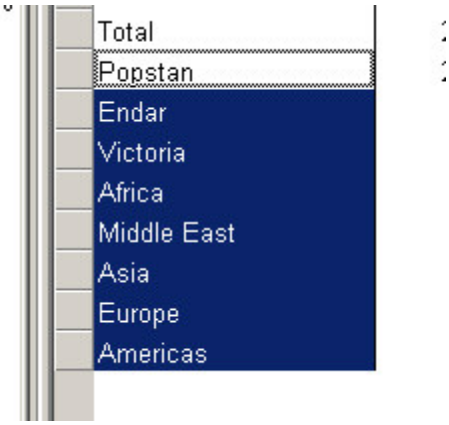
Formatting Row, Column, or Cell Data

Rows, columns and data cells are formatted in the same way as other parts of a table. Many formats, such as fonts, colors, indentation and lines, can be applied either to individual data cells or to entire rows or columns. To format an entire row, right-click on the corresponding stub and choose "Format (Stub)". To format an entire column, select the corresponding column head and choose "Format (Column Head)". Once you have changed the formats, make sure that "extend into cells" is checked in the Format (Table Element) dialog for the formats that you have modified. For example, if you wish to change the font for an entire row, make sure that "Extend font into cells" is checked for the stub, otherwise the font change will only apply to the stub itself and not to the rest of the row. See Formats for a Part of a Table for details on the Format (Table Element) dialog.

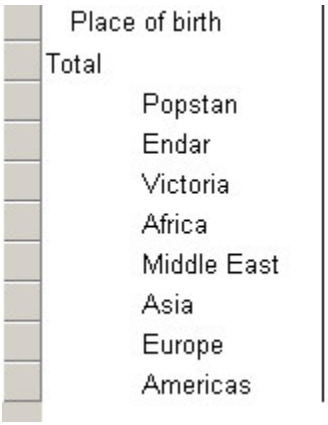
You can select multiple rows (stubs) or columns (column heads) to format. This is very useful to modify decimal places, indentation or specific attributes of specific groups of rows or columns; for example, to make ‘total’ rows bold.

Multiple selection is done in the usual Windows manner by left-clicking the mouse while using the Shift or Ctrl key. Using the Ctrl key adds each cell in the table that is clicked on to the selection. Using the shift key adds all the cells in between the two cells clicked on to the selection. Although you may select different types of cells at the same time (data cells, stubs, column heads), you can only format selected groups when all selected items are of the same type (for example, all selected items are stubs).

In this example multiple stubs have been selected but ‘Total’ is not.



Use the right-click to access the Format (Stub) menu. Change the indentation to .5 in. then select OK. The result is below.



Similar actions can be performed on columns by selecting multiple column heads. You can also select multiple captions or spanners.

Although the formatting option is available for each data cell, it should only be used in the rarest of cases.

When working with sub-groupings (one variable dropped on top of another), any formatting applied to one row or column in a sub-grouping will automatically be applied to the corresponding row or column in all of the other sub groupings. In the example below, changing the font for the column "Male" under the spanner "Total" also changes the font for the "Male" column under the "Urban" and "Rural" spanners.

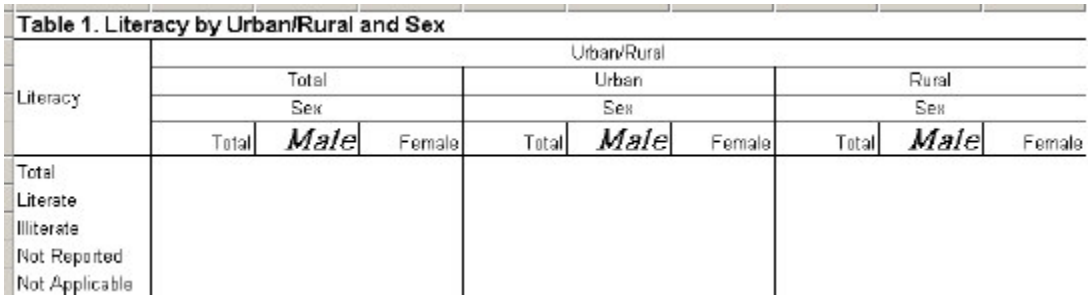


Table 1. Literacy by Urban/Rural and Sex

Literacy	Urban/Rural								
	Total			Urban			Rural		
	Sex			Sex			Sex		
	Total	*Male*	Female	Total	*Male*	Female	Total	*Male*	Female
Total									
Literate									
Illiterate									
Not Reported									
Not Applicable									

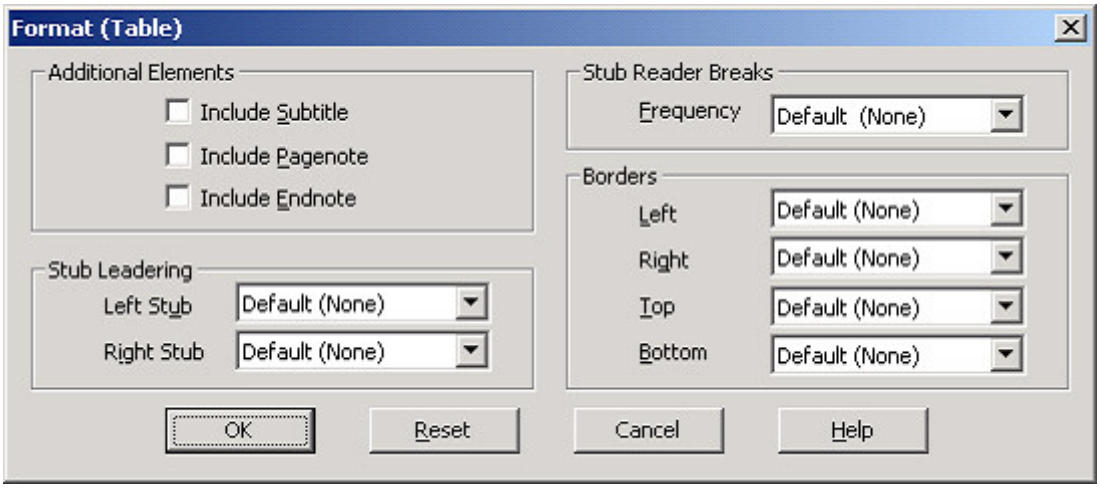
You cannot change the format for one row or column in only one sub-grouping but not the others. You can, however, achieve the same result using multiple subtables with universes.

See also: Change the Way Numbers are Displayed

Formats for a Table

The Format (Table) Dialog Box allows you to control the overall appearance of a table.

To bring up the Format (Table) Dialog Box, right-click anywhere on the table and choose "Format (Table)", or choose "Format (Table)" from the Edit menu. Any changes made will apply only to the current table.



The Format (Table) Dialog Box has the following settings:

Include Subtitle: When checked, an additional row below the title row is added to the table. You may enter a subtitle to the table by double-clicking on the new row and typing in the text for the subtitle.

Include Pagenote: When checked, an additional row is added at the end of the table. You may enter a pagenote for your table by double-clicking on the new row and typing in the text for the pagenote. Pagenotes are displayed at the bottom of each page of the table.

Include Endnote: When checked, an additional row is added at the end of the table. You may enter an endnote for your table by double-clicking on the new row and typing in the text for the endnote. Endnotes are displayed only once per table, at the bottom of the last page of the table.

Stub Reader Breaks: Sets the frequency of reader breaks in the rows of the table. Reader breaks are blank rows used to break up the table. By default there are no reader breaks. The frequency of reader breaks may be set anywhere from 1 (every other row is blank) to 10 (every 10th row is blank). Note that you can hide individual reader breaks by right-clicking on them and selecting Format (Stub). This allows for finer control over placement of reader breaks.

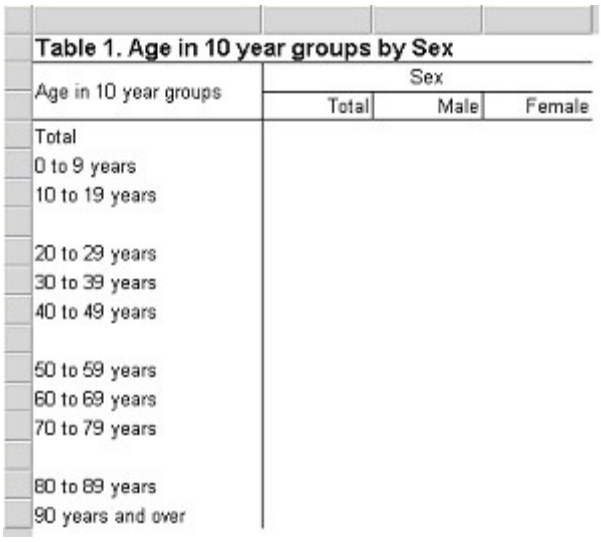


Table 1. Age in 10 year groups by Sex			
Age in 10 year groups	Sex		
	Total	Male	Female
Total			
0 to 9 years			
10 to 19 years			
20 to 29 years			
30 to 39 years			
40 to 49 years			
50 to 59 years			
60 to 69 years			
70 to 79 years			
80 to 89 years			
90 years and over			

(Stub reader break frequency set to 3.)

Stub Leadering: Allows you to set a pattern such as '….' or '_ _ _ _' to appear in each row of the table between the end of the stub text and the start of the next column. You may choose different patterns for the stubs on the left and right sides of the table. Note that stub leadering is only visible in print preview and on the printed page.

Table 1. Age in 10 year groups by Sex

Age in 10 year groups	Sex			Age in 10 year groups
	Total	Male	Female	
Total.....	24,273	11,789	12,484	---------- Total
0 to 9 years.....	7,467	3,734	3,733	---------- 0 to 9 years
10 to 19 years.....	6,217	3,000	3,217	-------- 10 to 19 years
20 to 29 years.....	3,929	1,850	2,079	-------- 20 to 29 years
30 to 39 years.....	2,809	1,362	1,447	-------- 30 to 39 years
40 to 49 years.....	1,618	797	821	-------- 40 to 49 years
50 to 59 years.....	1,085	528	557	-------- 50 to 59 years
60 to 69 years.....	726	336	390	-------- 60 to 69 years
70 to 79 years.....	328	141	187	-------- 70 to 79 years
80 to 89 years.....	80	39	41	-------- 80 to 89 years
90 years and over	14	2	12	----- 90 years and over

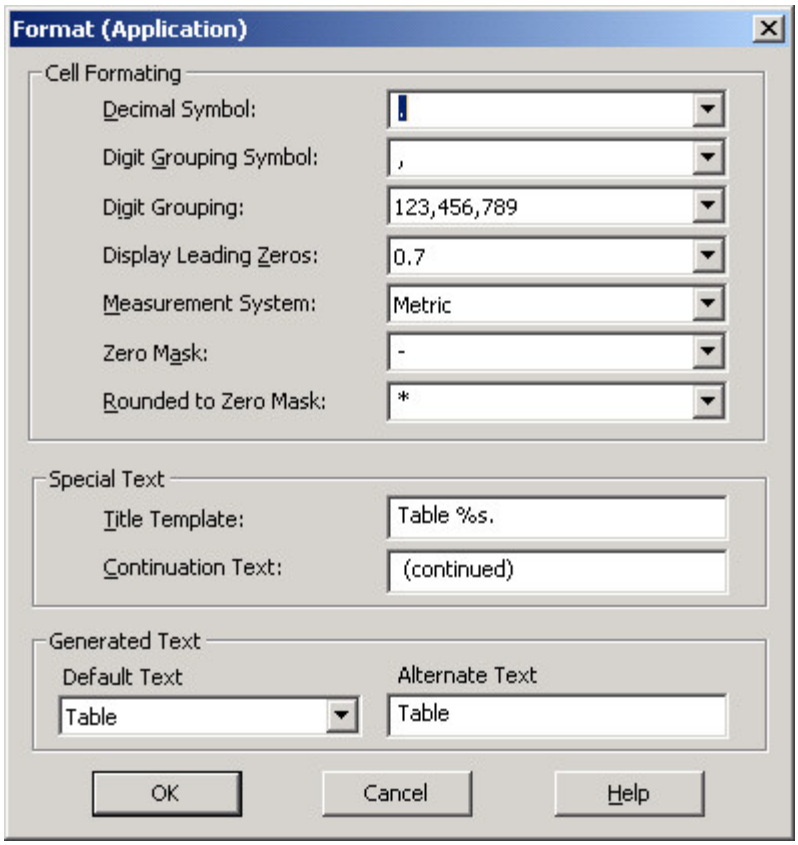
(Stub leadering on both left and right)

Borders: Allows you to set the borders on the top, left, bottom and right sides of the table. Each side may have no border, a thick border, or a thin border.

Formats for an Application

To bring up the Format (Application) Dialog Box, right-click on a table and select Format (Application), or choose "Format (Application)" from the Edit menu.

The Format (Application) Dialog Box controls the appearance of numbers and content of text labels in all tables in the file. It is particularly useful for customizing tables in languages other than English. Note that unlike Format (Table) and Format Print (Table), settings made in this dialog box apply to *all* tables in the file.



The following settings are available:

Decimal Symbol: Sets the symbol used to separate the whole and fractional parts of numbers in data cells. By default this is set to period; however, it may be set to comma or any other symbol. For example, changing the symbol from period to comma would change the display of a number from 3.14 to 3,14

Digit Grouping Symbol: Sets the symbol used to separate groups of digits in large numbers in data cells. By default this is set to comma so that numbers are displayed as 123,456,789 but it can be changed; for example, to a period so that numbers are displayed as 123.456.789

Digit Grouping: Controls how digits are grouped together to be separated by the digit grouping symbol. There are three options:

- 123,456,789 (default) – place the digit grouping symbol between each group of three digits.
- 123456789 – do not use the digit grouping symbol.
- 12,34,56,789 – place the digit grouping symbol between each group of two digits with one group of three digits at the right.

Display Leading Zeros: Controls whether or not zero should be displayed before the decimal point of a decimal number between zero and one. Leading zeros are displayed by default.

Measurement System: Allows you to switch between US (inches) and metric (centimeters) measurements for page margins. The measurement is used when specifying margins in the Format Print (Table) dialog as well as indentation in the Format (Table Element) dialog. The measurement system is set to US by default.

Zero Mask: Allows you to change the text that is displayed for data cells that contain a value of exactly zero. By default the zero mask is "-", so any data cell containing a value of zero is displayed as "-" rather than "0". If you want zero to be displayed, remove the zero mask text.

Rounded to Zero Mask: Allows you to change the text that is displayed for data cells that contain a value which is rounded off to zero because it cannot be displayed using the number of decimal places for that data cell. For example, if the value in a data cell is calculated to be 0.002 but the number of decimal places for that cell is set to 2, the rounded to zero mask will be displayed in that cell. Having the zero mask and rounded to zero masks be different makes it clear which values are exactly zero and which ones are rounded off. By default the rounded to zero mask is "*". If you want zero to be displayed, remove the rounded to zero mask text. Note that you can change the number of decimal places for a row, column or data cell by using the Format (Table Element) Dialog Box.

Title Template: Sets the text used to automatically generate the first portion of default table titles. When you create a new table, CSPro automatically gives the new table a title. By default, this title begins with "Table %s." where the table number replaces %s. You can change this by entering your own text in this edit box. If you put "%s" in your new title template, it will automatically be replaced with the table number when the table title is generated. As an example, if you put "Tableau %1" in the edit box, your table titles will begin with "Tableau 1", "Tableau 2", …

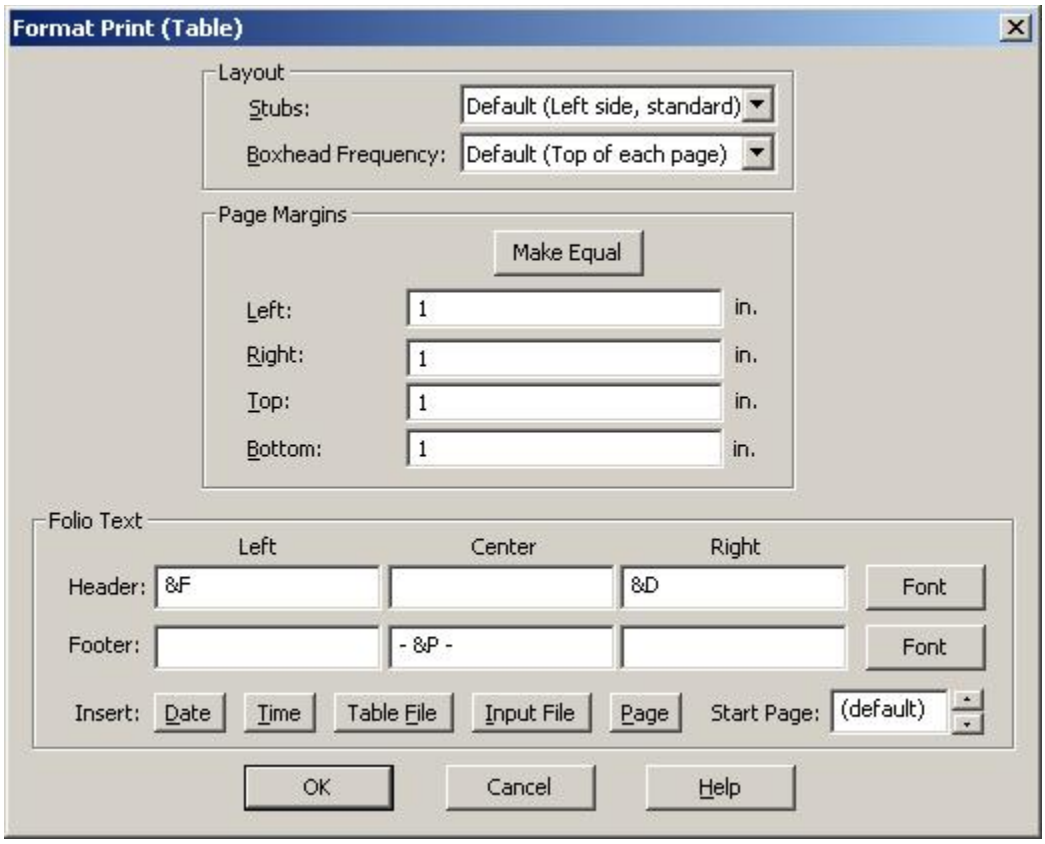
Continuation Text: Sets the text added to the end of the table title on the printed page and in print preview to indicate that the current page is not the first page of the table. By default this text is "(continued)", so that on all but the first printed page of a table, the title would be, for example, "Table 1: Age by Sex (continued)".

Generated Text: Sets other text (besides title template and continuation text) used by CSPro to generate default text in table titles, column heads, and stubs. To change a particular text string, first select the default string under "Default Text" and then enter the new string under "Alternate Text". For example, to replace the string "Percent" used in stubs and column heads with "%", first select "Percent" from "Default Text" and then type "%" into "Alternate Text". Now any rows or columns with percents in them will have "%" in the stub or column head.

Formats for Printing

To bring up the Format Print (Table) Dialog Box, either choose Page Setup from the File menu or right-click anywhere on the table and choose Format Print (Table) or choose "Format Print (Table)" from the Edit menu.

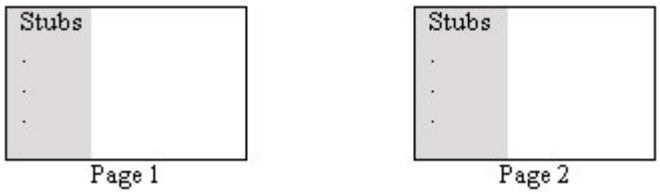
The Format Print (Table) Dialog Box allows you to change the appearance of the selected table on the printed page. Note that changes made in this dialog will only be visible when the file is printed or viewed in print preview. Changes made using Format Print (Table) only apply to the current table. To change the printed appearance of all tables in the file, use the Preferences Dialog Box available from the Edit menu.



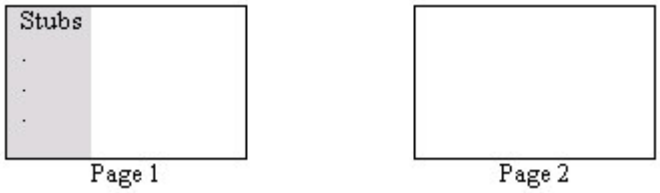
The Format Print (Table) Dialog Box has the following settings:

Stubs: Controls where stubs will be placed on the printed page. There are four options:

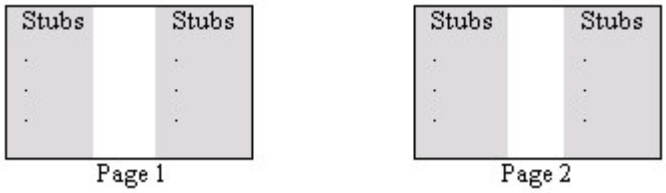
- Left side, standard (default): place stubs along the left side of every page. If the table is wide enough so that some of the columns go onto a second page, stubs will be placed on the left sides of both pages.



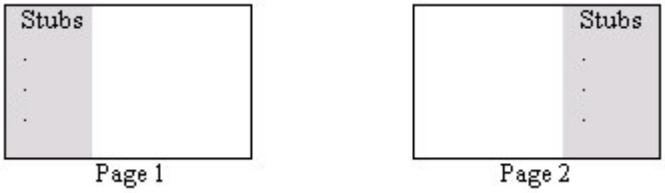
- Left side, facing pages: place stubs along the left side of the page but only for the first page of facing pages. If the table is wide enough so that some of the columns go onto a second page, no stubs will be placed on the second page. If the table columns all fit on one page then this is the same as Left side, standard.



- Left and right sides, standard: place stubs on both the left and right sides of the page for every page. If the table is wide enough so that some of the columns go onto a second page, stubs will be placed on the left and right sides of both pages.



- Left and right sides, facing pages: place stubs along the left side the first page of facing pages and on the right side of the second page of facing pages. If the table is wide enough so that some of the columns go onto a second page, stubs will be placed on the left side on the first page and the right side of the second page. If the table columns all fit on one page then this is the same as Left side, standard.



Boxhead Frequency: Controls where the boxheads will be placed on the printed pages. The box head is the area at the top of the table that contains the stub heads, column heads and spanners. You can set the boxhead frequency to any of the following:

- Top of each page (default): Places a boxhead at the top of every page of the table.
- No boxheads: No boxheads will be placed at all.
- Top of table only: Only place the boxhead on the first page of the table and not any of the other pages of the table.

Page Margins: Allows you to set the top, bottom, left and right margins of the printed pages for the selected table. By default, margins are measured in inches although you can change it to centimeters by choosing "metric measurement" in the Format (Application) Dialog Box. Use the "Make Equal" button to set all four margins to the currently selected number.

Folio Text: Allows you to set header and footer text for each page of the selected table. The header and footer each contain left, center and right portions that can be set separately by entering the desired text in the corresponding edit box.

You can modify the fonts used for the header and footer by clicking the Font buttons. The top Font button applies to the header and the bottom button applies to the footer.

You can place the current date, current time, page number and file name(s) in any of the header or footer sections by clicking on the appropriate buttons below the header and footer edit boxes or by adding the following text into the appropriate header/footer edit box:

Text in header/footer edit box	Replaced with on printed page
&D	Current date
&T	Current time
&P	Page number
&F	Name of table file
&I	Name of data input file(s)

By default, CSPro places the filename in the left header, the date in the right header and the page number in the center footer.

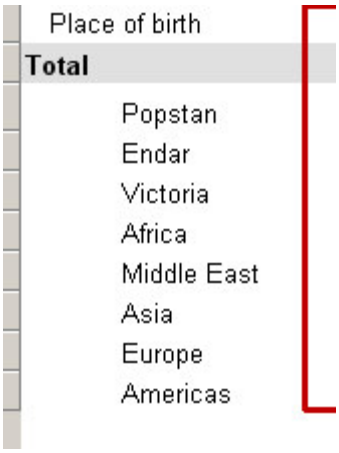
Start Page: Sets the page number to use on the first page for page numbering. If this is set to 1, then the first page of the table will be page 1; the second page will be page 2, etc… If it is set to default then the first page of the selected table will be one greater than the last page of the previous table in the file so that the numbers continue from one table to the next with no gaps. For the first table in the file, if the start page is set to default, then the page numbering for that table will start at page 1.

Views of Tables

Beside Print Preview, there are two views of every table. However, unless the table has hidden rows or columns, or multiple subtables, the two views are basically the same. One view, called the "Design view" shows the hidden rows or columns, and the other view, called the "Display" or "Data" view, does not. Users can switch between the two views by using Ctrl+D or checking/unchecking "Hidden parts" under the View menu.

Most often, users will see the Design view of the table. This is the one that shows hidden stubs or columns. Such stubs or columns are shaded in gray.

In example the "Total" stub is hidden.



This is the Display view of the same stub group.

When a table is first created, you see the design view of the table. When a table is run, the display view is automatically shown and hidden parts of the table are no longer visible. In order to switch back the design view (for example, so that you can select and unhide a part of the table that is hidden), use Ctrl+D or check "Hidden Parts" on the view menu.

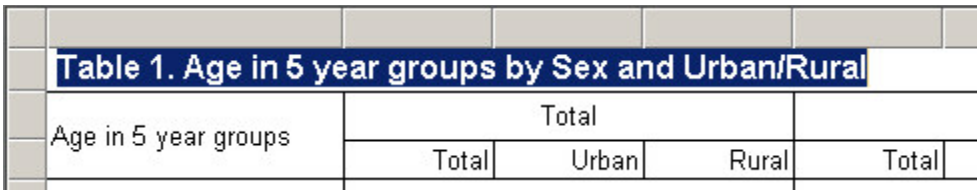
In addition to showing hidden parts, the design view also marks subtables with a colored outline while the display view does not. It is often useful to switch to the design view in order to be able to more easily set the tally attributes for a subtable. When the colored outline of subtables is displayed, you may right-click inside the outline and choose Tally Attributes (Subtable).

See Also: Using Print Preview, Hide or Show a Row or Column

How To ...

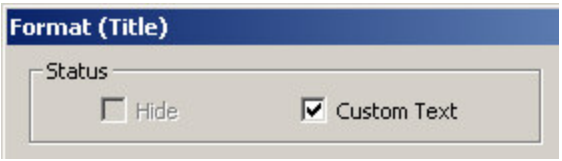
Customize Table Text

To customize the text in any table element (title, spanner, column head, stub, etc.) double click on the text you want to change. In the example below the table title has been selected.

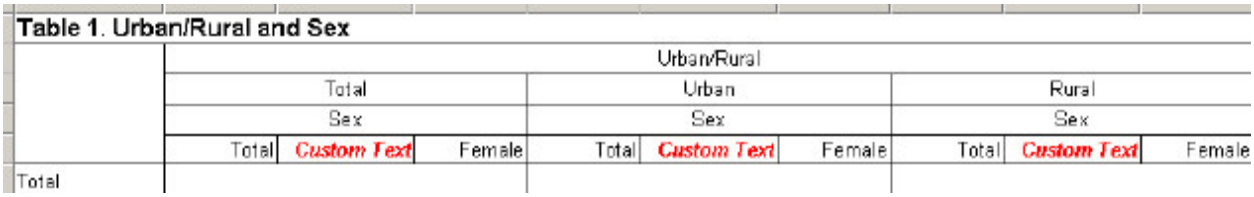


You can now change the text of the table title. If you want a second line of text, press **Ctlr+Enter** to create a new line of text.

To go back to the original default text, right click on the text you want to change, select **Format (*table part*)**, uncheck the **Custom Text** check box, then press OK.



Note that if you customize the text of a stub or column header in a sub grouping, the text will be changed for all corresponding rows or columns in the sub grouping. In the example below, the text for the column header Male has been changed in one column and the change is reflected in the other sub groupings.

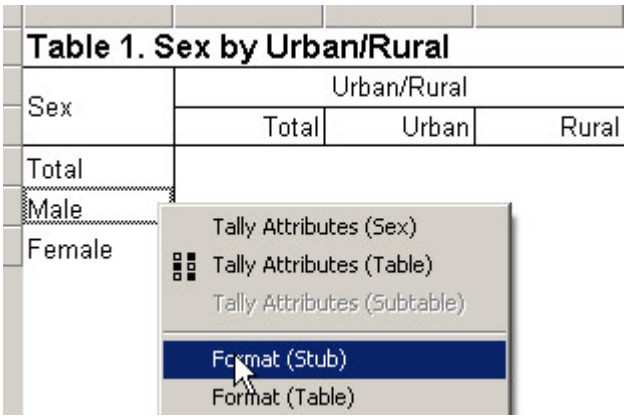


If you wish to change the text in only one sub grouping, you will have to use to multiple subtables with universes.

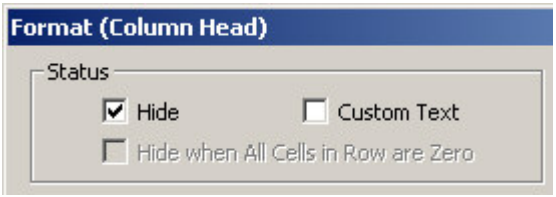
See Also: Change the Automatically Generated Text

Hide or Show a Row or Column

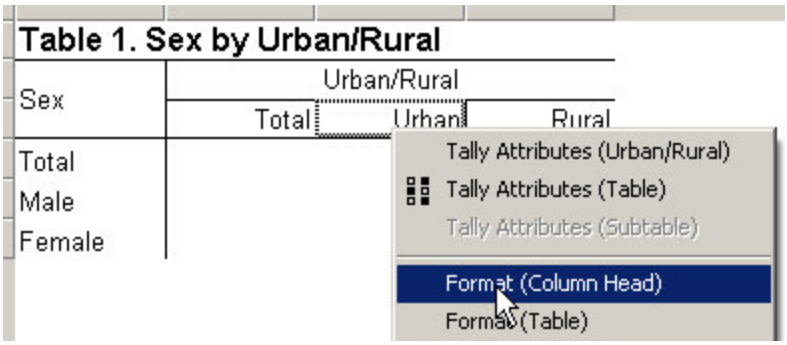
To hide a row, right-click on the stub for the row you wish to hide, select "Format (Stub)"



and check the "Hide" box.



Similarly, to hide a column, right-click on the column head for the column you wish to hide, select "Format (Column Head)"



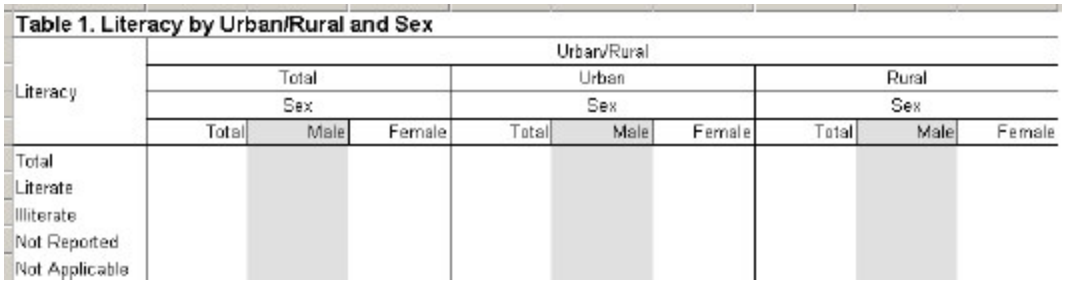
and check the "Hide" box.

You can hide multiple rows or multiple columns at the same time by first selecting the rows or columns to hide and then right-clicking and choosing "Format (Stub)" or "Format (Column Head)" as above. Multiple selection is done in the usual Windows manner by left-clicking the mouse while using the Shift or Ctrl key. Using the Ctrl key adds each cell in the table that is clicked on to the selection. Using the shift key adds all the cells in between the two cells clicked on to the selection.

Hidden rows and columns will be shown with a grey background in the design view and will not be shown at all in the display or print views or on the printed page. To show a hidden row or column, right-click on the stub or column head, choose "Format (Stub)" or "Format (Column Head)" and uncheck the "Hide" box. If you are not in the design view, you may need to first switch to the design view in order to see the row or column that you wish to unhide. To switch to the design view, use Ctrl-D or select "Hidden Parts" from the View menu.

When working with sub-groupings (one variable dropped on top of another), hiding or showing a row or column in one sub-grouping will automatically hide or show the corresponding row or column in all of the

sub groupings. In the example below, hiding the column "Male" under the spanner "Total", also hides the "Male" column under the "Urban" and "Rural" spanners.



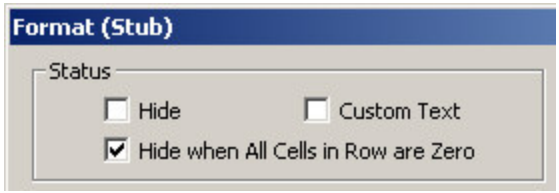
You cannot hide a row or column in only one sub-grouping but not the others. You can, however, achieve the same result using multiple subtables with universes.

To automatically hide rows when all data in the row is zero see Hide Rows Containing All Zeros.

See Also: Formats for a Part of a Table, Formatting Row, Column, or Cell Data, Hide or Change the Position of the Total

Hide Rows Containing All Zeros

It is possible to have CSPro automatically hide rows when all the cells in the row contain the value zero. This is useful when many rows in a table will have only zeros and you wish to view only the rows that have non-zero data.



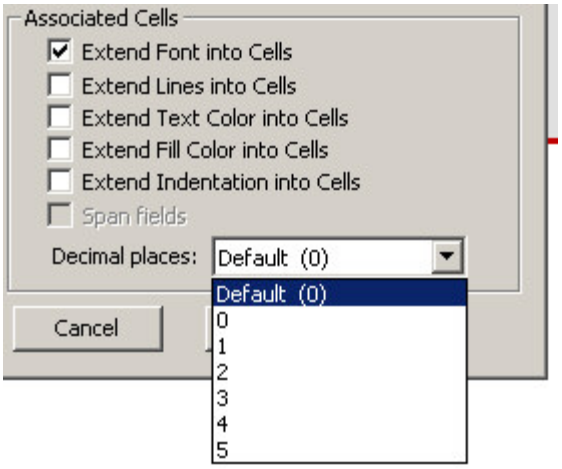
To have CSPro automatically hide a row if all values in the row are zero, right-click on the stub for the row and choose "Format (stub)". In the "Format (stub)" dialog, check the box "Hide when All Cells in Row are Zero". When you run the table, the row will be hidden if all cells are zero. To view the row again, in order to able to select it, switch to the design view. To switch to the design view, use Ctrl-D or select "Hidden Parts" from the View menu.

You can hide multiple rows if they contain all zeros at the same time by first selecting the rows and then right-clicking and choosing "Format (Stub)" as above. Multiple selection is done in the usual Windows manner by left-clicking the mouse while using the Shift or Ctrl key. Using the Ctrl key adds each cell in the table that is clicked on to the selection. Using the shift key adds all the cells in between the two cells clicked on to the selection.

Change the Number of Decimal Places Displayed

You can change the number of decimal places displayed in an individual data cell or for all cells in a row or column. To change the number of decimal places in a data cell, right click on the cell and choose Format (Data Cell). For rows, right click on the Stub and choose Format (Stub). For columns, right click on the column head and choose Format (Column Head).

In all cases, this brings up the Format (table part) dialog. Simply select the number of decimal places for the associated cells, i.e., the row, the column, or the data cell and click **OK**.



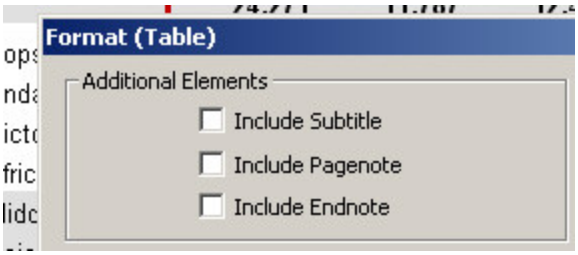
Rows or columns containing percentages are automatically designated as having one decimal place.

If you change the number of decimal places in a data cell to a number other than the default setting (indicated by the word "Default" in the pulldown), then the setting for the data cell will override any settings made for rows or columns containing that data cell. For example, if the data cell is set to one decimal place, and you change the row to have two decimal places, the data cell will still have one decimal place although other cells in the row will change. In order to make the number of decimal places in the data cell follow the number of decimal places set in the row or column, change the setting in the data cell to the default.

You can change the default number of decimal places for data cells in the preferences.

Add a Footnote (Pagenote or Endnote)

To add footnote to a table, right-click anywhere on the table and choose Format (Table) or choose Format (Table) from the Edit menu to launch the Format (Table) dialog.



Click to add a check mark to either the pagenote or endnote and then click **OK**. This will add a new row at the bottom of the table. Double click on this new row and type in the text for your footnote.

A pagenote will be displayed at the bottom of every page when the table is printed while an endnote will only be displayed on the last page of the printed table.

To remove a pagenote or endnote, uncheck the corresponding box in Format (Table).

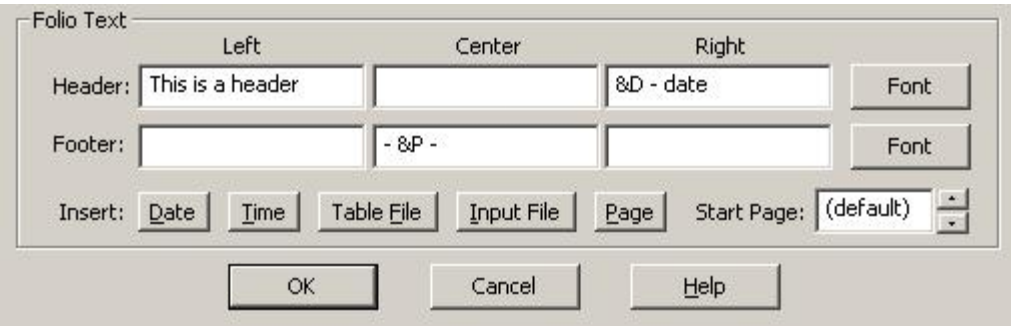
Format (Table) allows you add a pagenote or endnote to an individual table. To add an endnote or pagenote for all tables in a file, use the preferences.

See also: Add Header/Footer Text to a Table

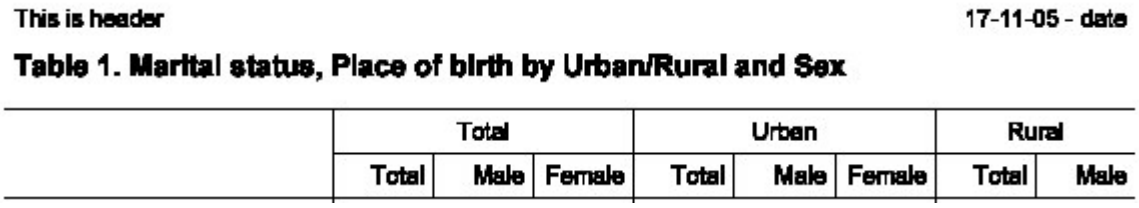
Add Header/Footer Text to a Table

To add header or footer text to a table, right click anywhere on the table and choose Format Print (Table) or choose Format (Table) from the Edit menu to launch the Format Print (Table) dialog.

This dialog has text boxes to enter left, right and center text for both the header and footer. "Header" refers to common text at the top of every printed page. "Footer" refers to common text at the bottom of every printed page. Left, Center, and Right refer to areas of the printed page.



The above options produce the following header:



The six text boxes can be filled with any text that you type. In addition, you can choose to display any combination of date, time, page number and file name in the header and/or footer by adding one of the following predefined text strings:

Text in header/footer edit box	Replaced with on printed page
&D	Date on which the tables were run
&T	Time at which the tables were run
&P	Page number relative to start page
&F	File name of the table file
&I	File name of the data input file(s)

To place predefined text in a text box, you can either type it directly or click in the text box where it should go and then click the button corresponding to the desired item. Predefined text strings may be mixed with other text, as in the right header of the example above. To delete text in the header or footer, delete it from the corresponding edit box.

Separate fonts are available for headers and footers. You can set these fonts using the font button to right of the text boxes.

If page numbering is used you may set the starting page number, the number to use for the first page of the table, in the "Start Page" text box. By default, the starting page number will be 1 for the first table in the file and for all subsequent tables in the file it will be one greater than the last page of the preceding table.

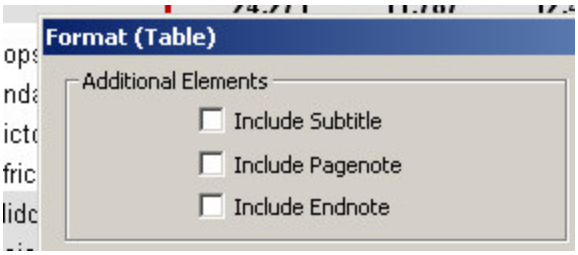
Using the Format Print (Table) dialog will set the header and footer text for the current table. You may set the header and footer text for all tables in the file using the preferences.

Note: Headers and Footers are visible only in Print Preview and on the printed page.

See also: Add a Footnote (Pagenote or Endnote)

Add a Subtitle

To add a subtitle to a table, right-click anywhere on the table and choose Format (Table) or choose Format (Table) from the Edit menu to launch the Format (Table) dialog.



Click to add a check mark next to "Subtitle" then click OK. This will add a new row in the table directly underneath the title. Double click on this new row and type in the text for your subtitle. You can then change the format (font, color, alignment, etc…) of your subtitle by right-clicking on it and choosing Format (Subtitle).

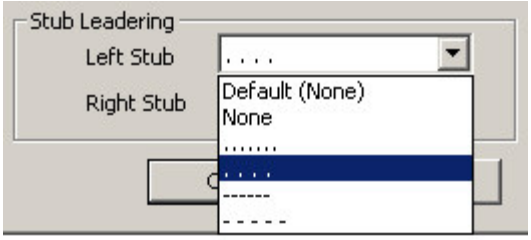
To remove the subtitle row, uncheck the corresponding box in Format (Table).

Add Stub Leadering

Stub leadering refers to text patterns such as "….." or "-----" that are placed between the text of a stub and the first column of the table.

Marital status	
Total.....	24,2
Married(code 1).....	6,9
Divorced(code 2)	7
Widowed(code 3).....	7
Never Married(code 4).....	15,6
Place of birth	
Popstan.....	23,3
Endar.....	5
Victoria.....	1
Africa	

By default, no stub leadering is displayed. To add stub leadering to a table, right-click anywhere on the table and choose Format (Table) or choose Format (Table) from the Edit menu to launch the Format (Table) dialog.



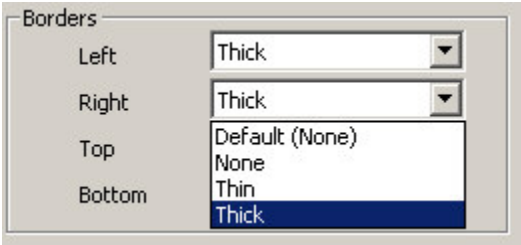
Choose the desired pattern from the Stub leadering pulldown menu. If you have chosen to display both left and right stubs, you can set different leadering patterns for each one.

Note: Stub leadering is only visible in Print Preview and on the printed page.

Using the Format Print (Table) dialog will stub leadering for the current table. You may set stub leadering for all tables in the file using the preferences.

Add Borders

To add borders to a table, right-click anywhere on the table and choose Format (Table) or choose Format (Table) from the Edit menu. This will launch the Format (Table) dialog.



Borders outline the table on the printed page. Different borders are available for each side of the table.

The options are:
None – no border on this side.
Thin – a normal line border on this side.
Thick – a "double" line border on this side.

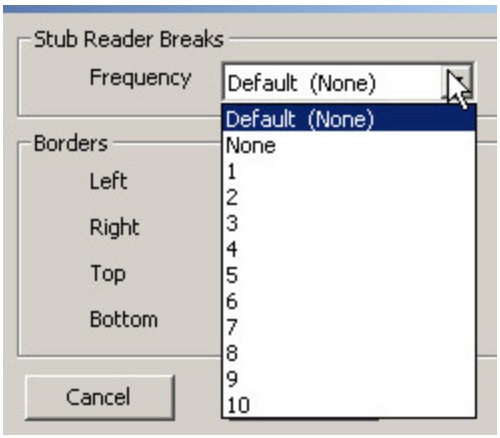
Using Format (Table) will set the borders for the current table. To set the borders for all tables in the file, use preferences.

See Also: Add Borders to Cells

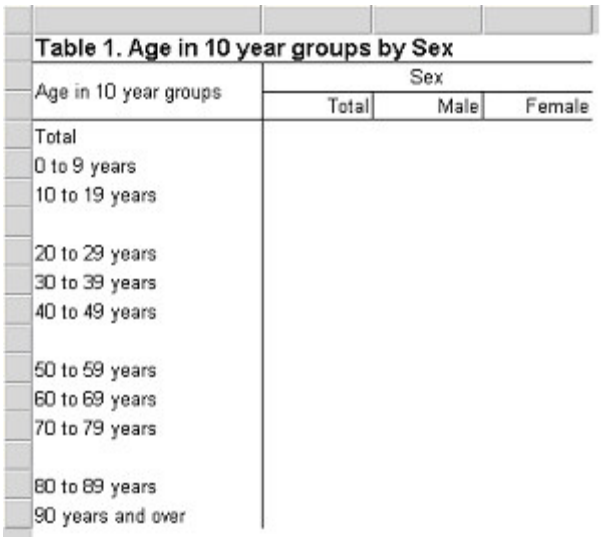
Add Reader Breaks

Reader breaks separate rows of data into readable groupings. This is done by inserting blank rows in the table at the specified intervals.

To add reader breaks to a table, right-click anywhere on the table and choose Format (Table). This will launch the Format (Table) dialog.



Select the number of lines (intervals) for each group.



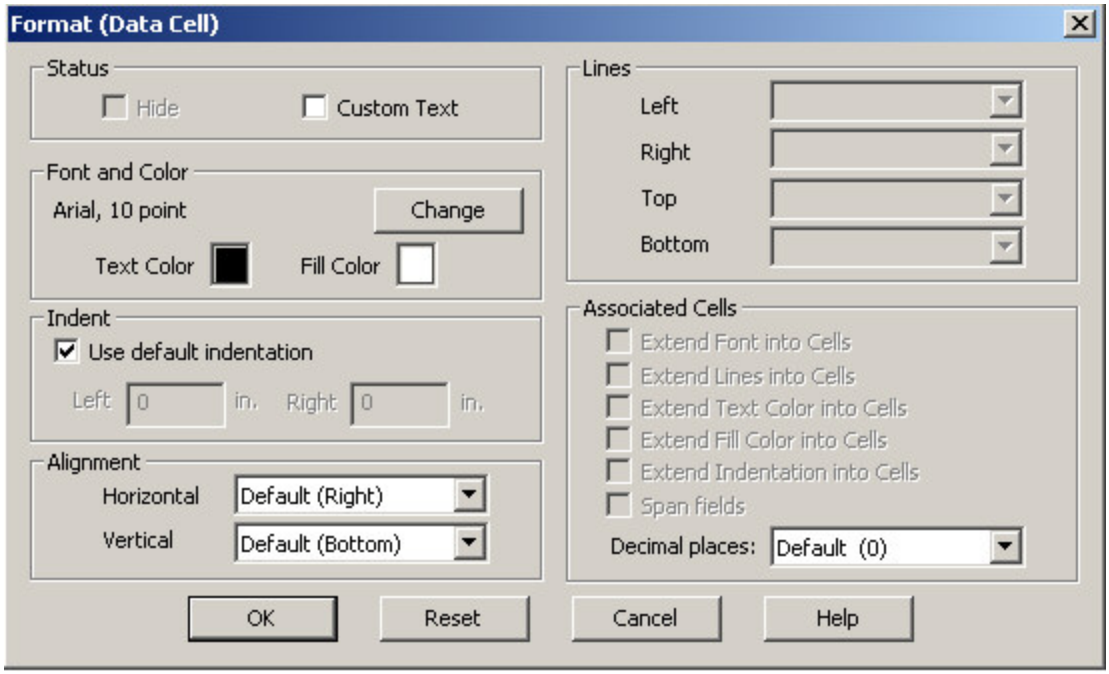
(Stub reader break frequency set to 3.)

The frequency of reader breaks may be set anywhere from 1 (every other row is blank) to 10 (every 10th row is blank). Note that you can hide individual reader breaks by right clicking on them and selecting Format (Stub). This allows for finer control over placement of reader breaks.

Note: Reader breaks apply to the entire table. Users may need to do additional spacing in the table rows to gain the desired result.

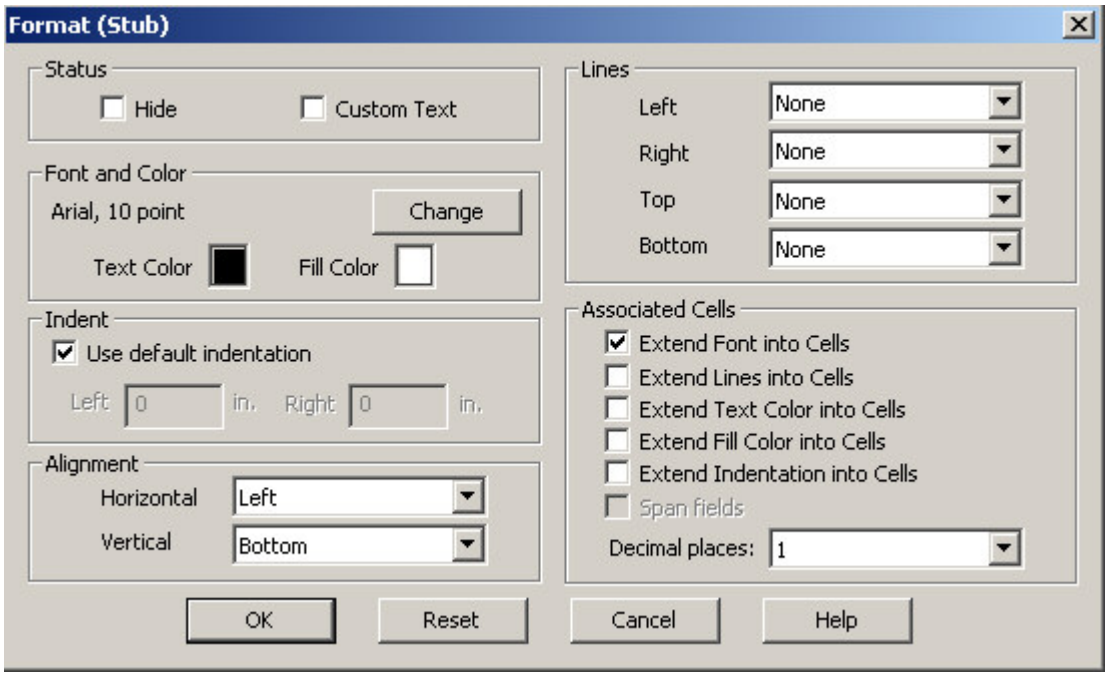
Change the Way Numbers are Displayed

If you want to modify the display of a single data cell or a group of unassociated data cells this is done by selecting the individual cells, right clicking, and using Format (Data Cell).



Individual Data cells cannot be hidden but you can replace their contents with Custom Text. Of course, the number of decimal places, fonts, alignment, and indentation can be changed using the options in the Format (Data Cell) dialog above.

You can hide or otherwise modify entire rows or columns of numbers. Select the rows or columns to be modified (multiple selection is done using Shift and/or Ctrl keys), right click and use **Format (Stub)** for rows or **Format (Column Head)** for columns.



The settings that have an effect on the numbers in the rows or columns associated with the stub or column head are in the "Associated Cells" section. These check boxes allow you to extend the format of the stub/column head into all the cells in the corresponding row/column. Of course, the number of decimal places affects only the data cells themselves and not the column head or stub.

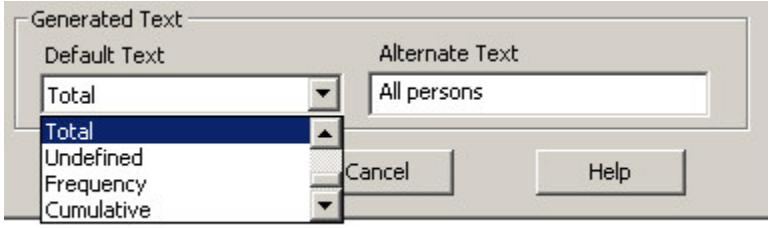
See Also: Formatting Row, Column, or Cell Data, Hide or Show a Row or Column

Change the Automatically Generated Text

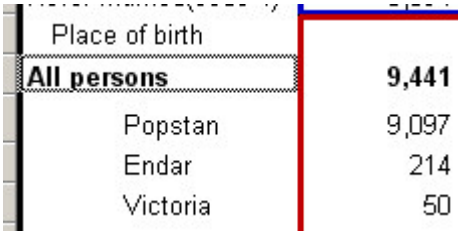
Most text in tables comes from value labels in the value sets for the variables that you drop onto the table. Some additional text, such as "Total", "Percent" and "by", is generated by the software. You can customize the text that the software will use for all of these terms.

This option is controlled through the Format (Application) dialog. Choose Format (Application) from the edit menu or right-click anywhere on the table and choose Format (Application).

Find a list of the words used in the generated text at the bottom of the menu. Select the "default text" which is to be replaced and enter the "alternate text" or replacement text.



Every occurrence of the default text will be replaced by the alternate text. In the example, every "total" text in every table will be replaced by "All persons".



If the text is not appropriate in certain places it can be replaced by "custom text ".

Note: Any text can always be replaced by custom text.

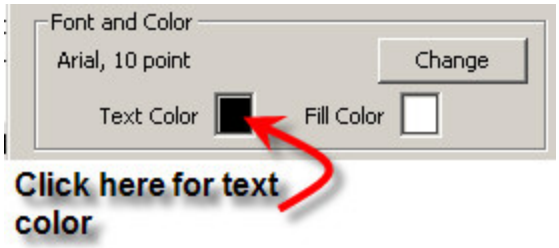
Changing the automatically generated text is particularly useful when generating tables in languages other than English. You can translate all of the automatically generated text into your language in the Format (Application) dialog and will be updated in all tables in your file.

Change Fonts or Colors

There is no "global" way to change fonts or colors in a table. The formats of the individual elements, e.g., Title, Spanner, Caption, etc., must be individually modified to replace the ‘default’ fonts and colors. To do so, right-click on the element and choose Format (<name of table part>) from the menu to bring up the Format (Table Part) dialog.

There is also the option to change the default formats for each of these elements. The modifications will take effect immediately in ALL the tables. See Preferences and Default Formats for more information.

In both circumstances, specific or default formats, the font and color are changed in the ‘Font and Color’ section of the format (Table Element) dialog.



To change the font, click on the ‘Change’ button and select the font characteristics from the available list.

To change the text or fill (background) click in the color box and select from the choices given.

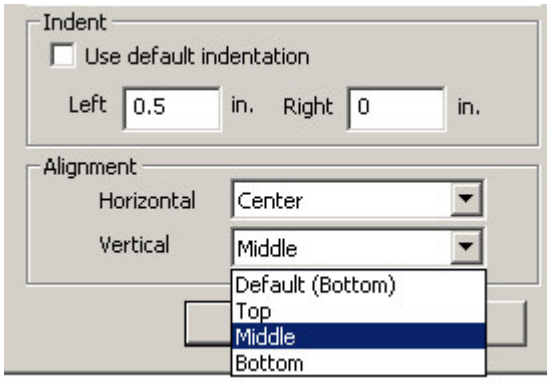
See Also: Preferences and Default Formats

Change Indentation or Alignment

There is no global way to change indentation or alignment of text. The formats of the individual elements, e.g., Title, Spanner, Caption, etc., must be individually modified to replace the 'default' alignment and indentation. To do so, right-click on the element and choose Format (<name of table part>) from the menu to bring up the Format (Table Part) dialog.

There is also the option to change the default formats for each of these elements. The modifications will take effect immediately in ALL the tables. See Preferences and Default Formats for more information.

In both circumstances, specific or default formats, alignment and indentation are changed in the 'Indent' and 'Alignment' sections of the format menu.



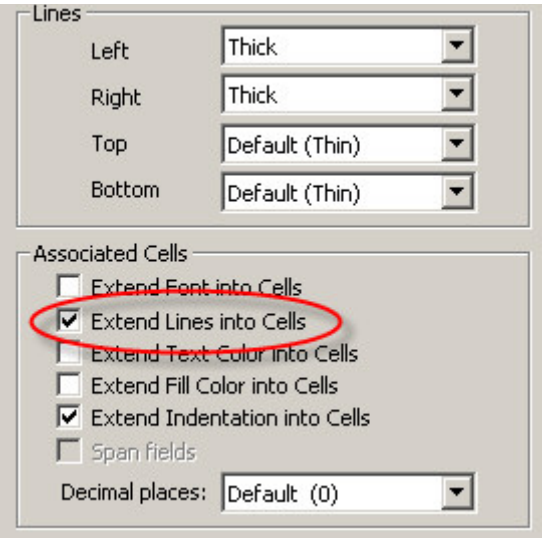
Indentation is given in inches (in.) or centimeters (cm.) depending upon the choice of Measurement System in the Format (Application) dialog. Indentation affects where in the text box, the text starts (Left) and/or ends (Right).

Alignment is the placement of text in the text box. The horizontal options are left(-justified), center, or right(-justified). Vertical choices (shown) are top, middle, and bottom.

See Also: Preferences and Default Formats

Add Borders to Cells

Individual cells have very few attributes that can be changed. Font, color, indentation, alignment, custom text, and number of decimal places are the only ones available.



Borders can only be created for cells by setting the borders on the rows and columns that contain the cell. To do this, right-click on the associated stub, caption or column head and choose Format (Stub), Format (Caption) or Format (Column Head) to bring up the Format (Table Part) dialog. For columns, set the left and right lines and for rows set the top and bottom lines. To ensure that line settings are applied to the entire row or column, make sure that **Extend Lines into Cells** is checked.

Table 2. Age in 5 year groups by Mother living

Age in 5 year groups	Mother living			
	Total	Yes	No	Don't know
Total	24,271	19,239	4,655	377
0 to 4 years	4,130	4,025	53	52
5 to 9 years	3,337	3,150	138	49
10 to 14 years	3,250	2,983	220	47
15 to 19 years	2,967	2,681	239	47
20 to 24 years	2,243	1,912	283	48
25 to 29 years	1,686	1,351	296	39
30 to 34 years	1,508	1,102	382	24

The column header "No" has thick lines set for left and right and extend into cells is checked

You can also set the left and right borders of a spanner and have them extend into the leftmost and rightmost columns under the spanner. This is done the same way it is done for individual columns. Right-click on the spanner, choose Format (Spanner), set the left and right lines and ensure that **Extend Lines into Cells** is checked.

Table 2. Age in 5 year groups by Sex, Urban/Rural

Age in 5 year groups	Sex			Urban/Rural		
	Total	Male	Female	Total	Urban	Rural
Total	24,271	11,787	12,484	24,271	13,920	10,351
0 to 4 years	4,130	2,088	2,042	4,130	2,083	2,047
5 to 9 years	3,337	1,646	1,691	3,337	1,869	1,468
10 to 14 years	3,250	1,606	1,644	3,250	1,943	1,307
15 to 19 years	2,967	1,394	1,573	2,967	1,865	1,102
20 to 24 years	2,243	1,061	1,182	2,243	1,334	909
25 to 29 years	1,686	789	897	1,686	964	722
30 to 34 years	1,508	761	747	1,508	829	679
35 to 39 years	1,300	600	700	1,300	757	543
40 to 44 years	957	474	483	957	613	344

Left border for "Sex" spanner

Left border for "Urban/Rural" spanner

By default, stubs and columns have no borders, but spanners have thin lines on the left. This means that cells in the leftmost columns under a spanner will have left borders. You can change these default settings by using the Preferences.

See Also: Formatting Row, Column, or Cell Data, Add Borders to a Table

Make Captions Span Data Cells

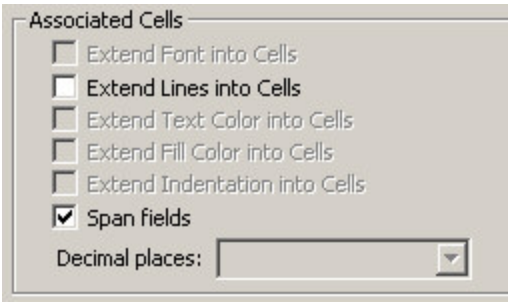
This option is controlled through the Format (Caption) menu.

Captions are rows without data. Normally the value set label is used as a caption before the stub group and the caption text box is restricted to the same size as the other stub text. However the caption text box can be widened to the width of the table by using its **Span fields** option. Right-click on the caption, choose Format (Caption) to bring up the Format (Table Part) dialog and check "Span fields".
An example:

	All persons	Male	Female
Marital status			
All persons	9,441	4,632	4,809

Caption

Select the "Span fields" option from the Format (Caption) menu.



The result:

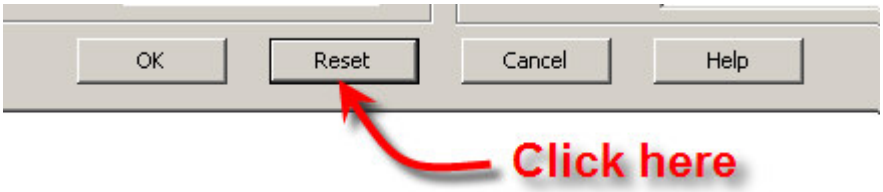
	All persons	Male	Female
	Marital status		
All persons	9,441	4,632	4,809

Note: The horizontal alignment of the caption was also changed to "center" to demonstrate the size of the associated text box.

By default, captions do not span fields. You can make all captions in all tables in the file span fields using Preferences and Default Formats.

Reset Format of Table Item to Default

All the format attributes of every element in a table have a "default" setting. These are established using the Preferences and Default Formats. Each attribute can be changed through certain entries in the Format (Table Part) dialog. To return any element to its "default" state, i.e., return to all the default settings in the dialog, just click on the **Reset** button at the bottom of the dialog.

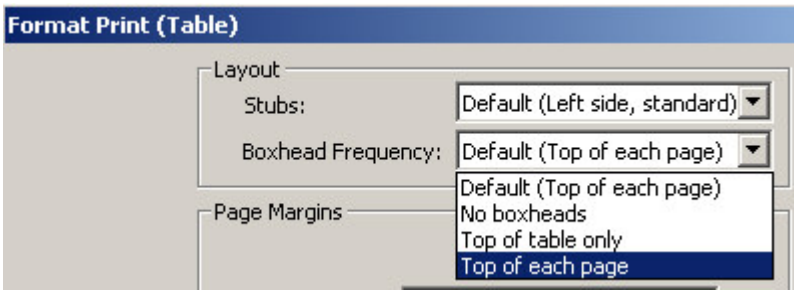


Note: There is **NO** "undo" for this operation. All custom entries are lost.

Change the Repeating of Boxheads

The boxhead refers to the area of the table containing the stub heads, column headers and spanners. You can control how the boxhead is displayed when the table is printed using the Format Print (Table) dialog. To bring up this dialog, choose Format Print (Table) from the Edit menu or right-click anywhere on the table and choose Format Print (Table).

The boxhead setting is found in the "Layout" section of the dialog. It only has meaning for a table that spans many pages and it only has an effect in the print preview and, of course, the printed table.



The options are:

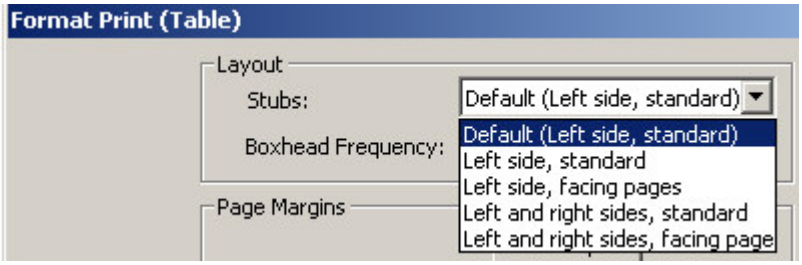
No boxheads – Only the table title is printed on each page.
Top of table only - Table title and box head only appear on the first page of table.
Top of each page – Table title and box head appear on each page of the table.

Use the Format Print (Table) dialog to set the boxhead frequency for a single table. Use Preferences and Default Formats to set the default boxhead frequency for all tables in the file.

Change Stub Column Position

Stubs are the text associated with each row of the table. By default the stubs appear to the left of the first data column. You can place the stubs for a table on the right, or on both the left and right as well as placing them on the left and right across a two-page spread when the table is printed. To bring up this dialog, choose Format Print (Table) from the Edit menu or right-click anywhere on the table and choose Format Print (Table).

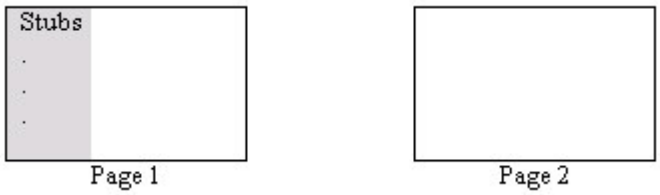
Stub position is found in the "Layout" section of the dialog. It only has an effect on the print preview and, of course, the printed table.



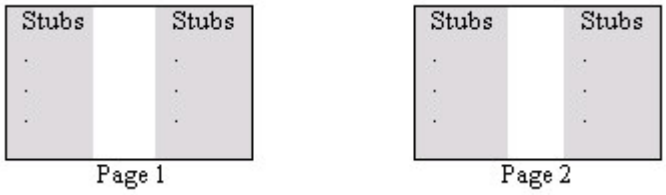
There are four options:

- Left side, standard (default): place stubs along the left side of every page. If the table is wide enough so that some of the columns go onto a second page, stubs will be placed on the left sides of both pages.

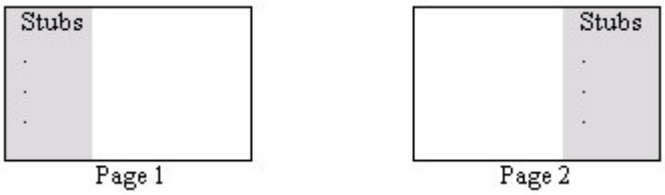
- Left side, facing pages: place stubs along the left side of the page but only for the first page of facing pages. If the table is wide enough so that some of the columns go onto a second page, no stubs will be placed on the second page. If the table columns all fit on one page then this is the same as Left side, standard.



- Left and right sides, standard: place stubs on both the left and right sides of the page for every page. If the table is wide enough so that some of the columns go onto a second page, stubs will be placed on the left and right sides of both pages.



- Left and right sides, facing pages: place stubs along the left side the first page of facing pages and on the right side of the second page of facing pages. If the table is wide enough so that some of the columns go onto a second page, stubs will be placed on the left side on the first page and the right side of the second page. If the table columns all fit on one page then this is the same as Left side, standard.



Use the Format Print (Table) dialog to set the boxhead frequency for a single table. Use Preferences and Default Formats to set the default boxhead frequency for all tables in the file.

See Also: Viewing Multiple and Facing Pages

Creating Tables by Geographic Area

Area Processing

Area Processing is the generation of tables by a defined hierarchical structure of area codes. If the geographic hierarchy of a hypothetical country is region, province, locality (major-to-minor order), it may be necessary to produce tables for each of these sub-divisions.

To use area processing you must have done the following:

• Created an area names file (.anm)

• Selected one or more Area IDs from the area IDs dialog box

For example, suppose we wish to perform geographic area processing for our top-most units of geography, which are Province and District (choose them in the Area IDs dialog box). Each tabulation will be repeated for each Province and District in the data file. In addition, a summary entry [Total] will be shown for the entire data file. The tabulation will be displayed in the following order:

Total
  Province 1
    District 1 (of Province 1)
    District 2 (of Province 1)
    District 3 (of Province 1)
  Province 2
    District 1 (of Province 2)
    District 2 (of Province 2)
    District 3 (of Province 2)
  :

:

Area Processing will apply to **all** tables in your application.

After producing the tables using area processing you can select a cell to display in a thematic map and view it spatially.

No File Processing: It is also possible to produce tables without creating an area names file. Select the levels of geography desired but after clicking on the Traffic Light button in CSPro to run your tables, leave the <Area Names> field blank. CSPro will use codes from the data file instead of names for each level of geography. For example, if province Gluten's code is 20, the tables will show code 20 instead of the name Gluten. This can be useful for generating a list of every geographic code found in a data file.

See also: Area Names File, Using Table Data in the Map Viewer

Create an Area Names File

The Area Names File (.anm) is a text file that you can create using any text editor. Be sure you save this file with the extension .anm. This file defines the levels of geography and assigns text names to the numeric codes for each geographic unit.

Note: It is similar in format to the area names file (.ARA) used in IMPS. You can also convert an IMPS area file (.ara) to a CSPro area file (.anm) using earlier versions of CSPro, e.g. 2.n.

The following is an excerpt from the area names file for Popstan (popstan.anm). The first section identifier, i.e., [Area Names], indicates the type of file, followed by the CSPro version number.

Next, the [Levels] section provides the names of the geographic hierarchy (levels) in major-to-minor order.

Finally the [Areas] section gives the correspondence of the geographic codes (major-to-minor order) found in the data with a name for the geographic area. More detail follows the listing of the area name file.

Note: Area code values can be positive, negative or zero.

If the data file contains area codes that do not have corresponding names in the area names file then the unmatched area codes will be displayed as a text string in place of the missing name.

```
[Area Names]
Version=CSPro 6.0

[Levels]
Name=Province
Name=District

[Areas]
X X = Popstan
1 X = Artesia
1 1 = Dongo
1 2 = Idfu
:
2 X = Copal
2 1 = Baja
2 2 = Bassac
:
3 X = Dari
3 1 = Argentina
```

```
3 2 = Benlata
3 3 = Bristol
:
```

The very first line following the [Areas] section are the codes and name of the ‘total’ area; ‘country’ in our case. It is considered the ‘Grand Total’ level and denoted by ‘X’ values for each level of the area hierarchy. In the example, the first ‘X’ represents the Province code and the second ‘X’ represents the District code. Basically, an ‘X’ value is similar to a ‘wildcard’ match, so any value in this field is acceptable (and thus part of the area).

Following the county area name is the set of codes and names for the lowest valued major level; ‘Province’ in our example. The lowest code for a province is ‘1’ associated with Artesia. Again, an ‘X’ value is given for the District code since any code here is acceptable. Next, we must give codes and name for all districts in Artesia starting with the lowest code value.

Note that each line for this province begins with ‘1’since the province code must be combined with the district code to uniquely distinguish this district. Dongo is district ‘1’ of Artesia (province ‘1’). Data for a questionnaire with Province code = 1 and District code = 1 will be tallied for Dongo District.

When districts for Artesia are all listed (codes and names), start with the next lowest province code followed by its districts. The process is repeated for each province.

Note: Area codes must be listed in ascending sort order from major to minor.

If the area name file has only one level, e.g., province, then only one code would be given. If three levels were needed, e.g., province, district, village, then three codes would be required. As always, ‘X’ represents the wildcard match.

Indentation associated with names above is for illustrative purposes only, i.e., area names shown in tables will not be indented. It can make the area name file easier to ‘read’.

Also, codes within the [Areas] section can be separated by commas, spaces, or a combination of both. Any of the following are acceptable to define an item at the district level:

- `3 15 = Sharif`
- `3, 15 = Sharif`
- `3,15 = Sharif`

Hiding Tables: To hide tables for certain level of geographies, preface the area name label with a tilde (~). For areas with this label, a table will still be created (and stored in the .tbw table file), but it will not be displayed and will thus be hidden. For example, if a table shows states (provinces) and counties (districts) but some states only contain one county, it may be desirable to suppress the repetition of the data at both the state and county level. For example:

```
X X = United States

1 X = Maryland
1 1 = Montgomery Country
1 2 = Prince George's County

2 X = District of Columbia
2 1 = ~ District of Columbia

3 X = Virginia
3 1 = Arlington County
```

```
3 2 = Fairfax County
```

Without the tilde, a table would be displayed for District of Columbia as both a state and a county. Adding the tilde leads to the desirable behavior of only displaying one table for that level of geography.

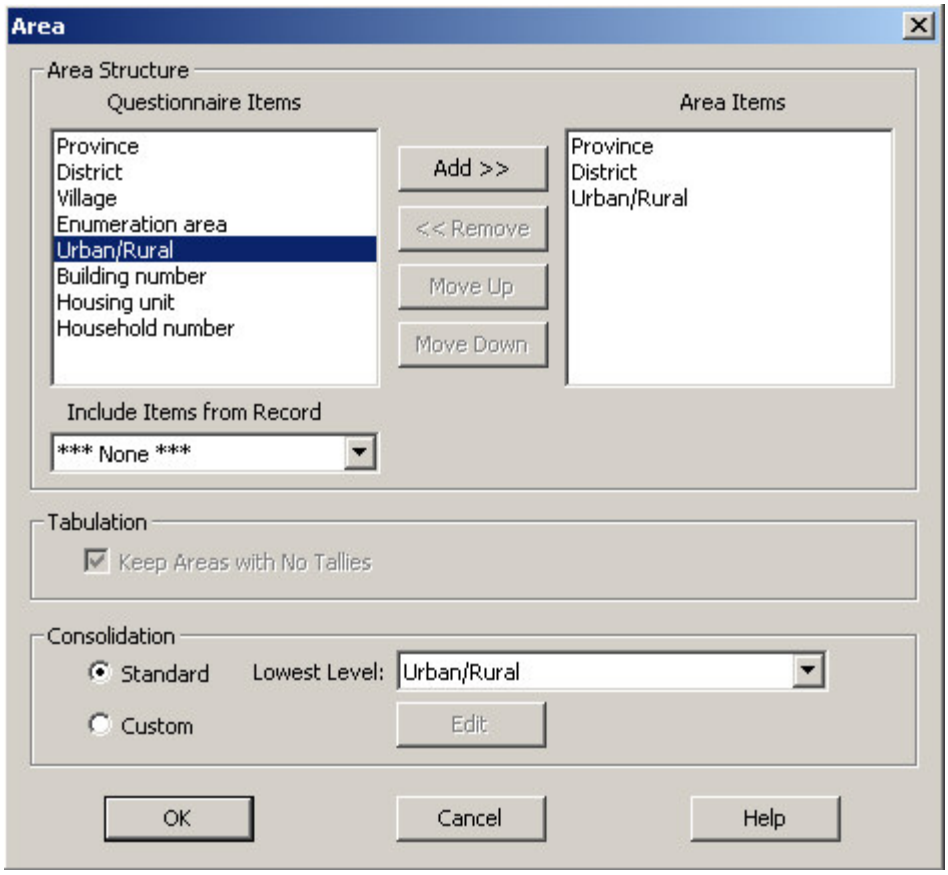
Tables created by Area Processing can be used to create thematic maps. The only real condition, besides having a CSPro map, is that the number of levels defined in the area name file, i.e., in [Levels] section, must not be greater than the number of geographical levels available for the map.

Tables created by Area Processing are easily incorporated into a Table Retrieval System for distribution.

See Also: Create a Thematic Map of Results, Add a CSPro or IMPS Tables File to a Table Retreival System

Area Dialog Box

This dialog box allows you to select the Questionnaire ID items to be used for area processing.



Area Structure Section

Codes for these items are used to 'tally' in the specific areas. Area items can come from the questionnaire ID items or items on singly occurring records. Available single record names will be available in the 'Include Items from Record' menu. If selected those items will be added to the list under 'Questionnaire Items'.

Note: The number of the Area IDs items determine the number of [Levels] needed in the Area Name File.

- Select one or more Questionnaire IDs from the left box (to select multiple items, hold down the Ctrl key when you make your selections).
- Click the 'Add' button to copy your choices to the Area IDs list.
- To remove an item from the Area IDs list, select it and click the 'Remove' button.
- To re-order items in the list, select one and click the 'Move Up'/'Move Down' button as appropriate.

Tabulation Section

In future versions, users will be able to decide if areas without any counts [no tallies] are displayed in the table or not.

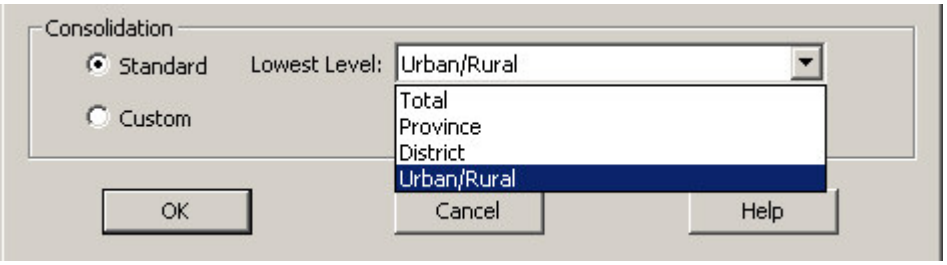
- Show area in table even if no counts are tallied [always in version 5.0].

Consolidation Section

Generally, only tables at the lowest level are created when the data file is processed. After these are completed these lowest levels are combined and recombined to create the higher level tables.

The two options here are Standard or Custom.

'Standard' follows the usual fashion of aggregation, i.e., adding to get total. Each lower level is combined to create the next highest level.



As an example from above, the original tables [created from the data file] would be an urban table for each district in each province and a rural table for each district in each province.
Standard Consolidation would:

- Add the appropriate urban and rural tables to create the province-district table.
- Add the appropriate district tables to create the province table.
- Add all province tables to create the 'country' table.

This procedure produces four levels of tables: Urban/Rural, District, Province, and Country (given in minor to major order).

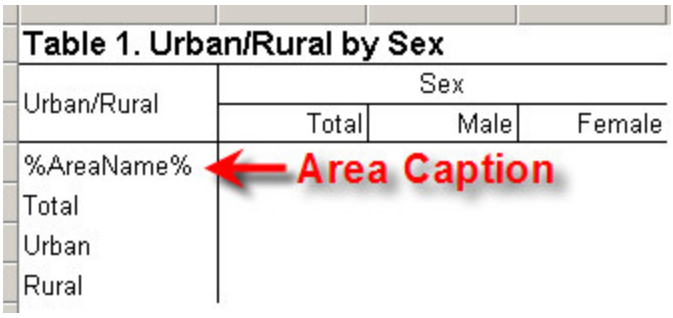
The **Lowest Level** allows users to select the most minor level to be produced. Counts at this level and higher will appear in the tabulation but any lower level counts will be discarded.

Custom option allows some freedom in defining the aggregation or consolidation scheme. See Custom Consolidation for more information.

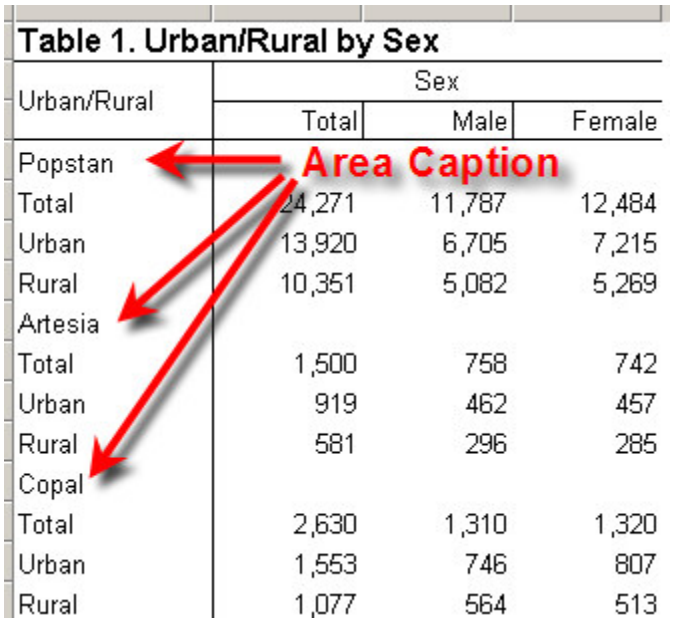
See also: Area Processing, Create an Area Names File

Area Captions

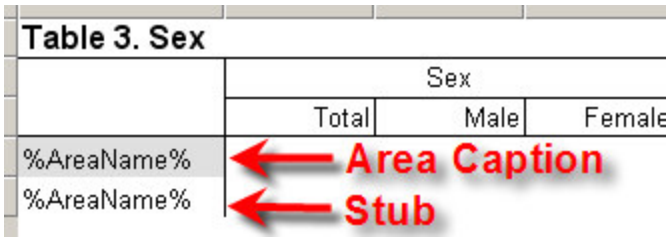
Area Captions are, as the name indicates, row captions in which the corresponding area name is placed for each table. Area captions are placed just above the total line of the first stub group in the table. An area caption is denoted by the "%AreaName%" text.



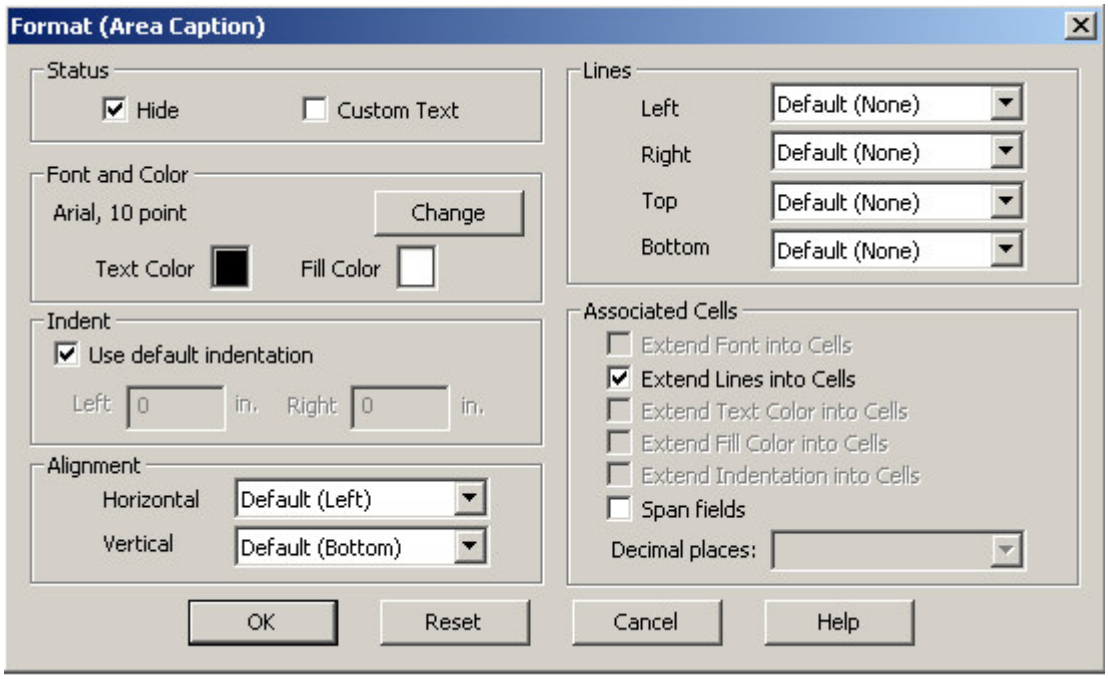
When the table is run, the %AreaName% is replaced by the names from the area names file.



If the table is a 'one-row' table, e.g., it has only column items but no stub groups, the area name caption replaces the 'Total' stub that would otherwise be present and the area caption is 'hidden'.



The area caption has the same formatting options as other captions, available by right clicking on the caption and choosing Format (Area caption). You can hide area captions, as in the example above, you can change the fonts, make them span rows, etc…

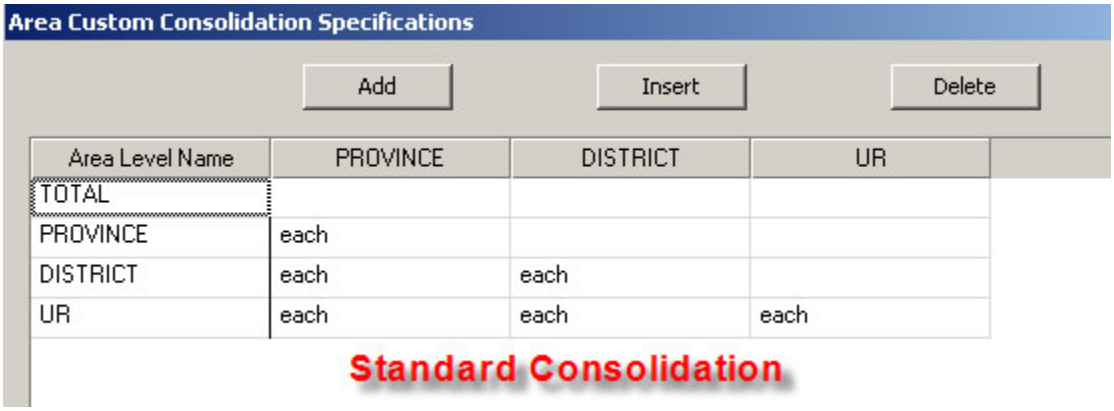


You can also set the default formats for all area captions in all tables in your file using Preferences and Default Formats.

See Also: Formats for a Part of a Table

Custom Consolidation

Custom Consolidation allows the aggregation of lowest level tables to create almost any reasonable higher level. Users define the consolidation scheme starting from the **Standard** consolidation, shown below.



Standard Consolidation would:

- Retain each Urban and Rural (UR) table for each district in each province.
- Add each group of UR tables to create a table for each district in each province.
- Add each group of district tables to create a table for each province.
- Add all province tables to create a single ‘country’ table.

The result of the consolidation is:

- One Country table

- A Province table for each province
- A District table for each district in each province
- One urban table and one rural table for each district in each province.

Each higher level is the aggregation of the levels below with the corresponding area code.

Using custom consolidation, you can produce tables at only the levels you wish. For example, if you did not want to produce tables at the district level but wanted to continue producing them for province, urban/rural and entire country; you would simply remove the district row:

Area Level Name	PROVINCE	DISTRICT	UR
TOTAL			
PROVINCE	each		
UR	each	each	each

Using Custom Consolidation you can also create user-defined schemes for combining the lowest level tables by adding new rows to the consolidation scheme:

Area Level Name	PROVINCE	DISTRICT	UR
TOTAL			
TOTAL_UR			each
PROVINCE	each		
PROVINCE_UR	each		each
DISTRICT	each	each	
UR	each	each	each

Each row of the scheme must have a CSPro 'name' for the set of tables to be created, listed under **Area Level Name** above.

In the example above, the following tables would be created:

- One Country table
- ***One urban table and one rural table for the entire country***
- A Province table for each province
- ***One urban table and one rural table for each province***
- A District table for each district in each province
- One urban table and one rural table for each district in each province.

In addition, you may place conditions in the cells of the consolidation scheme grid. These conditions are used to modify how the tables replaced for a given level (the column) are consolidated for a particular consolidation scheme (row).

The options are:

- Blank – Any code in this position is included.
- Each – Each different code in this position creates a separate table.
- Single value – One table created for this value of the area level
- Replacement formula – One 'summary' is table created for all areas that the meet the condition and the replacement code is substituted for the area code for the summary table. [start:end = replacement]

The last two options are illustrated by the following example:

Area Level Name	PROVINCE	DISTRICT	UR
TOTAL			
TOT_URB			1
PROV_1_4	1:4=99		

The TOT_URB table will include all tallies that had a value of ‘1’ for the UR area level. The area codes on this table will be Province=X, District=X, UR= 1.

The PROV_1_4 table will include all tallies that had Province codes of 1 through 4. The area codes for this table will be Province=99, District=X, UR= X. Since other province codes are 1 to 15, the table associated with ‘99’ would be displayed after any for those coded. (In the example above NO province tables were created hence they are not displayed)

In order to use these codes – a corresponding name should be in the area name file.

Table 1. Sex

	Sex		
	Total	Male	Female
Popstan	24,271	11,787	12,484
Total Urban	13,920	6,705	7,215
99	10,223	5,015	5,208

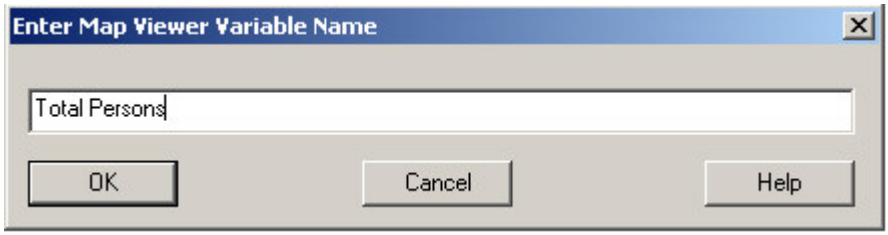
No area name for '99

Note: When custom consolidation is used it is the user’s responsibility to make sure the consolidation scheme is reasonable and that it works in the desired manner. As usual, every aspect of a data processing system should be tested for correctness.

Create a Thematic Map of Results

To Generate a Map from a Tabulation Application:

- Produce a table using area processing in a Tabulation Application. After running, select any representative cell for the data you want to map [a single data cell].
- Click on and enter a Map Viewer Variable Name for this data item [or accept the default title].



- Next choose the ‘map’ file that can be either an existing map data file (.mdf) to which the variable is appended or a map file (.mpc) which will have only this variable..
- MapViewer will be launched and a thematic map, representing the data you selected, will appear.

To Create a Map Data File (`.mdf`) from a Tabulation Application:

You can bring numerous tabulated variables into MapViewer and then save them all in a map data file (extension .mdf). This is an excellent way to build your own collection of mapped variables as a data dissemination tool.

To create an .mdf file:

- Generate the first variable as described above using an .mpc file (rather than an .mdf).
- Switch back to the Tabulation application.
- Select the next variable from the tabulation and provide a Map Viewer name for it. MapViewer will now hold both variables (look at MapViewer's Variable drop-down box).
- Repeat this process for as many variables as you would like to map. You can map different variables from different tables in the same run.
- Save the map data file (extension will be .mdf).
- Later you may add more variables to this map data file by loading this .mdf in the MapViewer the next time you wish to map.

See also: Introduction to Map Viewer

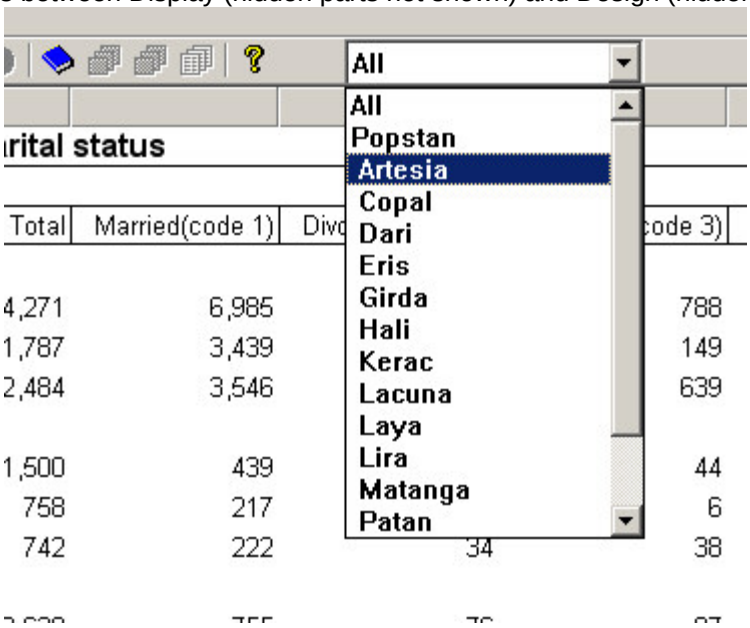
How To ...

Display Results for One Geographic Area

When a Tabulation Application uses the Area Processing option, the table usually displays the counts for each 'area' concatenated one after the other sorted by increasing area code values.

If you want to display the counts for a certain geographic then select it from the drop down menu available in Display view.

Note: Ctrl+D switches between Display (hidden parts not shown) and Design (hidden parts shown) views.



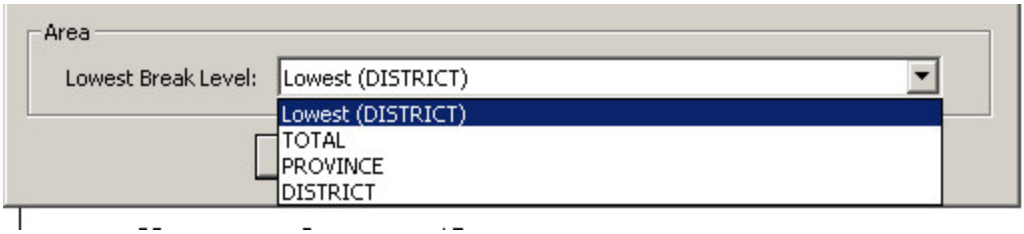
The 'All' selection represents the usual display. If area codes are found in the data file that are not defined in the Area Name File then those codes will be listed. The areas are listed in sort order, i.e., the same order that they are defined in the area names file.

Note that this only changes which area is currently displayed. Tables are still produced for all areas originally specified and all of these areas will still appear when the tables are printed or saved. To only produce tables for certain geographies either set the lowest break level for a table, or use custom consolidation.

Tabulate only Certain Levels of Geography

When the area structure is defined in the Area Dialog Box it defines the levels at which all the tables in the Tabulation application will be tallied. This can be modified for individual tables using the Tally Attributes for a Table.

The last section in the menu allows the selection of a lowest area break level for this table only. By default it will be set to the lowest area level available.



In the example above, if a certain table is to be produced only at the province level then that item would be selected as the **Lowest Break Level** instead of district.

Printing Tables

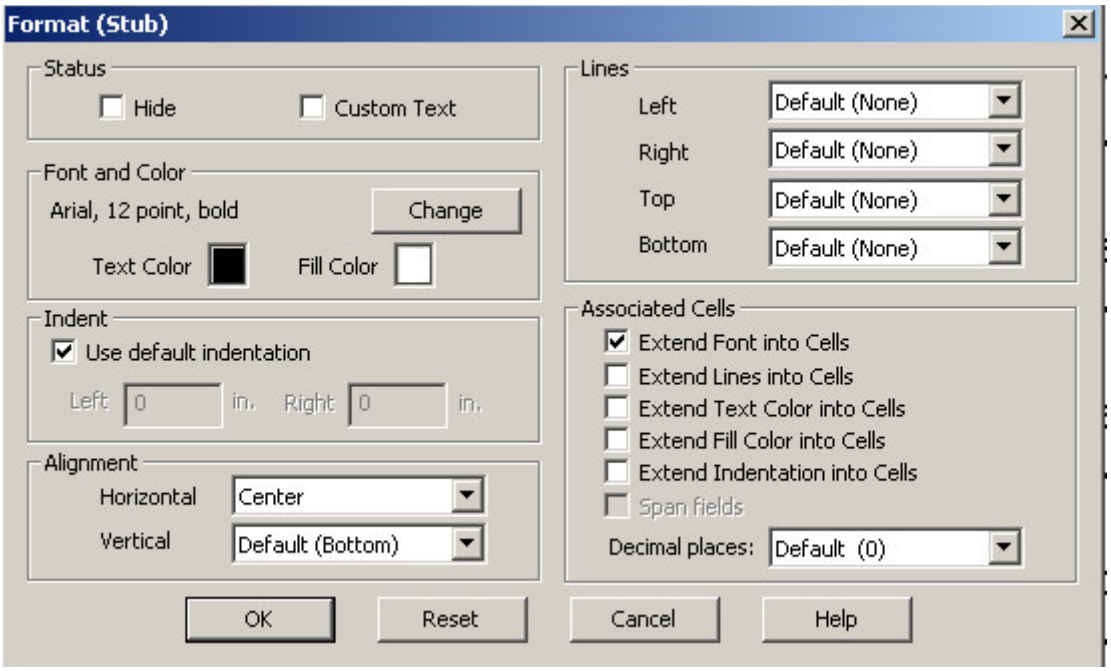
Using Print Preview

Print Preview is a view of the tables as they are display on the printed page, i.e., what they will look like when they are printed on the default printer. To view Print Preview, press the button on the toolbar or under **File** menu, select **Print Preview**.

Note: No Print Preview is available if there is no printer installed!

This is just one of three views available for a Tabulation Application. To return to Display or Design View use the "Close" button or just click the Print View icon [the icon is depressed when active and "undepressed" when not active], both are in the tool bar.

All the formatting done through the various format menus of a table are carried over to print preview. For example all the settings used in the following would be applied in this view.

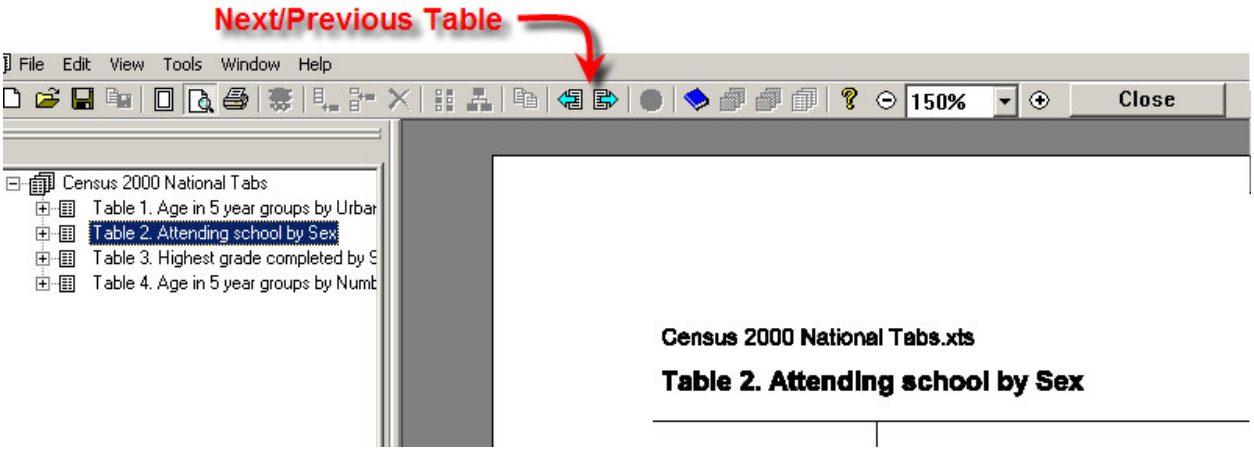


The spacing or sizing of rows or columns in Display view is NOT carried over to Print Preview.

Navigating Between Pages, Tables, and Areas

The easiest ways to move between tables are

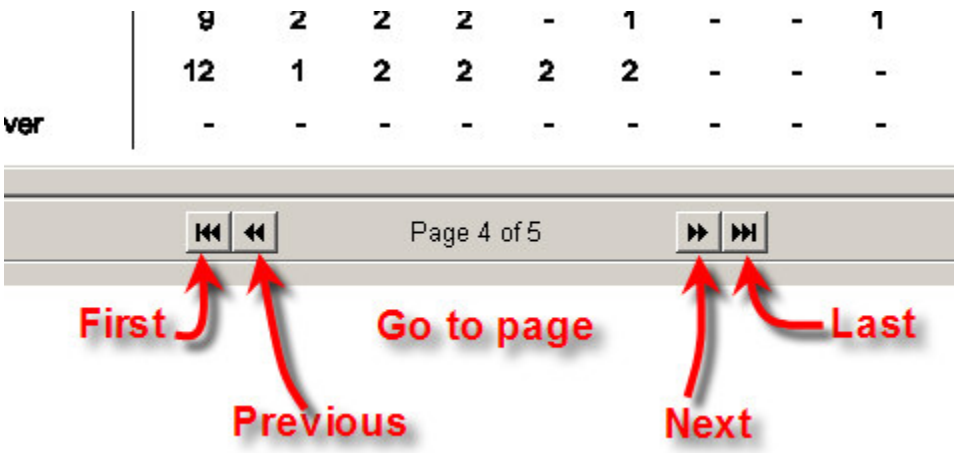
- Use the "Next Table" or "Previous Table" icons on the toolbar [see below].
- Select the table from the table tree on the left part of the screen [see below].



The easiest ways to move between pages of a table are:

- Use Home, End, Page Up or Page Down keys.
- Use navigation buttons at bottom of preview window (see below).
- Use the mouse "wheel" (if available and activated). One "click" (of the wheel) toward the user goes to the next page, one "click" away from the user goes to the previous page.

(Note: this behavior is only available in the page view (100%))



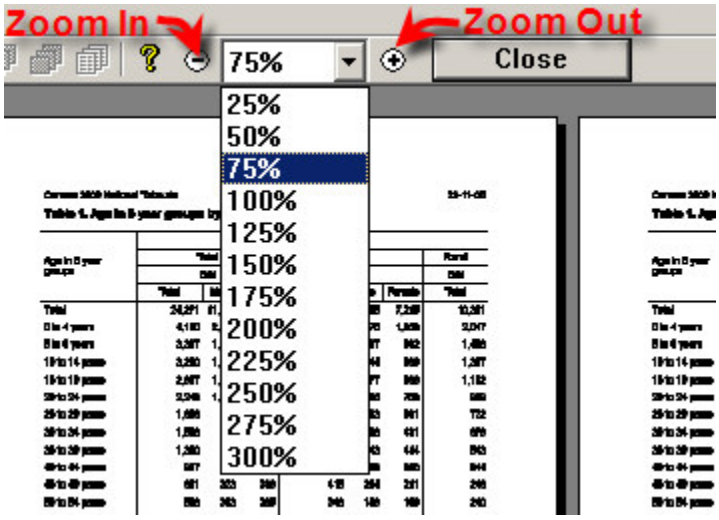
See the View menu for other combinations of keys that will allow you to move between tables.

There is also a "Goto ..." option available from the Edit menu and from the right-click menu which allows you to go directly to a particular page, table or area within a table.

See Also: Print Preview Options

Viewing Multiple and Facing Pages

The view presented when Print Preview is activated is that of the first page of the first table. Users can move between pages of the tables (see Navigating Between Pages, Tables, and Areas. Users can change magnification level [zoom in or zoom out] by using the drop down menu available at the top of the window or by clicking on the zoom in and zoom out buttons on the toolbar.



The "100%" setting is the full page and default setting. A value less than 100% shows multiple pages in the window [zoom out]. A value more than 100% shows less than one page in the window [zoom in].

If the current view is 100% or less -

A single wheel "click" toward the user will display the next page. A "click" away from the user will display to the previous page. This is true even if more than one page is displayed in the window. Of course, when an additional page is displayed one of the other pages is no longer in the window.

If the current view is greater than 100% -
The wheel activates the vertical scroll bar of the table.

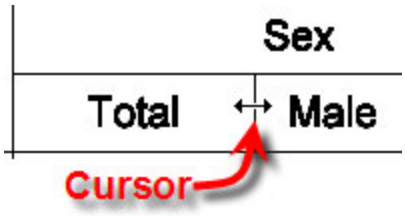
If the Ctrl key is held down and the wheel used –
A single wheel "click" toward the user will display the next "zoom in" option. A "click" away from the user will display the next "zoom out" option. Note that these wheel "click"s are executed one at a time so in the case of multiple clicks it may take time for the display to adjust.

You can activate **facing pages** mode by choosing it from the View menu. This mode simulates what the printed tables would look like if they were bound together in a book. The two pages that would face each other in the book are shown together. When in facing pages, the next page and previous page buttons will move to the next two pages that face each other rather than moving one page at a time. If you change the zoom level while in facing pages mode, facing pages is turned off.

Modifying Row and Column Spacing for Printing

The width of columns and the height of rows can be modified in Print Preview. Sizing is also available in Design and Display views but it does NOT carry over into Print Preview.

To modify the width of a column, place the cursor on the "vertical bar" in the box head that you want to move. The cursor will change to a horizontal double-sided arrow, left click the mouse and drag the vertical bar to the new position.



To modify the height of a row place the cursor below the row text until it changes to a vertical double-sided arrow, left mouse click and drag the (usually invisible) horizontal bar to the new position.

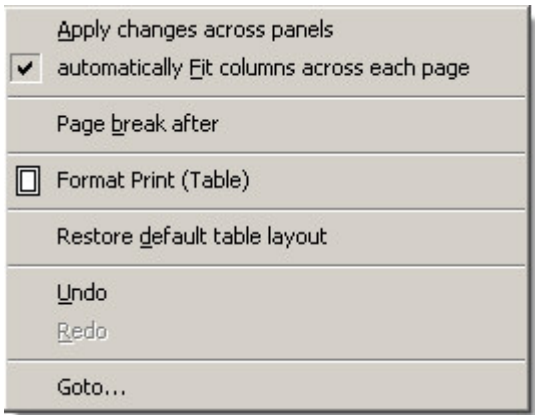
Modifications to row height and column width work in conjunction with the "Apply changes across panels" and "automatically Fit columns across each page" options. See Print Preview Options for more information.

Any changes made can be undone using the **undo** command. To reset the column and rows spacing back to its original settings, use **Restore Default Table Layout**. Both of these commands are available by right-clicking on the table while in the Print Preview.

See Also: Print Preview Options

Print Preview Options

You can make some modifications to the table in Print Preview. Most notably the rows and columns can be "sized" similar to the way it is done in spreadsheet software. There is also a "right-click" menu with additional options.



Briefly these options are:

Apply changes across panels

If unchecked this means that changes to row height or column width will be applied to only the selected row or column. If checked any sizing changes made to a row or column in a panel will be reflected in the corresponding row or column of each panel.

For example below are three panels for columns:

	Urban/Rural								
	Total			Urban			Rural		
	Sex			Sex			Sex		
	Total	Male	Female	Total	Male	Female	Total	Male	Female

'Sex' is a panel

If this option is checked and a change is made to the width of the "Male" column in any of the three panels then the width of all three will be modified. If the option is unchecked then only the specific column selected will have its width modified.

Automatically Fit columns across each page

If checked this option will "spread" the table columns for all tables across the entire printed page. If unchecked, the columns for all tables will be displayed in the minimal width possible given text, font, etc.

An example of unchecked:

Table 1. Sex

	Sex		
	Total	Male	Female
Popstan	24,271	11,787	12,484
Artesia	1,500	758	742

Columns NOT 'fit'

An example of checked:

Table 1. Sex

	Sex		
	Total	Male	Female
Popstan	24,271	11,787	12,484
Artesia	1,500	758	742

Columns Fit Across Page

Page Break After

Insert a page break [go to next page] after the row selected. This usually done when a stub group is "split" across two pages.

An Example before Page Break applied. The user right-clicks on the "10-14 years" row and selects page break after to place a break between the "10-14 years row" and the "15-19 years row".

Table 2. Age in 5 year groups by Sex

Age in 5 year groups	Sex		
	Total	Male	Female
Total	24,271	11,787	12,484
0 to 4 years	4,130	2,088	2,042
5 to 9 years	3,337	1,646	1,691
10 to 14 years	3,250	1,606	1,644
15 to 19 years	2,967	1,394	1,573
20 to 24 years	2,243	1,061	1,182
25 to 29 years	1,686	789	697
30 to 34 years	1,508	761	747

Cursor in "10 to 14 years" row

Next page after Page Break applied. This page starts with the "15-19 years row"

Table 2. Age in 5 year groups by Sex (continued)

Age in 5 year groups	Sex		
	Total	Male	Female
15 to 19 years	2,967	1,394	1,573
20 to 24 years	2,243	1,061	1,182
25 to 29 years	1,686	789	697
30 to 34 years	1,508	761	747

Format Print (Table)

See Formats for Printing

Restore default table layout
The default layout of a table is the layout produced for it when the table is first created. Any changes made to the print view of the table will be "undone". There is also an "undo" of a single previous action in Print Preview but this is an "undo" of ALL previous actions for this table.

Undo
Revert to view of table before the last action was taken. Ctrl+Z will also "Undo".

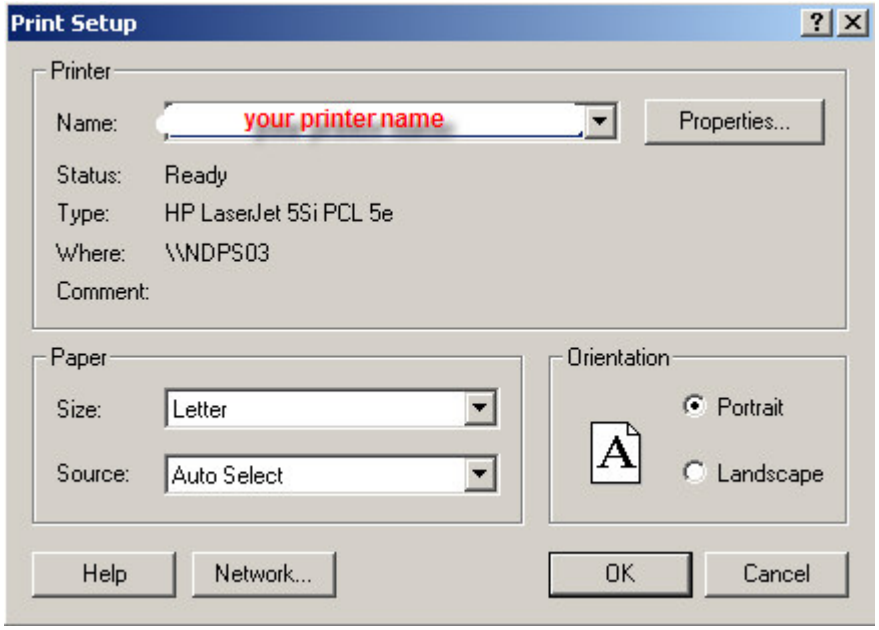
Redo
If an "Undo" was the last action then "Redo" is available.

Goto…
Allows you to move to a particular page, table or area within a table. This option activates the following menu:

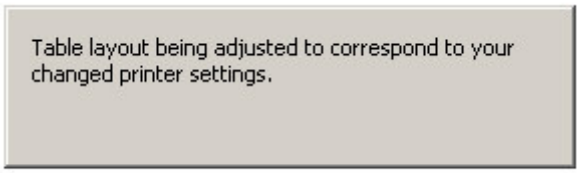
Page option – enter the page number to display.
Table option – choose the table (by name) to display.
Area option – choose the table and the area within that table to display. This option is only available if area processing is used.

Print Setup

This is the usual dialog box for Print Setup. It should be something like the following:



What is important is that this information is used to create the Print Preview of your tables. Changes made here will affect the "preview'' of your tables, particulary changes to the paper size and layout. If changes impact the print preview the following message will appear:



Sending Tables to the Printer

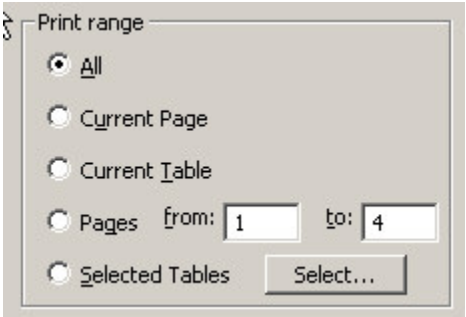
- **To preview the printing of tables**

  Click on the tool bar; or from the **File** menu, select **Print Preview**.

- **To print entire tables**

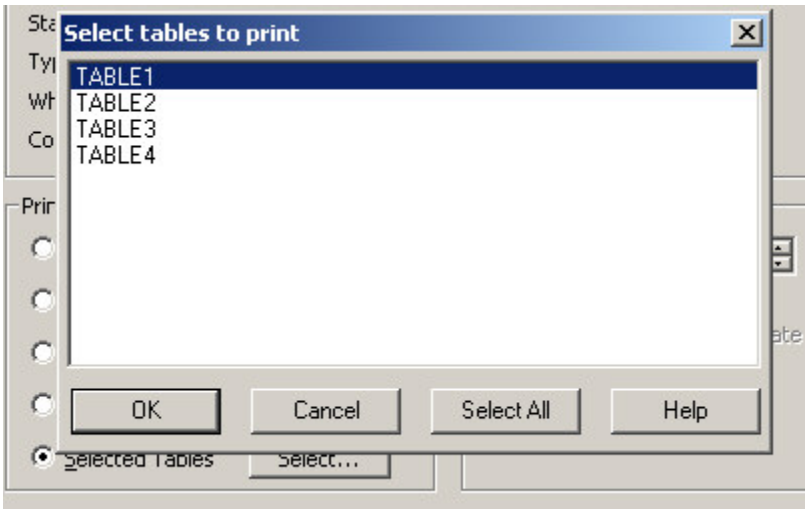
  Click on the toolbar; or from the **File** menu, select **Print**; or press **Ctrl+P**.

  When the printer menu is displayed the important part is the Print Range section that determines what is to be printed.



The selection here affects what is printed. The options:

- **All** – All pages of all tables.
- **Current Page** – Page currently displayed in window.
- **Current Table** – Table containing the currently displayed page.
- **Pages** – Range of page numbers for printing.
- **Selected Tables** – for this option click the "Select …" button to its right. A secondary menu is presented for table selection.



Other options will be included in the Print menu such as the number of copies to be printed. These options affect how the tables will be printed but not what will be printed.

How To ...

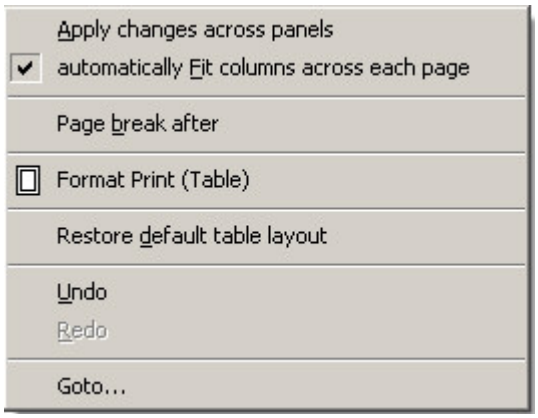
Undo or Reset Changes in Print Preview

Most formatting of tables is done using the various format menus available in the Display or Design View but not available in Print Preview.

The major formatting tools in Print Preview are

- Sizing of Columns or Rows [changing width or height]
- Add Page Breaks

If an action is taken and you want to "undo" it then right click in the Print Preview window. This gives the following menu:



Select "Undo" to reverse a single action.

If you just want to restore the table to its system default format then select "Restore default table layout".

See Also: Print Preview Options

Print Only Selected Tables or Pages

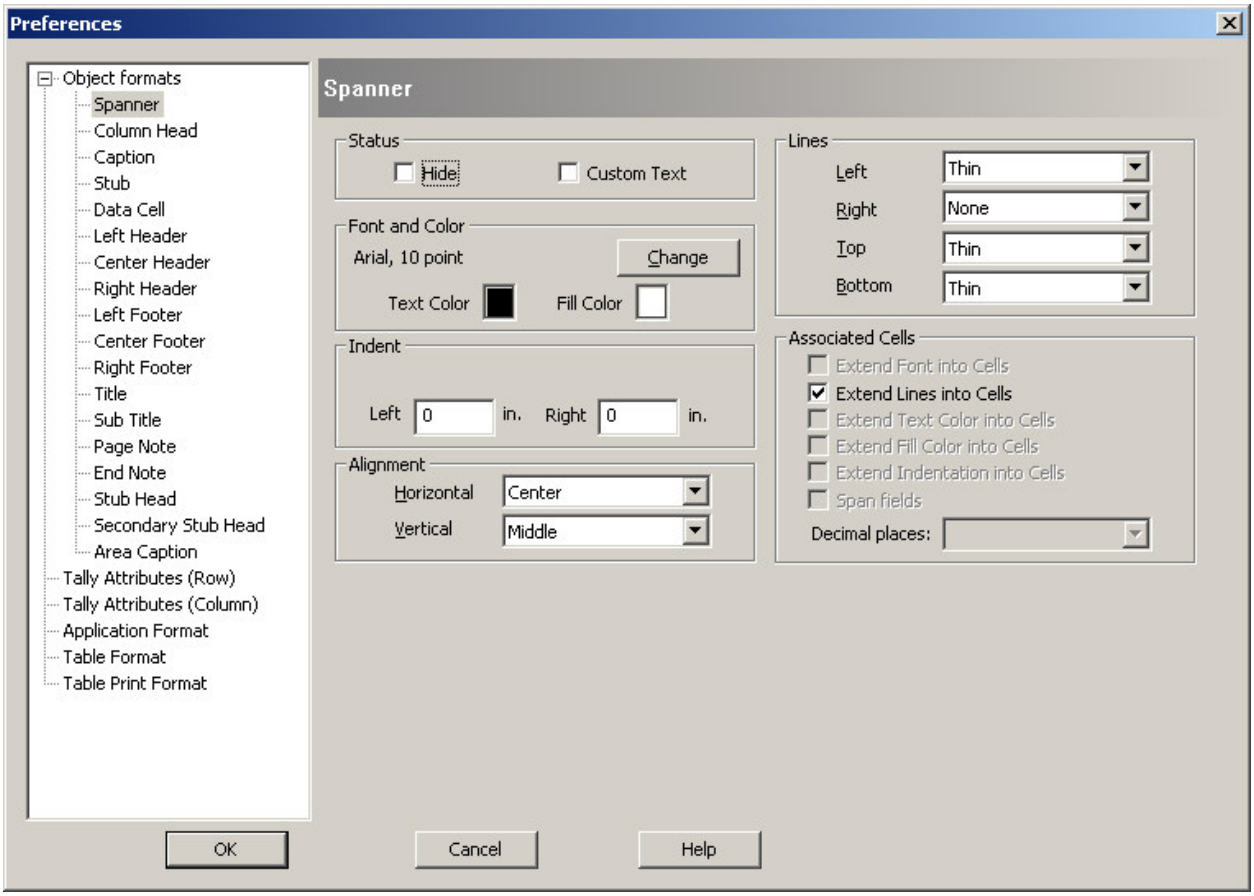
See Sending Tables to the Printer

Tabulation Preferences

Preferences and Default Formats

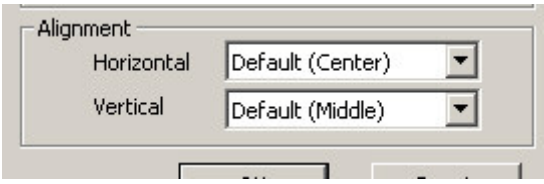
The "Default" settings used in all of the format menus (table element, table, table print, and application) can be set for the entire application. These "Default" settings are then accepted or changed for individual tables. Changing the default setting is the easiest way to change the formatting of all of your tables at once. It allows you to customize the look of all of your all tables.

The "Default" Options menu is available under the Edit drop down menu and listed as "Preferences".



Notice the alignment section in the above menu for Spanner.

Here is the alignment section in the Format (Spanner) menu for a specific table:

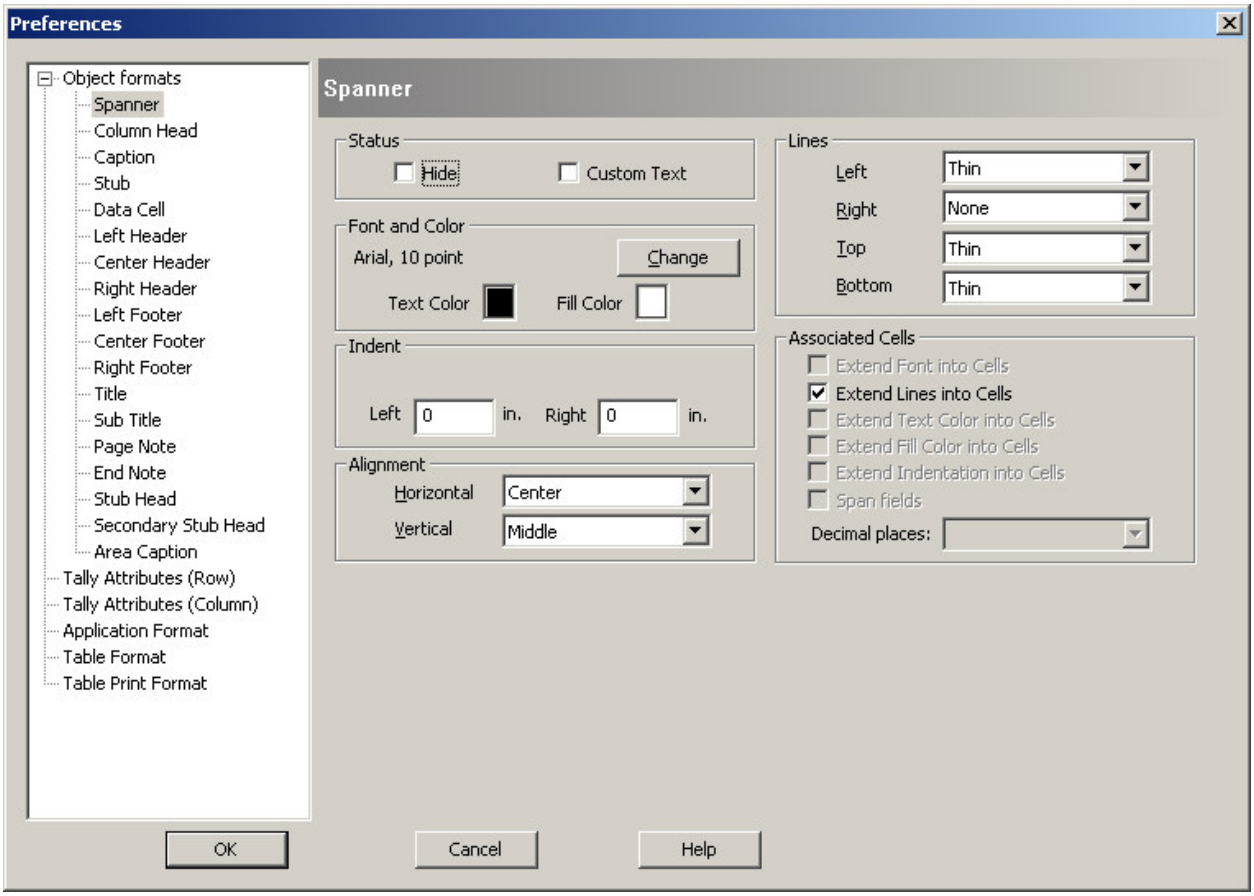


See Also: Modifying Preferences

Modifying Preferences

The option is available under the Edit drop down menu and listed as ‘Preferences’.

This brings up the Preferences Window:



Each "Object Format" on the left has a corresponding menu. Select the object in the tree then you can set the ‘default’ settings for each of the available options. The only difference is that ‘Use Default ’ check box may be missing since this is the menu that sets those defaults. It is important to note that if a setting is changed here it will be applied to all tables that use the default settings even the tables that have already been defined.

For example: If you select blue as "Text Color" for Spanner then all spanners that exist in tables already defined (and use the default setting) will now be displayed in blue as well as spanners in tables yet to be created.

For the meanings of the various options in each menu see the related topic.

See Also: Tally Attributes for a Variable, Formats for a Part of a Table, Formats for a Table, Formatting Row, Column, or Cell Data , Formats for an Application , Formats for Printing

Loading and Saving Preferences

Every Tabulation application starts with the "default preferences" set by CSPro. Users can easily modify these preferences to reflect their own choices of defaults. If you have a standard set of preferences that you want to use for your Tabulation applications, you do not need to make these same changes for each new application. Your settings can be saved after they are modified and then loaded into other applications.

Under the File menu are the two options: "Load Preferences..." and "Save Preferences...". Use the "Save" option to create a file that contains your selections. For this option, the user will supply a name for the

CSPro Table Format (.tft) file to be created. This file will contain all the ‘default’ settings currently active for this Tabulation application.

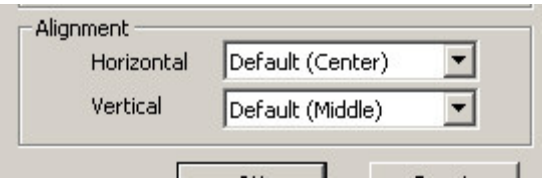
The "Load" option is used to change preferences to those in your CSPro Table Format (.tft) file. For this option, enter the name of the .tft file or browse to find it. Once the file is loaded the revised preferences will be in force.

See Also: Modifying Preferences

How To ...

Share the Same Format on Multiple Computers

In most applications the various options in the Format menus use the "Default" settings.



These "Default" settings are contained in Preferences and Default Formats. In any Tabulation application the initial set of Preferences is established by CSPro. They can be reset to other settings through "Load Preferences…" under the File menu.

Sets of user-defined preferences are contained in CSPro Table Format (.tft) files. Once the file has been created by the "Save Preferences…" option under the File menu, it can made available to other users in the same manner that any other file is shared.

See Also: Loading and Saving Preferences

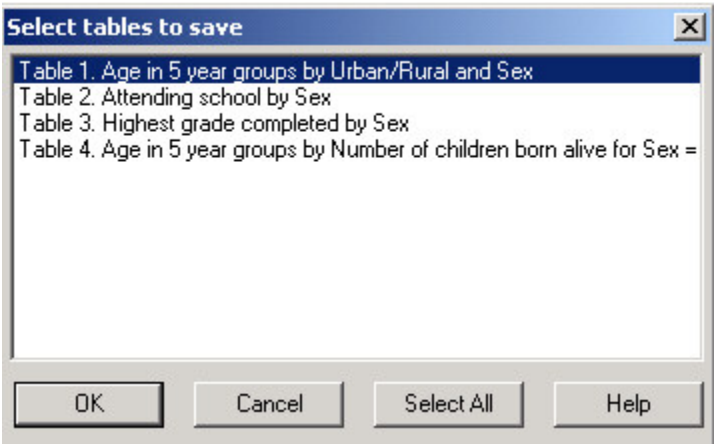
Saving and Copying Table Data

Save Tables for the Table Viewer

Click on the toolbar; or from the **File** menu select **Save Tables**.

If there is only one table in the Tabulation application, you will go directly to the Save As dialog.

If your Tabulation application has multiple tables defined within it, a **Select tables to save** dialog box listing the tables is displayed.



Use the **Select All** button if all tables are needed. Otherwise, select the individual table(s) that you would like to save in a single Table Viewer file. (Multiple tables are selected in the usual manner with the Shift and/or Ctrl keys.)

After selection press **OK**.

In the Save As dialog box enter the name of the Table Viewer file to be created or browse to select the file to be replaced. Table Viewer files must have the .tbw extension.

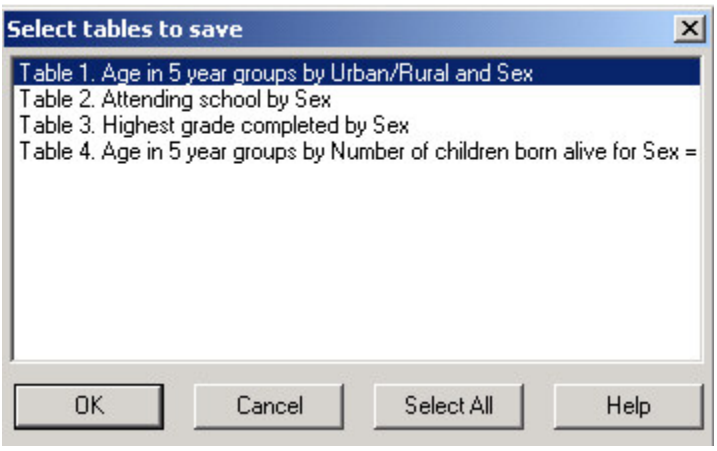
See Also: Saving Tables as Text, HTML and Rich Text Format

Saving Tables as Text, HTML or Rich Text

Click on the toolbar; or from the **File** menu select **Save Tables**.

If there is only one table in the Tabulation application, you will go directly to the Save As dialog.

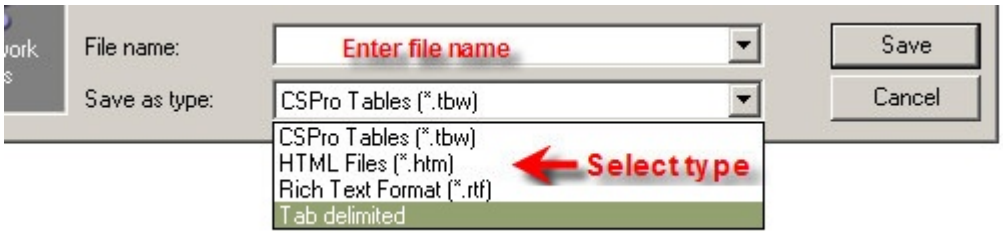
If your Tabulation application has multiple tables defined within it, a **Select tables to save** dialog box listing the tables is displayed.



Use the **Select All** button if all tables are needed. Otherwise, select the individual table(s) that you would like to save in a single Table Viewer file. (Multiple tables are selected in the usual manner with the Shift and/or Ctrl keys.)

After selection press **OK**.

In the "Save As" dialog box use the drop down "Save as type" menu to select the type of file other than .tbw.



Enter the name of the file to be created or browse to select the name of the file to be replaced.

Note: ONLY ONE table at a time can be in Rich Text (.rtf) or HTML (.htm) format. Tab delimited format supports saving multiple tables in a single text file.

Select and Copy Table Data to Other Applications

If you want to copy a table or selected cells (with or without associated text) of a table it must be done in the Display or Design View. There is no selection possible in the **Print Preview**.

• To select table cells

- Move the mouse pointer to the upper left-hand corner of the cells you wish to select.

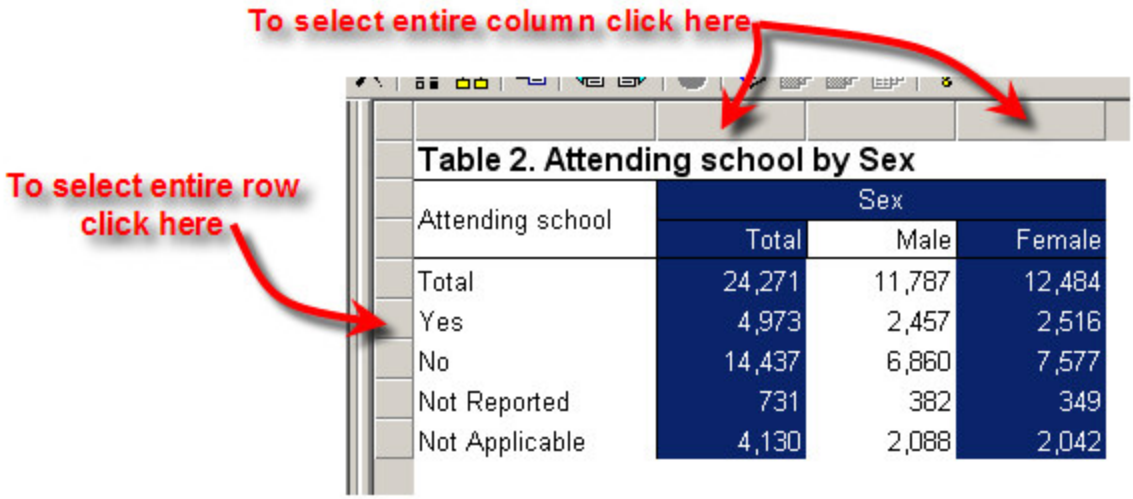
- Press the left mouse button and hold it down while you drag the mouse across the cells you want to select. The cells will change color to indicate that they have been selected. If you drag the mouse outside the cell area (top or bottom, left or right), the table will automatically scroll and continue to select.

- Release the mouse button. The selected cells are highlighted.

- To select additional cells hold the Ctrl Key down and follow the same technique.

• To select ALL cells from a row or column

- Entire Rows or columns of cells can be selected by clicking in the appropriate part of the grid.



- Hold the Ctrl key down and click to select non-contiguous rows/column (shown above). To select contiguous rows or columns use the Shift key and select the row or column for the other end of the range.

• **To select ALL cells in a Table**
Choose Select All from the Edit menu; or press Ctrl+A.

• **To deselect cells**
Press the **Esc** key; or from the **Edit** menu, select "**Cancel Selection"**.

Now that cells have been selected for copying you have the choice of copying the text associated with the selected cells (table title, stubs, column headers,…) or just copying the data in the selected cells.

- **To copy cell values and associated text**

- Click on the toolbar; or from the **Edit** menu, select **Copy**; or press **Ctrl+C**.

- **To copy cell values only**

- Select **Copy Cells Only** from the **Edit** menu.

Once the cells are copied to the clipboard they can be pasted into a word processor, spreadsheet or any software that accepts tabular or text format.

See also: Saving Tables as Text, HTML or Rich Text

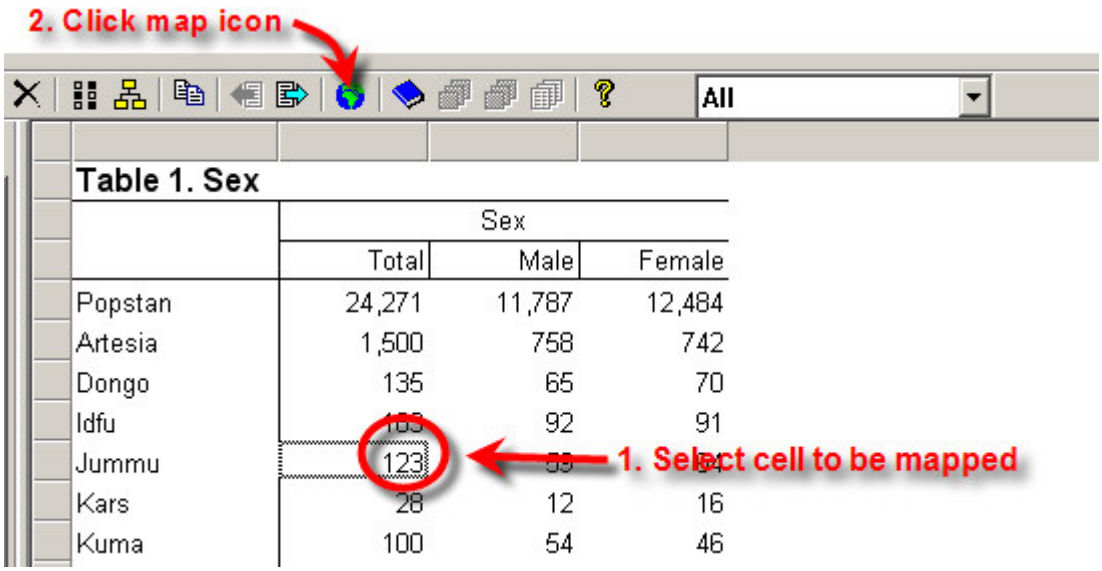
Using Table Data in the Map Viewer

Single cell values can be exported to a CSPro Map very easily.

In order to be able to export data to the map viewer:
- The Tabulation application must use Area Processing.
- You must have a CSPro map file or map data file which uses the same geographical coding scheme, i.e., same levels and codes as the in area names file used in your tabulation application.

If these conditions are met then
- Run the tables
- Select the cell to be mapped
- Click the Map Viewer icon on the toolbar
- Enter the label to be used in the map viewer for the variable you are mapping

Assume you wanted to map "Total Population". Run the table then click on ANY cell in the "Total" column. Next click the Map Viewer icon.



Next enter the Map Viewer label for the variable. This is the text that will appear for the variable when viewed in the Map Viewer.

Finally open the Map Viewer map file (.mpc) or the Map Viewer map data file (.mdf) to complete the mapping.

The process can be repeated for any number of data cells.

See Also: Area Processing, Create a Thematic Map of Results, Introduction to Map Viewer

How To ...

Distribute Finished Tables to Other Users

The tables from a Tabulation application can be saved in several different file formats. The format you choose depends on what you wish to do with the tables. To include tables in a word processing document, use the rich text format (rtf). To bring the table data into a spreadsheet so that users can do calculations on it, use the tab-delimited format. To post tables to the web, use the HTML format. If you want your users to be able to use your tables in all of these ways, you can distribute them in the CSPro Table Viewer format (tbw). Users can then view your tables in the CSPro Table Viewer tool and from there they can copy and paste or save tables in any of the formats supported by CSPro. Table Viewer will also allow them to print, and change the formatting of your tables, but it will not allow them to modify the table data

Table Viewer Files can be accessed via the CSPro Table Viewer Tool. The Tool, along with associated modules, can be installed on individual computers without installing the entire CSPro system (see Custom

Installation in Installing CSPro). This way other users can view your tables without installing the entire CSPro system.

If the tabulations use area processing, the table viewer files can also be imported into a CSPro Table Retrieval System (TRS). This is a much more structured way to distribute tables, especially for large numbers of tables (or documents, for that matter).

See Also: Introduction to Table Retrieval Setup, Add a CSPro or IMPS Tables File ,Save Tables for the Table Viewer, Saving Tables as Text, HTML or Rich Text, Introduction to Table Viewer

Copy Table Data to a Spreadsheet or Word Processor

CSPro allows you to transfer either part of a table, an entire table or multiple tables at once to a word processor or spreadsheet application.

To copy and paste a single table or part of a table follow the procedure outlined in Select and Copy Table Data to Other Applications.

To copy multiple tables, use the one of the file format options given for "Save Tables" (Tab Delimited Text, HTML, or Rich Text). These options are available for single tables or multiple tables. In some cases, you may need to save single tables in individual files to achieve the desired results.

See Also: Select and Copy Table Data to Other Applications, Saving Tables as Text, HTML or Rich Text

Prepare Tables for Posting to the Web

In most cases this means saving the tables as HTML files. HTML tables can only created one at a time. So the process must be repeated for each table to be saved in HTML format.

In Design or Display View (not Print Preview) select the Save Tables icon (or equivalent under File menu).

- If only one table is in the set then you will go directly to the Save As menu. If there is more than one table then the "Select tables to save" dialog box is displayed. Select only one from this menu.
- Click **OK** after selection.
- Select HTML Files (.htm) as Save as type.
- Enter the name of the file to be created or browse to select the name of an existing HTML file to be replaced.
- Click Save button. If file already exists you will be asked if you want to replace it.

These files can be viewed in a browser, e.g., Internet Explorer

See Also: Saving Tables as Text, HTML or Rich Text

Table Post Calculation

Introduction to Table Post Calculation

In some cases you may need to do additional processing on tables after all of the tabulation is completed. For example, you may need to calculate ratios or sums using the tabulated data. CSPro allows you to do this by adding program logic in the Postcalc section of the Tally Attributes (Table) dialog. You use program logic, as you would in a batch or data entry application, to access and modify the cells in a table.

Tables are treated as two dimensional arrays or matrices. You can use the full range of CSPro arithmetic operations on individual cells, as well as on rows, columns and cell ranges of tables.

This section contains the following information:

Adding Rows and Columns For Post Calculation
Post Calculation For Individual Cells
Post Calculation For Rows, Columns and Ranges
Row and Column Indexing for Post Calculation

Adding Rows and Columns For Post Calculation

In order to use table post calculations to add additional calculations to the tabulated data, you first need a place in the table to put these calculations. This section describes how to add additional rows or columns to a table.

As an example, take adding a third column to the following table in which we will place the male to female ratio.

Table 1. Sex

	Sex		
	Total	Male	Female
Total	24,271	11,787	12,484

There are two ways to add a new column. We can add a new value to the existing value set for the existing variable or we can drag a new variable onto the table next to the existing one. Adding a new value to the value set creates a new column under the existing spanner while adding a new variable creates a new column under a new spanner.

Table 1. Sex

	Sex			
	Total	Male	Female	Male/Female Ratio
Total				

Male/Female Ratio column added as new value to value set for the variable Sex. The new column is under the Sex spanner.

Table 1. Sex

	Sex			Male/Female Ratio
	Total	Male	Female	
Total				

Male/Female Ratio column added by dragging a new variable onto the table. The new column is under a separate spanner.

If you add a new value to the value set, make sure to set the value for the new value set entry to a number that does not exist in your data file, otherwise your totals will be incorrect. In the case of Sex, you can use 3, which should not appear in the edited data.

If you would rather add a new variable to the table to create the additional column, you can add any variable in the dictionary since the numbers in the new column will be overwritten by the postcalc logic. You can either add a new value set to the variable which uses the text you want for the column header (for example "Male/Female Ratio") or you can use an existing value set and customize the column header text once you have placed the variable on the table.

New rows can be added in the same ways, adding a new value to the value set of a variable on the rows or dragging a new variable onto the rows of the table.

See also: Add a Variable to a Tabulation, Implications of Data Dictionary Value Sets

Post Calculation For Individual Cells

CSPro supports modifying individual cells in a table through postcalc logic.

As an example, take adding a third cell to the following table in which we will place the male to female ratio. The male to female ratio is simply the number of males divided by the number of females. We need add a new column to the table and then add postcalc logic to divide the number of males by the number of females and put the result in the new cell.

Table 1. Sex

	Sex			
	Total	Male	Female	Male/Female Ratio
Total				

First, we must add the new column to the table as described in Adding Rows and Columns For Post Calculation.

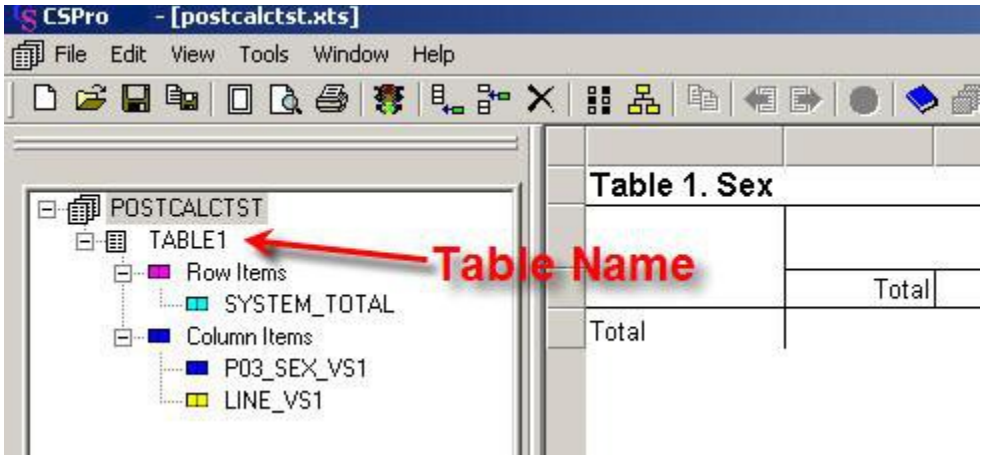
Once the new column has been created, all that is left is to add the postcalc logic. Postcalc logic in CSPro tabulation applications is similar to working with arrays in a CSPro batch edit or data entry application. To access an individual cell in a table, use the table name followed by the indices of the row and column in parentheses:

<table name>(<row>, <column>)

where:

<table name> is the name of the table.

To see the names of the tables, click on the Tables tab in the bottom left of the CSPro window to show the Tables Tree and then select "Names in Trees" from the View menu or press Ctrl+T. This toggles between showing the table titles and the table names in the Tables Tree.



<row> is the row number in the table.

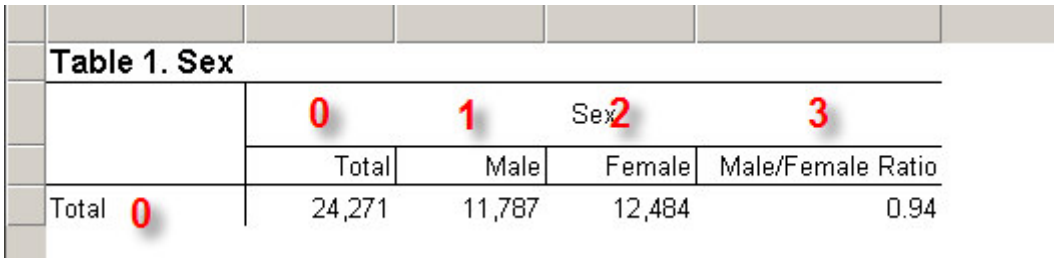
<column> is the column number in the table.

Row and column numbering starts at zero.

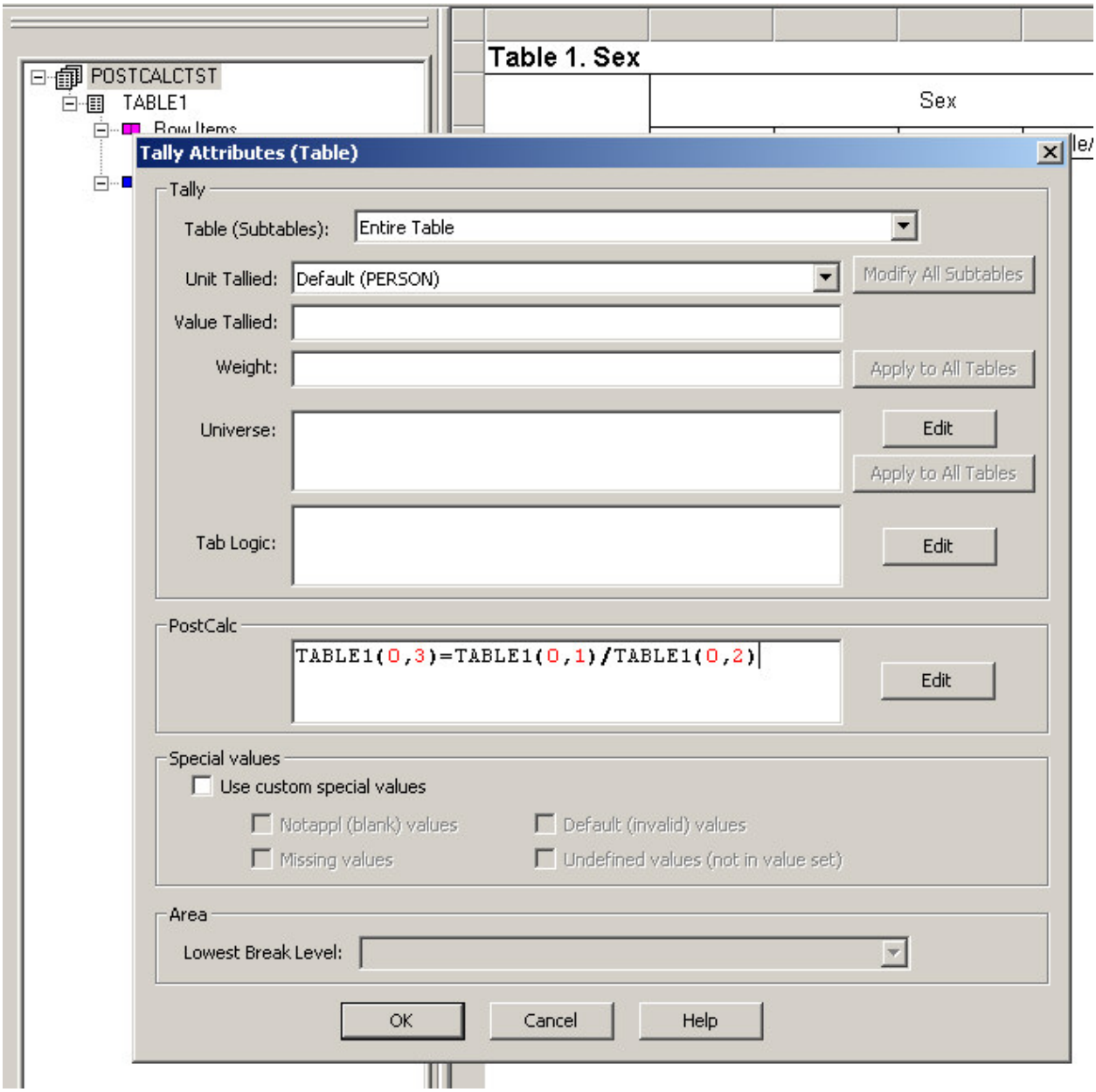
In this example, the following logic divides the number of males (row 0, column 1) by the number of females (row 0, column 2) and puts the result in male/female ratio (row 0, column 3):

```
TABLE1(0,3) = TABLE1(0,1) / TABLE1(0,2);
```

Note that column and row numbers start at zero, so the first column is column zero, the second column is column is column 1, etc…



To enter postcalc logic for a table, open the Tally Attributes (Table) dialog. You can then either type postcalc logic directly into the postcalc area or click on the "Edit" button next to the postcalc area to bring up a larger window to type in.



Once you have typed in the logic, press OK to dismiss the dialog and then run the table. You may need to change the number of the decimal places in the row, column or cell containing the calculated values.

Table 1. Sex

	Sex			
	Total	Male	Female	Male/Female Ratio
Total	24,271	11,787	12,484	0.94

Postcalc logic can contain numeric constants as well as table values. For example if you wanted men per 100 women rather than male to female ratio, you could use the following logic:

```
TABLE1(0,3) = 100 * TABLE1(0,1) / TABLE1(0,2);
```

This multiplies the result of the division by 100 to give the number of men per 100 women.

Table 1. Sex

	Sex			
	Total	Male	Female	Males per 100 Females
Total	24,271	11,787	12,484	94.42

Multiple statements may be entered in postcalc, provided that each one is terminated by a semicolon. For example, if you wanted to calculate male to female ratio in column 3 and males per 100 females in column 4 you would write:

```
TABLE1(0,3) = TABLE1(0,1) / TABLE1(0,2);
TABLE1(0,4) = 100 * TABLE1(0,1) / TABLE1(0,2);
```

In fact, postcalc logic can contain nearly any of the statements and functions available in program logic in batch edit and data entry applications.

See also: Post Calculation For Rows, Columns and Ranges, Row and Column Indexing for Post Calculation

Post Calculation For Rows, Columns and Ranges

In addition to performing operations on individual cells, you can perform operations on cell ranges, rows and columns. As an example, take the following table, in which an additional column is added to calculate the male to female ratio for each group.

Table 1. Age in 5 year groups by Sex

Age in 5 year groups	0	1	Sex 2	3
	Total	Male	Female	Male/Female Ratio
Total	24,271	11,787	12,484	0.94
0 to 4 years	4,130	2,088	2,042	1.02
5 to 9 years	3,337	1,646	1,691	0.97
10 to 14 years	3,250	1,606	1,644	0.98
15 to 19 years	2,967	1,394	1,573	0.89
20 to 24 years	2,243	1,061	1,182	0.90
25 to 29 years	1,686	789	897	0.88
30 to 34 years	1,508	761	747	1.02
35 to 39 years	1,300	600	700	0.86
40 to 44 years	957	474	483	0.98
45 to 49 years	661	323	338	0.96
50 to 54 years	588	303	285	1.06
55 to 59 years	497	225	272	0.83
60 to 64 years	418	194	224	0.87
65 to 69 years	307	141	166	0.85
70 to 74 years	213	90	123	0.73
75 to 79 years	115	51	64	0.80
80 to 84 years	63	31	32	0.97
85 to 89 years	17	8	9	0.89
90 to 94 years	14	2	12	0.17
95 years and over	-	-	-	-

For information on how to add the additional column, see the previous section Adding Rows and Columns For Post Calculation.

The postcalc logic for assigning rows and columns is similar to that of individual cells, however there is a simplified syntax for working with cell ranges, rows and columns. You could set each cell individually with multiple statements as follows:

```
TABLE1(0,3) = TABLE1(0,1) / TABLE1(0,2);
TABLE1(1,3) = TABLE1(1,1) / TABLE1(1,2);
                    …
TABLE1(2,3) = TABLE1(2,1) / TABLE1(2,2);
TABLE1(20,3) = TABLE1(20,1) / TABLE1(20,2);
```

However, that would take a lot of code. Instead, you can specify this one statement using a range or a wild card. To use a range, give the lower and upper limits separated by a colon:

```
TABLE1[0:20,3] = TABLE1[0:20,1] / TABLE1[0:20,2];
```

This means divide the cell in column 1 by the cell in column 2 and put the result in the cell in column 3 for each row from 0 to 20. You can also use a wildcard (an asterisk) to specify an entire row or column:

```
TABLE1[*,3] = TABLE1[*,1] / TABLE1[*,2];
```

Using an asterisk in place of the row index means that the operation applies to all rows in the table. In this case it is the same as specifying the range 0:20 since the table has 21 rows. An asterisk can also be used in place of the column index to specify every column in a table:

```
TABLE1[1,*] = TABLE1[2,*]; { copy row 2 into row 1 }
```

Note that when working with rows, columns and ranges, you must use square brackets "[]" rather than parentheses "( )". Parentheses may only be used when specifying individual cells.

You can only assign ranges or wildcards to each other if the dimensions of the ranges match. For example:

```
TABLE1[0:2, 0:3] = TABLE1[3:5, 0:3];
```

copies one 3 by 4 region of the table to another 3 by 4 region. However, the following code will fail since it attempts to copy a 3 by 4 region to a 3 by 3 region:

```
TABLE1[0:2, 0:2]= TABLE1[3:5, 0:3]; { This does not work ! }
```

See also: Post Calculation For Individual Cells, Row and Column Indexing for Post Calculation

Row and Column Indexing for Post Calculation

When determining the indices to use for particular rows and columns, always remember that the first row or column is at index zero and that caption rows (rows with no data) should be skipped. Also note that hidden data rows and columns should be counted even if they are not visible in the table. Always make sure that view hidden parts is turned on when determining row and column indices.

Table 2. Sex and Literacy by Urban/Rural for Age 15 and Above

		Urban/Rural		
		Total	Urban	Rural
	Sex			
	Total			
	Literacy			
0	Total	13,554	8,025	5,529
1	Literate	11,313	7,179	4,134
2	Illiterate	2,241	846	1,395
3	Percent Literate	83.5	89.5	74.8
	Male			
	Literacy			
4	Total	6,447	3,749	2,698
5	Literate	5,567	3,424	2,143
6	Illiterate	880	325	555
7	Percent Literate	86.4	91.3	79.4
	Female			
	Literacy			
8	Total	7,107	4,276	2,831
9	Literate	5,746	3,755	1,991
10	Illiterate	1,361	521	840
11	Percent Literate	80.8	87.8	70.3

For example, in the table above, in order to calculate the percentage of people in each sex category (Male, Female, Total), we set the values in rows 3, 7 and 9. The caption rows, "Sex", "Total", "Literacy", "Male", and "Female" are not counted, but the hidden rows, the three "Illiterate" rows, are counted. The postcalc logic follows:

```
TABLE2[3,*] = 100 * TABLE2[1,*]/TABLE2[0,*]; {% literate Total }
TABLE2[7,*] = 100 * TABLE2[5,*]/TABLE2[4,*]; {% literate Male }
TABLE2[11,*] = 100 * TABLE2[9,*]/TABLE2[8,*]; {% literate Female}
```

When a table includes percents, the row and column indices are a bit more complicated. Although the percent rows/columns are interleaved with the rows/columns for the counts, the percent rows/columns are numbered after the counts. In other words the index for the first percent row or column in a subtable always starts after the last count row or column. For example, in the table below, the rows for the values of marital status under the male caption are counted one after another as indices 0 through 4 and are then followed by the percent rows for marital status as indices 5 through 9. The first percent row is at index 5 which follows the last count row (Never Married) at index 4.

Table 3. Sex and Marital status by Urban/Rural for Age 15 and Above

		Urban/Rural		
		Total	Urban	Rural
	Male			
0	Married	3,434	1,888	1,546
5	Percent	25.3	23.5	28.0
1	Divorced	199	86	113
6	Percent	1.5	1.1	2.0
2	Separated	-	-	-
7	Percent	-	-	-
3	Widowed	148	68	80
8	Percent	1.1	0.8	1.4
4	Never Married	2,666	1,707	959
9	Percent	19.7	21.3	17.3
	Female			
10	Married	3,539	1,963	1,576
15	Percent	26.1	24.5	28.5
11	Divorced	599	311	288
16	Percent	4.4	3.9	5.2
12	Separated	-	-	-
17	Percent	-	-	-
13	Widowed	638	354	284
18	Percent	4.7	4.4	5.1
14	Never Married	2,331	1,648	683
19	Percent	17.2	20.5	12.4
20	Unmarried men per 100 unmarried women	84.4	80.5	91.8

In this example, to calculate the ratio of unmarried men per 100 unmarried women, we need to divide the number of unmarried men (divorced men + separated men + widowed men + never married men) by the number of unmarried women (divorced women + separated women + widowed women + never married women) and multiply the result by 100. The postcalc logic is therefore:

```
TABLE3[20,*] =
100 *
(TABLE3[1,*] + TABLE3[2,*] + TABLE3[3,*] + TABLE3[4,*])/
(TABLE3[11,*] + TABLE3[12,*] + TABLE3[13,*] + TABLE3[14,*]);
```

See also: Post Calculation For Individual Cells, Post Calculation For Rows, Columns and Ranges

Run Production Tabulations

Introduction to Production Tabulations

Production applications can be run either interactively from CSPro or using batch (*.BAT) files to execute one CSPro program after another. For small surveys all the tabulation work can usually be done within CSPro. For a population census or large survey, you may want to create batch files to perform processing.

When you run tabulation from CSPro, you can produce the entire tabulation in one run or you can tabulate in parts, that is one step, or process, at a time. The entire tabulation process consists of three processes:

Tabulate – One or more data files are processed to produce table matrices, row and columns of tabulated numbers.

Consolidate – Tables matrices are added together for different lower geographic areas, say districts, to produce tables for higher geographic areas such as provinces and country. This process is performed only when area specifications are given.

Format – Table matrices are surrounded with titles, headings, and stubs to be displayed or printed for the end user.

You can have CSPro Run in Parts to perform each of these processes one at a time.

Run All in Batch

You can run a complete tabulation from a batch program by executing **CSTab.exe** and using a PFF file as the command line parameter. For example, if your PFF file name is "MyTabs.pff", you can launch CSTab by:

```
Start /wait "C:\Program Files\CSPro 6.0\CSTab.exe" MyTabs.pff
```

This launches the program CSTab.exe to run with the parameters specified in the PFF file MyTabs.pff. Note that using Start /wait is not strictly necessary; it simply ensures that the command does not terminate until CSTab.exe has finished processing. This is useful when there are other commands that follow which depend on CSTab completing before they can be executed.

You can create a PFF file in two ways:

- Run the tabulation from CSPro. It will save the *.PFF file it generates in the same folder with your tabulation application. The *.PFF will have the same name as the application with .PFF appended. Rename and modify this file with a text editor (such as Notepad or Wordpad).
- Create a new *.PFF file using a text editor.

The following shows an example of a tabulation PFF file. Note that a PFF file is not case sensitive. You can use any combination of upper and lower case text.

```
[Run Information]
Version=CSPro 6.0
AppType=Tabulation
Operation=All

[Files]
Application=.\MyTabs.xtb
InputData=.\MyData.dat
Listing=.\MyTabs.xtb.lst
AreaNames=.\MyAreaNames.anm
OutputTBW=.\MyTables.xtb.tbw

[Parameters]
ViewListing=OnError
ViewResults=Yes
```

The **[Run Information]** block is required and must appear exactly as shown in the example above.

The **[Files]** block is required and defines all files used in the tabulation run. A description of the files is as follows:

- **Application =** the tabulation edit application you created
- **InputData =** the data file to be tabulated -- If there is more than one input data file, insert multiple InputData lines.
- **Listing =** a report of the tabulation processing
- **AreaNames =** the areanames file used only if there is area processing
- **OutputTBW =** the output formatted tables

If any required files are not coded or are missing, the file association dialog box will be displayed allowing you to fill in or change the missing file names.

The **[Parameters]** block is optional. If allows to specify additional aspects of the tabulation run.

- **ViewListing =** specifies how the tabulation run listing is displayed. If ViewListing is missing, Always is assumed.
  Always - the listing is always displayed
  OnError - the listing is displayed only when an error or an invalid subscript warning occurred
  Never - the listing is never displayed
- **ViewResults =** specifies whether or not the formatted tables file (*.TBW) is displayed in TableViewer at the end of the run. If ViewResults is missing, Yes is assumed.
  Yes - the tables are displayed
  No - the tables are not displayed

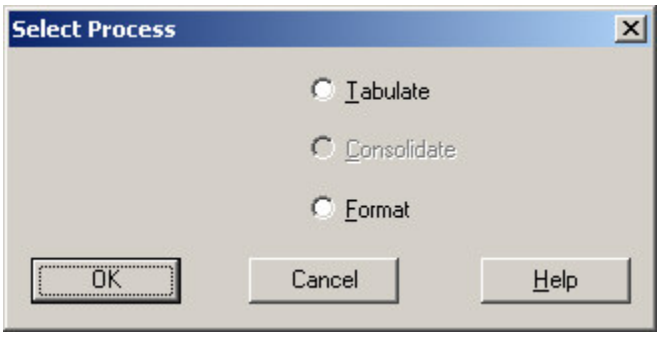
Run in Parts

Introduction to Run in Parts

When CSPro runs a tabulation application performs two or three processes. These processes are **Tabulate** the data file, **Consolidate** geographic areas (if the application uses area processing), and **Format** the tables with text. Run in parts allows you perform each of these processes separately.

Run in parts is used when you need to save the intermediate *.TAB files for later use.

To run in parts interactively, from the **File** menu, select **Run Parts**. Then select the process you want to run. The Consolidate process will be grayed out if you are not using area processing in your application.

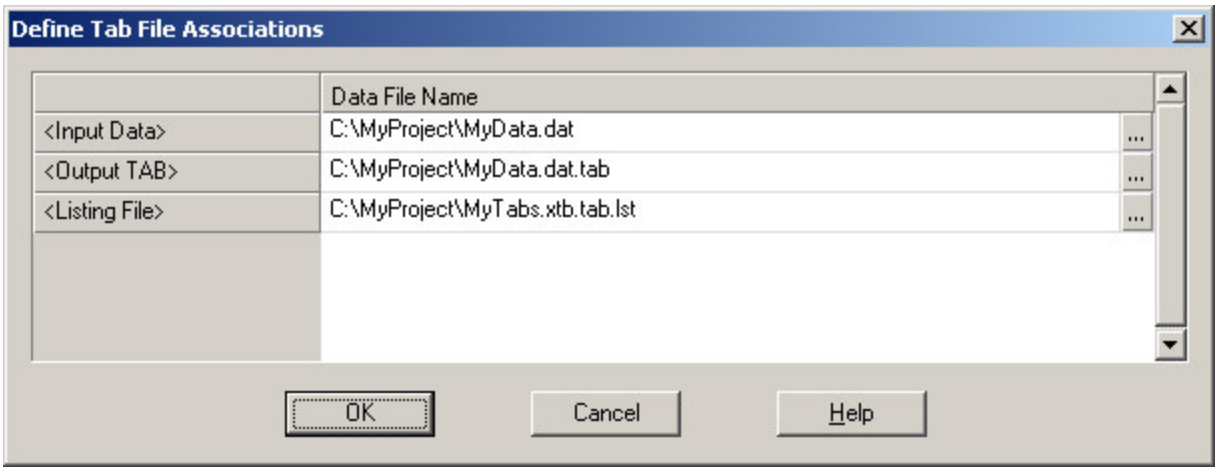


See also: Run Tabulate Interactively, Run Consolidate Interactively, Run Format Interactively, Run Tabulate in Batch, Run Format in Batch, Run Consolidate in Batch

Run Tabulate Interactively

The Tabulate process reads the input data file(s), runs the tabulation application, and produces a set of table matrices that are stored in a tables file (*.TAB).

When the Tabulate process is selected, the following files are requested:



Input Data: The data file(s) being tabulated. There is NO required extension for CSPro data files. Multiple input data files can be selected using the browse button.

Output TAB: The table matrix file created during tabulation. The *.TAB extension is required. The name should be associated with the corresponding data file especially if multiple data files are tabulated using the same application.

Listing File: The diary-type report generated by the tabulation run showing record and case counts and any problems encountered. The .LST extension is not required.

This is a sample of a listing file where no problems were encountered during tabulation:

```
CSPRO Process Summary
+-----------------------------------------------------+
| 29143 Records Read ( 100% of input file) |
| 0 Ignored ( 0 unknown, 0 erased) |
| 0 Messages ( 0 U, 0 W, 0 E) |
+---------+--------------+--------------+-------------+
| Level | Input Case | Bad Struct | Level Post |
+---------+--------------+--------------+-------------+
| 1 | 4872 | 0 | 4872 |
+---------+--------------+--------------+-------------+

Process Messages

CSPRO Executor Normal End
-----------------------------------------------------
```

See also: Run Consolidate Interactively, Run Format Interactively

Run Tabulate in Batch

You can run the tabulate process from a batch program by executing **CSTab.exe** and using a PFF file as the command line parameter. For example, if your PFF file name is "MyTabs.pff", you can launch CSTab by:

```
Start /wait "C:\Program Files\CSPro 6.0\CSTab.exe" MyTabs.tab.pff
```

This launches the program CSTab.exe to run with the parameters specified in the PFF file MyTabs.pff. Note that using Start /wait is not strictly necessary; it simply ensures that the command does not terminate until CSTab.exe has finished processing. This is useful when there are other commands that follow which depend on CSTab completing before they can be executed.

You can create a PFF file in two ways:

- Run the tabulation from CSPro. It will save the *.PFF file it generates in the same folder with your tabulation application. The *.PFF will have the same name as the application with .PFF appended. Rename and modify this file with a text editor (such as Notepad or Wordpad).
- Create a new *.PFF file using a text editor.

The following shows an example a PFF file for the tabulation process. Note that a PFF file is not case sensitive. You can use any combination of upper and lower case text.

```
[Run Information]
Version=CSPro 6.0
AppType=Tabulation
Operation=Tab

[Files]
Application=.\MyTabs.xtb
InputData=.\MyData.dat
Listing=.\MyTabs.xtb.tab.lst
TabOutputTAB=.\MyData.dat.tab

[Parameters]
ViewListing=OnError
```

The **[Run Information]** block is required and must appear exactly as shown in the example above.

The **[Files]** block is required and defines all files used in the tabulation run. A description of the files is as follows:

- **Application =** the tabulation application you created
- **InputData =** the data file to be tabulated -- If there is more than one input data file, insert multiple InputData lines.
- **Listing =** a report of the tabulate process
- **TabOutputTAB =** the output table matrices file

If any required files are not coded or are missing, the file association dialog box will be displayed allowing you to fill in or change the missing file names.

The **[Parameters]** block is optional. If allows to specify additional aspects of the tabulation run.

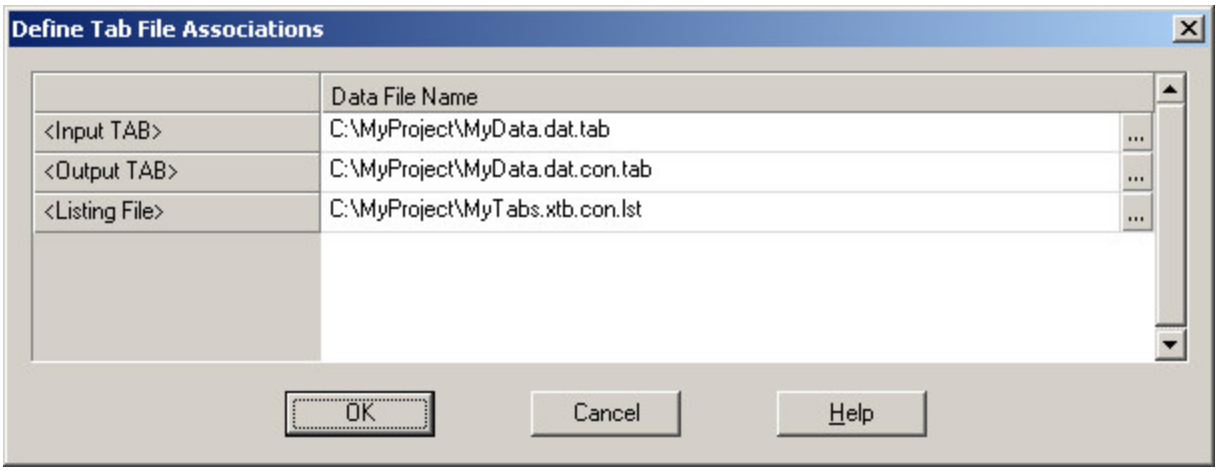
- **ViewListing =** specifies how the tabulation run listing is displayed. If ViewListing is missing, Always is assumed.
  Always - the listing is always displayed
  OnError - the listing is displayed only when an error or an invalid subscript warning occurred
  Never - the listing is never displayed

See also: Run Consolidate in Batch, Run Format in Batch

Run Consolidate Interactively

The Consolidation process reads the table matrix file(s) output from Tabulate process, and produces another table matrix file (*.TAB) based on consolidation specifications given in the tabulation application.

When the Consolidate process is selected, the following files are requested:



Input TAB: The tabulated table matrices file(s) to be consolidated. Multiple input *.TAB files can be selected using the browse button. The *.TAB extension is required.

Output TAB: The consolidated table matrices file created [or replaced] from the input TAB file(s). This TAB name must be different from the input name and the extension is required.

Listing File: The report generated for the run showing area codes found in the data and tables available for those codes. Blank ‘codes’ indicate summary levels. The .LST extension is not required.

This is a sample of a listing file for a Consolidation process:

```
CSPro Process Summary
+-----------------------------------------------+
| 1568 Slices Read ( 100% of Input file) |
+-----------------------------------------------+


Process Messages

PROVINCE DISTRICT Table Names
-------- -------- -----------
TABLE1 TABLE2 TABLE3 TABLE4
1 TABLE1 TABLE2 TABLE3 TABLE4
1 1 TABLE1 TABLE2 TABLE3 TABLE4
1 2 TABLE1 TABLE2 TABLE3 TABLE4
1 3 TABLE1 TABLE2 TABLE3 TABLE4
1 4 TABLE1 TABLE2 TABLE3 TABLE4
1 5 TABLE1 TABLE2 TABLE3 TABLE4
  . . .
```

Tables 1, 2, 3 and 4 are available for the following geographies:

Total [Country] indicated by Province and District = blank
Province 1 indicated by Province = 1 and District = blank
Province 1 and District 1 indicated by Province = 1 and District = 1
Etc.

See also: Run Tabulate Interactively, Run Format Interactively

Run Consolidate in Batch

You can run the consolidation process from a batch program by executing **CSTab.exe** and using a PFF file as the command line parameter. For example, if your PFF file name is "MyTabs.pff", you can launch CSTab by:

```
Start /wait "C:\Program Files\CSPro 6.0\CSTab.exe" MyTabs.con.pff
```

This launches the program CSTab.exe to run with the parameters specified in the PFF file MyTabs.pff. Note that using Start /wait is not strictly necessary; it simply ensures that the command does not terminate until CSTab.exe has finished processing. This is useful when there are other commands that follow which depend on CSTab completing before they can be executed.

You can create a PFF file in two ways:

- Run the tabulation from CSPro. It will save the *.PFF file it generates in the same folder with your tabulation application. The *.PFF will have the same name as the application with .PFF appended. Rename and modify this file with a text editor (such as Notepad or Wordpad).
- Create a new *.PFF file using a text editor.

The following shows an example of a PFF file for the consolidation process. Note that a PFF file is not case sensitive. You can use any combination of upper and lower case text.

```
[Run Information]
Version=CSPro 6.0
AppType=Tabulation
Operation=Con

[Files]
Application=.\MyTabs.xtb
ConInputTAB=.\MyData.dat.tab
Listing=.\MyTabs.xtb.con.lst
ConOutputTAB=.\MyData.dat.con.tab

[Parameters]
ViewListing=OnError
```

The **[Run Information]** block is required and must appear exactly as shown in the example above.

The **[Files]** block is required and defines all files used in the tabulation run. A description of the files is as follows:

- **Application =** the tabulation application you created
- **ConInputTAB =** the input table matrices file – These are the output from the tabulate process. If there are multiple input table matrices files, insert multiple ConInputTab lines.
- **ConOutputTAB =** the output table matrices file
- **Listing =** a report of the consolidate process

If any required files are not coded or are missing, the file association dialog box will be displayed allowing you to fill in or change the missing file names.

The **[Parameters]** block is optional. If allows to specify additional aspects of the tabulation run.

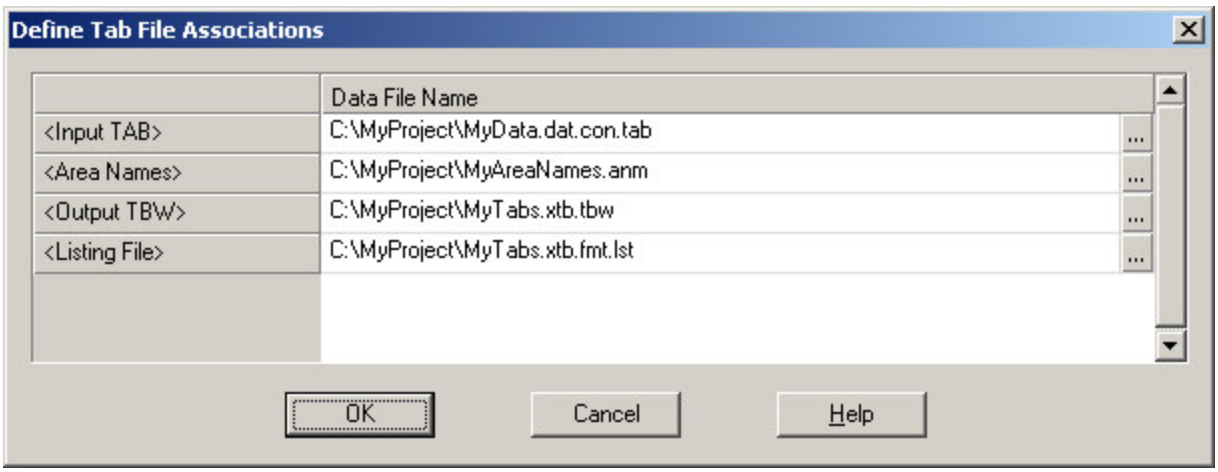
- **ViewListing =** specifies how the tabulation run listing is displayed. If ViewListing is missing, Always is assumed.
  Always - the listing is always displayed
  OnError - the listing is displayed only when an error occurred
  Never - the listing is never displayed

See also: Run Tabulate in Batch, Run Format in Batch

Run Format Interactively

The Format process reads the table matrix file(s) output from Tabulate or Consolidations process, and produces tables files (*.TBW) based on format specifications given in the tabulation application.

When the Format process is selected, the following files are requested:



Input TAB: The tabulated or consolidated matrices file. If application does not have area processing, it was created in the Tabulate process, if it has area processing was performed, it was created in the Consolidate process. The .TAB extension is required.

Area Names: Only used for applications with area processing. An Area Names File is used to associate the areas codes in the TAB file with their descriptive text. The .ANM extension is required.

Output TBW: The CSPro Table Viewer file created by merging the matrices from the input TAB file, the table format specifications from the tabulations application, and the optional area names. The .TBW extension is required.

Listing File: The report of the Format process.

This is a sample of a listing file for a Format process:

```
CSPRO Process Summary
+-----------------------------------------------------+
| 8 Slices Read ( 100% of input file) |
| 0 Ignored ( 0 unknown, 0 erased) |
| 0 Messages ( 0 U, 0 W, 0 E) |
+-----------------------------------------------------+
```

```
Process Messages

CSPRO Executor Normal End
-----------------------------------------------------------------------
```

See also: Run Tabulate Interactively, Run Consolidate Interactively, Introduction To Table Viewer

Run Format in Batch

You can run the format process from a batch program by executing **CSTab.exe** and using a PFF file as the command line parameter. For example, if your PFF file name is "MyTabs.pff", you can launch CSTab by:

```
Start /wait "C:\Program Files\CSPro 6.0\CSTab.exe" MyTabs.fmt.pff
```

This launches the program CSTab.exe to run with the parameters specified in the PFF file MyTabs.pff. Note that using Start /wait is not strictly necessary; it simply ensures that the command does not terminate until CSTab.exe has finished processing. This is useful when there are other commands that follow which depend on CSTab completing before they can be executed.

You can create a PFF file in two ways:

- Run the tabulation from CSPro. It will save the *.PFF file it generates in the same folder with your tabulation application. The *.PFF will have the same name as the application with .PFF appended. Rename and modify this file with a text editor (such as Notepad or Wordpad).
- Create a new *.PFF file using a text editor.

The following shows an example a PFF file for the format process. Note that a PFF file is not case sensitive. You can use any combination of upper and lower case text.

```
[Run Information]
Version=CSPro 6.0
AppType=Tabulation
Operation=Format

[Files]
Application=.\MyTabs.xtb
FormatInputTAB=.\MyData.dat.con.tab
AreaNames=.\MyAreaNames.anm
Listing=.\MyTabs.xtb.fmt.lst
OutputTBW=.\MyTables.xtb.tbw

[Parameters]
ViewListing=OnError
ViewResults=No
```

The **[Run Information]** block is required and must appear exactly as shown in the example above.

The **[Files]** block is required and defines all files used in the tabulation run. A description of the files is as follows:

- **Application =** the tabulation application you created
- **FormatInputTAB =** the input table matrices file – This is output from the tabulate or consolidate process.
- **Listing =** a report of the tabulation processing
- **AreaNames =** the area names file, only if there is area processing

- **OutputTBW =** the output formatted tables

If any required files are not coded or are missing, the file association dialog box will be displayed allowing you to fill in or change the missing file names.

The **[Parameters]** block is optional. If allows to specify additional aspects of the tabulation run.

- **ViewListing =** specifies how the tabulation run listing is displayed. If ViewListing is missing, Always is assumed.
  Always - the listing is always displayed
  OnError - the listing is displayed only when an error occurred
  Never - the listing is never displayed
- **ViewResults =** specifies whether or not the formatted tables file (*.TBW) is displayed in TableViewer at the end of the run. If ViewResults is missing, Yes is assumed.
  Yes - the tables are displayed
  No - the tables are not displayed

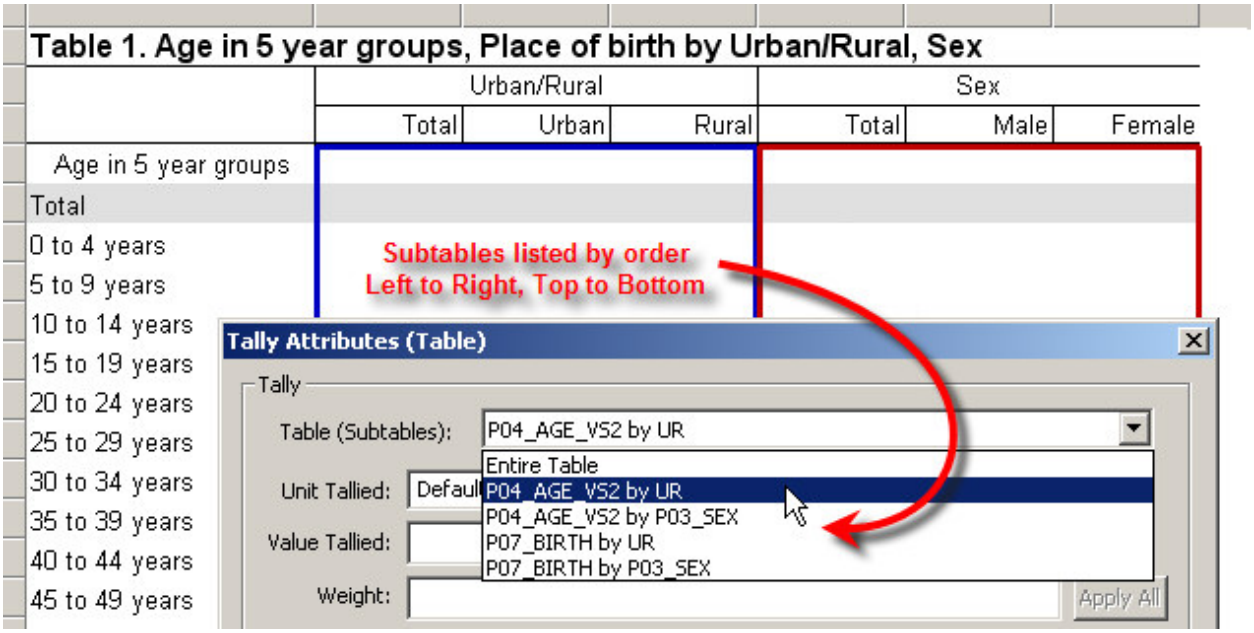
See also: Run Tabulate in Batch, Run Consolidate in Batch

Advanced Table Topics

Using Subtables

Subtables are separate and individual tabulations ("sub") contained within the definition of a single table. Subtables can be thought of as merged or concatenated to create the whole table.

Each subtable, being independent, has its own set of Tally Attributes as well as the Tally Attributes for the "Entire Table". To access the attributes of a subtable either right click while the cursor is in the area of the subtable and then select Tally Attributes (Subtable) or right click anywhere in the table, select Tally Attributes (Table), and then select the desired subtable from the drop down menu as shown below.



In the menu subtables are listed as they appear in the "big" table, from left to right then from top to bottom. The "Names" of the subtables are created from the value sets present. If an item has only one value set then the name of the item itself appears, e.g., P03_SEX in example. If an item has more than one value set then the name of the value set appears, e.g., P04_AGE_VS2 in the example.

Any attributes entered in the "Entire Table" menu except "Unit Tallied" are also incorporated into the subtables. Entries in this menu apply to all subtables in the table.

In the Tally Attributes (Table)

- If an item or value is given for "Value Tallied" or "Weight" it is also MULTIPLIED by any corresponding value give in any subtable.
- Any entry in the "Universe" box is combined with any universe in a subtable using "and". In other words, both universe criteria must be met in order for a tally to be made.
- Of course, any checked "Special values" or "Lowest Break Level" (applicable only for area processing) always refer to the entire table including any subtables.

With the exception of attributes applying to entire table, individual subtables can each be assigned different weights, universes, etc.

See Also: Create Tabulations with Multiple Variables, Tally Attributes for a Table

Changing the Unit of Tabulation

This attribute is available on the Tally Attribute (Table/Subtable) menu.

"Unit Tallied" allows you to change the unit of computation for the table or subtable. The unit of computation is the level, record or item in the dictionary that is counted for the tally. It is basically a "looping" unit for the tally, i.e., for each level, record or item do the tally.

For example, when tallying categories of the rent variable on the housing record, by default, the unit would be the housing record so each household would be tallied once in the appropriate category. However, if the "unit of tally" were the person record, this would tally in the household's rent category once for each person in the case.

For example, a household paying $850/month rent would be in the $800 to $899 category. If the "unit of tally" were the household record then the tally would be done once for this case. If the "unit of tally" were the person record and there are 4 persons in the household, then a tally is done for each person in the household resulting in 4 tallies for this case (instead of one).

Note: In most cases, the unit is left as the default assigned by CSPro.

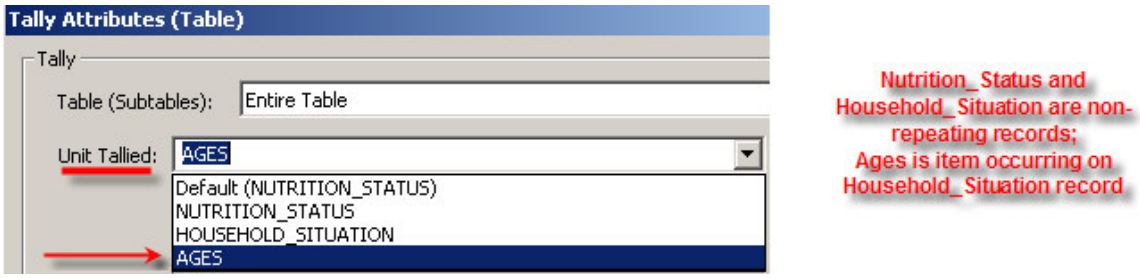
Basic rules of "units" (these refer to both tables or subtables):

- If table has all items from one record then the "unit of tally" is the record.
- If table has some items for a non-repeating record, e.g., house, and a repeating record, e.g., person, the "unit of tally" is the repeating record (person).
- If table has items from different non-repeating records then the "unit of tally" can be either record.
- If a non-repeating record contains an item with occurrences then the repeating item can also be the "unit of tally."
- If table has only ID items then the "unit of tally" can be the case (Quest) or any repeating record or repeating item.
- No table can contain items from different repeating records unless there is a relation defined between the records in the data dictionary (see Relation Description). For relations, the "unit of tally" must be the name of the link.

(Non-repeating means MAXimum records is one)

Note: If a table/subtable contains items from a non-repeating record and another record, there is no tally if the non-repeating record is not present for the case. For example, if the case is missing its House record then no tallies involving items from the House will be done for this case.

For example, the table has an item from the Household_Situation record and one from the Nutrition_Status record, both of which are non-repeating. AGES is an item with occurrences on the Household_Situation record. The available "Unit Tallied" are:



If the "unit of tally" is either non-repeating record then only tally is made per case. If the "unit of tally" is the occurring item AGES, then a tally is made for each occurrence of age.

See Also: Tabulations Using Relations

Table Logic (tablogic)

Table logic (or tablogic) allows you to add CSPro logic that is executed during tabulation. This is mainly used to add your own "recoded" variables to tables instead of using existing variables from your dictionary. This is useful when the categories you want in your table must be tallied based on the values of more than one variable from the dictionary, in other words, when the categories you want must be computed based on the values of multiple variables. In such a case, you can create a new variable with the categories you want and write logic to set its value based on the values of existing variables.

For example, if you want to tabulate houses that have "complete plumbing," meaning that they have piped water, bathing facilities and a toilet inside the housing unit. The categories for "complete plumbing" are:

- Complete – piped water inside the unit, and a private toilet inside the unit and bathing facilities inside the unit.
- Some but not all –one or more of the three conditions above, but not all three.
- None – none of the above conditions.

The variable "complete plumbing" does not exist in the main dictionary. It can be however, be determined based on the values of the following three variables that are in the dictionary:

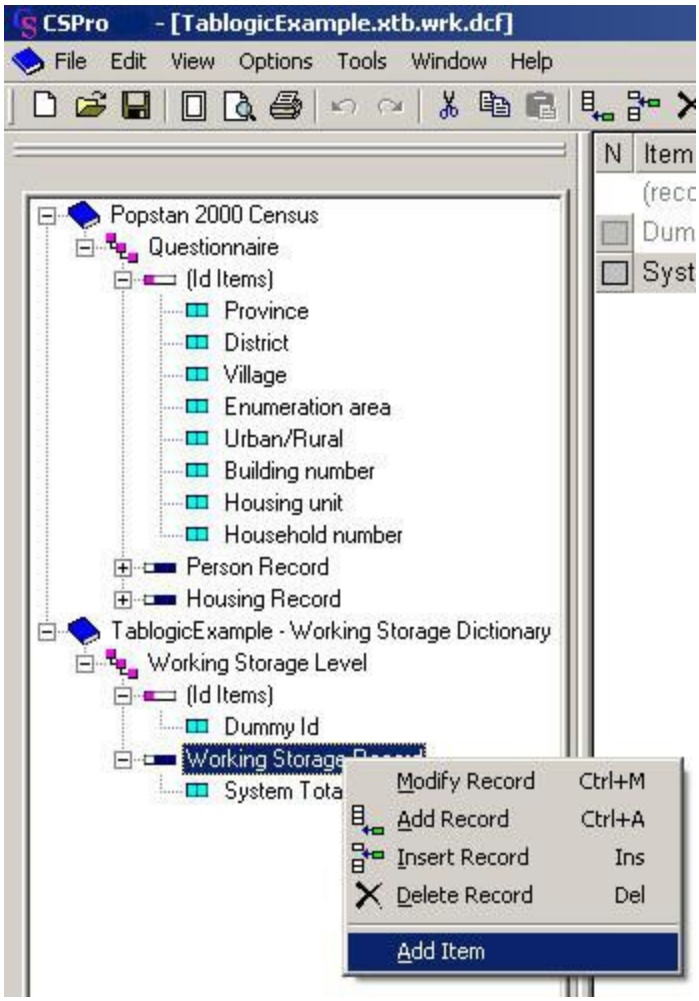
N	Value Set Label	Value Set Name	Value Label	From
☐	Source of water	H10_WATER_VS1		
☐			Inside piped	1
☐			Outside piped	2
☐			Bottled or canned	3
☐			Closed well or close	4
☐			Open well or spring	5
☐			River/Stream/Other	6

N	Value Set Label	Value Set Name	Value Label	From
☐	Type of bathing fac	H09_BATH_VS1		
☐			Exclusive	1
☐			Shared	2
☐			Hand basin	3
☐			Portable tub or basir	4
☐			Other	5
☐			None	6

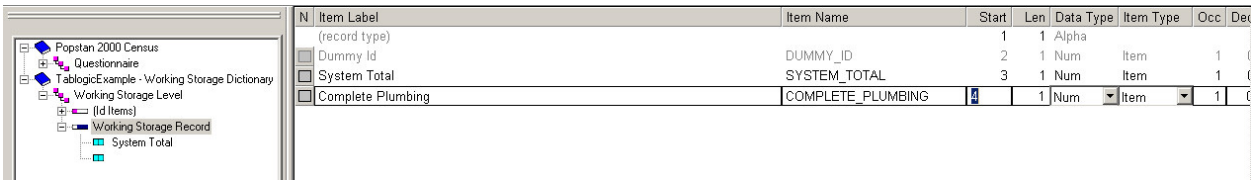
N	Value Set Label	Value Set Name	Value Label	From
☐	Type of toilet faciliti	H08_TOILET_VS1		
☐			Private toilet	1
☐			Shared toilet	2
☐			Outhouse	3
☐			Pit	4
☐			Other	5
☐			None	6

Rather than adding new variables to your existing dictionary, you can add new variables to the working storage dictionary. The working storage dictionary is a second dictionary that is created automatically when you create a tabulation application. This dictionary is similar to other dictionaries, however there is no data file associated with it. The variables in the working storage dictionary must be set by program logic.

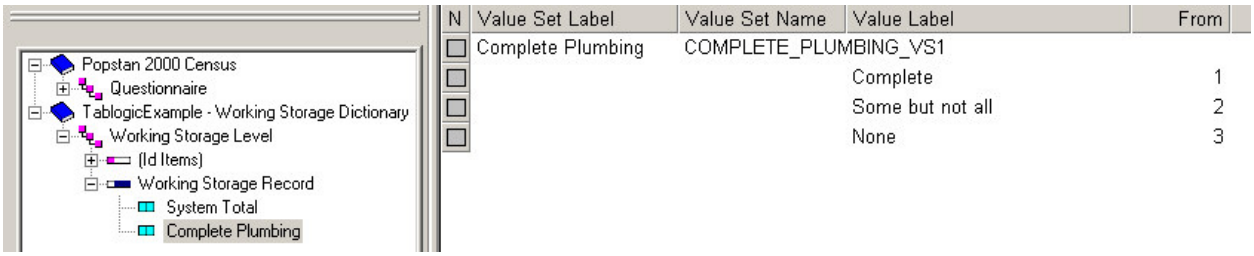
In this example, we will add the new "complete plumbing variable" to the working storage dictionary. The working storage dictionary appears just below the main dictionary in the dictionary tree. Adding a new variable to the working storage dictionary is the same as adding a variable to any dictionary. Right-click on the "Working Storage Record" under the working storage dictionary in the dictionary tree and choose "Add Item".



Then fill in the label, name and other properties for the new variable.



Next, create the value set for the new variable.

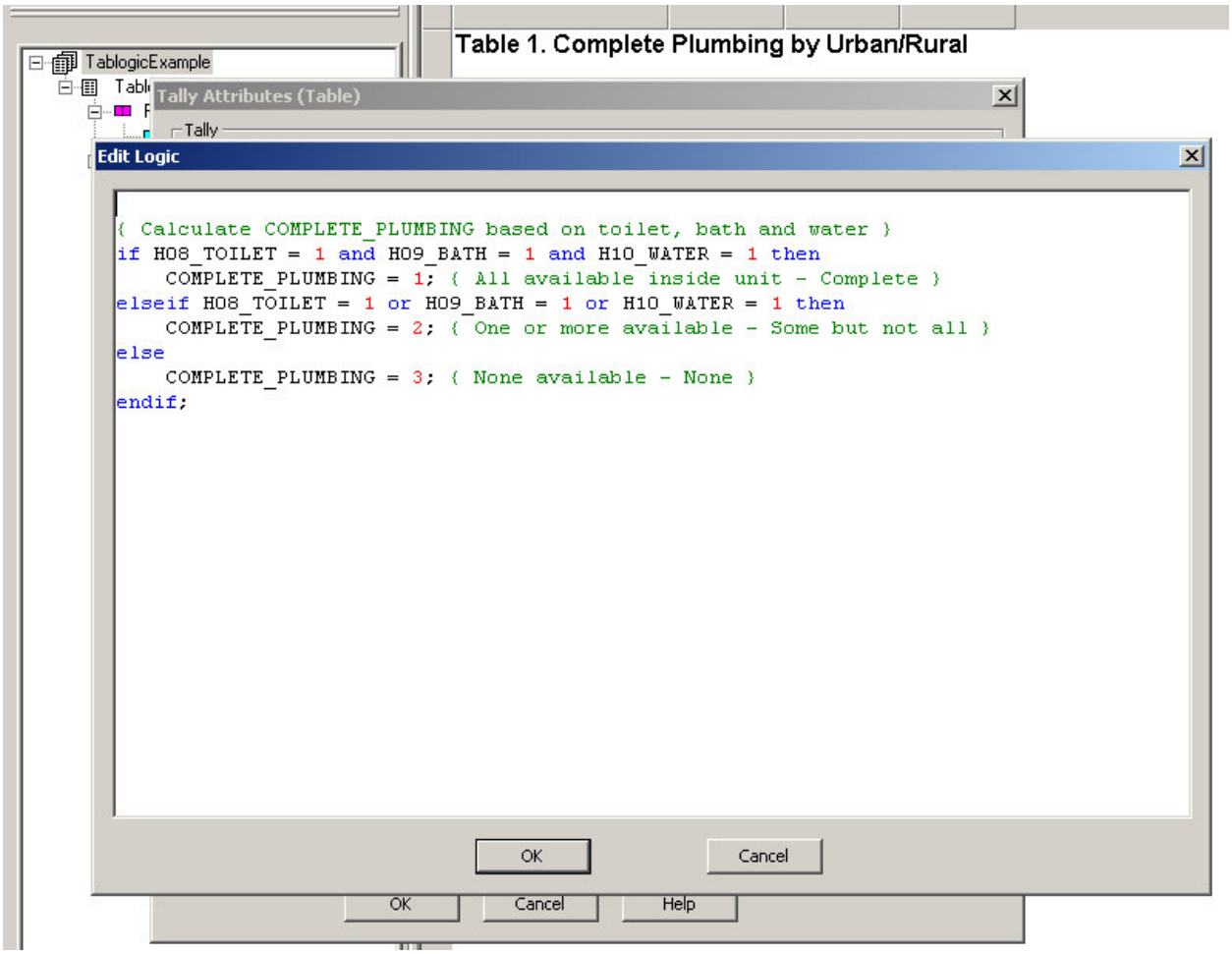


Now drag the new variable onto the table just as if it were a variable in the main dictionary.

Table 1. Complete Plumbing by Urban/Rural

Complete Plumbing	Urban/Rural		
	Total	Urban	Rural
Total			
Complete			
Some but not all			
None			

Finally add the tablogic to set the value of complete plumbing based on the values of source of water, toilet facilities and bathing facilities. Bring up the Tally Attributes (Table) dialog and enter the following code in the tab logic edit box:



The above logic will be executed once for each housing record to set the value of complete plumbing for that household. This will result in the following table:

Table 1. Complete Plumbing by Urban/Rural

Complete Plumbing	Urban/Rural		
	Total	Urban	Rural
Total	4,872	2,821	2,051
Complete	106	106	-
Some but not all	361	354	7
None	4,405	2,361	2,044

When using variables from the working storage dictionary, it is important to pay attention to the unit of tabulation. When using variables from the main dictionary, CSPro can determine the unit of tabulation based on the variables used. If you choose variables from the Person record, such as sex, the unit will be set to the person record. If you choose variables from the housing record, the unit will be set to the housing record. However, when you drag variables from the working storage dictionary onto the table, CSPro does not know which record or records the recodes for these variables will come from. It always sets the unit of tabulation to the first level (questionnaire). In the above example, this does not cause any problems because there is exactly one housing record per questionnaire. However, had we created a recode of variables from the person record, we would need to set the unit to the person record, otherwise we could get errors during tabulation. For more information see Changing the Unit of Tabulation.

Note that often it is preferable to create recoded variables in a batch edit program rather than during tabulation. This recode could have been accomplished by adding a new variable to the main dictionary for complete plumbing and writing a batch edit program to set the value of this variable for each case. Then the new complete plumbing variable could be dropped on the table directly, rather than using a working storage variable. The advantage of using batch edit is that the logic to set the value of the new variable is done once in a batch edit program rather than done as tablogic in each table in which the variable is used.

Another alternative is to use multiple subtables and value sets to simulate a recoded variable. This can be easier for simple variables, although for recoded variables with more than a few categories, it is generally simpler to use tablogic or batch edit. For more information see Recodes in Tables Using Value Sets and Subtables.

Tabulations Using Relations

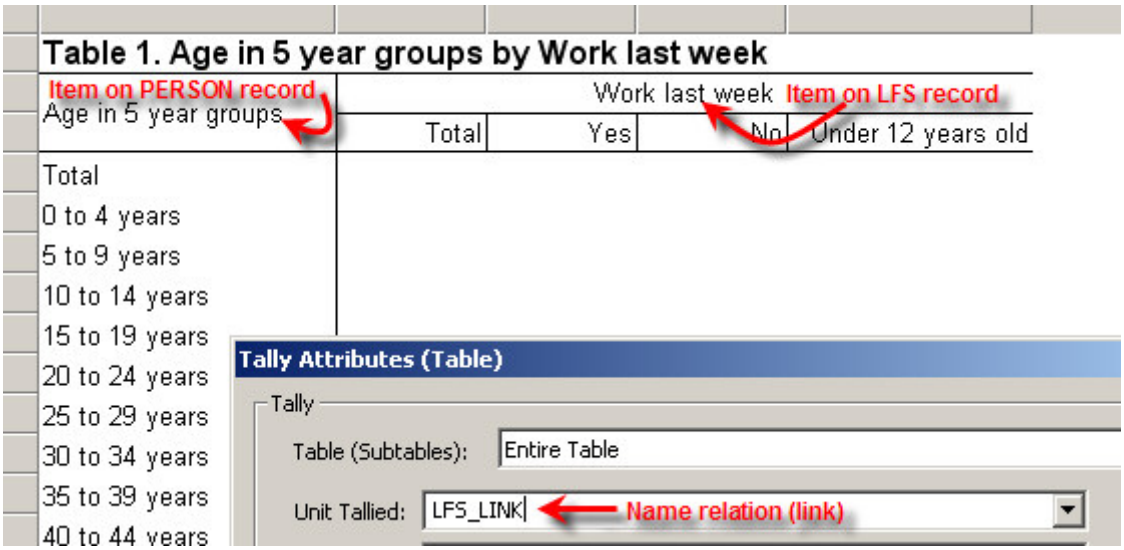
Relations provide a way of linking one multiple record or item to one or more multiple records or items. Once the items on different records are linked through a relation, tallies are done just as if all the items belonged to one record.

For example: LFS_LINK is a relation linking a labor force record (LFS) with a person record (PERSON). Both of these records are repeating records. The value that links two records is a number on each record (LFS_NUMBER and SERIAL_NUMBER) which identifies a single person, i.e., is the same on both records.

Here is the definition of the Link.

```
-----------------------------------------------------------------------------------------------------
Relation Name Primary Linked by Secondary Linked by
-----------------------------------------------------------------------------------------------------
LFS_LINK LFS LFS_NUMBER PERSON SERIAL_NUMBER
```

Here is table that tallies an item from both records:



If items from different multiply occurring records are used in the same subtable, CSPro will automatically choose the relation between them as the unit of tabulation if such a relation exists.

See Also: Relation Description

How To ...

Tabulate Items in Relations

A "relation" is created in the data dictionary. It links two record types through values on each record or occurrence number. If the two items have the same value then the records are linked, i.e., the items on the two record types are merged so all items "appear" to be on one big record.

A table or subtable using items from two record types linked through a relation is created in the same manner as any other table, i.e., drag and drop the items from each record. CSPro will automatically set the unit of tabulation to the relation between the two items. When the table is run, the tallies will be made from the linked records only.

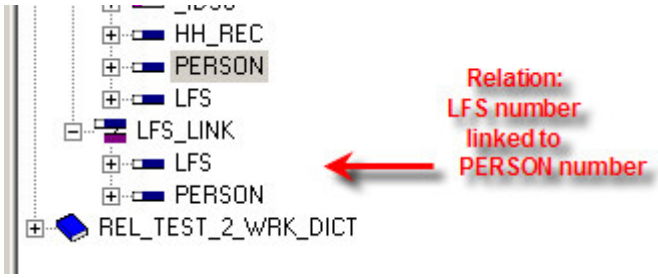
Example:
In a Labor Force Survey, the record with labor force information is related to the record with person information through sequence numbers.

Here is the definition of the Link.

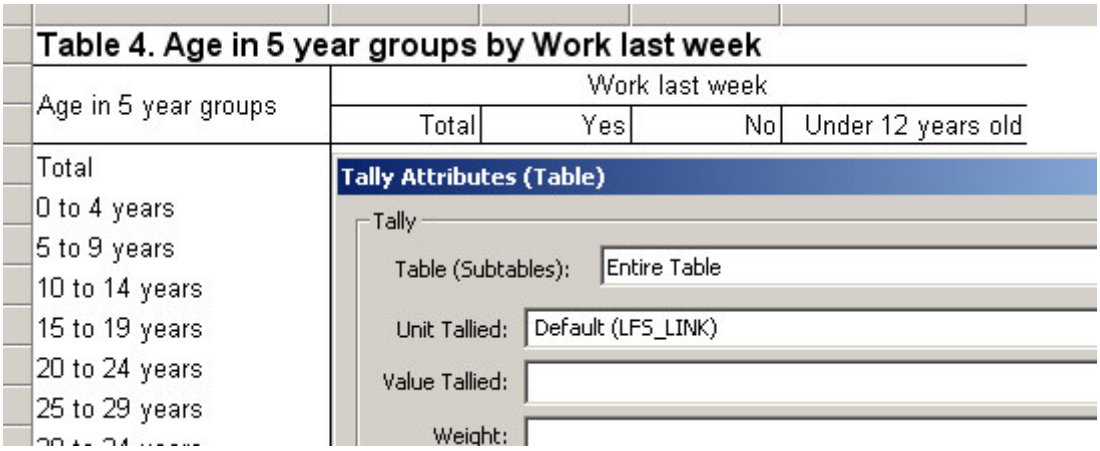
Relation Name Primary Linked by Secondary Linked by

LFS_LINK LFS LFS_NUMBER PERSON SERIAL_NUMBER

And how it is displayed in the data dictionary:



If "Age" value set is dropped for the Person record and the "Work last week?" value set is dropped from the LFS record then the following table is created.



Since the LFS record is the primary link, the tallies will be done for each occurrence of the LFS record. (This assumes that every LFS record in the case can be linked to a Person record in the case and that no "universe" has been specified.)

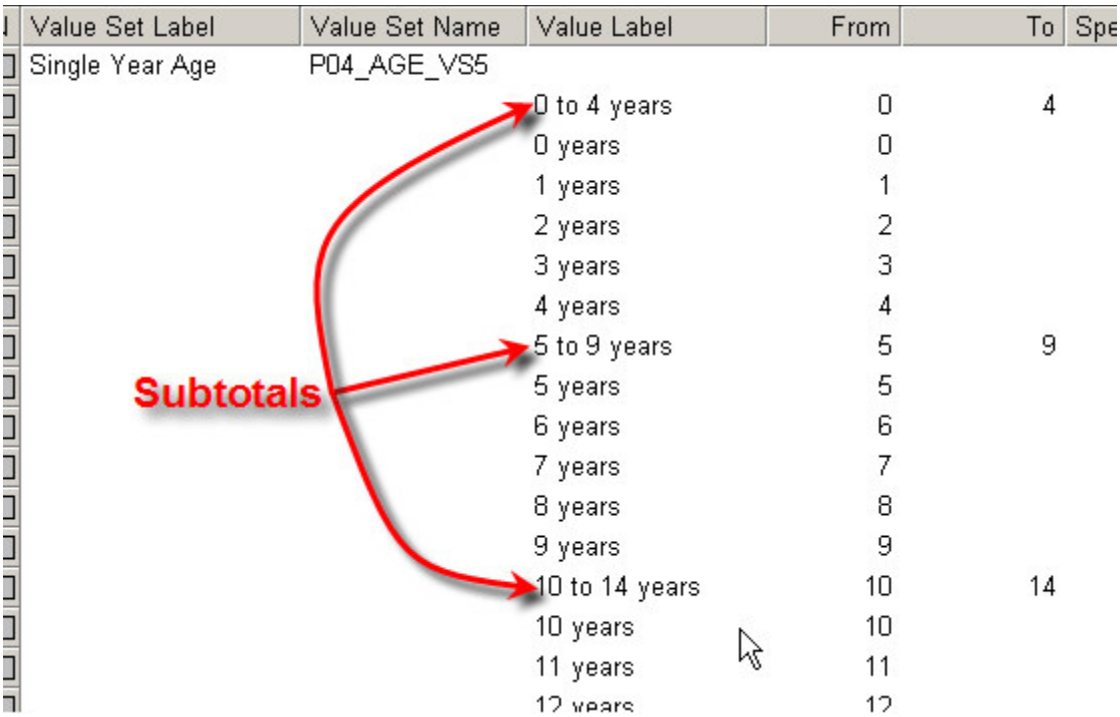
Note: The "Unit Tallied" must be the relation defined in the data dictionary. CSPro will automatically set the default to the relation if such a relation exists.

See Also: Relation Description

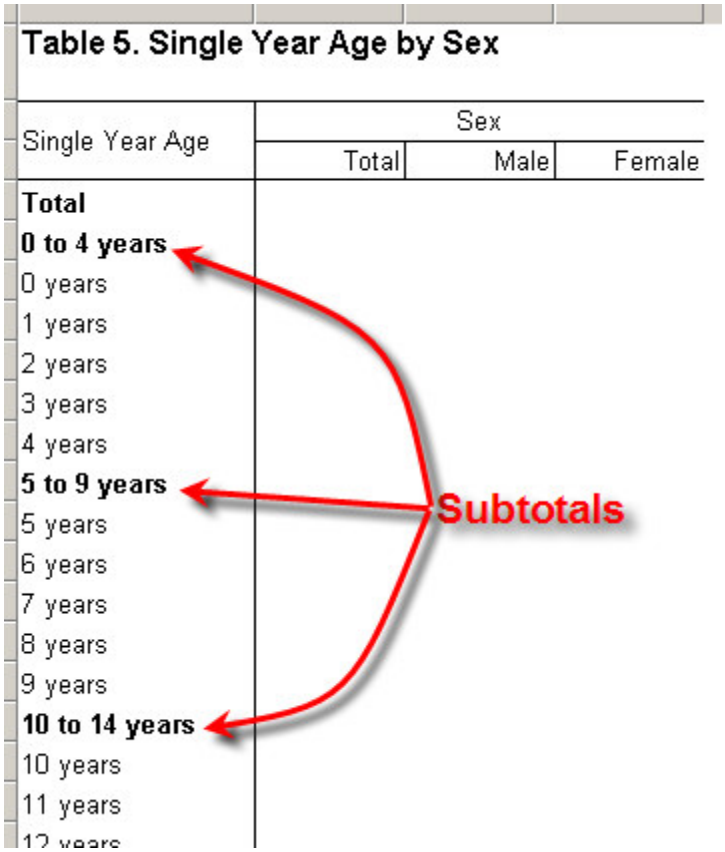
Table Tips and Tricks

Add Subtotals to a Table

You can add subtotals to a table using overlapping value sets. In your value set, add an item for each subtotal in addition to the existing items. For example, if you have a value set with single years of age and you wanted to add subtotals for each 5-year age group, you would insert additional items in the value set for each of the 5-year groups:



Then drag the modified value set onto the table. For additional clarity you can format the stubs corresponding to the subtotals to make them standout. In the table below, the font for the subtotal stubs is set to bold.



Tabulate Categories With Disjoint Values

At times it is useful to create categories in a tabulation that contain disjoint values. For example, if you have the following value set for marital status where divorced is code 2 and you wish to create a table with only two categories: "divorced" and "not divorced".

N	Value Set Label	Value Set Name	Value Label	From	To	Special
☐	Marital status	P05_MS_VS1				
☐			Married	1		
☐			Divorced	2		
☐			Separated	3		
☐			Widowed	4		
☐			Never Married	5		
☐			NA	◻		NotAppl

Table 1. Divorced by Urban/Rural

Divorced	Urban/Rural		
	Total	Urban	Rural
Total			
Divorced			
Not divorced			
NA			

To create such a table, we need a value set that contains only the two categories divorced and not divorced. Creating the category for "divorced" is simple, it includes only the value 2. The "not divorced" category, however, needs to contain the value 1 (Married) plus the values 3, 4 and 5 (Separated, Widowed, Never Married). These values do not make up a single range. In order to create the category you must create two entries in the value set, one that contains the value 1 and the second that contains the values 3 through 5. The first entry must contain the label for the category and the label for the second entry must be a single space character. CSPro recognizes the single space as a continuation of the previous category rather than a separate category. In this case it will merge the value range in the second entry with that of the first entry, creating a single category that includes the value 1 and the values 3 through 5.

N	Value Set Label	Value Set Name	Value Label	From	To	Special
☐	Divorced	P05_MS_VS2				
☐			Divorced	2		
☐			Not divorced	1		
				3	5	
☐			NA	◻		NotAppl

No notes box

Space character as label

It is important that you use a space rather than no text at all since CSPro will treat an empty label all as a new category in the value set. You can tell when CSPro has combined the entries by the absence of the notes box in the second entry.

You can combine any number of entries in the value set into a single category.

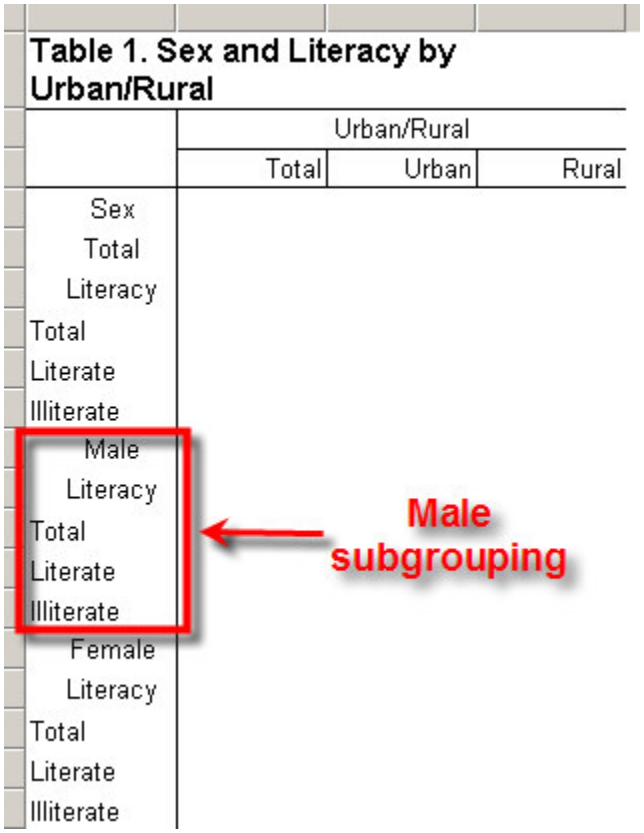
See also: Implications of Data Dictionary Value Sets

Format/Hide Rows and Columns in Subgroupings

When you have a table that uses subgroupings (one variable that was dropped onto another), there are often problems when trying to hide rows or columns in those subgroupings. When you hide or format a row or column in a subgrouping, the corresponding rows/columns in the other sub-groupings are also hidden. This makes it difficult to hide or format a row or column in one subgrouping but not the others. In addition, it is not possible to hide an entire subgrouping by hiding the associated caption or spanner. Fortunately, using value sets and multiple subtables, there are workarounds for these restrictions.

Hiding Subgroupings

There are two ways to hide an entire subgrouping. One way is to modify the value set used and the other is to use multiple subtables rather than subgroupings. Take the example of a table with Sex and Literacy as subgroupings on the row where you want to only show the sub-groupings for Total and Female. In other words, you would like to hide the subgrouping for Male.



If you try to hide the rows for the Male subgrouping using hide in the Format (Stub) dialog, it will not work. Hiding the Literate and Illiterate rows under Male will also hide those rows under Total and Female. Hide is not available in the Format (Caption) dialog if you right-click on the Male caption.

N	Value Set Label	Value Set Name	Value Label	From	To	Special
☐	Sex	P03_SEX_VS2				
☐			Total	1	2	
☐			Female	2		

One way to work around this is to modify the value set for Sex to remove the Male category. Note that removing the Male category will also affect the Total (since males will no longer be counted). To fix this, add your own category to the Sex value set called Total, which includes the values for Male and Female. Finally hide the system generated total in the table (see Hide or Change the Position of the Total). The result is the following table:

Table 1. Sex and Literacy by Urban/Rural

	Urban/Rural		
	Total	Urban	Rural
Sex			
Total			
Literacy			
Total	16,804	9,968	6,836
Literate	13,908	8,840	5,068
Illiterate	2,896	1,128	1,768
Female			
Literacy			
Total	8,751	5,275	3,476
Literate	7,063	4,607	2,456
Illiterate	1,688	668	1,020

A second approach is to use multiple subtables rather subgrouping. In our example you would drop the Literacy variable twice onto the rows to make two separate subtables. This would create the following table:

Table 1. Sex and Literacy by Urban/Rural

	Urban/Rural		
	Total	Urban	Rural
Literacy	Subtable 1		
Total			
Literate			
Illiterate			
Literacy	Subtable 2		
Total			
Literate			
Illiterate			

Next, modify the first subtable so that it becomes the subgrouping for Total and modify the second subtable so that it becomes the subgrouping for Male. To do this, first edit the captions for the two subtables to be "Total" and "Female" (see Customize Table Text). Next modify the universe of the second subtable to include only females (see Restrict a Universe). The first subtable will give the counts for all persons while the second subtable will give the counts for only females. This will give the same results as using the subgroupings:

Table 1. Sex and Literacy by Urban/Rural

	Urban/Rural		
	Total	Urban	Rural
Total			
Total	16,804	9,968	6,836
Literate	13,908	8,840	5,068
Illiterate	2,896	1,128	1,768
Females			
Total	8,751	5,275	3,476
Literate	7,063	4,607	2,456
Illiterate	1,688	668	1,020

Formatting and Hiding Rows/Columns in a Subgrouping

To hide or format a row or column in one subgrouping without effecting the other subgroupings you must use multiple subtables rather than subgroupings. This is because changing a row or column in one subgrouping will automatically cause the same change to the corresponding row in the other subgroupings. This is true for all format changes (font, color, alignment,…) as well as hiding rows/columns and modifying stub/column head text.

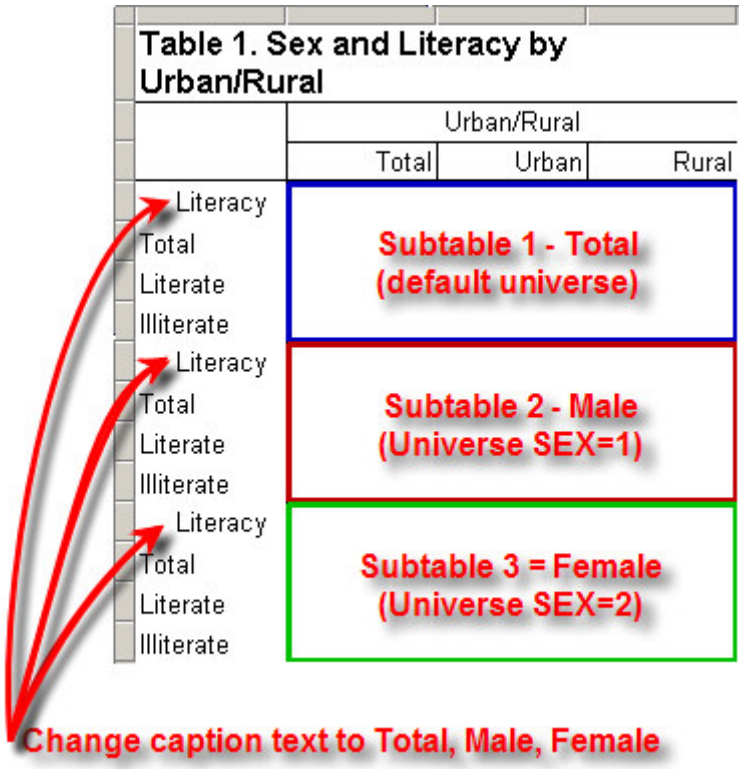
Take the earlier example of the Sex and Literacy table and try to make the text in the literacy row under the Total subgrouping bold without making the other literacy rows bold. If you simply set the format for this row to bold, the literacy rows in the Male and Female subgroupings will also become bold.

Table 1. Sex and Literacy by Urban/Rural

	Urban/Rural		
	Total	Urban	Rural
Sex			
Total			
Literacy			
Total	16,804	9,968	6,836
Literate	13,908	8,840	5,068
Illiterate	2,896	1,128	1,768
Male			
Literacy			
Total	8,053	4,693	3,360
Literate	6,845	4,233	2,612
Illiterate	1,208	460	748
Female			
Literacy			
Total	8,751	5,275	3,476
Literate	7,063	4,607	2,456
Illiterate	1,688	668	1,020

Format only this row

Instead of using subgroupings, drag the variable Literacy onto the rows three times to create separate subtables for the Total, Male and Female subgroupings.



Then edit the universe for the second subtable to only include males (see Restrict a Universe). In this case set the universe to SEX=1. Next edit the universe for the third subtable to include only females (SEX=2). Note that you do not need to edit the universe for the first subtable since you want to include both males and females (the entire population) which is the default universe. Next, edit the captions of the subtables to be "Total", "Male" and "Female" (see Customize Table Text). Finally right-click on the Literacy stub in the first (Total) subtable and set the font to bold. This results in the following table:

Table 1. Sex and Literacy by Urban/Rural

	Urban/Rural		
	Total	Urban	Rural
Total			
Total	16,804	9,968	6,836
Literate	**13,908**	**8,840**	**5,068**
Illiterate	2,896	1,128	1,768
Male			
Total	8,053	4,693	3,360
Literate	6,845	4,233	2,612
Illiterate	1,208	460	748
Female			
Total	8,751	5,275	3,476
Literate	7,063	4,607	2,456
Illiterate	1,688	668	1,020

The same procedure works for hiding or modifying the stub text of an item in a subgrouping, only the last step changes. Rather than setting the font to bold, you would hide the row or modify the stub/column head text.

See also: Hide or Show a Row or Column, Hide or Change the Position of the Total, Formats for a Part of a Table, Implications of Data Dictionary Value Sets, Using Subtables

Recodes in Tables Using Value Sets and Subtables

Often the variables in your dictionary do not map exactly to the categories desired in your tables and you need to "recode" one or more variables to map it to the groupings you want in the rows and columns of your table. Recodes involving a single variable can be accomplished by creating additional value sets while recodes of multiple variables can be accomplished using multiple subtables.

Single variable recodes

To recode a single variable, simply create an additional value set for that variable in your dictionary, which regroups the values as desired. Then drag this new value set onto the table.

For example if you have the following value set for marital status but would like to count only two categories, Married and Unmarried, in your table.

N	Value Set Label	Value Set Name	Value Label	From	To	Special
☐	Marital status	P05_MS_VS1				
☐			Married	1		
☐			Divorced	2		
☐			Separated	3		
☐			Widowed	4		
☐			Never Married	5		

Create a new value set that regroups the original values for marital status into these new categories.

N	Value Set Label	Value Set Name	Value Label	From	To	Special
☐	Marital Status	P05_MS_VS2				
☐			Married	1		
☐			Unmarried	2	5	

Now drag the new value set onto your table.

Table 1. Marital Status by Urban/Rural

Marital Status	Urban/Rural		
	Total	Urban	Rural
Total	24,271	13,920	10,351
Married	6,985	3,854	3,131
Unmarried	17,286	10,066	7,220

Value sets may have disjoint values, one category which contains values from different ranges, for example, one category for values 23, 45, and 67-71 (see Tabulate categories with disjoint values).

Multiple variable recodes

To do a recode using multiple variables, create multiple subtables by dragging the variables onto the table multiple times and set universes on these subtables. For example, if you want to tabulate houses that have "complete plumbing", meaning that they have piped water, bathing facilities and a toilet inside the housing unit. This involves the following three variables:

N	Value Set Label	Value Set Name	Value Label	From
☐	Source of water	H10_WATER_VS1		
☐			Inside piped	1
☐			Outside piped	2
☐			Bottled or canned	3
☐			Closed well or close	4
☐			Open well or spring	5
☐			River/Stream/Other	6

N	Value Set Label	Value Set Name	Value Label	From
☐	Type of bathing fac	H09_BATH_VS1		
☐			Exclusive	1
☐			Shared	2
☐			Hand basin	3
☐			Portable tub or basir	4
☐			Other	5
☐			None	6

N	Value Set Label	Value Set Name	Value Label	From
☐	Type of toilet faciliti	H08_TOILET_VS1		
☐			Private toilet	1
☐			Shared toilet	2
☐			Outhouse	3
☐			Pit	4
☐			Other	5
☐			None	6

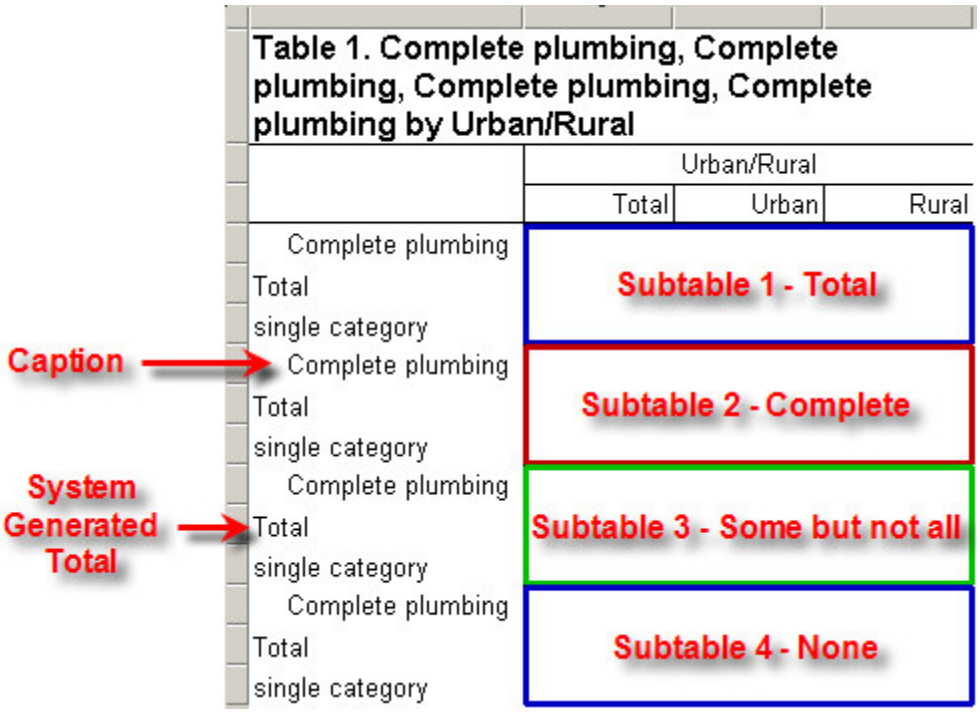
The "recoded" variable, "complete plumbing", has three categories:

- Complete – piped water inside the unit (source of water = 1), and a private toilet inside the unit (type of toilet = 1) and bathing facilities inside the unit (type of bathing = 1).
- Some but not all – any one of the three variables (piped water, flush toilet, bathing facilities) inside the unit but not all three.
- None – all other cases.

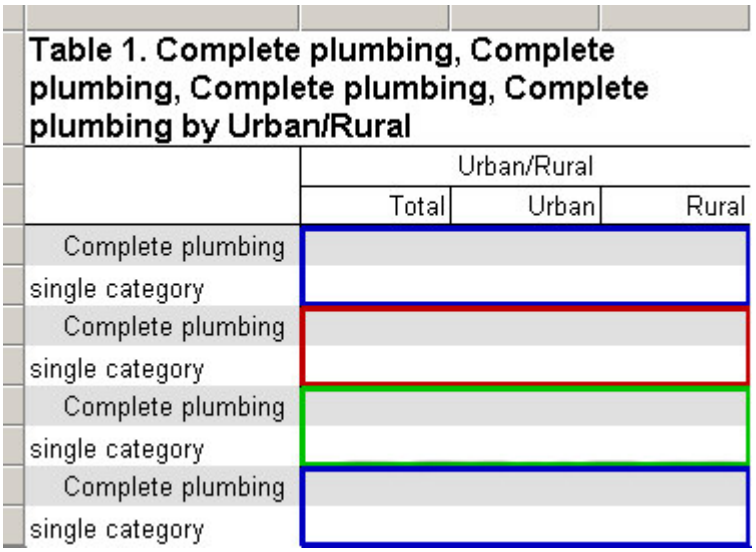
To create the table for complete plumbing, use 3 subtables, one for each of the 3 possible values of the recoded variable (complete, some but not all, none). Each of these subtables should have only one row. The easiest way to create a subtable with one row is to drag a variable whose value set has only one category onto the table. If no such value set exists, you can create one. In this case, create the value set on the variable "source of water". In fact, you could use any variable on the same record as the variables we are using in the recode (the housing record in this case). Note that the single category in this value set must include all valid values for the variable so that all housing units will be counted when tabulating the variable.

N	Value Set Label	Value Set Name	Value Label	From	To
☐	Source of water	H10_WATER_VS1			
☐			Inside piped	1	
☐			Outside piped	2	
☐			Bottled or canned	3	
☐			Closed well or close	4	
☐			Open well or spring	5	
☐			River/Stream/Other	6	
☐	Complete plumbing	H10_WATER_VS2			
☐			single category	1	9

Drag this new value set onto the table 4 times, once for each of three possible values of complete plumbing and once for the total.



Since we only need one row for each subtable, hide the system generated total in each of the four subtables (see Hide or Change the Position of the Total). Also hide the captions for each of the subtables (see Formats for a Part of a Table).



Next customize the text in the stubs to match the categories for the recode (Total, Complete, Some but not all, None). Also edit the title of the table and put the name of the variable in the Stub Head. See Customize Table Text for how to edit text in the table.

Table 1. Complete plumbing by Urban/Rural

Complete Plumbing	Urban/Rural		
	Total	Urban	Rural
Complete plumbing Total			
Complete plumbing Complete	WATER = 1 and TOILET = 1 and BATH = 1		
Complete plumbing Some but not all	(WATER = 1 or TOILET = 1 or BATH = 1) and not (WATER = 1 and TOILET = 1 and BATH = 1)		
Complete plumbing None	WATER <> 1 and TOILET <> 1 and BATH <> 1		

Now set the universes for the subtables to correspond to the appropriate values of the complete plumbing recode. The first subtable will be the total so it should use the default universe, which includes all cases. The second subtable represents "complete plumbing" and must include only those cases where piped water, flush toilet AND bathing facilities all equal 1. The third subtable represents "some but not all" and must include the cases where one or two of the variables are equal to 1 but not all of them. The fourth subtable represents "none" and must include all cases where none of the variables are equal to 1. For more information on setting the universe on a subtable see Restrict a Universe. This produces the following table:

Table 1. Complete plumbing by Urban/Rural

Complete Plumbing	Urban/Rural		
	Total	Urban	Rural
Total	4,420	2,469	1,951
Complete	106	106	-
Some but not all	361	354	7
None	3,953	2,009	1,944

Note that often it is preferable to create recoded variables in a batch edit program rather than during tabulation. This recode could have been accomplished by adding a new variable to the dictionary for complete plumbing and writing a batch edit program to set the value of this variable for each case. Then the new complete plumbing variable could be dropped on the table directly rather than creating multiple subtables with universes. This makes the creation of the table much simpler although it involves creating a batch edit application and writing a small amount of logic. Using batch edit is best when the recoded variable will be used in multiple tables.

Another alternative would be to create the recoded variable in the working storage dictionary and use tablogic to set its value to each case. See Table Logic (tablogic) for more information.

Use Expressions in Universe and Value Tallied

You can enter any valid CSPro expression as a universe, weight or value tallied. This includes not only variables, constants, logical operators (and, or, not), and mathematical operators (+,-,*,/,…) but also CSPro functions (count, sum, max, special, ...). Making full use of these expressions allows you to create complex tables very easily. Here are some examples:

Universe

Tabulate only households where at least one child was born last year with the following universe:

```
COUNT(CHILDREN_BORN_LAST_YEAR) > 0
```

This is assuming that CHILDREN_BORN_LAST_YEAR is a variable on the person record.

Count only households with "married couple families", i.e. a household with both a head of household and a spouse present, with the following universe:

```
COUNT(PERSON where RELATIONSHIP = 1) > 0 and
COUNT(PERSON where RELATIONSHIP = 2) > 0
```

Restrict a table to only households with total household income of greater than $20,000 by using the following universe:

```
SUM(PERSONAL_INCOME) > 20000
```

This assumes that PERSONAL_INCOME is a variable on the person record. The sum of the incomes of each person in the household is the total income for the entire household.

Restrict a table to only households where the head of household is female with the following universe:

```
SEX(1) = 2
```

This assumes that the head of the household is always the first person record (index 1). This will only be true if your data entry program and/or edit program ensure this. If this is not true, then you would need a more complicated expression such as:

```
COUNT(PERSON where RELATIONSHIP = 1 and SEX = 2) = 1
```

This will be true only if there is a person in the household who is the head of household and is female. This assumes that there is only one head of household in the household, which should be the case for properly edited data.

Value Tallied

Often fertility information is captured separately for male and female children and you wish to tabulate it for both sexes. For example, you have variables for MALE_CHILDREN_BORN and FEMALE_CHILDREN_BORN but no variable for total children born and you want to count the total number of children born. You can use the sum of the two variables in the value tallied:

```
MALE_CHILDREN_BORN + FEMALE_CHILDREN_BORN
```

Note that if one or more of the variables is a special value, the counts will not be correct. This is because the sum of a special value and a number is a special value.

See also: Restrict a Universe, Tabulate Values Instead of Frequencies, Tally Attributes for a Table

CSPro Statements and Functions

Alphabetical List

abs	Returns the absolute value of a numeric expression.
accept	Returns the number of a choice from a list made by the data entry operator.
advance	Moves forward field by field to a specified field during data entry.
alias	Creates an aliased name for a dictionary item, typically to shorten or standardize names.
alpha	Declares alphanumeric variables used in the application.
(assignment)	Sets a variable equal to the value of an expression.
array	Declares a 1- to 3-dimension array of numeric values.

average	Returns the average of an item that occurs multiple times.
box	Old name for recode statement.
break	Exits a do, while, or for loop early and continues execution with the first statement after the enddo.
changekeyboard	Modifies the keyboard input associated with certain fields.
clear	Initializes the memory values of data items defined in external files to zero or blank.
close	Closes a previously opened external file.
cmcode	Returns the number of months since the year 1900 given a month and year.
compare	Returns alphabetical order (i.e., collating sequence) of the two strings.
concat	Joins two or more strings into one string.
count	Returns the number of occurrences for a repeating form or roster.
countnonspecial	Returns the number of non-special values in a group of data items.
curocc	Returns the current occurrence number for a repeating form, roster, or record.
dateadd	Calculates a new date from a starting date and a period of elapsed time.
datediff	Calculates the difference between two dates.
datevalid	Determines whether a date in the format YYYYMMDD is valid.
delcase	Marks a case for deletion in an external file based on a key.
delete	The delete function removes a record or item occurrence from the current case.
demode	Returns the current data entry mode.
display	This function, to display a message, has been superceded by errmsg.
do	Executes one or more statements repeatedly while a logical condition remains true or until a logical condition is no longer true.
edit	Converts a number to a string.
editnote	Displays data entry field note box for adding or changing.
endcase	Ends batch editing for the current case (but outputs the case to file).
endgroup	Ends data entry for the current record or group/roster.
endlevel	Ends data entry for the current level.
enter	Enters data from a secondary form file.
errmsg	Displays or writes a message.
execpff	Starts another CSPro application.
execsystem	Starts another application or process.
exit	Ends a procedure before normal processing is expected to end.
exp	Returns the value of **e** raised to a given power.
export	Writes a record to an export file.
file	Declares one or more files used in the application.
fileconcat	Concatenates a list of files or a set of files described by a wildcard specification.
filecopy	Copies a file to another file.
filecreate	Creates a new file with the given file name.
filedelete	Deletes an already existing file.
fileempty	Determines whether a file exists but is empty.
fileexist	Determines whether a file exists.
filename	Returns the data file name currently associated with a data dictionary.

fileread	Reads a text line from a file into an item or variable.
filerename	Changes the name of a file.
filesize	Returns the size of a file in bytes.
filewrite	Writes a line of text to a file.
find	Determines the existence of a case in an external file that matches a condition.
for	Loops through multiple records or items.
function	Declares a user-defined function.
getbuffer	Returns a string containing the contents of a data item.
getcapturetype	Returns the capture type associated with a field.
getdeck	Retrieves a value from a DeckArray used for editing data.
getlabel	Returns the label of a dictionary symbol or text associated to symbol's value.
getlanguage	Returns the current language being used while in CAPI data entry mode.
getnote	Returns then data entry field note.
getocclabel	Returns an occurrence label associated with a repeating dictionary item or a roster.
getoperatorid	Returns the text entered in the operator id.
getorientation	Returns the display orientation.
getos	Returns information about the operating system, including whether it is running Android or Windows.
getrecord	Returns the name of the record containing a particular item.
getsymbol	Returns the name of the current procedure being executed.
getusername	Returns the name of the user logged into Windows or the Android device account.
getvalue	Returns the value of a numeric variable whose identity is only known at runtime.
getvaluealpha	Returns the value of a alpha variable whose identity is only known at runtime.
getvaluenumeric	Returns the value of a numeric variable whose identity is only known at runtime.
gps	Controls and returns values from a GPS receiver.
has	Determines whether any of a group of repeating items is within a range of values.
high	Returns the maximum value in a group of numeric expressions
highlighted	Returns whether field is on path or reached during data entry.
if	Executes statements conditionally.
impute	Assigns a value to a data item and logs the frequency of assignments.
in	Determines whether a variable is within a range of values.
inc	Increments a numeric item.
insert	The insert function creates a record or item occurrence in the current case.
int	Returns the integer portion of a numeric expression.
invalueset	Determines whether a data item's value is within the items value set.
ispartial	Determines whether the current case was opened from a partial case or not.
key	Returns the key of the case at the current position in an external file.
killfocus	Declares that following statements are executed object stops being active.
length	Returns the length of a dictionary item or a string.
loadcase	Reads a case from an external file into memory based on a key.

locate	Finds but does not load a case in an external file that matches a condition.
log	Returns the base-10 logarithm of a numeric expression.
low	Returns the minimum value in a group of numeric expressions
maketext	Returns a formatted string with inserted values.
max	Returns the maximum value of an item that occurs multiple times.
min	Returns the minimum value of an item that occurs multiple times.
move	Moves backward or forward to a specified field during data entry.
next	Ends a do, while, or for loop early and continues execution with the next iteration of the loop.
noccurs	Returns the number of occurrences for a repeating form or roster.
noinput	Prevents input for the current field during data entry.
numeric	Declares numeric variables used in the application.
onchangelanguage	Provides control over actions to occur after a user has changed the CAPI language.
onchar	Allows users to trap characters in order to perform special actions or to change the action of the character.
onfocus	Declares that following statements are executed object becomes active.
onkey	Allows users to trap keystrokes in order to perform special actions or to change the action of the key.
onstop	Provides control over stopping or exiting data entry.
open	Opens and keeps open an external file.
pathname	Returns the name of various paths (folders)
pos	Returns the position of a string within another string.
preproc	Declares that following statements are executed at the beginning of a block.
proc	Declares the beginning of a new procedure.
postproc	Declares that following statements are executed at the end of a block.
putdeck	Puts a value in a DeckArray used for editing data.
putnote	Puts the contents of string to a data entry field note.
random	Returns a pseudo-random integer in a given range.
randomin	Returns a pseudo-random integer in a non-continuous range, or using a value set.
randomizevs	Randomizes the order of a value set; useful for data entry applications using extended controls.
recode	Assigns a value to a variable based on the value of one or more other variables.
reenter	Forces the data entry operator to re-enter a previous entered field.
relation	Defines a relation between multiple records or items.
retrieve	Reads a case from the current position of an external file into memory.
savepartial	Saves the current case as a partially added, modified, or verified case.
seed	Initializes the random number generator to a particular starting place.
seek	Searches a multiply occurring item for an item that meets a condition.
seekmax	Searches a multiply occurring item for an item with the highest value that meets a condition.
seekmin	Searches a multiply occurring item for an item with the lowest value that meets

	a condition.
selcase	Allows a data entry operator to select and load a case from an external file.
setcapturepos	Allows the manual declaration of the window coordinates of extended controls.
setcapturetype	Sets the capture type associated with a field.
set	Switches the values of various system parameters.
setfile	Assigns a new physical file to a dictionary or declared file.
setfont	Changes the default display font in data entry applications.
setlanguage	Dynamically modifies the current language being used while in CAPI data entry mode.
setorientation	Modifies the display orientation, allowing dynamic rotation of the display.
setoutput	Programmatically changes the output file where cases are saved in a batch application.
setvalue	Assigns a value to a variable whose identity is only known at runtime.
setvalueset	Dynamically modifies value set of an item.
setvaluesets	Dynamically modifies the value sets for all items in the dictionary.
show	Displays the occurrences of one or more items from a record or multiply occurring item.
showarray	Displays the values of one or more elements from an array.
skip	Jumps forward to a specified field during data entry.
skip case	Ends processing of the current case in a batch edit run.
soccurs	Returns the number of occurrences of a record.
sort	Sorts occurrences of records or items based on the value of an item.
special	Determines whether a variable's value is MISSING, NOTAPPL, or DEFAULT.
sqrt	Returns the square root of a numeric expression.
stop	Ends a data entry session or batch edit run.
string	Declares variable-length alphanumeric variables used in the application.
strip	Removes leading and trailing blanks from a string.
sum	Returns the sum of an item that occurs multiple times.
swap	Switches the order of two occurrences within a group.
sysdate	Returns the current system date as an integer.
sysparm	Returns a parameter from your data entry or batch edit pff file.
systime	Returns the current system time as an integer.
tolower	Changes all uppercase letters in a string to lowercase.
tonumber	Converts a string to a number.
totocc	Returns the total occurrences for a repeating form, roster, or record.
toupper	Changes all lowercase letters in a string to uppercase.
trace	Controls a window or file that can contain debugging information.
userbar	Controls the optional display bar in a data entry application.
universe	Specifies a condition that must be true for execution of a procedure or function to continue.
visualvalue	Returns the value of a data item prior to its input.
while	Executes one or more statements repeatedly while a logical condition remains true.
write	Write to a text file.
writecase	Writes a case from memory to an external file.

Statement Format Symbols

The formats of statements and function use the following symbols:

`reserved` Name of the statement or function, or a another reserved word.

`keyword` Keyword used within this statement or function.

`element` Type of object used as a part of a statement or parameter of a function.

`A | B` One or the other of the symbols A or B may be used.

`[ A ]` The symbol A is optional.

`A, B, ...` More symbols of the same type.

List of Reserved Words

CSPro does not allow certain names to be used as dictionary unique names, or as variables in the programming logic, as they are part of CSPro's procedural language. But don't worry about accidental usage—when you attempt to name something in the CSPro system with a reserved word, the system will notify you that you have used a reserved word.

In addition to the list of reserved words below, there are a few reserved words used internally by CSPro. But again, CSPro will alert you when you try to create a dictionary item or variable with this name. Further, if you are writing logic, reserved words are shown in blue—therefore, if you attempt to create a variable using one of these reserved words, you will know this name is not available when it turns blue.

abs	endlevel	key	savepartial
accept	enter	killfocus	seed
add	errmsg	length	seek
advance	exec	level	seekmax
all	execpff	linked	seekmin
alias	execsystem	loadcase	selcase
alpha	exit	locate	select
and	exp	log	set
array	export	low	setcapturepos
average	file	maketext	setcapturetype
box	fileconcat	max	setfile
break	filecopy	maxocc	setfont
by	filecreate	min	setlanguage
case	fileempty	missing	setlb
changekeyboard	fileexist	modify	setorientation
clear	filedelete	move	setoutput
close	filename	next	setub
cmcode	fileread	noccurs	setvalue
compare	filerename	noinput	setvalueset
concat	filesize	not	setvaluesets
count	filewrite	notappl	show
countnonspecial	find	numeric	showarray
crosstab	float	onchangelanguage	skip
curocc	for	onchar	soccurs
dateadd	function	onfocus	sort
datediff	getbuffer	onkey	special

datevalid	getcapturetype	onstop	specific
default	getdeck	open	sqrt
delcase	getlabel	or	stat
delete	getlanguage	pathname	stop
demenu	getnote	pos	string
demode	getocclabel	poschar	strip
denom	getoperatorid	postproc	sum
disjoint	getorientation	preproc	summary
display	getos	proc	swap
do	getrecord	putdeck	sysdate
edit	getsymbol	putnote	sysparm
editnote	getusername	random	systime
else	getvalue	randomin	then
elseif	getvaluealpha	randomizevs	title
end	getvaluenumeric	recode	to
endbox	gps	reenter	tolower
enddo	has	relation	tonumber
endif	high	retrieve	totocc
endcase	highlighted		toupper
endgroup	if		trace
	impute		universe
	in		until
	inc		update
	insert		userbar
	int		verify
	invalueset		visualvalue
	ioerror		vset
	ispartial		where
	item		while
			write
			writecase

In general, reserved words have been linked to the function of the same name, if one exists. If no link exists for a word, it is either because [1] there was more than one association for the word, or [2] the word is for internal usage only.

Declaration Statements

Set Statement

Format:

```
set explicit | implicit;
```

Description:

The **set** statement overrides the compiler's default setting of how numeric variable are declared. If used, it must be code as the first line coded in the **PROC GLOBAL** section.

By default, CSPro sets the **set explicit** option to ON -- meaning that all user-declared variables must be declared in the PROC GLOBAL section via the numeric or alpha statement. If they are not, a compiler error message is generated when an undeclared variable is used. It is good programming practice to use **set explicit** either as the computer default or to code it in Proc Global. If variables are explicitly defined, the compiler can detect misspellings of variable names, which can otherwise hard to find.

If you do not wish to declare your variables, you can change the behavior to implicit by going to the **Options** menu, and unchecking **Set Explicit**. This will allow you to create variables whenever they are first used or referenced. However, this is not recommended by the CSPro team. It is much better programming practice to use the set explicit option and harness the compiler's error detection capabilities to prevent execution failures due to misspelled variable names.

The following explains the impact of programmatically setting this switch, as opposed to using the system setting:

System Setting	Program Setting Result
✓ Set Explicit set explicit;	No effect, as program matches system setting
✓ Set Explicit set implicit;	Program overrides system setting, variables do **not** need to be declared
Set Explicit set explicit;	Program overrides system setting of implicit—variables must be declared
Set Explicit set implicit;	No effect, as program matches system setting

CSPro defaults to **explicit** mode, but CSBatch defaults to **implicit** mode. Therefore, if you developed your program in CSPro leaving the set explicit option checked, and there were no errors, you can rest assured that your application will run correctly under CSBatch, even though the mode will be implicit.

Example 1:

```
PROC GLOBAL
  set explicit;
  numeric x,y,z;
```

Example 2:

```
PROC GLOBAL
  set implicit;
```

See also: Numeric, Alpha

File Statement

Format:

```
file file-1[, file-2[..., file-N]]
```

Description:

The **file** statement defines files, not associated with dictionaries, that are used by the application and whose physical names are not given until run time. It must be coded in the **PROC GLOBAL** section of the application.

The *file-N* is a CSPro name that is used in functions and statements that control the file.

The physical folder and file name of each file is requested when the application is run. The associated folder and file name can be changed during the run by using the **setfile** function

Example:

```
PROC GLOBAL
  File FILE_PERSON, FILE_HOUSEHOLD;
```

See also: Setfile Function

Numeric Statement

Format:
```
numeric var1[, var2[..., var-n]];
```

Description:
The **numeric** statement declares temporary numeric variables used only in this application. They will not be save a data file defined by a dictionary. Temporary numeric variables must be declared with the **numeric** statement if the set explicit menu option is checked (from the menu Options/Set Explicit) or if a set explicit statement is included in the PROC GLOBAL section of your program. A numeric variable is an integer or decimal number significant to 15 digits.

Example:
```
PROC GLOBAL
  numeric x, NumOfKids;

PROC CHILDREN
  x = 0;
  NumOfKids = NumOfKids + 1;
```

See also: Set Statement, Array Statement, Alpha Statement, String Statement

Alpha Statement

Feature Upgrade: *As of CSPro 6.0, you can now use variable-length strings. You are encouraged to use strings instead of fixed-width alpha variables whenever possible.*

Format:
```
alpha [(len)] var-1[, var-2[..., var-n]];
```

Description:
The **alpha** statement is used to define alphanumeric variables used in the application. The **len** is the number of characters in the variable. The **len** applies to all variables which follow in the same statement. If no **len** is given, 16 is assumed. The maximum string length that can be declared is 8,192. If you attempt to assign to an alpha variable a string that is longer than the variable's size, the string will be truncated from the right. Conversely, it you assign a string that is shorter than the variable's size, the trailing character positions will be blank-filled.

The following two examples, using the declaration of x below, should clarify this point:

Example 1:
```
PROC GLOBAL
  alpha(10) x,y;

PROC A1
  x = "hi mom";

     x will now equal "hi mom    "
                      1234567890

  x = "good night, mom";

       x will now equal "good night"
                        1234567890
```

Example 2:

```
PROC GLOBAL
  alpha (3) reply;
  alpha flag;

PROC Q5
if q5 = 1 then
  reply = "Yes";
  flag = "Y";
endif;
```

If the user attempts to assign the string "Not reported" to the variable "reply," CSPro would place the letters "Not" in the variable and drop the remaining characters of the string.

See also: String Statement, Array Statement, Numeric Statement,

String Statement

Format:

```
string var-1[, var-2[..., var-n]];
```

Description:

The **string** statement is used to define alphanumeric variables temporarily used in the application. It is similar to the **alpha** statement but, unlike variables created with that statement, variables created as strings have no fixed length. Strings can be used anywhere an alpha variable can be used. (Strings were only introduced starting in CSPro 6.0 so much of the documentation refers to alpha variables.) Strings start out with a length of 0 and then automatically resize to match whatever is assigned to it. This feature make strings the preferred text variable to use, unless the length of the string is needed for formatting and calculation purposes.

Examples:

```
PROC GLOBAL

alpha (10) myAlpha;
string myString;

PROC EXAMPLE

    // myString starts out empty, with length 0

    myString = "This is a test." // myString's length is now 15

    myAlpha = "Hello";
    myString = myAlpha; // myString's length is now 10: "Hello     "

    myString = "Hello"; // myString's length is now 5: "Hello"
```

See also: Alpha Statement, Array Statement, Numeric Statement

Array Statement

Format:

```
array [alpha[(len)]] array-name(dim-1[,dim-2[,dim-3]]) [save];
```

Description:

CSPro supports numeric and alphanumeric arrays of up to three dimensions. You must declare arrays in the global procedure of the application, using the array statement.

Only one array at a time can be declared with the array statement. The array name must be unique and contain only letters, numbers, or the underscore ('_') character. It must begin with a letter.

The keyword alpha indicates that the array is alphanumeric. If the keyword alpha is not used, the array is numeric. The len is optional. If it is coded it give the number of characters in each occurrence of the alphanumeric array variable. If it is not coded, the length of each occurrence is 16.

The values dim-1, dim-2, and dim-3 are numbers giving the size of each dimension.

The initial values of a numeric array are zeroes and of alphanumeric arrays are blanks.

The optional keyword **save** indicates that the array values should be saved to a file and loaded from that file when the program is run again. This allows you to maintain the values of the arrays across multiple runs of the same program. This is particularly useful for setting the initial values of hot decks. The program is run twice, the first run fills the hotdeck and saves the hotdeck array to the file. The second run loads the values saved from the first run and uses them as the initial values for the hotdeck for imputation. See Initialize Hot Decks Using Saved Arrays for more information.

When one or more arrays in the program are marked with save, the first time the application is run, a saved array file is created and the values of the arrays at the end of program execution are written to the file. On consecutive runs of the program, the initial values of the arrays are read in from the file. If the file is not present (such as for the first run of the program) or does not contain values for a particular array (such as when the name of an array changes or a new array is added), the initial values of a numeric array are set to the special value default and the initial values of an alphanumeric array are set to blank.

All arrays marked with save in the application are written to the same file. This file has the same name as the application but with a .sva file extension appended to it (for example, myapplication.bch.sva).

Saved arrays are not supported in data entry applications.

Example 1: (numeric array)

```
PROC GLOBAL
  array age_hd (2,8); { sex by relationship }
  array MyArray (5,10);
  numeric X, Y, male, female;

PROC MY_PROGRAM
preproc
  male = 1;
  female = 2;

  age_hd (male,1) = 20; { male head }
  age_hd (male,2) = 24; { male spouse }
  age_hd (male,3) = 8; { male child }
  { continue with male initializations }

  age_hd (female,1) = 26; { female head }
  age_hd (female,2) = 32; { female spouse }
  age_hd (female,3) = 5; { female child }
  { continue with female initializations }
```

```
MyArray (1,3) = 0;
X = 2;
Y = 1;
MyArray (X,Y) = 0;
Z = MyArray (X,Y);
```

Example 2: (alphanumeric array)

```
PROC GLOBAL
  array alpha(10) crop (20); {20 crop names, each up to 10 chars long}

PROC MY_PROGRAM
preproc
  crop(1)= "maize";
  crop(2)= "wheat";
  crop(3)= "rice";
  crop(4)= "potatoes";
```

If you attempt to assign to an element of an alpha array a string that is longer than the element's size, the additional portion will be truncated. For example, if the following were written:

```
crop(1)= "sweet potatoes";
```

the variable would be assigned the string "sweet pota". There is no "spillover" effect (such as exists in some programming languages) that would corrupt subsequent array cells.

If the string length of (10) had not been given above, the string would have defaulted to a length of 16.

Example 3: (saved array)

```
array age_hd(2,8) save; {sex by relationship, init from file }

PROC AGE
```

`if AGE = notappl then`
`impute(AGE, age_hd(SEX, RELATIONSHIP));`
 	
`else`
`age_hd(SEX, RELATIONSHIP)= AGE;`
`endif;`
	`{if the value for age is not valid}`
	`{assign the value from the hot deck based on sex and relationship}`
	
	`{update the value of the hot deck}`

The array age_hd is not initialized in the program logic. Instead, the program is run twice. During the first run, the values of the array will be filled in by the assignment in the else clause above and the results will be stored in the saved array file. When the program is run a second time, these values are loaded in automatically as the initial values for the array.

See also: Arrays, Numeric Statement, Alpha Statement, String Statement

Relation Statement

Format:

```
relation relation-name primary to secondary-1 method
[to secondary-2 method] ... [to secondary-n method];
```

where *method* is

```
parallel | linked by arith-exp | where condition
```

Description:

The **relation** statement allows you define additional relations beyond those defined in the data dictionary.

The *relation-name* is a unique CSPro name which contains only letters, numbers, or the underscore ('_') character. It must begin with a letter.

The *primary* is the name of a multiply occurring record or item. Items defined as *secondary* are linked to the *primary* by the *method* specified.

The *secondary* is the name of a multiply occurring record or item which is linked to the *primary*.

The *method* type is specified by one of the keywords parallel, linked by, or where.

In the **parallel** method corresponding occurrences of the primary record or item and secondary record or item are linked, that is first occurrences are linked, second occurrences are linked and so on.

In the **linked by** method the value of the arithmetic expression containing values from one record item is a pointer to the occurrence in the other record or item.

In the **where** method the value of an item on the primary record is compared to the value of an item on the secondary record. If the values are equal, the records are linked.

Example 1:

```
PROC GLOBAL
  relation PERSON POP1 to POP2 parallel
      to POP3 parallel;
```

Example 2:

```
PROC GLOBAL
  relation MOTHER-CHILD CHILD to MOTHER linked by MOTHER_LINE;
```

Example 3:

```
PROC GLOBAL
  relation MOTHER-ALL PERSON to MOTHER
      where PERSON_LINE = MOTHER_LINE;
```

Function Statement

Format:

```
function function-name([p-1[,p-2[...,p-n]]])
  statements;
  function-name = numeric-exp;
end;

return-value = function-name([p-1[,p-2[...,p-n]]]);
```

[] indicates that this part is optional.

Description:

Numeric and alpha expressions, and arrays, can be passed to a user-defined function as parameters. Variables mentioned in the parameter list will by default be considered numeric. To specify an

alphanumeric parameter, you must place the keyword **alpha** before the parameter. By default, the length of an alphanumeric parameter is 16 characters. To specify a different length, place **alpha (***length***)** before the parameter name.

Be aware that the names used in the parameter list of a function may not be the same as names that are defined in any dictionary associated with the application, nor may the parameter names be the same as variables defined in a **numeric** or **alpha** declaration statement.

To specify an array parameter, place the keyword **array** (or **array alpha**) before the parameter name. By default the array is considered one-dimensional. For a two-dimensional array, place (,) after the parameter name. For a three-dimensional array, place (,,) after the parameter name. Within the function, the functions **tblrow**, **tblcol**, and **tbllay** return the number of dimensions of the passed array. As opposed to numeric or alpha expressions, wherein modifying the value does not alter the source items, modifying the value of elements of an array also changes the values of the source array.

Functions always return a value, generally a numeric value. To specify the return value, assign a numeric value to the name of the function. If no return value is assigned to the function, a DEFAULT value is returned. To return an alphanumeric value, please the keyword **alpha** before the function name. To specify a length other than 16, place **alpha (***length***)** before the function name.

Example 1:

```
PROC GLOBAL
function absvalue(VALUE)
  if VALUE < 0 then
    absvalue = (-VALUE);
  else
    absvalue = VALUE;
  endif;
end;

PROC AGE
  AGE = absvalue(AGE); // call user-defined function
```

Example 2:

```
PROC GLOBAL
function isvalidname(alpha (32) VALUE)
  LOOKUP_NAME = VALUE;
  isvalidname = loadcase(LOOKUP,LOOKUP_NAME);
end;

PROC AGE
  if isvalidname("JIM") then // call user-defined function
```

Example 3:

```
PROC GLOBAL
function calcval(VAL1, VAL2, alpha(3) OPER);
   [instructions]
   . . .
end;
```

Variables VAL1 and VAL2 are implicitly numeric. Variable OPER is alphanumeric with a length of three characters. None of the three is declared elsewhere in the program or in any of the dictionaries used.

Example 4:

```
PROC GLOBAL

array exampleArray(2,3) = 1 2 3 4 5 6;

function sumArray(array values(,))

    numeric rowIdx,colIdx,sumValue;

    do rowIdx = 1 while rowIdx <= tblrow(values)
        do colIdx = 1 while colIdx <= tblcol(values)
            inc(sumValue,values(rowIdx,colIdx));
        enddo;
    enddo;

    sumArray = sumValue;

end;

PROC EXAMPLE

    errmsg("Sum of array is %d",sumArray(exampleArray));
```

Example 5:

```
PROC GLOBAL

function string getAccount(string accountName)

    // this function, when passed an email address, returns the
    // string with the domain part of the address removed
    numeric atPos = pos("@",accountName);

    if atPos > 0 then   getAccount = accountName[1:(atPos - 1)];
    else                getAccount = accountName;
    endif;

end;

PROC EXAMPLE

    errmsg("User account is: %s",getAccount(getusername()));
```

More examples of user-defined functions can be found in the ***Date Checking*** folder in the CSPro Examples folder.

See also: User-Defined Functions

Program Control Statements

Break Statement

Format:

```
break;
```

Description:

The break statement exits a do, while, or for loop early and continues execution with the first statement after the enddo.

Example:

```
{ find the spouse }
N = 0;
for I in PERSON do
  if P02_REL = 2 then
    N = I;
    break;
  endif;
enddo;
```

See also: Do Statement, For Statement, While Statement, Next Statement, Exit Statement

Do Statement

Format:

```
do [[varying] var = expression] while/until condition [by expression]
  statements;
enddo;
```

[ ] indicates that this part is optional.

Description:

The **do** statement executes one or more statements repeatedly, in a loop, either while a logical condition is true, or until a logical condition is no longer true. The **do** and **enddo** keywords are required. You must use a **while** or **until** phrase to terminate the loop. The condition is evaluated on each repetition of the loop before any of the statements within the loop are executed.

When the **while** option is used, it means the statements within the loop [between **do** and **enddo**] are executed **while** the condition remains true. That is, if the condition is true, the statements are executed. If the condition becomes false, execution moves to the first statement following the **enddo** keyword.

When the **until** option is used, the statements within the **do** are executed **until** the condition becomes true. That is, if the condition is false the statements are executed. If the condition becomes **true**, execution moves to the first statement following the **enddo** keyword.

The **by** phase adds the indicated number or numeric expression (expression) to the variable after each repetition of the loop. If the **by** phase is present, at the end of each repetition of the loop, the expression is evaluated. The result of the expression is added to the numeric variable in the varying clause. If the **by** phase is omitted, 1 is added to the variable at the end of each repetition of the loop. For example, if you wanted to process only odd-numbered records, you could increment your loop **by** 2.

In the **varying** clause, the variable must be a numeric variable. The variable assignment is performed once, before the first repetition of the loop. The **varying** keyword has no affect on the command, and so may be omitted.

Example:

```
HEAD = 0;
do varying i = 1 until HEAD > 0 or i > totocc(PERSON)
  if RELATIONSHIP(i) = 1 then
    HEAD = i;
  endif;
enddo;
```

This same example could be rewritten using the **while** condition as follows:

```
HEAD = 0;
do varying i = 1 while HEAD = 0 and i <= totocc(PERSON)
  if RELATIONSHIP(i) = 1 then
    HEAD = i;
  endif;
enddo;
```

It is purely a matter of preference as to which method should be used.

See also: For Statement, While Statement , If Statement

Exit Statement

Format:

```
exit;
```

Description:

The **exit** statement terminates a procedure or function before normal processing is expected to end. When the **exit** statement is executed, processing stops for the current procedure or user-defined function, and control is passed to the next procedure or user-defined function.

Example:

```
function FIRST_WOMAN();
  FIRST_WOMAN = 0;
  do i = 1 while i <= HH_MEMBERS
    if SEX(i) = 2 then
      FIRST_WOMAN = i;
      exit; {exit the function, we've found our first woman!}
    endif;
  enddo;
end; {end the function}
```

See also: Universe Statement, Skip Case Statement, Stop Function, Endcase Statement

For Statement

Format:

```
for num-var in [record | item | relation] group do
    statements;
enddo;
```

Description:

The **for** statement executes one or more statements repeatedly within a loop for each occurrence of a multiply occurring form, roster, record, item, or relation. In the example below, PERSON_EDT (i.e., the number of people in a household) would control how many times the **for** loop is executed.

Num-var contains the number of the current occurrence being examined. It cannot be changed inside the loop, but it can be referenced. Its starting value is 1, and its ending value is determined by the number of occurrences of *group* named.

The *group* is the name of a form, roster, record, item, or relation. If the group name is a record, item, or relation then the appropriate keyword **record**, **item**, or **relation** must be used before the name.

The **for** statement should be coded outside *group* it references. In the example below, note that the code is executed in the QUEST procedure. It should not be located in the PROC PERSON_EDT, or in a **proc** for any of the data items within the person record.

Example:

```
PROC QUEST
  spouse = 0;
  for i in PERSON_EDT do
    if relationship = 2 then
      spouse = i;
      break;
    endif;
  enddo;
```

See also: Do Statement, While Statement

If Statement

Format:

```
if condition then    [] indicates that this part is optional.
  statements;
[elseif condition then
  statements;]
[else
  statements;]
endif;
```

Description:

The **if** statement executes different statements based on the value of "condition". The condition following the **if** command is evaluated. If the condition is **true**, then the statements following it are executed and execution moves to the first statement after the **endif** keyword. If the condition is **false**, execution moves to the first **elseif** keyword or the **else** keyword (if there are no **elseif** keywords).

The **elseif** blocks are evaluated in the same way as the first **if** block. When CSPro finds a condition that is **true** it executes the statements following it and moves to the first statement after the **endif** keyword. If all the conditions are **false**, the statements following the **else** keyword are executed. If none of the conditions are true and there is no **else** keyword, execution moves to the first statement after the **endif** keyword without the execution of any statements within the **if** statement.

Every **if** statement must contain an **endif** keyword. However, if multiple **elseif** keywords are nested within an **if** block, they may be terminated with a single **endif** keyword. The statements within the **if** statement can be any number of CSPro statements. If a condition contains an inequality (e.g., >, <, >=, <=) and one of the values tested in the inequality is a special value (e.g., MISSING, NOTAPPL, or DEFAULT), the result of the condition is false and execution skips to the statement following the else.

Example:

```
if X = 3 then
  z = 6;
elseif x in 4:5 or y in 7:9,12 then
  z = 7;
else
  z = 8;
endif;
```

Next Statement

Format:
```
next;
```

Description:
The next statement ends a do, while, or for loop early and continues execution with the next iteration of the loop. If the next iteration causes termination of the loop, then execution will begin with the first statement after the enddo.

Example:
```
{ find all spouses }
NUMSP = 0;
for I in PERSON do
  if P02_REL <> 2 then
    next;
  endif;
  NUMSP = NUMSP + 1;
  SP(NUMSP) = I;
enddo;
```

See also: Do Statement, For Statement, While Statement, Break Statement, Exit Statement

Universe Statement

Format:
```
universe condition [case];
```
[ ] indicates that this part is optional.

Description:
The **universe** statement determines whether to allow normal execution of a procedure or function based on the value of condition. The condition following the **universe** command is evaluated. If the condition is **true**, then the statements following it are executed. If the condition is **false**, processing stops for the current procedure or user-defined function, and control is passed to the next procedure or user-defined function. In batch editing mode, if the optional **case** keyword is specified and the condition is **false,** the program will stop processing the case but will still write it to the output file if one is specified).

Example 1:
```
PROC FERTILITY

universe SEX = 2 and AGE in 12:49;
```

Example 2:
```
universe HHTYPE = 1 case;
```

Example 2 (rewritten without the **universe** statement):
```
if not HHTYPE = 1 then
    endcase;
endif;
```

See also: Exit Statement, Endcase Statement, If Statement

While Statement

Format:
```
while condition do
  statements;
enddo;
```

Description:

The **while** statement executes one or more statements repeatedly, in a loop, while the logical condition is true. The **while** and **enddo** keywords are required. Unlike the do statement, the index is not automatically incremented. The user must therefore remember to save (or decrement) the controlling variable(s) as needed to ensure termination of the loop. The condition is evaluated on each repetition of the loop before any of the statements are executed.

Example:

```
i = 1;
NumPeople = totocc (Person);
while i <= NumPeople do
  if rel(i) = notappl and sex(i) = notappl and age(i) = notappl then
    delete (PERSON(i)); {remove "blank" population records}
  else
    i = i + 1; {only increment i if we DON'T delete someone}
  endif;
enddo;
```

See also: Do Statement, For Statement, If Statement

Assignment Statements

Assignment Statement

Format:

```
numeric-variable = numeric-expression;
string-variable = string-expression;
```

Description:

The assignment statement sets a variable equal to the value of an expression. If the expression is a string-expression, then the variable must be alphanumeric. If the expression is numeric or conditional, then the variable must be numeric.

Examples:

```
AGE = 10;
Q102 = PREV_AGE;
Y = sqrt(X);
NAME = "John Doe";
```

See also: Setvalue Function

Recode (Box) Statement

Format:

```
recode var-1 [:var-2 [:var-n]]   => var-out;
      [range-1] [:range-2 [:range-n]]  => exp;
      [range-1] [:range-2 [:range-n]]  => exp;
          : : :
                [: [:]]   => other-exp;
endrecode;
```

[ ] indicates that this part is optional.

Description:

The **recode** statement assigns a value to a variable based on the value of one or more other variables. It is used to rescale variables, to assign values to variables, and to create new variables from existing ones. It works like a multiple if statement but is easier to use. The recode statement evaluates each line within it sequentially, beginning with the first line.

If the values of variables "var-1" to "var-n" lie within the ranges "range-1" to "range-n", respectively, then "var-out" is assigned the value given by the expression on the first line and the **recode** statement is ended. If the values of the variables "var-1" to "var-n" do not all lie within their specified ranges, then the next line of the **recode** statement is evaluated. This process continues until either a value is assigned to "var-out" or the end of the **recode** statement is reached.

A variable in a multiple record or group cannot be used in the **recode** statement except in data entry applications (where it may be specified without an index and the current occurrence of a variable is assumed). Use working variables to refer to or to assign values to variables in multiple sections or groups.

Variables "var-1" through "var-n" are referred to as independent variables and must be separated by colons [**:**]. "Var-out", the variable whose value is assigned by the recode statement, is referred to as the dependent variable. A **recode** statement can have any number of independent variables, but only one dependent variable. The dependent variable can also be included among the independent variables. The dependent variable is separated from the independent variables by **=>**.

The ranges specified in the **recode** statement (i.e., "range-1" through "range-n") can take the following formats:

- A range between two values, e.g., 12-15
- An individual value, e.g., 9
- A comparison with another value, using >, <, >=, <=, or <>, e.g., < 5
- A special value , e.g., NOTAPPL
- Some combination of these formats separated by a comma, e.g., < 5, 9, 12-15, missing

A blank range for an independent variable includes all values. A blank range for all independent variables on the last line of a **recode** statement acts as a catch-all condition. It ensures that a value is always assigned to "var-out" by the recode statement. If a value is not assigned by the **recode** statement, the value of "var-out" will not change. The number of ranges on each line must equal the number of independent variables.

The expression for the dependent variable must result in a numeric value (if "var-out" is a numeric variable) or a string (if "var-out" is an alphanumeric variable).

(Note to ISSA users: The Recode and Box statements are identical.)

Example 1:

```
recode AGE => AGE_GROUP;
       0-19 => 1;
      20-29 => 2;
      30-39 => 3;
      40-49 => 4;
      >= 50 => 5;
            => 9;
endrecode;
```

is equivalent to the following if statements:

```
if AGE in 0:19 then
  AGE_GROUP = 1;
```

```
elseif AGE in 20:29 then
  AGE_GROUP = 2;
elseif AGE in 30:39 then
  AGE_GROUP = 3;
elseif AGE in 40:49 then
  AGE_GROUP = 4;
elseif AGE >= 50 then
  AGE_GROUP = 5;
else
  AGE_GROUP = 9;
endif;
```

Example 2:

```
recode ATTEND : ED_LEVEL => EDUC;
    2,notappl :          => 1;
            1 : 1       => 2;
            1 : 2,3     => 3;
              :          => 9;
endrecode;

is equivalent to the following if statements:

if (ATTEND = 2 or ATTEND = notappl) then
  EDUC = 1;
elseif ATTEND = 1 then
  if ED_LEVEL = 1 then
    EDUC = 2;
  elseif ED_LEVEL in 2:3 then
    EDUC = 3;
  endif;
else
  EDUC = 9;
endif;
```

Example 3:

```
recode UNITS : NUMBER  => DAYS;
             : notappl => notappl;
             : missing => missing;
           1 :         => NUMBER;
           2 :         => NUMBER*7;
           3 :         => NUMBER*30;
           4 :         => NUMBER*365;
             :         => missing;
endrecode;
```

is equivalent to the following if statements:

```
if NUMBER = notappl then DAYS = notappl;
elseif NUMBER = missing then DAYS = missing;
elseif UNITS = 1 then DAYS = NUMBER;
elseif UNITS = 2 then DAYS = NUMBER*7;
elseif UNITS = 3 then DAYS = NUMBER*30;
elseif UNITS = 4 then DAYS = NUMBER*365;
else DAYS = missing;
endif;
```

See also: If Statement

Impute Function

Format:

```
impute (item-name, expression)
    [stat (item-name1, item-name2,....,item-nameN)]
    [title (alpha-expression)]
    [vset (vset-number)]
    [specific];
```

[ ] indicates that this part is optional.

Description:

The **impute** statement assigns a value to a data item and logs the frequency of assignments. The parameters are:

- **item_name**:
  The dictionary data item to impute. The item must be numeric, with or without decimals, and can be single or multiple. If the item is multiple and is being used inside its PROC, the current occurrence is assumed. If the item is multiple and is being referenced outside its PROC, an occurrence must be specified. Occurrence numbers are 1-based.

- **expression**:
  Is a variable name, a number, or an arithmetic expression; for example, 'TEMPAGE', ‘5’, or ‘2+3’.

The options are:

- **STAT** (item_name1[, item_name2]):
  Tells the system to generate the secondary .dcf and .dat files. These files contain references to the data items that were changed; i.e., identifying ID values, the data item being imputed, and each subsequent data item named in the **stat** parameter list.

- **TITLE** (alpha_expr):
  Under the "IMPUTE STATISTICS" heading at the top of each page, this line will replace the default line that is generated ("IMPUTED Item (<unique name of data item>): <label name of data item>").

- **VSET** (vset_number):
  Is the 1-based value set number of the item being imputed (i.e., impute_item_name), and corresponds to the order of listing in the data dictionary (i.e., the first value set of an item will be number 1, the second value set will be number 2, etc). This may yield a different number of frequencies than what occurred when not using this option. For example, if you are imputing "age", and do not use the VSET option, your report will show the total number of imputations that occurred. However, if you use the VSET option, and the value set you choose does not list all possible values, then the total number of imputations listed in the frequency report will most likely be less than that given if you did not use this option.

- **SPECIFIC**:
  Indicates if the frequency will be generated alone or not. You can have multiple **impute** statements for a single data item. If you do this, you may want to have frequency reports separated for each **impute**. If **specific** is not used, all imputations for a given data item will be lumped together in the frequency report.

The **impute** command can generate up to three files:

<xxx>.frq
<xxx>.dat (only generated if the STAT option is used)
<xxx>.dcf (only generated if the STAT option is used)

where XXX corresponds to the name of the data file used in the run. These files will be placed in the directory where the .bch application is located.

The format of the report contained in the .frq file is divided up into five columns as follows:

Category	Freq	CumFreq	Percent	CumPct
1	3432	3432.0	14.8	14.8
2	193	3625.0	0.8	15.7
:				

Column one:	lists the values that were assigned during the imputations (1, 2, etc)
Column two:	shows the frequency (that is, the total number of times) each value was assigned (i.e., number '2' was assigned 193 times)
Column three:	displays cumulative totals of the "Freq" column
Column four:	indicates what percentage each imputation represents from the total number of imputations made (i.e., number '2' was imputed 193 times, representing barely one percent (0.8) of the total number of imputations made)
Column five:	lists the cumulative totals of the "Percent" column

Example:

```
impute(P04_AGE, TEMPAGE) title("Age updated via TempAge") vset(2);
```

Code Example:

A code example of this statement can be found in the *Examples/Impute* folder in the CSPro software folder.

Setvalue Function

Format:

```
b = setvalue(alpha-expression,numeric-expression | alpha-
expression[,numeric-expressions]);
```

Description:

The **setvalue** function executes an assignment when the name of the variable to be assigned to is not known at compilation time (when writing the application). The function searches for the first parameter *alpha-expression* and then attempts to assign the second parameter expression, numeric or alpha, to the variable. Optional parameters allow you to pass occurrence numbers to the function. This function is especially useful when combined with a userbar button or an onkey character sequence.

Return value:

The function returns 1 if the variable name was found and the value could was successfully changed. The function returns 0 if the variable was not found, or if an alpha value was assigned to a numeric variable or a numeric value was assigned to an alpha variable.

Examples:

```
// the following statements are equivalent but you would not use setvalue
because the left-hand side of the assignment is already known

setvalue("NAME","Amelia");
NAME = "Amelia";

setvalue("AGE",20);
AGE = 20;


// below, the left-hand side of the assignment is not known

setvalue(getsymbol(),missing); // set the current field to missing

setvalue(getsymbol(),"DK",3); // set the third occurrence of the current
field to "DK"
```

See also: Assignment Statement, Getvalue Function

Getvalue Function

Formats:

```
d = getvalue(alpha-expression[,numeric-expressions]);
d = getvaluenumeric(alpha-expression[,numeric-expressions]);
s = getvaluealpha(alpha-expression[,numeric-expressions]);
```

Description:

The **getvalue** function returns the value of a variable when the variable name is not known at compilation time (when writing the application). The function searches for the first parameter *alpha-expression* and then returns the value of a variable with that name, if found. **Getvalue** and **getvaluenumeric** return the value of numeric variables. **Getvaluealpha** returns the value of an alpha variable. Optional parameters allow you to pass occurrence numbers to the function.

Return value:

The function returns the variable's value if the variable name was found. If a numeric variable was not found, the functions **getvalue** and **getvaluenumeric** return DEFAULT. If an alpha variable was not found, **getvaluealpha** returns a blank string.

Examples:

```
errmsg("The current field's value is %d",getvalue(getsymbol()));
```

See also: Setvalue Function

Data Entry Statements and Functions

Accept Function

Format 1:

```
i = accept(heading, opt-1, opt-2[,...opt-n]);
```

Format 2:

```
i = accept(heading, alpha-array);
```

Description:

The **accept** function displays a menu with the heading and list of choices ("opt-1", "opt-2", etc.). The operator can move the down- or up-arrow keys to select the desired options and press **Enter**. The operator can also use the mouse to click on the desired option.

The number of options will be calculated dynamically if an alpha array is passed to the accept function. Each element of the array will be displayed until a blank string is found in the array.

Return value:

The function returns the number of the option selected: 1 for the first option, 2 for the second option, etc. The value 0 is returned if the **Esc** key is pressed and none of the options is chosen.

Example 1:

```
PROC UR
preproc
  I = 0;
  do until I in 1:2
    I = accept("Area Designation?", "Urban", "Rural");
  enddo;
  $ = I;
  noinput;
```

Example 2:

```
PROC GLOBAL

array string acceptOptions(20);

PROC ACTION

preproc

    numeric ctr;

    do ctr = 1 while ctr <= count(NAME)
        acceptOptions(ctr) = NAME(ctr);
    enddo;

    acceptOptions(ctr) = ""; // mark the end of the array

    ACTION = accept("Select a Person to Interview",acceptOptions);
```

See also: Do Statement, Noinput Statement

Advance Statement

Format 1:

```
advance [to] field-name;
```

Format 2:

```
advance [to] alpha-variable;
```

[] indicates that this part is optional.

Description:

The **advance** statement moves forward field-by-field to the specified field, executing preprocs and postprocs as it goes. It acts as though the **Enter** key were pressed repeatedly until either the specified field appears or one of the procedures executed during the advance goes to a different field.

The field name can be given directly by naming the field or indirectly by using the contents of an alpha variable.

Field name can be located in any record at the same level as the current field, but it cannot be located at a different level. Field name must be the name of a field that has not yet been entered. If field name has already been entered, an error message will be displayed during data entry and data entry will be aborted. If you don't know whether the field has already been entered, use the move statement.

Note that the **advance** statement behaves differently from the skip statement.

Example:

```
FIRST_WOMAN = 0;
do i = 1 while i <= totocc(PERSON)
  if SEX(I) = 2 then
    FIRST_WOMAN = i;
    advance to CHILD;
  endif;
enddo;
```

See also: Move Statement, Reenter Statement, Skip Statement

Changekeyboard Function

Format:

```
i = changekeyboard(dictionary-symbol[,keyboard-id-as-numeric-expression]);
```

Description:

The **changekeyboard** function modifies the keyboard input associated with a field on a form. The dictionary symbol may be an item, group, form, or an entire dictionary. The *numeric-expression* refers to a keyboard ID (which can be determined using the Keyboard Input dialog box). A keyboard ID of 0 will reset the field to the default keyboard for the application. Calls to this function are ignored when executed on Android devices.

Return value:

The function returns the number of items whose capture positions were successfully changed. If a keyboard ID is not specified, the function will instead return the current keyboard ID associated with a given field.

Example:

```
PROC NAME

preproc

    changekeyboard(NAME,66569); // use the Dvorak keyboard to key in the
name
```

See also: Keyboard Input

Demode Function

Format:

```
i = demode();
```

Description:

The **demode** function is often used to limit the execution of certain statements to a specific mode. For example, a variable may need initialization when the operator invokes **add** or **verify** mode, but can be left unaltered for **modify** mode.

Return value:

There are three data entry operator modes:

- **add**, to input new cases; the demode function returns a '1'
- **modify**, to change cases that have already been entered; the demode function returns a '2'.
- **verify**, to reenter the cases as a check for differences between the first and second entry; the demode function returns a '3'.

Example:

```
if demode() = add then
  V103 = 3;
endif;
```

Display Orientation

In a data entry application that uses a rich visual interface, you may want to exert programmatic control over the orientation of the screen, e.g., whether the display is horizontal or vertical. For computers with video cards that support changing the display orientation, which includes most new computers as well as tablet PCs, two functions can be used to control the orientation. Calls to these functions are ignored when executed on Android devices.

GetOrientation Function

Format:

```
b = getorientation();
```

Description:

The **getorientation** function returns the current display orientation. The function will return one of four values:

- 0: The natural orientation of the display device
- 90: The display orientation is rotated 90 degrees from the natural orientation
- 180: The display orientation is rotated 180 degrees from the natural orientation
- 270: The display orientation is rotated 270 degrees from the natural orientation

SetOrientation Function

Format:

```
b = setorientation(numeric-expression);
```

Description:

The **setorientation** function changes the orientation of the display. The numeric expression must be one of the four values listed above. The function returns 1 if the display orientation was successfully changed, 0 otherwise.

Example:

```
function rotateScreen()

    numeric nextOrientation = getorientation() + 90;

    if nextOrientation = 360 then
        nextOrientation = 0;
    endif;

    setorientation(nextOrientation);

end;
```

Editnote Function

Format:

```
s = editnote();
```

Description:

The editnote function displays the note entry dialog box for adding or changing the note for the current field. You can use this function to force the collection of note text under program control. The operator can always create or edit a note manually by pressing **Ctrl+N**. Notes are stored in a file called <data file name>.NOT.

Return value:

The function returns a string, representing the contents of the field's note. If there is no note associated with the field, the string will be empty.

Example:

```
PROC COOKING
  if $ = 9 then
    OTHER = editnote();
  endif;
```

See also: Getnote Function, Putnote Function

Endlevel Statement

Format:

```
endlevel;
```

Description:

The **endlevel** statement ends data entry for the current level of the current questionnaire. The effect of this statement depends on where it is used. If **endlevel** is used in a field, roster, or form procedure, all remaining procedures within that level are skipped and control passes to the level **postproc**.

If the **endlevel** statement is executed in a level **preproc** or **postproc**, control passes to the **postproc** of the next-highest level. If it is used in the highest-level **postproc**, control passes to the form file's **postproc** (if there is one), and then data entry is terminated for the current case.

In system-controlled applications, CSPro will continue to add cases at the lowest level of a multiple-level dictionary until it is told to stop by endlevel. Therefore, an endlevel statement should be used in the postproc of the lowest level to end data entry at that level.

Example:

```
if MORE_WOMEN = 0 then
  endlevel;
endif;
```

Endgroup Statement

Format:

```
endgroup;
```

Description:

The endgroup statement finishes data entry for the current group (roster or multiply-occurring form) in a data entry application. It can not be used in a batch application. If the endgroup statement is used in an item procedure, it causes an automatic skip to the postproc of the current group/record. If the endgroup statement is executed in the preproc of the group/record, the entire group/record is skipped and control passes to the group/record's postproc.

Note: this function has superseded the endsect statement. Where **endsect** exists in an application, it will continue to work, but users creating new applications should adopt the **endgroup** instruction.

Example:

```
if KIDSBORN = 0 then
  endgroup;
endif;
```

Code Example:

A more thorough example of this statement can be found in the examples folder.

Enter Statement

Format:

```
enter form-file-name
```

Description:

The **enter** statement allows the use of a secondary form file to capture data in a secondary data file.

The *form-file-name* is the name of the secondary form file you want to use. The secondary form file must be part of your data entry application. To see the name of form files, from the **View** menu make sure "Names in Tree" is checked, or press **Ctrl+T**.

The **enter** statement cannot be used inside a function.

Example:

```
if V108 = 6 then
  enter OTHERS;
endif;
```

Getcapturetype Function

Format:

```
i = getcapturetype(dictionary-item);
```

Description:

The **getcapturetype** function returns the capture type currently associated with a field on a form. The dictionary item must be located on one of the application's forms.

Return value:

The function returns the capture type, which is one of the following values:

0	Textbox
1	Radio Button
2	Checkbox
3	Drop Down Box
4	Combo Box
5	Date Picker
6	Number Pad

Example:

```
// change SEX from a radio button to a drop down box
if getcapturetype(SEX) = 1 then
    setcapturetype(SEX,3);
endif;
```

See also: Extended Controls, Setcapturetype Function

Getlanguage Function

Format:

```
s = getlanguage();
```

Return value:

The **getlanguage** function returns a character string with the name of the language being used by the CAPI interface. This function can only be used when CAPI mode is activated.

Example:

```
if getlanguage() = "ENG" then
     errmsg("Hello");
elseif getlanguage() = "FRA" then
     errmsg("Bonjour");
endif;
```

See also: Setlanguage Function, Define Languages

Getnote Function

Format:

```
s = getnote()
```

Description:

The getnote function returns a string containing the note for the current field. If there is no note, the length of the string will be 0. Notes are stored in a file called <data file name>.NOT.

Return value:
The function returns a string containing the note text.

Example:
```
PROC BIRTH_PLACE
if length(getnote()) > 0 then
  FLD_NOTE = editnote();
endif;
```

See also: Putnote Function, Editnote Function, If Statement, Length Function

Getocclabel Function

Format:
```
s = getocclabel([item-name | group]);
```

Description:
The **getocclabel** function returns the occurrence label for a repeating item or group (i.e., the row label of a roster). If no *item name* or *group* is given, the function returns the occurrence label of the current field, as calculated as follows:

```
getocclabel($(curocc($)))
```

Return value:
The function returns a string containing the occurrence label. If an occurrence label for the item or group item does not exist, the function returns a blank string.

Example:
```
if EXPENSES > SALES then
    errmsg("You could not have made a profit on %s if expenses were greater
than sales.",getocclabel());
    reenter;
endif;
```

See also: Occurrence Labels, Create Fills In Questions

Getoperatorid Function

Format:
```
s = getoperatorid();
```

Description:
This function returns the operator id for the current operator. The operator id may have been entered by the operator or passed as a parameter in the .PFF file for the run.

Return value:
The function returns the string containing the operator id assigned to or entered by the current operator.

Example:
```
errmsg("Operator = %s", getoperatorid());
```

See also: Getusername Function, Getos Function

Getos Function

Format:

```
i = getos([alpha-variable]);
```

Description:

The **getos** function returns a code that indicates what operating system is running the program. The codes are:

- 10 - 19: Windows
- 20 - 29: Android

At the moment, only the codes 10 and 20 are returned, but in the future other codes in the range may be used that will differentiate between different versions of the operating system. An optional alphanumeric variable can be passed to the function that will give detailed information on the operating system, including the operating system name and the version number. Note, however, that Windows version numbers do not necessary match the official release numbers. For example, Windows 8.1 has a version number of 6.3.

Return value:

The function returns the operating system code or, optionally, a text description of the operating system.

Example:

```
string operatingSystemName;
getos(operatingSystemName);

errmsg("Code = %d, Text = %s",getos(),operatingSystemName);

// the above code might display, for example:

// Code = 10, Text = Windows;6.1

// Code = 20, Text = Android;4.2.2
```

See also: GetOperatorId Function, Getusername Function

Getrecord Function

Format:

```
s = getrecord(alpha-expression);
```

Description:

The **getrecord** function returns the name of the record that contains the item identified by the *alpha expression*.

Return value:

The function returns a string up to 32 characters long. If the item does not exist, the function returns a blank string.

Example:

```
errmsg("SEX belongs to %s",getrecord("SEX")); // SEX belongs to POPULATION
errmsg("WATERSOURCE belongs to %s",getrecord("WATERSOURCE")); // WATERSOURCE
belongs to HOUSING
```

See also: Getsymbol Function

Getusername Function

Format:

```
s = getusername();
```

Description:

This function returns the login of the current user. On Windows this is the name of the user currently logged in, whereas on Android this is the first account registered on the device.

Return value:

The function returns a string containing the user name.

Example:

```
errmsg("User name = %s", getusername());
```

See also: GetOperatorId Function, Getos Function

GPS Function

It is possible to take advantage of the functionality of a Global Positioning System (GPS) receiver when designing an application for use on either a laptop or tablet with a GPS receiver or using either Android or Windows Mobile operating systems. Manipulating the GPS receiver is done from within the program's logic.

Basic Example

```
gps(open,3,4800); // on a laptop or tablet; COM3, 4800 baud
gps(open); // on Android or the Pocket PC

if gps(read,5) then // a successful attempt at a read, for up to five
seconds
    errmsg("Latitude is %f, longitude is %f",gps(latitude),gps(longitude));

else
    errmsg("GPS signal could not be acquired");

endif;

gps(close);
```

Opening and Closing the GPS Receiver

```
b = gps(open | close);
```

Before using the PDA's GPS receiver, it is necessary to open a connection to the GPS unit. After making all necessary GPS readings, close the connection. The function returns 1 if successful, 0 otherwise. On the Pocket PC, the GPS settings are controlled using the External GPS option under the device's

settings. However, is using CSPro on a laptop or tablet, it is necessary to indicate the hardware settings of the GPS unit, specifically the number of the COM port and the baud rate (see the above example).

Obtaining the Last Successful GPS Reading

```
b = gps(readlast);
```

The **readlast** command obtains the last successful GPS reading, a reading that might be very old. If the GPS unit has been turned on for some time and the PDA is being used outdoors, it is likely that the reading is very fresh (recent), but if, for example, an enumerator walks inside a building to conduct an interview, the reading may be minutes or hours old, from the last time that the enumerator was outside. The function returns 1 if there was a successful previous reading, 0 otherwise.

Obtaining a New GPS Reading

```
b = gps(read);
b = gps(read,numeric-expression);
i = gps(read,numeric-expression,alpha-expression);
```

The **read** command obtains a new GPS reading in a specified time period (in seconds). If no time period is specified, the program will pause for up to three seconds to obtain a reading. A reading time of up to ten minutes (600 seconds) may be specified. An optional third parameter displays a message while the program attempts to obtain a GPS reading. The message box has a cancel button and if the user cancels the operation, the function returns -1. Otherwise the function returns 1 if a reading was successful, 0 otherwise. Unlike the readlast command, a successful function call with read guarantees a fresh GPS reading.

It is possible on Android devices to specify a desired level of accuracy for the reading. When specified, the function will continue to take GPS readings until a reading at or below the level of accuracy is achieved. The function will still return 1 if, though timing out, a successful reading at an accuracy level greater than specified was achieved during the allotted time period. This ensures that a GPS reading will be as accurate as possible, but with the assumption that any reading is better than no reading. Use a second numeric expression to specify the level of the accuracy.

```
b = gps(read,time-period,accuracy-level);
i = gps(read,time-period,accuracy-level,alpha-expression);
```

Querying a Successful GPS Read

If the readlast or read commands returned successfully, the GPS system has valid values for latitude and longitude. Further attributes of the reading can be queried though they are not guaranteed to be valid.

```
f = gps(latitude);
f = gps(longitude);
f = gps(altitude);
f = gps(satellites);
f = gps(accuracy);
f = gps(readtime);
```

Latitude and **longitude** return coordinates in degrees. **Altitude** returns the number of meters above sea level of the reading. **Satellites** returns the number of satellites used to calculate the values in the last reading. Generally, the greater number of satellites, the better the quality of the reading. **Accuracy** is a calculation of the precision of the last reading. On Windows devices, an accuracy value of 1 is the most

accurate and 50 is the least accurate reading. On Android devices, the value signifies the accuracy of the reading, measured in meters. **Readtime** returns, in local time, the time of the last successful reading. If the queried value (other than latitude and longitude) is not available, or if the last GPS read was unsuccessful, the function will return DEFAULT.

Calculating Distances

The gps function can also calculate great-circle distances between two sets of GPS coordinates. Great-circle calculations give a rough approximation of the distance between two points. The function returns the distance as measured in meters.

```
f = gps(distance,numeric-expression,numeric-expression,numeric-
expression,numeric-expression);

numeric kilometersToCensusBureau = gps(distance,38.846261,-
76.929445,gps(latitude),gps(longitude)) / 1000;
```

Highlighted Function

Format:

```
b = highlighted(field-name);
```

Description:

The **highlighted** function is used to determine whether a field is on the path (green) in system controlled mode, or has been passed through (green or yellow) either directly or by skips in operator controlled mode. This can be used to determine whether an item has been entered or skipped because of logic in system controlled mode, or whether is before or after the high water mark in operator controlled mode.

Return value:

The function returns true (1) if the field has been passed (green or yellow) and false (0) if the field is yet to be entered (white).

Example:

```
if highlighted(HOURS_WORKED) then
  errmsg("Hours Worked = %d",HOURS_WORKED);
else
  errmsg("Skipped or Not entered yet!");
endif;
```

See also: Operator vs. System Controlled, Color of Fields in Data Entry User's Guide

Ispartial Function

Format:

```
b = ispartial();
```

Description:

The ispartial function determines whether the current case was opened from a partial case or not. This function can be coded in any procedure except the preproc and postproc of the run.

Return value:

The function returns a logical value of 1 (true) if the case was opened from a partial case and 0 (false) otherwise.

Example:
```
if ispartial() then
  errmsg("Entering a partially saved case");
endif;
```

See also: Savepartial Function, Onstop Global Function

Killfocus Statement

Format:
```
Killfocus;
```

Description:
The **killfocus** statement indicates that the following statements are executed when a form, roster, or field stops being active. A **killfocus** procedure can be coded in a **proc** for any form, roster, or field data entry applications. **Killfocus** procedures are only executed in data entry applications. Statements in a **killfocus** procedure are executed whenever you move off the object in which they are coded. If **postproc** statements are executed, they are executed after **killfocus** statements. **Killfocus** statements are executed when you complete an object; that is, when either logic or operator intervention (such as clicking with the mouse on another field, manually skipping or backtabbing, etc.) moves the cursor off the field. They are also executed when you perform noinput of a field or when the field is protected.

Example:
```
PROC SEX
{preproc would go here, if desired }
{onfocus would go here, if desired }
killfocus
  if ($ = 2 and AGE < 5) then
    reenter;
  endif;
{postproc would go here, if desired }
```

See also: Onfocus Event, Proc Statement, Preproc Statement, Postproc Statement, Order of Executing Data Entry Events, Order of Executing Batch Edit Events, Reenter Statement, If Statement

Move Statement

Format 1:
```
move [to] field-name [advance|skip]
```

Format 2:
```
move [to] alpha-variable [advance|skip]
```

[ ] indicates that this part is optional.
| indicates that one or the other keywords may be used

Description:
The move statement allows movement to a field without regard to whether it is before or after the current field. This is particularly useful when coding in the Onkey function or when using a alpha variable to give the field name.

The field name can be given directly by naming the field or indirectly by using the contents of an alpha variable.

Movement to a field before the current field acts exactly like a reenter statement. The action of move a field after the current field depends on the keywords **skip** or **advance**. If no keyword or **skip** is coded, forward movement acts exactly like a skip statement. If **advance** is coded, forward movement acts exactly like an advance statement.

Example 1:
```
move SEX;
```

Example 2:
```
move to SEX advance;
```

Example 3:
```
move to LAST_FIELD advance;
```

where SEX is a field and LAST_FIELD is an alpha variable.

See also: Reenter Statement, Skip Statement, Advance Statement, Onkey Function

Noinput Statement

Format:
```
noinput;
```

Description:

The **noinput** statement prevents input of a field during data entry. This command can be coded only in the **preproc** or **onfocus** procedures.

When the statement is executed in a **preproc**, control passes directly from the field's **preproc** to the field's **postproc**, executing the **onfocus** and **killfocus** procedures (if present) and performing the item range check, but not permitting input of the field. When the statement is executed in an **onfocus**, control passes directly from the field's **onfocus** to the field's postproc, executing the **killfocus** procedure if present and performing the item range check, but not permitting input of the field. The field is on the data entry path even though entry is prevented.

The effect of the **noinput** statement is similar, but not identical, to that of a protected field. If a **noinput** statement is used, it is possible to back-tab to the field. It is not possible to back-tab to a field that is protected.

Example:
```
PROC Q102
preproc
 if Q101 <> 1 then
   noinput;
 endif;
```

Onchangelanguage Global Function

Format:
```
function OnChangeLanguage();
```

Description:

The **onchangelanguage** global function allows you to execute certain code after the user has changed the CAPI language via the CSEntry Options menu. This function must be placed in the Global Procedure.

Returned value:
The return value of the function is ignored.

Example:

```
function OnChangeLanguage()
    setvaluesets(concat("_",getlanguage()));
end;
```

See also: Global Procedure, User Defined Functions

Onchar Global Function

Format:

```
function OnChar(key-value);
```

Description:
The **onchar** global function allows you to trap characters in order to perform special actions or to change the action of the character. It can also be used to disable or remap characters. This function must be placed in the Global Procedure.

If an onchar global function is coded, every character the operator types is sent to the onchar function for processing. If the onchar function returns a value, then the return value is processed by the field as the character. If a statement in the onchar function causes movement to another field within the case, then the movement is executed. If no onchar function is coded, then characters are unmodified.

The **key value** is a number code identifying what character was typed using the keyboard. Its value can be used within the function.

You can use the OnKey Character Map to determine the value of characters.

Calls to this function are ignored when executed on Android devices.

Differences between onkey and onchar:
The onchar function differs from the onkey function. A keystroke contains information about the key pressed on the keyboard, regardless of what this keystroke eventually maps to. A character refers to the final representation of one or more keystrokes. This is important when using non-Latin languages that require multiple keystrokes to create one character. For example, creating the Chinese character '马' using a Pinyin input system requires two keystrokes: 'm' and 'a.' The code for this character is 39532. If typing such a keystroke, onkey will be called several times before onchar is called. If both onkey and onchar functions exist, onkey will always be called before onchar is called.

The onchar function also returns values different from the onkey function for some Latin keystrokes. For example, with Caps Lock off, if a keyer holds down Shift and types 'M,' onkey will return 1077 (1000 for the shift, 77 for 'm'). Onchar, on the other hand, will return 77, the character code for 'M.' For a lowercase 'm,' onkey returns 77 and onchar returns 109, the character code for 'm.'

The onchar function does not return any information about whether any of the Shift, Ctrl, or Alt keys were held down when the character was typed.

Returned value:
Like any global function, the onchar global function returns an integer value. The value should be either the value of the character pressed (the same as the value passed to the function), a substituted character value (remapping the character), or zero (0) to indicate that the character is to be ignored.

Example:

```
function OnChar(keystroke)

    if keystroke = 24110 then // 帮 (bang)
        move to HELP_FORM;

    else
        OnChar = keystroke;

    endif;

end;
```

See also: Global Procedure, User Defined Functions, OnKey Character Map, Onkey Global Function

Onfocus Statement

Format:

```
Onfocus;
```

Description:

The **onfocus** statement indicates that the following statements are executed when a form, roster, or field becomes active. An onfocus procedure can be coded in a proc for any form, roster, or field data entry applications. Onfocus procedures are only executed in data entry applications.

Statements in an onfocus procedure are executed when you move onto the object in which they are coded. If preproc statements are executed, they are executed before onfocus statements. Onfocus statements are executed when you move backward onto an object either via the reenter statement, moving backwards to it with the cursor, or back-tabbing to it. They are also executed when you perform noinput of a field or when the field is protected.

Example:

```
PROC TOTAL_INCOME
{preproc would go here, if desired}
onfocus
  TOTAL_TEMP = WAGES + OTHER;
{killfocus would go here, if desired }
{postproc would go here, if desired }
```

See also: Killfocus Event, Proc Statement, Preproc Statement, Postproc Statement, Order of Executing Data Entry Events, Order of Executing Batch Edit Events

Onkey Global Function

Format:

```
function OnKey(key-value);
```

Description:

The **onkey** global function allows you to trap keystrokes in order to perform special actions or to change the action of the key. It also can be used to disable or remap keys. This function must be placed in the Global Procedure.

If an onkey global function is coded, every keystroke the operator types is sent to the onkey function for processing. If the onkey function returns a value, then the return value is processed by the field as the keystroke. If a statement in the onkey function causes movement to another field within the case, then the movement is executed. If no onkey function is coded, then keystrokes are unmodified.

The **key value** is a number code identifying what key was pressed on the keyboard. Its value can be used within the function. See detailed description below.

You can use the OnKey Character Map to determine the value of characters.

Calls to this function are ignored when executed on Android devices.

Returned value:

Like any global function, the onkey global function returns an integer value. The value should be either the value of the key pressed (the same as the value passed to the function), a substituted key value (remapping the key), or zero (0) to indicate that the key is to be ignored.

Example:

```
function OnKey(x)
  if x in 114:116 then { F3, F4, F5 }
    { don't allow these keys to work, eat the key }
    OnKey = 0;
  elseif x = 2069 then { Ctrl+E go to END_FIELD }
    move to END_FIELD;
  else
    OnKey = x; { return rest of keys }
  endif;
end;
```

Keyboard Key Values

The key values passed into and out of the **onkey** function are given below.

Single key values are given below. When Shift, Ctrl, and/or Alt are held down while pressing another key, the following values are added to the code of the other key.

Keys	Add
Shift	1000
Ctrl	2000
Shift+Ctrl	3000
Alt	4000
Alt+Shift	5000
Alt+Ctrl	6000
Alt+Shift+Ctrl	7000

For example, if Shift, Ctrl, and/or Alt are held down when A is pressed

Keys	Code
A	65
Shift+A	1065
Ctrl+A	2065
Shift+Ctrl+A	3065
Alt+A	4065

Alt+Shift+A	5065
Alt+Ctrl+A	6065
Alt+Shift+Ctrl+A	7065

Numbers

For the number keys across the top of the keyboard, or on the numeric keypad when the "NumLock" button is depressed, the following keyboard key value will be returned:

Key	Code
0	48
1	49
2	50
3	51
4	52
5	53
6	54
7	55
8	56
9	57

Letters

For the letters a-z and A-Z, the following keyboard key values will be returned. Please note that the status of the CapsLock key does not effect the code. However, if the Ctrl, Shift, and/or Alt keys are being held down, they will change the code returned. See above.

Key	Code
a	65
b	66
c	67
d	68
e	69
f	70
g	71
h	72
i	73
j	74
k	75
l	76
m	77
n	78
o	79
p	80
q	81
r	82
s	83
t	84
u	85
v	86
w	87
x	88
y	89

z	90

Function Keys

Key	Code
F1	112
F2	113
F3	114
F4	115
F5	116
F6	117
F7	118
F8	119
F9	120
F10	121
F11	122
F12	123

Non-numeric Numpad Keys:

Key	Code
Num Lock	144
/	111 (with or without num lock)
*	106 (with or without num lock)
-	109 (with or without num lock)
+	107 (with or without num lock)
.	110 (with num lock—46 without)

Miscellaneous Keys:

Key	Code
SysReq	no code returned
Bksp	8
Tab	9
Enter	13 (with or without num lock)
Break	19
Caps	20
Escape	27
Space	32
Page Up	33
Page Down	34

End	35
Home	36
Left Arrow	37
Up Arrow	38
Right Arrow	39
Down Arrow	40
Insert	45
Delete	46
Wnd	91 (flying window)
Scroll Lock	145
;	186
=	187
,	188 (comma)
-	189
.	190 (period)
/	191
`	192 (accent)
[	219
]	221
\	220
'	222

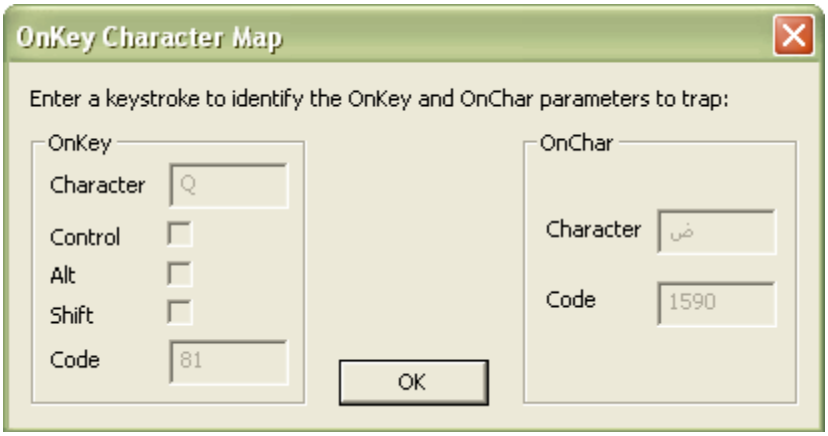
Control Keys:

Key	Code
Shift	1016
Ctrl	2017
Alt	4018

See also: Global Procedure, User Defined Functions, OnKey Character Map, OnKey Character Map

OnKey Character Map

With the logic view active, select View -> OnKey Character Map.



The OnKey Character Map is used to determine the codes for keystrokes or characters. Type a character and the OnKey and OnChar codes for that character will be displayed in the Code field. In the above image, an Arabic keyboard was used to generate the letter Dad. In this example, the OnKey function will be called with the keystroke value 81 because the 'Q' key was pressed to generate the letter 'Dad'. The OnChar function will be called with the character value 1590, representing the Arabic letter.

See also: Onchar Global Function, Onkey Global Function

Onstop Global Function

Format:

```
function OnStop();
```

Description:

Onstop is a special global function. It has no return value and must be placed in the Global section just like any other user-defined functions.
When defined, it provides control over what happens when the data entry operator tries to stop data entry using the ESC key, the Stop button, pressing Ctrl+S, or attempting to exit data entry. When any of the above events occur, the onstop function is executed and no stop dialog (discard, save, cancel) occurs.

If an onstop function has been coded in a data entry application, then when resuming a partial case, no resume dialog ("Do you want to go to last") occurs. If special actions are required when entering a partial case, check whether a partial case has been entered using the ispartial function and program the appropriate action. You can retrieve the name and occurrence number of the last field entered (on a one-level application) by calling `getsymbol(savepartial)` from the preproc of the questionnaire.

The onstop function is not executed when the stop function is executed.

The onstop function can be used to keep the operator from stopping data entry (see Example 1 below) or to allow stopping only under certain conditions (see Example 2 below)

Example 1:

```
Proc Global
function OnStop();
 last_field = getsymbol();
  reenter last_field;
end;
```

Example 2:

```
Proc Global
function OnStop();
  last_field = getsymbol();
  if last_field in "FIRST_NAME", "LAST_NAME" then
    reenter last_field;
  else
    savepartial();
    stop(1);
  endif;
end;
```

See also: User-defined Functions, Function Statement, Stop Function, Savepartial Function, Ispartial Function, Endlevel Statement

Putnote Function

Format:

```
b = putnote(string);
```

Description:
The putnote function places a string in the note for the current field. If the string is empty, there will be no note for the field. Notes are stored in a file called <data file name>.NOT.

Return value:
The function returns the length of the note placed in the notes file, or -1 if called in error.

Example:

```
PROC COOKING
if $ <> 9 then
  putnote(""); {this will delete the note}
endif;
```

See also: Getnote Function, Editnote Function

Randomizevs Function

Format:

```
i = randomizevs(dictionary-symbol[,exclude(numeric-variable,...)]);
```

Description:

The **randomizevs** function scrambles the order of values in a value set. The function is useful when using extended controls in a data entry application. Some survey applications choose to display the possible response categories in a randomized way so as to minimize an enumerator or respondent's selection bias.

The dictionary symbol may be an item, group, form, or an entire dictionary. An optional exclusion list allows the user to prevent certain values from being given a random order. This is useful for variables like Don't Know, which, after the randomization, would remain at the bottom of the list of values.

Return value:

The function returns the number of items for which the value sets were successfully randomized.

Example:

```
randomizevs(FOOD,exclude(5));
```

becomes

See also: Random Function, **Randomin Function**

Reenter Statement

Format 1:

```
reenter [field-name];
```

Format 2:

```
reenter alpha-variable;
```

[ ] indicates that this part is optional.

Description:

The reenter statement is used to force the entry operator to reenter the contents for the current variable, or for a field that was entered earlier. "Field-name" specifies the field to be reentered. If no "field-name" is specified, the current field is reentered. "Field-name" must be earlier on the data path than the current variable. If it is not, an error message will be displayed during data entry and data entry will be aborted. If you don't know whether the field is earlier in the data path, use the move statement.

The field name can be given directly by naming the field or indirectly by using the contents of an alpha variable.

When a reenter statement is executed, the preproc for "field-name" will not be executed. If "field-name" is on a different form than the current field, that form will be displayed automatically. The postproc of "field-name" will be executed normally after "field-name" has been reentered.

Example:

```
if KIDS = 1 & BOYS = 0 & GIRLS = 0 then
  reenter KIDS;
endif;
```

See also: Move Statement, Skip Statement, Advance Statement

Savepartial Function

Format:

```
b = savepartial();
```

Description:

The savepartial function saves the current case as a partially added, modified or verified case. It is useful in a large data entry application to perform intermediate backups. It can also be used to automatically perform a partial save when the keyer stops the case before completing it.

The savepartial function can be coded only in the preproc or postproc of the field. The function cannot be used before all id values for the case have been entered.

Return value:
The function returns a logical value of 1 (true) if the case was successfully saved as partial case and 0 (false) otherwise.

Example:
```
OK = savepartial();
```

See also: Ispartial Function, Stop Function

Selcase Function

Format:
```
b = selcase(ext-dict-name,alphanumeric-expression[,offset])
     [include(dictionary-item])] [where logical-expression];
```

[] indicates that this part is optional.

Description:
The **selcase** function allows a data entry operator to select and load a case from an external file. This function can only be used in data entry applications. It searches the index of the external file named by "*ext-dict-name*" for all cases whose keys match the criterion specified by "*alphanumeric-expression*." If two or more matching keys are found, they will be presented to the entry operator in a display box. Using a highlight bar, the operator can select one of the keys. The case identified by that key is then read into memory. If only one key is found, the case with that key will be read into memory without operator input.

The "*offset*" tells CSPro the number of characters, beginning with the first character of the ID items for the external file, that should be suppressed if multiple matches are found.

"*alphanumeric-expression*" can be a literal or a CSPro expression. The matching is case sensitive. If an empty string is passed, all cases in the external file are returned.

"*include*" tells CSPro to list additional items from the specified dictionary in the display box.

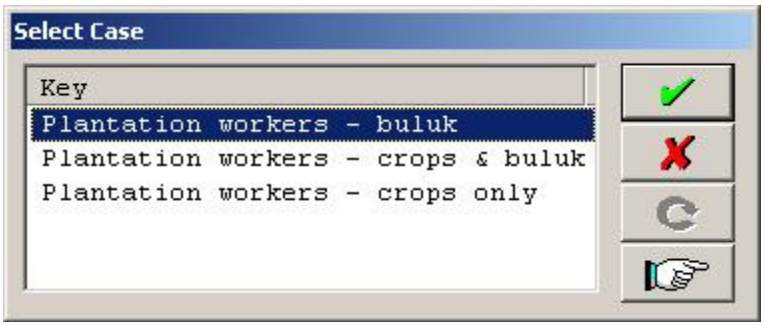
"*where*" applies the logical expression to all cases returned by the **selcase** statement. The resulting display box will only show cases in which the logical expression evaluated to true (returning a nonzero value).

Return value:
The function returns a logical value of **true** (1) if a case is found or selected by the entry operator and **false** (0) otherwise.

Example 1:
```
OK = selcase(LOOKUP,concat(PROV, DIST));
```

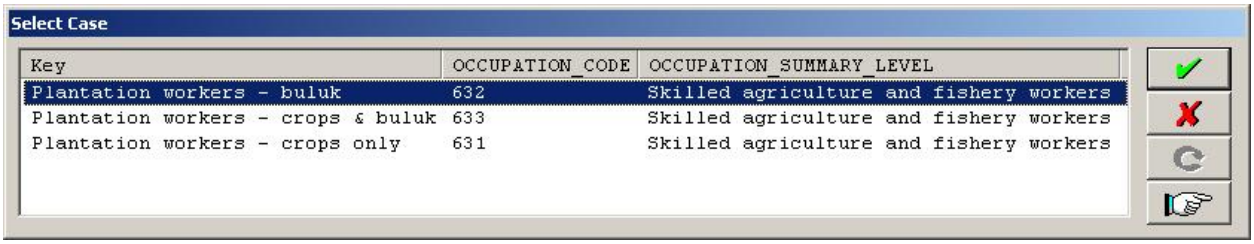
Example 2:
```
OK = selcase(OCCUPATION_DICT, "Plantation");
```

will return cases whose key begins with "Plantation."



Example 3:

```
OK = selcase(OCCUPATION_DICT, "Plantation")
     include(OCCUPATION_CODE, OCCUPATION_SUMMARY_LEVEL);
```



Example 4:

```
OK = selcase(OCCUPATION_DICT, "")
     include(OCCUPATION_CODE, OCCUPATION_SUMMARY_LEVEL)
     where OCCUPATION_CODE >= 631 and OCCUPATION_CODE <= 633;
```

will result in the same screen as appeared in Example 3.

Set Attributes Statement

Format 1:

```
set attributes (field-1[, field-2, ..., field-N])
    display | visible | autoskip | return | protect | hidden | native;
```

Format 2:

```
set attributes(field-1[, field-2, ..., field-N])
  assisted on|off[(question, responses)];
```

[ ] indicates that this part is optional
| indicates that one of the attributes may be selected

Description of Format 1:

This **set attributes** statement switches the values of various field properties. Field properties can be set statically, via the field properties dialog box, or dynamically at run time via the set attributes command. A dynamically set field property will override any statically set property. Field properties can be set dynamically anywhere in the program **except** in the `PROC GLOBAL` section.

One or more dictionary items can be named in the field list. If the dictionary name is used, all the fields in the dictionary are affected. If a form name is used, all the fields on the form are affected.

In Format 1, only one attribute setting can be used in any single set attributes statement. The options are as follows:

`visible`	If a field is hidden, its value will now be visible; if it was already visible, the setting has no effect
`autoskip`	This is equivalent to leaving the statically-set field property "Use Enter Key" unchecked. If this option is used, the cursor automatically advances to the next field, after the maximum number of characters have been entered. This option will override any statically-set field property settings.
`return`	This is equivalent to checking the statically-set field property "Use Enter Key." If this option is used, the operator must press the <Enter> key to advance from the listed field(s). This option will override any statically-set field property settings.
`protect`	This is identical to the statically-set field property "protected." If a field is set to 'protect', the operator will not be able to enter it. If the field was already statically set to "protected," the setting has no effect.
`hidden`	If a field is visible, its value will now be hidden from view; if it was already hidden, the setting has no effect.
`native`	Regardless of what settings have been made dynamically in the program, if a field is set to native, all field settings will revert to their initial, statically-set properties.

Example of Format 1:

```
set attributes (total_HH_income) protect;
```

Description of Format 2:

The set attributes statement with the assisted keyword switches on or off a pop-up responses box during data entry. The values in the responses box come from the first value set in the data dictionary for that field. The user can either select a response or type a response. This behavior is true in any CSPro data entry application.

One or more dictionary items can be named in the field list. If the dictionary name is used, all the fields in the dictionary are affected. If a form name is used, all the fields on the form are affected.

By default, the responses are taken from the first value set of the item. You can modify the values and responses using the function setvalueset.

Description of Format 2 in CAPI applications

This statement has further meaning in CAPI data entry applications. Note that when you create a CAPI data entry application, the question text for each item is automatically shown during data entry, but NOT the responses box. There are two ways to make the responses box appear (or disappear):

1. Use this statement in the application's logic. For example:

```
set attributes(MYDICT) assisted on;
set attributes(MYDICT) assisted off;
set attributes(REL, SEX, EDUC) assisted on;
```

2. Have the operator turn the response box on and off:

To show or hide responses for **this field**, press **Ctrl+C** or from the **Options** menu, select **Show Responses (This Field)**.

To show or hide responses for **all fields**, press **Ctrl+K** or from the **Options** menu, select **Show Responses (All Fields).**

The operator can also move the response box around on the screen using either the mouse or the keyboard. To move the responses dialog box with the keyboard, press **Ctrl+F8** one or more times. Each time you press **Ctrl+F8** the box will move to different position.

Use the following forms of this command to get the desired behavior in your program logic for CAPI applications:

```
// show both questions and responses
set attributes(MYDICT) assisted on;
set attributes(MYDICT) assisted on (question, responses);

// show question text but not responses
set attributes(MYDICT) assisted on (question);
< no set attributes command >

// show responses but not question text
set attributes(MYDICT) assisted on (responses);
```

See also: Change Field Properties, Change Data Entry Options, Introduction to CAPI

Set Behavior Canenter Statement

Format 1:

```
set behavior() canenter(notappl|outofrange) on (confirm|noconfirm);
```

Format 2:

```
set behavior() canenter(notappl|outofrange) off;
```

Description:

The **set behavior canenter** statement allows the entry of blanks (notappl) for numeric data items during data entry or to bypass the system 'Out of Range' message during data entry. You may wish to enter blanks when answers are missing from the form. You may wish to bypass the system 'Out of Range' message in order to code your own message. The set behavior statement affects all numeric data items from the point where it is executed onward. To limit its scope, it must be turned on and off at appropriate times.

In operator-controlled applications, notappl defined as a value in the value set for the item usually allows blank to be accepted. In system-controlled applications the Set Behavior function must be used to allow blanks even if notappl is in the value set.

The keywords **confirm** or **noconfirm** must be coded when **on** is used. Confirm means that a message box is displayed asking if it OK to enter this value. Noconfirm means that no message box is displayed.

Example:

```
set behavior() canenter(notappl) on (noconfirm);
```

See also: Special Values, Invalueset

Setcapturepos Function

Format:

```
i = setcapturepos(dictionary-symbol,x-coord-as-numeric-expression,y-coord-as-numeric-expression);
```

Description:

The **setcapturepos** function modifies the capture position for the extended control associated with a field on a form. The dictionary symbol may be an item, group, form, or an entire dictionary. The two *numeric-expressions* refer to the x (horizontal) and y (vertical) positions of the top-left corner of the extended control window. This position is relative to the form window, not the whole CSEntry window, with (0,0) referring to the top-left corner of the form. If the position given for a field is greater than the size of the form, CSEntry will ignore the parameter when displaying the extended control window. This function is useful for CAPI applications in which a part of the screen is left blank specifically for extended control windows. Calls to this function are ignored when executed on Android devices.

Return value:

The function returns the number of items whose capture positions were successfully changed.

Example:

```
setcapturepos(CAPI_DICT,500,20); // draw all windows on the right side of the form
```

See also: Extended Controls, Setcapturetype Function, Change Form Properties

Setcapturetype Function

Format:

```
i = setcapturetype(dictionary-symbol,numeric-expression[,alpha-expression]);
```

Description:

The **setcapturetype** function modifies the capture type currently associated with a field on a form. The dictionary symbol may be an item, group, form, or an entire dictionary. The *numeric-expression* refers to a capture type code, listed in the table below. If specifying a date, the optional *alpha-expression* specifies the date format.

0	Textbox
1	Radio Button
2	Checkbox
3	Drop Down Box
4	Combo Box
5	Date Picker
6	Number Pad

Return value:

The function returns the number of items that were successfully changed to the specified capture type. An item's capture type can be successfully changed if the current value set associated with the item supports the requested capture type. For instance, if an item's value set contains ranges, the capture type cannot be changed to a radio button. The success of changing a field to a date picker does not depend on the date format, so if the date format is not valid for the field the capture type will be changed to the default date format for the item.

Example:

```
setcapturetype(YOB,5,"YYYYMMDD");
errmsg("%d fields changed on the housing
form",setcapturetype(HOUSING_FORM,1));
```

See also: Extended Controls, Getcapturetype Function, Setcapturepos Function

Set Errmsg Statement

Format 1:
set errmsg(**default** | **system**);

Format 2:
set errmsg(**operator**[,*alpha_expr*,*numeric-expr*]);

Description:
CSPro data entry applications have two operating modes: operator controlled and system controlled modes. Each mode has a different style error message box. System controlled mode uses a standard Windows dialog box whereas operator controlled mode uses a customized yellow box. The **set errmsg** statement allows the user to choose which kind of box to use.

For operator controlled message boxes, the statement can modify the text that appears beneath the error message, which by default is "Press F8 to clear." Another optional parameter modifies the keystroke required to close the box.

Calls to this function are ignored when executed on Android devices.

Examples:

```
set errmsg(system);    // use system controlled style
set errmsg(operator);  // use operator controlled style
set errmsg(default);   // use the default style for the operating mode

// defines the error message and defaults to F8 to close the box
set errmsg(operator,"Appuyez sur F8 pour fermer");

// 67 (C) is the key that will close the box
set errmsg(operator,"Appuyez sur C pour fermer",67);

// 67 (C) is the key that will close the box, the default error message text
is used
set errmsg(operator,67);
```

See also: Setfont Function

Setfont Function

Format 1:
b = **setfont**(**ErrMsg** | **ValueSets** | **UserBar** | **Notes** | **All**,*alpha_expr*[,**bold**,**italics**]);

Format 2:
b = **setfont**(**ErrMsg** | **ValueSets** | **UserBar** | **Notes** | **All**,**default**);

Description:

In a data entry application, the **setfont** function allows you to modify the font that CSPro uses to display text in:

- **ErrMsg**: The font that appears in boxes generated by the errmsg function
- **ValueSets**: The font that appears when using CAPI mode with extended controls
- **UserBar**: The font that appears in buttons and text strings on the userbar
- **Notes**: The font that appears when a user edits a field's note
- **All**: The font is changed for all of the four above entities

The font is changed to the font indicated by the *alpha_expr*, or if format 2 is used, the font is restored to CSPro's default font selection. The user must ensure that the font is installed on the machine that will run the data entry application. Optional bold and italics markers may be indicated. Calls to this function are ignored when executed on Android devices.

Return value:

The function returns 1 if the font(s) were changed successfully, 0 otherwise.

Example:

```
function majorError(string message)

    setfont(ErrMsg,"Arial",24,bold,italics);
    errmsg(message);

end;

function minorError(string message)

    setfont(ErrMsg,"Arial",12,bold);
    errmsg(message);

end;

function minorWarning(string message)

    setfont(ErrMsg,"Arial",12,italics);
    errmsg(message);

end;


PROC AGE

    if AGE > 95 then
        minorWarning(maketext("Age (%d) is over 95, set to 95",AGE));
        AGE = 95;

    elseif not AGE in 12:95 then
        minorError(maketext("Age (%d) is invalid for this survey,
reenter",AGE));
        reenter;
    endif;
```

See also: Set Errmsg Statement, Setvaluesets Function

Setlanguage Function

Format:

```
b = setlanguage(alpha-expression);
```

Description:

The **setlanguage** function changes the language mode being used by the CAPI interface. Pass the function the name, not label, of the language to which you want to change.

Return value:

The **setlanguage** function returns 1 if the language exists and the language mode was successfully be changed. Otherwise it returns 0.

Example:

```
if X = 1 then
     setlanguage("FRA");
elseif X = 2 then
     setlanguage("GER");
else
     setlanguage("ENG");
endif;
```

See also: Gettlanguage Function, Define Languages

Setvalueset Function

Format 1:

```
setvalueset(item-name | @alpha_expr, valueset-name);
```

Format 2:

```
setvalueset(item-name | @alpha_expr, numeric-array | alpha-array, alpha-
array);
```

Description:

The **setvalueset** function allows you to dynamically change the first value set of an item. The first value set of an item is used to determine whether or not inputted values are out of range. The first value set is also used to provide the value choices displayed in the pop-up menu when assisted mode is enabled using set attributes. The **setvalueset** function therefore allows you to programmatically change the value set used in these cases. The change to the value set is not permanent; it remains in effect only during the current execution of the program.

The *item-name* is the name of the item in the data dictionary whose value set is to be changed. This may also be an alpha expression that evaluates to the name of an item preceded by the @ symbol, for example:

```
setvalueset(@getsymbol(), MY_VALUESET_VS1)
```

In the first format, *valueset-name* is the name of an existing value set in the dictionary for the specified item. The **setvalueset** function will replace the values of the first value set for the item with the values from the value set specified by *valueset-name*.

In the second format, the second parameter is an array of values to use in the new value set and the third parameter is an array of labels that correspond to the values in the first array (the nth element of the labels array is the label for the nth element in the values array). The last element in the array of values must be the special value **notappl**. This indicates to the system when to stop loading values from the array.

For a detailed example, see the **setvalueset** example in the CSPro examples directory.

In versions of CSPro earlier than 6.0, the value set labels and codes array needed to start at index position 0. Zero-based arrays are still permitted, but if a one-based array is passed to the function (and the zeroth element has never been assigned to), then CSPro will start processing the dynamic value set at array position 1.

Example 1:

```
PROC DISTRICT
preproc
  if STATE = 1 then
    setvalueset(DISTRICT, DISTRICT_VS1);
  elseif STATE = 2 then
    setvalueset(DISTRICT, DISTRICT_VS2);
  elseif STATE = 3 then
    setvalueset(DISTRICT, DISTRICT_VS3);
  endif;
  set attributes(DISTRICT) assisted on (responses);
```

Example 2:

```
PROC GLOBAL
  string codes(4);
  array string labels(4);

PROC DISTRICT
preproc
  codes(1) = 1;
  codes(2) = 2;
  codes(3) = 3;
  codes(4) = notappl; // mark end of array
  labels(1) = "District 1";
  labels(2) = "District 2";
  labels(3) = "District 3";

  setvalueset(DISTRICT, codes, labels);
  set attributes(DISTRICT) assisted on (responses);
```

See also: Setvaluesets Function, Invalueset Function

Setvaluesets Function

Format:

```
i = setvaluesets(alpha_expr);
```

Description:

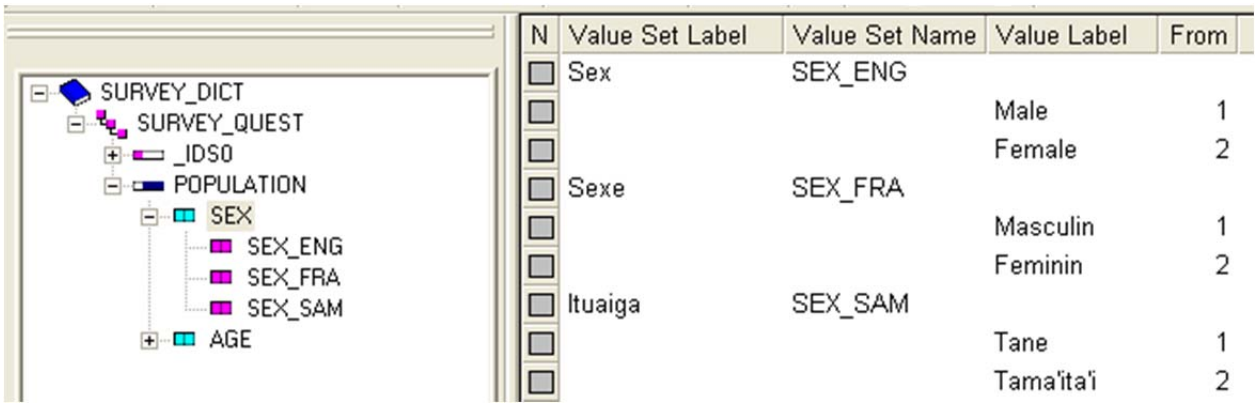
The **setvaluesets** function allows you to dynamically change the value sets associated with multiple items. Unlike the setvalueset function, which only operates on one item, the **setvaluesets** function traverses every item in the primary dictionary and searches for the *alpha_expr* in the names of each item's value sets. If the expression is found, the first value set with the name containing the expression is the value set used for the item. Otherwise the currently used value set is maintained.

This function can be useful for changing the language of value sets in a multi-language application but requires care when creating names for the value sets.

Return value:
The function returns a value containing the number of items whose value sets were changed.

Example 1:

```
if      language = 1  then  setvaluesets("_ENG");  // English
elseif  language = 2  then  setvaluesets("_FRA");  // French
else                  then  setvaluesets("_SAM");  // Samoan
endif;
```



Example 2:

```
function resetLanguage(language)

    if language = 1 then // English
        setvaluesets("_ENG");
        setfont(valuesets,default);

    elseif language = 2 then // Russian
        setvaluesets("_RUS");
        setfont(valuesets,"Cyrillic",12);

    else // Tajik
        setvaluesets("_TAJ");
        setfont(valuesets,"Tajik",12);

    endif;

end;
```

See also: Setvalueset Function, Invalueset Function, Setfont Function

Show Function

Format:

```
i = show(group-name, item-list, [where condition], [title(text-list)]);
```

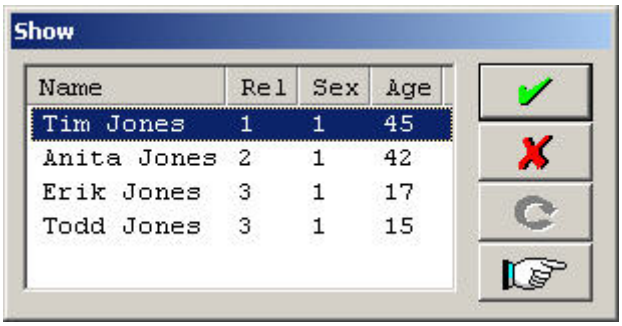
Description:

The **show** function displays items from a roster in the form or a menu that looks like a roster. This function is similar to the Accept function. This function is useful as a menu or simply as a way to show roster values in another part of the questionnaire.

Return value:
The function returns the number of the item selected: 1 for the first item, 2 for the second item, etc. This is the number of the item on the display, not in the roster. The value 0 is returned if the Esc key is pressed and none of the options is chosen.

Example 1

```
if RELATIONSHIP = 2 then
  if SEX = SEX(ptrHead) then
    errmsg("Sex of spouse is the same as the sex of head!");
    i = SHOW(PERSON_REC, NAME, RELATIONSHIP, SEX, AGE,
          TITLE("Name", "Rel", "Sex", "Age"));
  endif;
endif;
```



Example 2

```
if ctrHead > 1 then {More than 1 head in the household}
  errmsg("More than 1 head of Household (Count of Heads=%d)", ctrHead);
  i = SHOW(PERSON_REC, SEQUENCE, RELATIONSHIP, SEX, AGE,
        TITLE("Seq Number", "Rel", "Sex", "Age"));
endif;
```

Example 3

```
if ctrHead > 1 then {More than 1 head in the household}
  errmsg("More than 1 head of Household (Count of Heads=%d)", ctrHead);
  i = SHOW(PERSON_REC, SEQUENCE, RELATIONSHIP, SEX, AGE,
        WHERE RELATIONSHIP = 1 TITLE("Seq Number", "Rel", "Sex", "Age"));
endif;
```

Note that in Example 3 the WHERE clause restricts the display to rosters for Heads of Household only. This means that there will NOT be a correspondence between the occurrence number (Curocc) and the return value. The number returned is the number of the item selected from the display as shown in the display.

See also: Accept Function, Showarray Function

Showarray Function

Format:

```
i = showarray(array-name, [row-count], [column-count], [title(text-list)]);
```

Description:
The **showarray** function, similar to the show and accept functions, displays items from an array in a grid. This function is useful as a menu or simply as a way to show values that are relevant to data

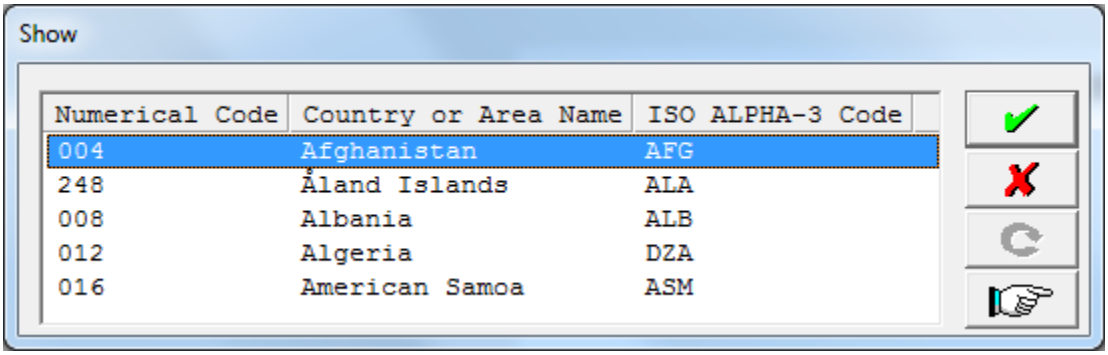
collection. If the row and column counts are not supplied, then the function will parse the array, determining the size of the menu based on where the first blank alphanumeric element is found.

Return value:

The function returns the number of the item selected: 1 for the first item, 2 for the second item, etc. The value 0 is returned if the Esc key is pressed and none of the options is chosen.

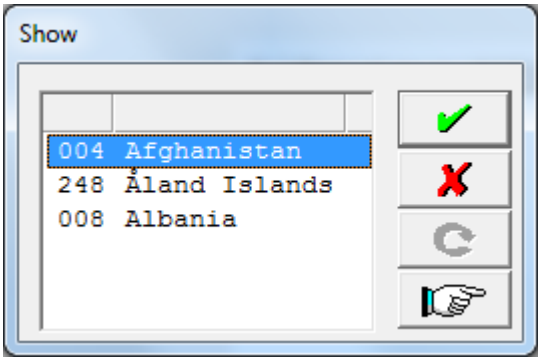
Example 1:

```
PROC GLOBAL

array string unitedNationsCountryCodes(5,3) =
    "004"   "Afghanistan"       "AFG"
    "248"   "Åland Islands"     "ALA"
    "008"   "Albania"           "ALB"
    "012"   "Algeria"           "DZA"
    "016"   "American Samoa"    "ASM"
    // ...
    ;

PROC EXAMPLE

    numeric countrySelection = showarray(unitedNationsCountryCodes,
                title("Numerical Code","Country or Area Name","ISO ALPHA-3
Code"));
```



Example 2:

```
    numeric countrySelection = showarray(unitedNationsCountryCodes,3,2);
```



See also: Accept Function, Show Function

Skip Statement

Format 1:
```
skip [to [next]] field-name;
```

Format 2:
```
skip [to [next]] alpha-variable;
```

Format 3:
```
skip [to] next;
```

[ ] indicates that this part is optional.

Description:

The **skip** statement skips to the specified field. If the field has multiple occurrences, either record or item, the occurrence number must be specified to skip to the correct occurrence.

The "next" keyword skips to the next occurrence of field-name where "field-name" is a multiple-occurrence field. If "field-name" is in the same record or group as the current field, control will move to the next occurrence of "field-name". If "field-name" is not in the same record or group as the current field, control will move to the first occurrence of "field-name". Occurrence numbers cannot be used with the **next** keyword.

If the "field-name" is not specified after "next" (format 3), control will move to the next occurrence of the first item in the group. This is a useful way to skip to the beginning of the next occurrence.

"Field-name" can be located in any record at the same level as the current field, but it cannot be located at a different level. "Field-name" must be the name of a field that has not yet been entered. If "field-name" has already been entered, an error message will be displayed during data entry and data entry will be aborted. If you don't know whether the field has already been entered, use the move statement.

The field name can be given directly by naming the field or indirectly by using the contents of an alpha variable.

When a **skip** statement is executed, the **preproc** of field-name, if any, will be executed. None of the statements between the **skip** statement and the **preproc** of "field-name" will be executed. Skipped fields are assigned the special value of NOTAPPL.

Note that the skip statement behaves differently from the advance statement.

Example 1:
```
if Q305 <> 2 then
  skip to Q307;
endif;
```

Example 2:
```
if Q202 <> 1 then
  skip to next Q201;
endif;
```

See also: Move Statement, Reenter Statement, Advance Statement

Userbar Function

It is possible to add a bar at the top of a data entry application to facilitate certain kinds of communication between a user and the intentions of the application programmer. For instance, a button can be added

that allows the user to jump to a certain section of a questionnaire, or the bar can display reference text useful for the user. This "userbar" can display text, buttons, and text fields.

General Userbar Commands

```
b = userbar(show | hide | clear);
```

Show adds a userbar to the application, or if one has been added and is hidden, the show command displays it again. The **hide** command removes the userbar from the data entry window but keeps the userbar's contents in case it is shown again later. The **clear** command removes the userbar from the data entry window and deletes the contents of the userbar. The function returns 1 if successful, 0 otherwise. If the first command to a userbar is an add command, the userbar will be automatically displayed. If you want to add items to the userbar without displaying it, call hide or clear before adding the items.

Adding Items to the Userbar

```
i = userbar(add text | button | field | spacing,[...]);
```

All of the add commands, detailed below, return a resource identifier, an integer that uniquely points to the added object. If you plan to modify or remove objects added to the userbar, it is necessary to maintain this resource identifier as a way to identify on what object the modify or remove command should work. Objects are added to the userbar in left-to-right format. There is no way to add an item in between two existing items.

Adding Text to the Userbar

```
i = userbar(add text,alpha-expression);
```

The *alpha-expression* is a string of text that will be displayed on the userbar. The color of the text can be modified using the set color command.

Adding Buttons to the Userbar

```
i = userbar(add button,alpha-expression[,function-name]);
```

The button text comes from the *alpha-expression*. An optional parameter specifies a function that is to be called when the button is pressed. If the function has parameters, it is necessary to specify the values of the parameters. The values of these parameters are evaluated when the button is activated by a user, not at the point that the button is added to the userbar. Several existing functions (mimicking options available on the CSEntry menu) can be specified using the do command:

```
i = userbar(add button,alpha-expression,do(alpha-expression));
```

The *alpha-expression* within the **do** command can be one of the following values:

"NextField", "PreviousField", "AdvanceToEnd", "EditNote", "ChangeLanguage", "PartialSave", "FieldHelp", "InsertLevelOcc", "AddLevelOcc", "DeleteLevelOcc", "InsertGroupOcc", "InsertGroupOccAfter", "DeleteGroupOcc", "SortGroupOcc", "PreviousScreen", "NextScreen", "EndGroupOcc", "EndGroup", "EndLevelOcc", "EndLevel", "FullScreen", "ToggleResponses", or "ToggleAllResponses"

Adding Fields to the Userbar

```
i = userbar(add field,alpha-expression[,function-name]);
```

The field's initial text comes from the *alpha-expression*. As with adding buttons, an optional parameter specifies a function that is called when the user presses enter while typing text in the field. The width of the field depends on the initial text, so if you want a large field that starts with a blank value, use a long but blank string to initialize the field. Modifying the text of the field will not affect its size.

Adding Spacing to the Userbar

```
i = userbar(add spacing,numeric-expression);
```

If you want to space out the items on the userbar, specify in *numeric-expression* the number of pixels to leave blank between the last-to-be and next-to-be added items.

Modifying Items on the Userbar

```
b = userbar(modify,resource-identifier,alpha-expression); // for text,
buttons, and fields
b = userbar(modify,resource-identifier,function-name); // for buttons and
fields
b = userbar(modify,resource-identifier,alpha-expression,function-name); //
for buttons and fields
b = userbar(modify,resource-identifier,numeric-expression); // for spacing
```

To modify an item on the userbar, use the resource identifier that the function returned when the item was added to the userbar and specify the changes to the item. For buttons and fields, the text of the item and/or the function can be modified. The function returns 1 if successful, 0 otherwise.

Removing Items from the Userbar

```
b = userbar(remove,resource-identifier);
```

The remove command takes the item pointed to by the resource identifier off the userbar. The function returns 1 if successful, 0 otherwise.

Changing the Color of Items

```
b = userbar(set color[,resource-identifier],red-value,green-value,blue-
value);
```

The background color of the userbar is changed by specifying the RGB (0 to 255) values of the desired color. Alternatively, if a resource identifier of a text item is passed, the color of the text item is changed. The function returns 1 if successful, 0 otherwise.

Identifying the Source of an Action

```
i = userbar(get);
```

The get command returns the resource identifier of the last clicked button or the last userbar field in which data was entered. If no event has occurred, the function returns 0.

Obtaining the Contents of a Field

```
b = userbar(get,resource-identifier,alpha-variable);
```

In this format, the get command returns the contents of a userbar field identified by the resource identifier. *Alpha-variable* must be an alphanumeric variable declared in the PROC GLOBAL section of the code. The function returns 1 if successful, 0 otherwise.

Userbar on Android Devices

The userbar shows up as an icon to the left of the field note icon. Only buttons are displayed on the userbar so any color changes or additions of fields or spacing will be ignored. Options that mimic the CSEntry menu, added via the do(...) command, will also be ignored.

Example

```
PROC GLOBAL

numeric ADD_OP = 1, SUBTRACT_OP = 2, MULT_OP = 3, DIVIDE_OP = 4, MOD_OP = 5,
EXP_OP = 6;

numeric leftOperatorRID,rightOperatorRID,resultsRID;

alpha (30) leftOperator,rightOperator;
alpha (1) operatorTypeString;

function performMathOperation(operationType)

    userbar(get,leftOperatorRID,leftOperator);
    userbar(get,rightOperatorRID,rightOperator);

    numeric leftNumber = tonumber(leftOperator);
    numeric rightNumber = tonumber(rightOperator);

    if countnonspecial(leftNumber,rightNumber) <> 2 or leftOperator = "" or
rightOperator = "" then
        userbar(modify,resultsRID,"The supplied values are invalid!");
        userbar(set color,255,0,0);

    elseif rightNumber = 0 and operationType in DIVIDE_OP,MOD_OP then
        userbar(modify,resultsRID,"0 cannot be a denominator in a divide or
modulo expression!");
        userbar(set color,255,0,0);

    else
        numeric result;

        if      operationType = ADD_OP      then result = leftNumber +
rightNumber; operatorTypeString = '+';
        elseif  operationType = SUBTRACT_OP then result = leftNumber -
rightNumber; operatorTypeString = '-';
        elseif  operationType = MULT_OP     then result = leftNumber *
rightNumber; operatorTypeString = '*';
        elseif  operationType = DIVIDE_OP   then result = leftNumber /
rightNumber; operatorTypeString = '/';
        elseif  operationType = MOD_OP      then result = leftNumber %
rightNumber; operatorTypeString = '%';
```

```
        elseif  operationType = EXP_OP      then result = leftNumber ^
rightNumber; operatorTypeString = '^';
        endif;

        userbar(modify,resultsRID,maketext("%0.2f %s %0.2f =
%0.2f",leftNumber,operatorTypeString,rightNumber,result));
        userbar(set color,0,255,0);

    endif;

end;


PROC USERBAR_FF

preproc

    userbar(clear);
    userbar(add text,"Enter two values and select an operation:");
    leftOperatorRID = userbar(add field,"                    ");
    rightOperatorRID = userbar(add field,"                    ");
    userbar(add button,"Addition",performMathOperation(ADD_OP));
    userbar(add button,"Subtraction",performMathOperation(SUBTRACT_OP));
    userbar(add button,"Multiplication",performMathOperation(MULT_OP));
    userbar(add button,"Division",performMathOperation(DIVIDE_OP));
    userbar(add button,"Modulo",performMathOperation(MOD_OP));
    userbar(add button,"Exponentiation",performMathOperation(EXP_OP));
    userbar(add spacing,50);
    resultsRID = userbar(add text," ");
    userbar(show);
```

See also: Extended Controls

Visualvalue Function

Format:

```
i = visualvalue(numeric-item);
```

Description:

The **visualvalue** function is used to access the contents of a data item before the data item has been keyed. In the example below, the value of UR is being examined in the preproc of the item, that is, before any input can be attempted.

Return value:
The function returns the numeric of value of the item.

Example:
```
PROC UR
preproc
  if not visualvalue($) in 1:2 then
    do until visualvalue($) in 1:2
      $ = accept("Area Designation?", "Urban", "Rural");
    enddo;
    noinput;
  endif;
```

See also: Accept Function, If Statement, Do Statement, Noinput Statement

Batch Edit Statements

Endcase Statement

Format:
```
endcase;
```

Description:
The **endcase** statement ends batch editing for the current questionnaire (case). All remaining procedures beyond where the statement is executed will be skipped. The statement is similar to the **skip case** command, but in the latter command the case is not saved to the output file. When using **endcase** the case will be saved to the output file.

Example:
```
if HHTYPE = 2 then
  endcase;
endif;
```

See also: Skip Case Statement, Stop Function, Exit Statement

Export Statement

Format:
```
Export to file-name
    [rec_name(rec-name | alpha-exp)]
    [rec_type(rec-name | alpha-exp)]
    [case_id ([item-list])]
    rec-item-list
```

Description:
The **export** statement writes a record to an export file. Export statements can only be coded in level procedures.

In the **to** phase, the *file-name* is a name declared in the **file** statement in PROC GLOBAL.

The **rec_name**, **rec_type**, and **case_id** phrases can each be coded only once but can be coded in any order. They all must be coded before the *rec-item-list*. The order in which **rec_type** and **case_id** are coded determines the order of output of the record type and case ids in the exported record.

The **rec_name** phrase is optional and only used when data are exported in CSPro format. When coded the **rec_name** phase is used to give a label and name to the record type to the CSPro data dictionary created by the export statement. If a *rec-name* is coded, then the label and name from that record name in the input data dictionary is used for the label and name of the record type created in the exported data dictionary. If an *alpha-exp* is coded, then the label of the record type in the exported data dictionary is the result of the alphanumeric expression and the name is derived from the label. If **rec_name** is not coded, the labels and record names in CSPro will be RECORD001, RECORD002, etc.

The **rec_type** phrase is optional. When coded it is used to place a record type on the exported data record. If a *rec-name* is coded, then the record type value from the record name in the input data dictionary is placed on the exported data file. If an *alpha-exp* is coded, then the value of the expression is placed on the exported data file.

The **case_id** phrase is optional. When coded it is used to place case id items on the exported data record. If **case_id()** is coded then ALL the case ids from the level in which the export statement is coded are placed on the exported data record. If no **case_id** phrase is coded and export format is CSPro, the ALL case ids from ALL levels will be output.

The *rec-item-list* specifies the contents of the exported data record. This can be any combination of record names or item names.

Where possible users are encouraged to use the **Export Data** tool instead of the **export** statement.

Example:

```
{Export fertility data for women 15 to 54 years old}
PROC GLOBAL
numeric i;
file SPSS_EXPORT;

PROC CENSUS_2000_DICTIONARY_FF

PROC QUEST {export done at LEVEL procedure}
  set behavior() export ( SPSS , ItemOnly );

  do I = 1 until I > totocc(PERSON)
    if P03_SEX (i) = 2 and P04_AGE(i) in 15:54 then
      export to SPSS_EXPORT case_id (PROVINCE,DISTRICT,EA,HU,HH)
          LINE(i), P18_BORN(i), P19_LIVING(i), P20_BORN12(i);
      errmsg ("Record exported for female, 15 to 54") summary;
    endif;
  enddo;
```

See also: Set Behavior Export Statement, File Statement, SetFile Function, Export Data Tool

Getdeck Function

Format:

```
f = getdeck(array-name[,override-dim1,override-dim2,override-dim3]);
```

Description:

The **getdeck** function returns the value in the DeckArray hotdeck using the current values in the items identified by the value sets used in the declaration of the DeckArray. The function automatically recodes the values and accesses the proper cell in the hotdeck. If any of the dimensions of the DeckArray are not value set dimensions, you must specify the numeric index when calling the function.

Return value:
The function returns the value in the hotdeck or DEFAULT in the case that the values supplied are not valid entries in the value sets and thus could not be recoded to a proper cell in the hotdeck.

Example:
```
array education_HD_SexAge(SEX_VS,AGE_FOR_EDUCATION_HD_VS) save;

...

PROC EDUCATION

    EDUCATION = getdeck(education_HD_SexAge); // use current values for sex
and age
    EDUCATION = getdeck(education_HD_SexAge,1); // override sex only
    EDUCATION = getdeck(education_HD_SexAge,,28); // override age only
    EDUCATION = getdeck(education_HD_SexAge,1,28); // override both value
and age
```

See also: DeckArrays, DeckArray Leftover Rows, Putdeck Function

Putdeck Function

Format:
f = **putdeck**(*array-name*,*numeric-expression*[,*override-dim1*,*override-dim2*,*override-dim3*]);

Description:
The **putdeck** function updates the value in the DeckArray hotdeck using the current values in the items identified by the value sets used in the declaration of the DeckArray. The function automatically recodes the values and accesses the proper cell in the hotdeck, where it places the value of the *numeric-expression*. If any of the dimensions of the DeckArray are not value set dimensions, you must specify the numeric index when calling the function.

If a (+) is specified after the *array-name*, the "leftover" rows for the hotdeck will also be updated with the value. See the leftover rows page for more information.

Return value:
The function returns 1 if successful or DEFAULT in the case that the values supplied are not valid entries in the value sets and thus could not be recoded to a proper cell in the hotdeck.

Example:
```
array education_HD_SexAge(SEX_VS,AGE_FOR_EDUCATION_HD_VS) save;

...

PROC EDUCATION

    putdeck(education_HD_SexAge,EDUCATION); // use current values for sex
and age
    putdeck(education_HD_SexAge,EDUCATION,1); // override sex only
    putdeck(education_HD_SexAge,EDUCATION,,28); // override age only
    putdeck(education_HD_SexAge,EDUCATION,1,28); // override both value and
age
```

See also: DeckArrays, DeckArray Leftover Rows, Getdeck Function

Set Behavior Export Statement

Format:

```
set behavior() export (model[, item-type, text-encoding, decimal-mark]);
```

Description:

The **set behavior export** statement is coded before the first **export** statement.

The *model* is required. It is one of the keywords **SPSS**, **SAS**, **Stata**, **R**, **All**, **CSPro**, **TabDelim**, **CommaDelim**, or **SemiColonDelim**, indicating the type of file being exported.

Item-type is optional. It is one of the keywords **ItemOnly**, **SubitemOnly**, or **ItemSubitem**, indicating how subitems and their parent item are handled when entire records are exported. If not coded, **SubitemOnly** is assumed.

Text-encoding is optional. It is either **ANSI** or **Unicode**, and specifies the text encoding of the exported data file and any description or script files. If not coded, **ANSI** is assumed. The **Unicode** option will output UTF-8 files.

Decimal-mark is optional. If specified with the keyword **CommaDecimal**, decimal marks will use commas instead of periods in the exported data file.

See also: Export Statement

Setoutput Function

Format:

```
b = setoutput(alpha-exp);
```

Description:

The **setoutput** function redirects the output cases of a batch application to the data file specified in the *alpha-exp*. All data will be appended to (added to the end of) the file. If the file exists prior to the application run, you may, depending on the circumstances, want to use the filedelete function to remove the file. This function can not be used in data entry applications.

Alpha-exp is an alphanumeric expression containing the folder and file name of the file to be attached to the batch output.

Return value:

The function returns a logical value of 1 (true) if the physical file is successfully assigned and 0 (false) otherwise. If the new file cannot be created or opened, the batch application will terminate.

Example:

```
PROC PROVINCE

    // this will split the contents of the input data file into
    // one file for each province code
    setoutput(maketext(".\split\%02d.dat",PROVINCE));
```

See also: Setfile Function, Skip Case Statement

Skip Case Statement

Format:

```
skip case;
```

Description:
The **skip case** statement ends batch edit processing of the current case and skips to the next case in the input file. If an output file is specified, the skipped case is **not** placed in the output file.

Example:
```
if totocc(HOUSING) > 1 then
  errmsg("Too many housing records");
  skip case;
endif;
```

See also: Endcase Statement, Stop Function, Exit Statement, Errmsg Function, If Statement, Totocc Function

Numeric Functions

Abs Function

Format:
```
d = abs(numeric-expression);
```

Description:
The **abs** function returns the absolute value of a numeric expression.

Return value:
The function returns the absolute value. If the value of the numeric expression is a special value (e.g., MISSING, NOTAPPL, or DEFAULT), the function returns the special value.

Example:
```
if abs(X - target) < abs(Y - target) then
    // ...
endif;
```

Cmcode Function

Format:
```
i = cmcode(month,year);
```

Description:
The **cmcode** function returns the century month code of the given date by the **month** and **year** parameters. The "century month code" is the number of months since January 1900 (the century month code for January 1900 = 1). It is calculated by multiplying the number of years between the parameter **year** and 1900 by twelve, then adding the value of parameter **month**.

The **cmcode** function returns the value 9999 if the month is less than one or greater than 12, or if either the month or year are equal to any of the special values DEFAULT, MISSING, or NOTAPPL. **Cmcode** will accept either 2- or 4-digit years. If a 2-digit year is used, the **cmcode** function assumes that the year is in the 20th (i.e., 19xx) century. Four-digit years can be used for years in the 20th or 21st century.

Return value:
The function returns the number of months as an integer.

Example 1:

```
XMONTH = 06;
XYEAR = 81;
DATE = cmcode(XMONTH,XYEAR);
```

The value of DATE for the given parameters [June 1981], would be (81 x 12) + 6 = 978.

Example 2:

```
XMONTH = 2;
XYEAR = 2000;
DATE = cmcode(XMONTH,XYEAR);
```

The value of **DATE** in this example would be ((2000 – 1900) * 12) + 2, or 1202.

Countnonspecial Function

Format:

i = **countnonspecial**(*item* | *record* | *array* | *numeric-expression*);

Description:

The function **countnonspecial** counts the number of non-special values within the passed parameters. Special parameters include **missing**, **notappl**, and **default**. The function can receive multiple numeric parameters, including items, records, arrays, and numeric expressions. This function can greatly simplify some programming tasks; for example, the following two lines of code are the same:

```
if countnonspecial(RELATIONSHIP,SEX,AGE) = 3 then
if not special(RELATIONSHIP) and not special(SEX) and not special(AGE) then
```

If an array is passed, the function will count the non-special values among all the values in the array. For a multiply occurring item or record, to count the non-special values for all the items or records instead of just the current occurrence, include **(*)** after the item or record name. **Countvalid** is a synonym for **countnonspecial.**

Example 1:

```
numeric tempVal = notappl;
array tempArray(2,3) = 1 2 3 notappl missing default;

countnonspecial(5,tempVal,default,28 + 31,3 / 0); // returns 2

tempVal = 0.123456789;
countnonspecial(5,tempVal,default,28 + 31,3 / 0); // returns 3

countnonspecial(5,tempVal,default,28 + 31,3 / 0,tempArray); // returns 6

tempArray(2,1) = 0;
countnonspecial(5,tempVal,default,28 + 31,3 / 0,tempArray); // returns 7
```

Example 2:

```
PROC SEX // in this example SEX is not on the FERTILITY record

if SEX <> 2 and countnonspecial(FERTILITY(curocc())) > 3 then
    errmsg("Sex (%d) is not female but 4+ fertility entries defined so sex
changed to female",SEX);
```

```
    impute(SEX,2);
endif;
```

Example 3:

```
numeric numDefinedValues = countnonspecial(SEX(*),AGE(*));
numeric numPossibleValues = 2 * totocc(POPULATION);

errmsg("%d% of sex and age values are missing",100 * ( 1 - (
numDefinedValues / numPossibleValues ) ));
```

See also: Special Function, Special Values

Exp Function

Format:

```
d = exp(numeric-expression);
```

Description:

The exp function raises the value of **e** (2.7182818…) to the power given by "numeric-expression". This value (**e**) is called Napier's constant or Euler's number and is the basis of natural logarithms.

Return value:

The function returns a decimal number. If the value of "numeric-expression" is a special value (MISSING, NOTAPPL, or DEFAULT), the function returns that value.

Example:

```
X = exp(Y);
```

High Function

Format:

```
d = high(numeric-expression,[...]);
```

Description:

The **high** function returns the maximum (highest) value in a group of numbers. The function ignores special values.

Return value:

The function returns the highest value, or DEFAULT in the case that no valid values were passed.

Example:

```
errmsg("Lowest value is %f",low(50,-123.45,1982.0605,20)); // displays -
123.45
errmsg("Highest value is %f",high(50,-123.45,1982.0605,20)); // displays
1982.0605
```

See also: Low Function

Inc Function

Format:

```
i = inc(numeric item-name [,numeric increment expression]);
```

Description:

The **inc** function increments a numeric item that is either a dictionary item or a numeric variable. If an increment expression is present, then the value of the expression is added to first parameter. If no expression is present, 1 is added to the first parameter. The increment expression can be negative or nonnegative. `inc(X)` is essentially shorthand for `X = X + 1;`

Return value:

The function returns the value of the increment expression added to the numeric item.

Example:

```
X = 5;
inc(X);  yields X = 6, is the same as X = X + 1;
inc(X,4);  yields X = 10, is the same as X = X + 4;
inc(X,-5);  yields X = 5, is the same as X = X + (-5);
X = 5 + inc(X,inc(X));  yields X = 17, is the same as X = X + 1; X = X + X;
X = 5 + X;
```

Int Function

Format:

```
i = int(numeric-expression);
```

Description:

The **int** function returns the integer portion of the result of the "numeric-expression".

Return value:

The function returns an integer value. If the value of "numeric-expression" is a special value (MISSING, NOTAPPL, or DEFAULT), the function returns that value.

Example:

```
x = int(5 / 3);
```

The value of x would be 1.

Log Function

Format:

```
d = log(numeric-expression);
```

Description:

The log function calculates the base-10 logarithm of "numeric-expression".

Return value:

The function returns a decimal number logarithm. If the value of "numeric-expression" is a special value (MISSING, NOTAPPL, or DEFAULT), the function returns that value. If the value of numeric-expression is negative, the special value DEFAULT is returned.

Example:

```
X = log(Y);
```

Low Function

Format:

```
d = low(numeric-expression,[...]);
```

Description:
The **low** function returns the minimum (lowest) value in a group of numbers. The function ignores special values.

Return value:
The function returns the lowest value, or DEFAULT in the case that no valid values were passed.

Example:
```
errmsg("Lowest value is %f",low(50,-123.45,1982.0605,20)); // displays -
123.45
errmsg("Highest value is %f",high(50,-123.45,1982.0605,20)); // displays
1982.0605
```

See also: High Function

Random Function

Format:
```
i = random(min-value,max-value);
```

Description:
The random function returns a uniformly distributed random integer between "min-value" and "max-value". "Min-value" and "max-value" are numeric expressions that must have integer values in the range -32767 to +32767. Use seed to initialize the random function.

Return value:
The function returns an integer random value. The function will return a value DEFAULT if "min-value" is greater than "max-value" or if either limit is equal to one of the special values DEFAULT, MISSING, and NOTAPPL.

Example:
```
NUM = random(1,100);
```

See also: Randomin Function, Randomizevs Function

Randomin Function

Format:
```
i = randomin(in-list | value-set);
```

Description:
The **randomin** function returns a uniformly distributed random integer from one of two categories:
- A grouping of non-continuous integers expressed as an in list.
- A value set.

Use seed to initialize the randomin function.

Return value:
The function returns an integer random value. The function will return a value DEFAULT if there were no applicable values to construct a table of non-continuous integers from which to pick a random number. If a value appears more than once in the in list or the value set, it will have a higher probability of being selected by the randomin function.

Example:

```
errmsg("Random tribe code: %d",randomin(TRIBE_VS1));
errmsg("Non-continuous random number: %d",randomin(-100:-50,50:100,999));
errmsg("Over time 5 will be selected 75% of the time:
%d",randomin(5,5,5,8));
errmsg("Random month: %d",randomin(1:12)); // same as random(1,12)
```

See also: Random Function, **Randomizevs Function**

Seed Function

Format:

```
b = seed(numeric-expression);
```

Description:

The seed function is used to determine the first value generated by the random function. For best results, "numeric-expression" should be set to a prime number, such as 1009.

Versions of CSPro 6.0 and greater will return different random numbers, given the same seed value, than earlier versions of CSPro. If you expect a certain sequence to the random numbers and want to match a previously recorded sequence, then you will want to use an old version of CSPro to run your application.

Return value:

The function returns a logical value "**true"** if the seeding is successful, "**false"** otherwise.

Example:

```
OK = seed(1009);
```

Sqrt Function

Format:

```
d = sqrt(numeric-expression);
```

Description:

The sqrt function returns the square root of "numeric-expression". "Numeric-expression" should be a positive value.

Return value:

The function returns a decimal value of the square root of the expression. If the value of the numeric-expression is a special value (e.g., MISSING, NOTAPPL, or DEFAULT), the function returns the special value given. If the value of "numeric-expression" is negative, the function returns the special value DEFAULT.

Example:

```
X = sqrt(Y);
X = sqrt(12);
```

Set Behavior SpecialValues Statement

Format:

```
set behavior() specialvalues(zero) on | off;
```

Description:

The **set behavior specialvalues** statement allows special values to be treated as zero (0) values during arithmetic operations, including addition, subtraction, multiplication, division, and modulo. By default this behavior is disabled. The set behavior statement affects all numeric data items from the point where it is executed onward. To limit its scope, it must be turned on and off at appropriate times.

Example:

```
PROC GLOBAL

   numeric var1,var2,var3;
   numeric result;

PROC SUMMATION

   var1 = 5;
   var2 = 10;
   var3 = default;

   set behavior() specialvalues(zero) on;
   result = var1 + var2 + var3; { result will be 15 }

   set behavior() specialvalues(zero) off;
   result = var1 + var2 + var3; { result will be DEFAULT }
```

See also: Special Values

String Functions

Compare Function

Format:

```
i = compare(string-1,string-2);
```

Description:

The compare function compares the two strings character by character to determine the alphabetical (collating sequence) order of the strings. If "string-1" and "string-2" are of different lengths, the compare function will pad the shorter string with blanks to carry out the comparison.

Return value:

The function returns an integer value of

-1 if "string-1" would be listed alphabetically before "string-2"
0 if the strings are identical
1 if "string-1" would be listed alphabetically after "string-2"

Example 1:

```
ORDER = compare(RESPONSE, ANSWER);
```

where ORDER is an integer variable and RESPONSE and ANSWER are string variables.

Example 2:

```
Y = compare("survey", "census");
```

where Y is an integer variable and the arguments are string constants.

Direct string comparisons can also be made. For example, the following code is permissible (and in fact preferable to usage of the compare function:

Example 3:

```
if string1 < string2 then
  <statements>;
endif;
```

Concat Function

Format:

```
s = concat(string-2,string-2[,..,string-n]);
```

[] indicates that this part is optional.

Description:

The **concat** function concatenates the values of two or more strings. The strings can be alphanumeric items, text strings, or functions that return strings. You can also use the + operator to concatenate strings.

Return value:

The function returns the concatenated string.

Example:

```
PROC GLOBAL
  alpha (30) FIRST_NAME, LAST_NAME, FULL_NAME;

PROC EXAMPLE
  FIRST_NAME = "John"
  LAST_NAME = "Henry"
  FULL_NAME = concat(strip(FIRST_NAME)," ", strip(LAST_NAME));
```

Results in the following values:

```
  FIRST_NAME = "John "
  LAST_NAME = "Henry "
  FULL_NAME = "John Henry "
```

See also: Plus String Concatenation

String Concatenation Operator

In addition to serving as the addition operator for numeric values, the + operator can be used to concatenate strings, similar to the concat function. For example, both lines of code below will give the same result:

```
PROC GLOBAL

string  text1 = "Hello",
        text2 = "World!";

PROC EXAMPLE

    string messagePlus = text1 + ", " + text2;

    string messageConcat = concat(text1,", ",text2);
```

The + operator is a convenient way to build strings and can be used along with other functions that return alphanumeric values.

Example:

```
PROC GLOBAL

string message;
numeric counter;

PROC EXAMPLE

    do counter = 1 while counter <= 13 by 2
        message = message + edit("99 ",counter);
    enddo;

    // message is now: "01 03 05 07 09 11 13"
```

Edit Function

Format:

```
s = edit(edit-pattern,numeric-expression);
```

Description:

The edit function converts a number to a character string defined by the given "edit pattern". The "edit pattern" is a string containing "Z"s or "9"s (i.e., "9999" or "ZZ9.99"). Both "9" and "Z" represent a digit.

9 display a digit
Z display a digit, but if it is a leading zero, display a blank
. display the decimal character
, display the thousands separator character

Any other character will be displayed as itself.

Return value:

The function returns a string derived from the "numeric-expression" parameter.

Example 1:

```
X = 87;
A1 = edit("ZZZ9",X); yields A1 = " 87"
A2 = edit("9999",X); yields A2 = "0087"
A3 = edit("Z999",X); yields A3 = " 087"
```

Example 2:

```
Y = 0;
A4 = edit("ZZ9",Y); yields A4 = " 0"
A5 = edit("999",Y); yields A5 = "000"
A6 = edit("ZZZ",Y); yields A6 = " "
```

Example 3:

```
A = edit("99:99:99",sysdate());
```

Example 4:

```
A = edit("99/99/99",sysdate("DDMMYY"));
```

Example 5:
```
A = edit("ZZZ,ZZZ,ZZ9",INCOME);
```

See also: Tonumber Function , Sysdate Function

Getbuffer Function

Format:
```
s = getbuffer(item-name);
```

Description:
The data item specified by "item-name" may be numeric or alphanumeric. The **getbuffer** function always returns a string containing the data item's contents. Therefore, in both examples below it does not matter whether **getbuffer** specifies a numeric item (AGE), or an alphanumeric item (NAME), it will always return a string of the data item's contents.

This function is especially useful when a numeric data item in a data file contains a non-numeric value, such as "*", "-", or "a". You cannot test the contents of the numeric data item for alphanumeric values because CSPro stores DEFAULT as the value of any numeric data item which contains non-numeric values. Therefore, to find out what non-numeric value(s) exist in a data item, you would use the **getbuffer** to return its contents in the form of a string of characters (at least one of which is non-numeric).

Return value:
The function returns a string containing the data item's contents.

Example 1:
```
if special(AGE) then
  errmsg("Person's Age is invalid, Age = %s", getbuffer(AGE));
endif;
```

Example 2:
```
errmsg("Household Head's Name = %s", getbuffer(NAME(1)));
```

Length Function

Format:
```
i = length(string-exp);
```

Description:
If the "string-exp" is a data dictionary item, the value returned is the length of the item. If the "string-exp" is the result of a function, the value returned is the length of the string returned by the function.

Return value:
The function returns the length of the string as an integer value.

Example:
```
PROC GLOBAL
  alpha (30) NAME;

PROC ABC
  NAME = "John Henry"
  LEN1 = length(NAME);
  LEN2 = length(strip(NAME));
```

Results in the following values:

```
NAME = "John Henry "
LEN1 = 30
LEN2 = 10
```

See also: Alpha Statement, Strip Function

Maketext Function

Format:

```
s = maketext(string-exp[,p1[,p2[,...,pn]]]);
```

[] indicates that this part is optional.

Description:

The maketext function formats a text string with inserted values. Each parameter (e.g., "p**1**") is sequentially inserted into the text string. Parameters can be numeric or alphanumeric expressions, but the type of parameter must match the type of the receiving field in the string expression.

In the "string expression",

%[*n*]d = Insert a number and display it as an integer
%[*n.d*]f = Insert a number and display it as a decimal value
%[*n.d*]s = Insert a text string

where "***n"*** is the size of the field and "***d"*** is the number of decimal places to show for a number.

Numbers are never truncated. Text strings are truncated only if "***.d"*** is used. If "***n"*** is positive, the insert is right justified in the size of the field. If "***n"*** is negative, the insert is left-justified in the size of the field. If "***n"*** is a positive number with a leading zero, the insert is right-justified in the size of the field and zero-filled to the left. When inserting a number, if "***n"*** is preceded by a "**+**", the sign of the number is always displayed.

Return Value:

The function returns the formatted string.

Example 1:

```
TEXT = maketext("Sex = %d", SEX);
```

Example 2::

```
Value    %d       23456
=
23456
         %10d          23456
         %-10d    23456
         %010d    0000023456
         %+10d        +23456
         %+010d   +000023456
         %f       23456.000000


Value    %f         12.567
=
12.567
         %10.3f         12.567
```

```
%-10.3f    12.567
%10.2f          12.57
%10.5f       12.56700
%010.3f    000012.567
%+10.3f       +12.567
%+010.3f   +00012.567
%d         12


Value =    %s       abcdef
"abcdef"
           %10s         abcdef
           %-10s    abcdef
           %10.3s          abc
           %-       abc
           10.3s
```

Pos Function

Format:

```
i = pos (substring, source);
```

Description:

The pos function searches for a "pattern string" within a "source-string". It returns the beginning position of the first occurrence of "pattern string". The "source-strings" and "pattern-spring" are case-sensitive, meaning that "children" is recognized as different from "CHILDREN."

Return value:

The function returns an integer position. If the "pattern-string" is not found, 0 is returned.

Example 1:

```
X = pos("L","FOR THE CHILDREN");
```

The value of X will be 12; this is where the pattern-string [the letter "L"] occurs in the source string.

Example 2:

```
X = pos("DRE","CHILDREN");
```

The value of X will be 5; this is where the pattern-string ["dre"] begins in the source string.

Example 3:

```
X = pos("DCN","CHILDREN");
```

The value of X will be 0. The pattern string ["dcn"] does not occur in the source string.

Please note unless an alpha string is declared to be the exact length of the string that is being assigned to it, any trailing character positions will be blank-filled. This can have ramifications on your search, if you are searching for the blank character. There may be none within the string, but it will find one at the end of your string, if your assigned string is shorter than the space allocated to the alpha variable. In this case, you should strip the string first. The following example should clarify this situation:

```
PROC GLOBAL
  alpha (8) myStr;

PROC FOO
  myStr = "Kids";
  pos (" ", myStr);
  {will return 5, as it finds a blank after the ‘s’ in Kids }
  pos (" ", strip (myStr));
  {will return 0, as it does not find a blank, since the string
  has been stripped of all trailing blanks before the search
  begins}
```

See also: Poschar Function, Alpha Statement, Strip Function

Poschar Function

Format:

```
i = poschar (pattern-string, source-string);
```

Description:

The poschar function searches for a collection of characters ["pattern-string"] within "source string". It returns the beginning position of the first occurrence of the pattern string. The source-string and pattern-string are case-sensitive, meaning that "c" is recognized as different from "C."

Return value:

The function returns an integer position. If no characters from the pattern string are found, 0 is returned.

Example 1:

```
X = poschar("L","CHILDREN");
```

The value of X will be 4; this is where the pattern string [the letter "L"] occurs in the source string {"CHILDREN"].

Example 2:

```
X = poschar("LCN","CHILDREN");
```

The value of X will be 1; of the characters in the pattern-string, "C" is the first character encountered in the source string, and it is found in position 1 of the source.

See also: Pos Function, Alpha Statement, Strip Function

Strip Function

Format:

```
s = strip(string-exp);
```

Description:

The strip function removes trailing blanks from the given string. The result of a strip function is often used as a parameter to other functions (such as the length and concat functions above).

You will generally not have to use the strip function when working with variable-length strings since they do not have spaces automatically added to the end of the text to match a fixed width.

Return value:

The function returns a string with no trailing blanks.

Example:

```
PROC GLOBAL
  alpha(30) FIRST_NAME, LAST_NAME, FULL_NAME;

PROC ABC
  FIRST_NAME = "John";
  LAST_NAME = "Henry";
  FULL_NAME = concat(strip(FIRST_NAME)," ", strip(LAST_NAME));
  LEN = length(strip(FULL_NAME));
```

Results in the following values:

```
FIRST_NAME = "John "
LAST_NAME = "Henry "
FULL_NAME = "John Henry "
LEN = 10
```

See also: Alpha Statement, Concat Function, Length Function

Tolower Function

Format:

```
s = tolower(string-exp);
```

Description:

The **tolower** function scans the given string and converts any uppercase letters to lowercase letters.

Return value:

The function returns a string with all uppercase letters converted to lowercase letters.

Example:

```
X = tolower("hello james!");
Y = tolower("Hello JaMeS!");
Z = tolower("HELLO JAMES!");
```

Results in the following values:

```
X = "hello james!"
Y = "hello james!"
Z = "hello james!"
```

See also: Alpha Statement, Toupper Function

Tonumber Function

Format:

```
d = tonumber(string-exp);
```

Description:

The tonumber function converts a string ["string-exp"] into a number. Leading blanks in the string are ignored. The conversion stops at the first non-numeric character encountered.

Return value:

The function returns a decimal number. If the string begins with a non-numeric character, the function returns DEFAULT.

Example:

```
n = tonumber(CASEID);
```

See also: Edit Function

Toupper Function

Format:

```
s = toupper(string-exp);
```

Description:

The **toupper** function scans the given string and converts any lowercase letters to uppercase letters.

Return value:

The function returns a string with all lowercase letters converted to uppercase letters.

Example:

```
X = toupper("hello james!");
Y = toupper("Hello JaMeS!");
Z = toupper("HELLO JAMES!");
```

Results in the following values:

```
X = "HELLO JAMES!"
Y = "HELLO JAMES!"
Z = "HELLO JAMES!"
```

See also: Alpha Statement, Tolower Function

Multiple Occurrence Functions

Average Function

Format:

```
d = average(multiple-item [where condition]);
```

Description:

The average function returns the average of an item that occurs multiple times. During data entry, the result of the **average** calculation depends on where the statement is located. If the average function is executed prior to the form or roster containing the item, it returns DEFAULT. If it is executed within the form or roster containing the item, it returns the average up to the current occurrence number. If it is executed after the form or roster containing the item, it returns the average for all occurrences of the item.

During batch edit, **average** returns the average value for all occurrences of the item, regardless of the statement's placement in the program.

If a **where** condition is included, the function returns the average of the occurrences for which the condition is true.

If the value of an occurrence of the item is a special value (DEFAULT, MISSING, or NOTAPPL), the occurrence will not be included in the calculation. If none of the occurrences have values other than special values, DEFAULT is returned.

Return value:

The function returns the decimal value of the average.

Examples:

```
AVG_INCOME = average(INCOME);
AVG_FEMALE_INCOME = average(INCOME where SEX = 2);
```

Count Function

Format:

```
i = count(multiple-item [where condition]);
```

Description:

The count function returns the number of occurrences for a repeating form or roster. During data entry, the occurrence value is updated after the postproc of the first field within a repeating form or roster is executed. If the count function is executed prior to the form or roster, it returns 0. If it is executed from a field within the form or roster, it returns the current occurrence number. If it is executed after the form or roster, it returns the total number of occurrences in the form or roster.

During batch editing, count always returns the total number of occurrences in the multiply-repeating item/record.

If a **where** condition is included, the function returns the number of occurrences for which the condition is true. If the **where** condition is not included, the count function and the noccurs function return the same result.

Return value:

The function returns an integer count value.

Examples:

```
TOTAL_PERSONS = count(PERSONS);
NUM_CHILDREN = count(PERSONS where RELATIONSHIP = 3);
```

See also: Noccurs Function, Soccurs Function, Totocc Function, Curocc Function, Seek Function, Has Operator

Curocc Function

Format:

```
i = curocc([group]);
```

Description:

The **curocc** function returns the current occurrence number for a roster, form, or record.
During data entry, you may determine the current occurrence of a roster or form. If the form does not repeat, **curocc** will return 1 (a roster must always repeat). The current occurrence can be determined by calling the **curocc** function from any field contained within the roster or form. If it is executed prior to the roster or repeating form it names, it returns 0. If it is invoked after entry of the roster or form has completed, it returns the total number of occurrences keyed.

During batch editing, you may determine the current occurrence of a record or repeating item. If the **curocc** function is used in a procedure not associated with an item on a record then **curocc** will return the total number of occurrences found. If the **curocc** function is used in a procedure associated with an item on the record, it will return the sequence number of the record in the case. The **curocc** of a repeating item will be its sequence number within the group.

Return value:

The function returns the occurrence number as an integer.

Examples 1:

```
PROC RELATION
  if curocc(PERSON_REC) = 1 then
    if (RELATION <> 1) then
      errmsg("First person must be head of household.");
    endif;
  endif;
```

See also: Maxocc Function, Totocc Function, Noccurs Function, Soccurs Function, Count Function , If Statement, Errmsg Function

Delete Function

Format:

```
b = delete(group[occ]);
```

[occ] is required for multiply-occurring records or items; is not required for singly-occurring records or items

Description:

The **delete** function removes incomplete, or otherwise unneeded, records or item occurrences from the current case. It can be used to remove singly- or multiply-occurring records, although if the record is non-repeating (such as the housing record in a typical census application), then it must be defined as "required=no" in the dictionary. This function was primarily intended for batch applications. It should be used with extreme caution in data entry applications because of possible conflicts between the operator's actions and the program logic.

Return value:

The function returns a logical value 1 (true) if successful and 0 (false) otherwise.

Example 1 (for multiply-occurring records)**:**

```
do varying i = totocc(PERSON_REC) until i <= 0 by (-1)
  if rel(i) = notappl and
    sex(i) = notappl and
    age(i) = notappl then
    delete (PERSON_REC(i)); { remove "blank" person records }
  endif;
enddo;
```

In this example blank person records are deleted from the case. Records following any deleted record are 'shifted up' to cover the vacated area. For example, if you delete the 2nd of four records, the 3rd record shifts to the 2nd position and the 4th records shifts to the 3rd position.

It is best to delete the records starting with the last record and moving toward the first. Use a subscript that starts at the last occurrence then is decremented [decreased by 1]. In this way you will not need to worry about the records that are shifting positions.

Example 2 (for singly-occurring records)**:**

```
if h01_type = 6 then { person is homeless, delete record }
  delete (HOUSING); { notice the absence of a subscript }
endif;
```

See also: If Statement, Totocc Function, Do Statement

Insert Function

Format:

```
b = insert(group[occ]);
```

[occ] is required for multiply-occurring records or items, and is not required for singly-occurring records or items

Description:

The insert function inserts missing or otherwise needed data records or item occurrences in the current case. It is primarily intended for use in batch applications. It should be used with extreme caution in data entry applications because of possible conflicts between the operator's actions and the program logic.

Return value:

The function returns a logical value 1 (true) if successful and 0 (false) otherwise.

Example 1 (for multiply-occurring records):

In the following example there is a data item in the housing record called "H13_persons", which contains the total number of people living in the household. We have decided that if the number of population records found in the household is less than this variable, we will insert the missing number of population record(s).

```
NumPersons = count (PERSON_REC);
do varying I=NumPersons+1 while I <= h13_persons
  insert (PERSON_REC(I)); { note the need for a subscript }
enddo;
```

It makes no difference if the population record has been defined in the dictionary as required or not. What is important is that it was defined as a multiply-occurring record.

Example 2 (for singly-occurring records):

For this example, we are processing a datafile that did not require housing records to be present. However, now we want to force the existence of housing records. We could implement this as follows:

```
if totocc(HOUSING) = 0 then
  insert(HOUSING); { note the absence of a subscript }
endif;
```

To accomplish this, the housing record must be set to "required = no" in the dictionary. You cannot use this function for a singly-occurring record when the "required" property setting is "yes".

See also: Do Statement, If Statement, Totocc Function, Count Function

Max Function

Format:

```
d = max(multiple-item [where condition]);
```

Description:

The **max** function returns the maximum value of an item that occurs multiple times. During batch editing, if the values of the items are not changed, the result of the maximum calculation is the same, no matter where the function is located.

During data entry, the result of the maximum calculation depends on where the statement is located. If the **max** function is executed prior to the form or roster containing the item, it returns DEFAULT. If it is executed within the form or roster containing the item, it returns the maximum value up to the current

occurrence number. If it is executed after the form or roster containing the item, it returns the maximum value for all occurrences of the item.

During batch editing, max always returns the maximum value for all occurrences of the item.

If a **where** condition is included, the function returns the maximum value of the occurrences for which the condition is true.

If the value of an occurrence of the item is a special value (DEFAULT, MISSING, or NOTAPPL), the occurrence will not be included in the calculation. If none of the occurrences have values other than special values, DEFAULT is returned.

Return value:
The function returns a decimal value.

Examples:
```
MAX_INCOME = max(INCOME);
MAX_FEMALE_INCOME = max(INCOME where SEX = 2);
```

See also: Min Function

Maxocc Function

Format:
```
i = maxocc([group]);
```

Description:
The **maxocc** function returns the maximum number of multiply occurring records or the maximum number of multiply-occurring items defined for a group in the dictionary.

This value remains the same throughout the application run.

Return value:
The function returns an integer value of the maximum number of occurrences.

Example 1:
```
PROC HOUSING
  errmsg("Maximum number of persons is %d", maxocc(PERSON));
```

Example 2:
```
PROC PERSON
  errmsg("Maximum number of persons is %d", maxocc());
```

See also: Curocc Function, Totocc Function, Soccurs Function, Noccurs Function

Min Function

Format:
```
d = min(multiple-item [where condition]);
```

Description:
The **min** function returns the minimum value of an item that occurs multiple times. During batch editing, if the values of the items are not changed, the result of the minimum calculation is the same, no matter where the function is located.

During data entry, the result of the minimum calculation depends on where the statement is located. If the **min** function is executed prior to the form or roster containing the item, it returns DEFAULT. If it is executed within the form or roster containing the item, it returns the minimum value up to the current occurrence number. If it is executed after the form or roster containing the item, it returns the minimum value for all occurrences of the item.

During batch editing, **min** always returns the minimum value for all occurrences of the item.

If a **where** condition is included, the function returns the minimum value of the occurrences for which the condition is true.

If the value of an occurrence of the item is a special value (DEFAULT, MISSING, or NOTAPPL) the occurrence will not be included in the calculation. If none of the occurrences have values other than special values, DEFAULT is returned.

Return value:
The function return a decimal value.

Examples:
```
MIN_INCOME = min(INCOME);
MIN_MALE_INCOME = min(INCOME where SEX = 1);
```

See also: Max Function

Noccurs Function

Format:
```
i = noccurs(group);
```

Description:
The **noccurs** function returns the number of occurrences of a roster, form, or record. It is equivalent to the count function without the **where** phrase.

During data entry, you may determine the current occurrence of a roster or form. The occurrence value is updated after the first entry into the first field. If the **noccurs** function is executed prior to the roster or form it specifies then it returns 0. If it is executed from a field within the roster or form, it returns the current occurrence number after the first field is on the path. For example, its value in the PREPROC of the first occurrence of the first item entered in a form or roster is zero (0), i.e., before entry. After entry its value will always be one (1). This is true for each occurrence keyed, the **noccurs** function is not incremented until the cursor is in or has passed through the first field on the roster or form. If it is executed after the form or roster, it returns the total number of occurrences in the form or roster. If the form does not repeat, **noccurs** will return 1 (a roster must always repeat). When used in Data entry **noccurs** and soccurs functions are the same.

During batch editing, **noccurs** always returns the total number of occurrences in the group.

Return value:
The function returns the number of occurrences as an integer value.

Example:
```
TOTAL_PERSONS = noccurs(PERSON);
```

See also: Totocc Function, Curocc Function, Soccurs Function, Count Function

Seek Function

Format:

```
i = seek(multiple-item condition[,numeric-expr]);
```

Description:

The **seek** function returns the occurrence number of the first item in a multiply occurring item that satisfies a certain condition. If *numeric-expr* is included, the function starts searching the multiply occurring record for a true condition starting at occurrence *numeric-expr*. If a @ symbol precedes the *numeric-expr*, the function searches for the nth occurrence of the condition.

Return value:

The function returns the occurrence number of an item that meets the condition or 0 if no such item is found.

Example 1:

```
numeric femaleIndex = seek(SEX = 2);

while femaleIndex do
    write("Person #%d is a female with name
'%s'",femaleIndex,NAME(femaleIndex));
    femaleIndex = seek(SEX = 2,femaleIndex + 1);
enddo;
```

Example 2:

```
numeric secondSpouse = seek(RELATIONSHIP = 2,@2);
```

See also: Count Function, Has Operator, **Seekmax** Function, Seekmin Function

Seekmax Function

Format:

```
i = seekmax(multiple-item condition[,numeric-expr]);
```

Description:

The **seekmax** function returns the occurrence number of the item with the greatest value that satisfies a certain condition in a multiply occurring item. If *numeric-expr* is included, the function starts searching the multiply occurring record for a true condition starting at occurrence *numeric-expr*.

Return value:

The function returns the occurrence number of an item that meets the condition or 0 if no such item is found.

Example 1:

```
numeric ptrEldest = seekmax(AGE in 0:95),
        ptrEldest12_64 = seekmax(AGE in 12:64),
        ptrYoungest65plus = seekmin(AGE in 65:95);
```

See also: Count Function, Has Operator, Seek Function, Seekmin Function, Max Function

Seekmin Function

Format:

```
i = seekmin(multiple-item condition[,numeric-expr]);
```

Description:

The **seekmin** function returns the occurrence number of the item with the lowest value that satisfies a certain condition in a multiply occurring item. If *numeric-expr* is included, the function starts searching the multiply occurring record for a true condition starting at occurrence *numeric-expr*.

Return value:

The function returns the occurrence number of an item that meets the condition or 0 if no such item is found.

Example 1:

```
numeric ptrEldest = seekmax(AGE in 0:95),
        ptrEldest12_64 = seekmax(AGE in 12:64),
        ptrYoungest65plus = seekmin(AGE in 65:95);
```

See also: Count Function, Has Operator, Seek Function, Seekmax Function, Min Function

Soccurs Function

Format:

```
i = soccurs(record-name);
```

Description:

The **soccurs** function returns the total number of occurrences of a record.

During data entry, you may determine the current occurrence of a record. The occurrence value is updated after the first entry into the first field. If the **soccurs** function is executed prior to the roster or form it specifies then it returns 0. If it is executed from a field within the roster or form, it returns the current occurrence number after the first field is on the path. For example, its value in the PREPROC of the first occurrence of the first item entered in a form or roster is zero (0), i.e., before entry. After entry its value will always be one (1). This is true for each occurrence keyed, the **soccurs** function is not incremented until the cursor is in or has passed through the first field on the record. If it is executed after the form or roster, it returns the total number of occurrences in the form or roster. If the form does not repeat, noccurs will return 1 (a roster must always repeat). When used in Data entry noccurs and **soccurs** functions are the same.

During batch editing, **soccurs** always returns the total number of record occurrences found.

Return value:

The function returns the number of occurrences as an integer value.

Example:

```
NUM_HH_MEMBERS = soccurs(PERSON_REC);
```

See also: Totocc Function, Curocc Function, Noccurs Function, Count Function

Sort Function

Format:

```
b = sort(group using [-]item [where condition]);
```

Description:

The **sort** function will sort occurrences of records or items based on the value of an item. It will order the multiple records or items in the specified group in ascending sequence **using** the specified data item as the sort key. The sort key item must be contained within the record or item sorted. If a dash (**negative sign**) is included before the item name, the sort will be in descending order. If a **where** condition is included, the function sorts only the occurrences for which the condition is true.

Sort is primarily intended for use in batch applications. It should be used with extreme caution in data entry applications because of possible conflicts between the operator's actions and the program logic.

Return value:

The function returns a logical value 1 (true) if successful and 0 (false) otherwise.

Example 1:

```
sort(PERSON using LINE_NUM);
```

Example 2:

```
// sort the children by age from oldest to youngest
sort(PERSON using -AGE where RELATIONSHIP = 3);
```

See also: Swap Function

Sum Function

Format:

```
d = sum(multiple-item [where condition]);
```

Description:

The **sum** function returns the sum of an item that occurs multiple times. If a **where** condition is included, the function returns the sum of the occurrences for which the condition is true.

During data entry, the result of the **sum** calculation depends on where the statement is located. If the **sum** function is executed prior to the form or roster containing the item, it returns DEFAULT. If it is executed within the form or roster containing the item, it returns the sum up to the current occurrence number. If it is executed after the form or roster containing the item, it returns the sum for all occurrences of the item.

During batch editing, **sum** always returns the sum for all occurrences of the item.

If the value of an occurrence of the item is a special value (DEFAULT, MISSING, or NOTAPPL) the occurrence will not be included in the calculation. If none of the occurrences have values other than special values, DEFAULT is returned.

Return value:

The function returns a decimal value of the sum.

Example:

```
TOTAL_INCOME = sum(INCOME);
TOTAL_FEMALE_INCOME = sum(INCOME where SEX = 2);
```

Swap Function

Format:

```
b = swap(group-name,record-number,record-number);
```

Description:

The **swap** function reorders the sequence of occurrences of records or items. It is useful for reorganizing the position (occurrence number) of items in a roster, for instance if you want to ensure that the head of household is the first person on the roster.

Swap is primarily intended for use in batch applications. It should be used with extreme caution in data entry applications because of possible conflicts between the operator's actions and the program logic.

Return value:

The function returns a logical value 1 (true) if successful and 0 (false) otherwise.

Example:

```
swap(PERSON,1,ptrHead);
```

See also: Sort Function

Totocc Function

Format:

```
i = totocc([group]);
```

Description:

The **totocc** function returns the total number of multiply occurring records or the total number of multiply-occurring items that a group currently contains.

During data entry, the occurrence value is updated in the preproc of the first field within a repeating form, record or roster. If the **totocc** function is executed prior to the entry of form or roster, it returns 0. If it is executed from a group or field within the form or roster, it returns the total number of occurrences currently entered. If it is executed after the form or roster, it returns the total number of occurrences in the form or roster.

During batch editing, **totocc** always returns the total number of occurrences in the group.

Return value:

The function returns an integer value of the number of occurrences.

Example 1:

```
if totocc(HOUSING) > 1 then
  errmsg("More than 1 housing record");
endif;
```

Example 2:

```
PROC HOUSING
if totocc() > 1 then
  errmsg("More than 1 housing record");
endif;
```

See also: Curocc Function, Maxocc Function, Count Function, Soccurs Function, Noccurs Function

General Functions

Errmsg (Display) Function

Format 1:

```
[b =] errmsg(alpha-exp[,p1[,p2[,...,pn]]])
   [select(caption-1, field-name-1, ..., caption-n, field-name-n)]
   [denom=var] [case|summary]);
```

Format 2:

```
[b =] errmsg(msg-num[,p1[,p2[,...,pn]]])
   [select(caption-1, field-name-1, ..., caption-n, field-name-n)]
   [denom=var] [case|summary];
```

Description:

The **errmsg** function displays a message on the data entry screen (when used in a Data Entry application) or writes a message to the batch edit report (when used in a Batch Edit application). If messages are defined via the message number *msg-num*, then those messages will be stored in a message file [.mgf].

msg-num can be a number or numeric expression

Note to ISSA users: Use the **errmsg** function with the **case** keyword to replace the "**display"** function.

Each parameter (e.g., "p1") is sequentially inserted into the error message. Parameters can be numeric or alphanumeric expressions, but the type of parameter must match the type of the receiving field in the message text. The maximum number of parameters in an errmsg function is 20.

In the message text

%[*n*]d = Insert a number and display it as an integer
%[*n.d*]f = Insert a number and display it as a decimal value
%[*n.d*]s = Insert a text string

"n" is the size of the field and "***d"*** is the number of decimal places to show for a number.

Numbers are never truncated. Text strings are truncated only if "***.d"*** is used.

If "***n"*** is positive, the insert is right justified in the size of the field. If "***n"*** is negative, the insert is left justified in the size of the field. If "***n"*** is a positive number with a leading zero, the insert is right justified in the size of the field and zero filled to the left.

When inserting a number, if "***n"*** is preceded by a **+**, the sign of the number is always displayed.

Examples:

```
Value = 23456        %d          23456
                     %10d             23456
                     %-10d       23456
                     %010d       0000023456
                     %+10d           +23456
                     %+010d      +000023456
                     %f          23456.000000


Value = 12.567       %f           12.567000
                     %10.3f           12.567
                     %-10.3f      12.567
                     %10.2f            12.57
                     %10.5f         12.56700
```

```
                    %010.3f      000012.567
                    %+10.3f         +12.567
                    %+010.3f     +00012.567
                    %d           12


Value = "abcdef"    %s           abcdef
                    %10s             abcdef
                    %-10s        abcdef
                    %10.3s               abc
                    %-10.3s      abc
```

The **denom** keyword allows you to specify a denominator, so that you can show percentages in the summary portion of the output listing. This is very useful for showing edit failure rates. In Example 2 below, the output listing will show the number of times there was more than one head of household divided by the number of households processed during the run. Note that it is the responsibility of the application designer to write logic to put the proper values into the denominator variable.

The **case** and **summary** keywords give you some control over the output listing. By default, the output listing shows you messages case by case, and also shows you a summary of the number of times the message was triggered (with an optional denominator, described above). You can limit the output listing to only case-by-case reporting, or only summary reporting by using these keywords.

The **select** keyword is used to give the option of specifying the field to go to in response to the error message. A button is placed on the error message window for each *caption and field-name* specified as a **select** parameter. The caption is what is displayed on the button. When the button is clicked, the system goes to the field specified as *field-name*. If **continue** is used as the field name, the system continues executing the code following the errmsg function.

Return Value:

The function returns a logical value 1 (true) if successful and 0 (false) otherwise. When the select keyword is used, the function returns the number of the button that was pressed, with the first button having the value 1.

Format 1 Examples:

Example 1:

```
errmsg("Head of household is %d years old.", AGE);
```

Example 2:

```
errmsg("More than 1 head of household") denom = PERSON_COUNT summary;
```

Example 3:

```
errmsg("Head of household is %d years old.  Age must be >= 12",AGE)
  select("Go to RELATIONSHIP", RELATIONSHIP,"Go To AGE", AGE)
  denom = PERSON_COUNT;
```

Format 2 Example:

```
OK = errmsg (1,"June",30,31);
```

where the message file contains the following text:

```
1 %s has only %d days. You entered %d!
```

Note the errmsg call could have also been invoked as follows:

```
   i = 1;
OK = errmsg (i,"June",30,31);
```

Execsystem Function (Windows)

Format:

```
d = execsystem(alpha-exp, [maximized | normal | minimized],
                          [focus | nofocus], [wait | nowait | stop]);
```

Description:

The **execsystem** function starts another Windows application or process. There is a related version for applications running on Android devices.

The *alpha-exp* is the name of the application or process to be started. Command line parameters may be included in this expression. If folders names or file names contain blanks, then quotation marks (") must surround the names. In this case CSPro text strings must be surrounded by apostrophes (').

Coding one of the parameters **maximized**, **normal**, or **minimized** determines how the window for the new application is opened. If this parameter is not coded, the default is **normal**.

Coding one of the parameters **focus** or **nofocus** determines whether the new application will receive focus, or "be active," when it is started. If this parameter is not coded, the default is **focus**.

Coding one of parameters **wait** or **nowait** determines whether the current application will wait until the new application is finished before control returns to it. If this parameter is not coded, the default is **nowait**. Alternatively, if **stop** is coded, the application will immediately close after launching the application, in the same way that would occur if using the stop function with a passed parameter of 1.

Return value:

The function returns a logical value of 1 (true) if the new application is started successfully and NOTAPPL otherwise.

Example 1:

```
execsystem("C:\Windows\calc.exe");
```

Example 2:

```
execsystem('textview "C:\My Online Helps\DE_Helps.txt"',maximized,wait);
```

See also: ExecPFF Function, Execsystem Function (Android)

Execsystem Function (Android)

Format:

```
d = execsystem(alpha-exp[,stop]);
```

Description:

The **execsystem** function starts another Android application or function. There is a related version for applications running on Windows machines.

If **stop** is coded, the application will immediately close after launching the application, in the same way that would occur if using the stop function with a passed parameter of 1.

The *alpha-expression* command argument is one of the following attributes: **app**, **browse**, **call**, **camera**, **gps**, or **view**.

Return value:
The function returns a logical value of 1 (true) if the new application is started successfully and NOTAPPL otherwise.

Examples:

```
// app - opens another application on the device
execsystem("app:com.google.android.gm"); // open GMail

// browse - views a website in the Internet browser
execsystem("browse:https://www.census.gov");

// call - makes a phone call
execsystem("call:+13017631451"); // call the CSPro support number

// camera - takes a photo in .jpg or .png formats
execsystem("camera:/storage/emulated/0/csentry/photo.jpg");
execsystem("camera:/storage/emulated/0/csentry/photo.png");

// gps - views a latitude/longitude point using an installed mapping engine
(e.g., Google Maps)
execsystem("gps:38.84839,-76.931098");
execsystem("gps:38.84839,-76.931098,CSPro Team at the U.S. Census Bureau");

// view - displays an image, plays a sound clip, or views a movie
execsystem("view:/storage/emulated/0/csentry/picture.jpg");
execsystem("view:/storage/emulated/0/csentry/audio.mp3");
execsystem("view:/storage/emulated/0/csentry/movie.3gp");
```

See also: ExecPFF Function, Execsystem Function (Windows)

ExecPFF Function

Format:

```
b = execPFF(alpha-exp, [maximized | normal | minimized],
                        [focus | nofocus], [wait | nowait | stop]);
```

Description:
The **execPFF** function starts another CSPro application. The function is very similar to execsystem, but instead of passing the function the name of a Windows executable application, only pass the name of a CSPro application (a Program Information File). This function is useful for surveys that use different machines on which the path of the CSPro executables may differ.

Multiple data entry applications cannot be run simultaneously on Android devices so the function simply stores the filename of the passed .PFF. When the Android application returns to the Entry Applications screen, whether via user interaction or via the stop function, the passed .PFF will be launched. Using the **stop** flag will cause the .PFF to be launched immediately.

Return value:
The function returns a logical value of 1 (true) if the new application is started successfully and NOTAPPL otherwise.

Example 1:

```
execPFF("H:\Survey\AdditionalQuestions.pff");
```

See also: Execsystem Function (Android), Execsystem Function (Windows)

Getlabel Function

Format:

```
s = getlabel(dictionary-symbol[,value]);
```

Description:

The **getlabel** function returns the label of a dictionary symbol or the text associated with a particular value of the symbol as defined in a value set.

If the value parameter is not used, then the dictionary symbol's label is returned. The symbol can be the name of the dictionary, level, record, item, or value set.

The value parameter can only be used if the dictionary symbol is an item or a value set. If the value parameter is used, the label associated with the specified value is returned. If the symbol is an item name, then the value labels from the first value set are returned. If the symbol is a value set, then the value labels from that value set are returned. If no label is associated with the value, then an empty string is returned.

The dollar sign ($) can be used to refer to the current item for both the dictionary symbol and the value. See Example 3 below.

Return value:

The function returns a string up to 255 characters long.

Example 1:

```
write("%s = %s", getlabel(SEX), getlabel(SEX,1));
```

Example 2:

```
write("Crop Type = %s", getlabel(CROP_VS2,23));
```

Example 3:

```
PROC P02_REL
  write("%s = %s", getlabel($), getlabel($,$));
```

gives

```
Relationship = Head
Relationship = Spouse
Relationship = Child
```

See also: Getsymbol Function

Getsymbol Function

Format:

```
s = getsymbol();
```

Description:

The **getsymbol** function returns the name of the current procedure being executed.

Return value:

The function returns a string up to 32 characters long.

Example:
```
errmsg("Procedure %s",getsymbol());
```

See also: Getlabel Function, Getrecord Function

Invalueset Function

Format:
```
b = invalueset(item-name[, valueset-name]);
```

Description:
The **invalueset** function determines whether a data item's value is within the item's value set.

The *item-name* is the name of the item with the dictionary.

The *valueset-name* is the name of a value set with the specified item. If the value set name is omitted, the first value set for the item is used.

Return value:
The function returns a logical value of 1 (true) if the data item is within the value set and 0 (false) otherwise. If the item has no value set and only the item name is given, the function return 1 (true).

Example 1:
```
if not invalueset(P03_SEX) then
  errmsg("Sex is out of range. Value is %d", P03_SEX);
endif;
```

Example 2:
```
if invalueset(P04_AGE(1), P04_AGE_VS2) then
  errmsg("Heads age is in group '%s'",
  getlabel(P04_AGE_VS2, P04_AGE(1)));
endif;
```

See also: Special Function, Set Behavior Canenter

Pathname Function

Format:
```
s = pathname(name-listed-below | dict-name | file-name);
```

Description:
The **pathname** function returns the fully qualified name of one of the following paths (folders):

- **Application**: The folder where the application file (.ent or .bch) is located.
- **InputFile**: The folder where the main data file (associated with the primary dictionary) is located.
- **CSPro**: The folder where the CSPro executable files are located.
- **Documents**: The current user's My Documents folder.
- **Desktop**: The folder of the current user's Desktop.
- **Temp**: The operating system's temporary directory.
- **ProgramFiles32**: The folder containing installed 32-bit applications.
- **ProgramFiles64**: The folder containing installed 64-bit applications. If the computer has only a 32-bit processor, this returns the same value as ProgramFiles32.
- **Windows**: The folder contains the Windows operating system.

A dictionary or file handler name can also be passed as an argument to the function. Passing a dictionary name will return the folder name of the dictionary file (.dcf) itself, not the data file associated with the dictionary.

The following attributes are ignored on the Android platform: Documents, Desktop, ProgramFiles32, ProgramFiles64, and Windows. Additionally, passing a dictionary name will return the folder name of the .pen file because dictionary files (.dcf) are not used directly when running Android applications.

Return value:
The function returns a string containing the fully qualified folder name.

Example:

```
setfile(LOG_DICT,concat(pathname(Application),"log.txt"),create);
```

See also: Filename Function

Special Function

Format:

```
b = special(numeric-exp);
```

Description:
The **special** function determines whether the value of a *numeric-exp* is one of the three "special" values: **missing**, **notappl**, or **default**.

The *numeric-exp* can be a data item, a variable, or any numeric expression.

Return value:
The function returns a logical value of 1 (true) if the variable is a special value and 0 (false) otherwise.

Example:

```
if special(XVAR) then
  YVAR = 99;
else
  YVAR = XVAR;
endif;
```

See also: Special Values, Countnonspecial Function

Stop Function

Format:

```
stop([0 | 1]);
```

Description:
The **stop** function ends a data entry session or batch edit run.

If the **stop** function is used in a data entry application, the parameter determines whether data entry is stopped just for the current case or whether the data entry application is closed. If the parameter is empty or 0, entry of the current case is stopped (the same a pressing the stop button on the tool bar), but the data entry application remains active. If the parameter is non-zero, for example 1, entry of the current case is stopped and the data entry application is terminated.

In a data entry application, if the stop function is executed in the Postproc of the first (highest) level then the data for the case is saved. Otherwise, any data entered for the current case is lost. If you want to avoid losing data, you should include a savepartial function just before the stop function (see example below).

If the stop function is used in a batch edit application, the parameter (0 | 1) has no effect, the run is always terminated. If an output file was specified in the batch run, neither the current case nor subsequent cases will be placed in the output file.

Example 1 (data entry):

```
if $ = 99 then
  savepartial();
  stop(1);
endif;
```

Example 2 (batch edit):

```
if TOTAL_ERRORS > 100 then
  stop();
endif;
```

See also: Savepartial Function, Exit Statement, Break Statement, Next Statement, Skip Case Statement

Sysparm Function

Format:

```
s = sysparm();
```

Description:

The **sysparm** function returns the value of the ‘parameter’ variable stipulated in the data entry or batch edit pff file. The sysparm function returns the passed-in parameter as a left-justified string.

If no parameter was given in the pff file, then **sysparm** returns the null (empty) string. If the string given in the pff file is longer than the size allocated for your program’s string variable, then the string will be truncated.

Return value:

The function returns an alphanumeric string containing the system parameter.

Example:

```
PROC GLOBAL
  alpha(30) MyParam;

PROC MyFile
preproc
  MyParam = sysparm();
```

Trace Function

Programmers use tracing to obtain low-level information about how an application runs, with the information often used for debugging purposes to understand why a program does not execute as expected. CSPro offers limited tracing functionality for two objectives: logging user-generated information and tracking executed statements. Tracing messages can be displayed in a window or saved to a file.

Activating and Disabling Tracing

```
trace(on); // turns on tracing and outputs messages to a window

trace(on,"filename"); // turns on tracing and appends messages to "filename"

trace(on,"filename",clear); // turns on tracing, clears any contents in the
file, and writes messages to "filename"

trace(off); // turns off tracing, closing all open trace windows or files
```

It is possible, by calling the **trace** function twice, to send messages to both a window and a file. If the filename does not contain a path the file will be placed in the application folder. On Android it is only possible to trace to a file and any trace statements output to a window will be ignored.

Logging User-Generated Information

To send a message to the trace log, simply pass a string to the **trace** function.

```
trace("This is a trace message");

if distance > 100 then
    trace(maketext("distance (%d) > 100!",distance));
endif;
```

Tracking Executed Statements

Occasionally a programmer wants to observe how logic statements are executed, particularly when the logic behaves in a manner inconsistent with the programmer's expectations. The trace window or file can display each line of logic as it is executed. Because in some applications this may be a very large number of statements, the programmer must specify what elements of the logic should be outputted:

```
set trace; // logic statements after this point will be outputted

set trace(on); // same as above

set trace(off); // logic statements after this point will not be outputted
```

The **set trace** statement indicates to CSPro that logic statements should or should not be outputted but the statements will only be outputted if tracing is activated, thus the **trace** function and **set trace** statements must be used together.

Example

```
trace("There is no trace window open so this message is discarded");

trace(on,"trace.txt",clear); // opens the trace file and clears previous
contents

trace("This message appears in the file");

trace(maketext("Complex strings can be outputted using maketext; e.g., e =
%0.3f",exp(1)));

trace(off); // closes the trace file
```

```
trace(on,"trace.txt"); // opens the trace file and now messages will be
appended to the end of the file

set trace;

numeric value = 10;

if value > 10 then
    errmsg("A");
elseif value < 10 then
    errmsg("B");
else
    errmsg("C");
endif;

errmsg("This statement will appear on the trace window");

set trace(off);

errmsg("This statement will not appear on the trace window");
```

Trace Output

As the following trace results show, the output for conditional statements (e.g., **if**) and loops (e.g., **do**) is limited. Trace results show the line numbers to the left of the executed statements.

```
Trace started at 02/03/11 20:28:35

TRACE   This message appears in the file
TRACE   Complex strings can be outputted using maketext; e.g., e = 2.718

Trace stopped at 02/03/11 20:28:35
Trace started at 02/03/11 20:28:35

31   :  numeric value = 10;
33   :  if value > 10 then
38   :  errmsg("C");
41   :  errmsg("This statement will appear on the trace window");
43   :  set trace(off);

Trace stopped at 02/03/11 20:28:35
```

Date and Time Functions

Dateadd Function

Format:

```
i = dateadd(numeric-start-date,numeric-period[,string-expression]);   []
```

indicates that this part is optional.

Description:

The **dateadd** function calculates a new date by adding a period to a starting date. The dates must be passed in YYYYMMDD format. If no year is present then the current or previous year is assumed in

order satisfy the condition that the date is not in the future (based on the computer's system clock). An optional third parameter indicates the format of the period, which can be:

"d" days (default)
"m" months
"y" years

Return Value:
The function returns a date calculated by adding the period to the starting date. If the starting date cannot be processed, the function returns DEFAULT.

Example:

```
dateadd(20121225,7); // returns 20130101

dateadd(20120228,1); // returns 20120229
dateadd(20130228,1); // returns 20130301

dateadd(20040820,3,"m"); // returns 20041120
dateadd(20040820,3,"y"); // returns 20070820

dateadd(20001010,-3,"m"); // returns 20000710
```

See also: Datediff Function, Datevalid Function, **Sysdate Function**

Datediff Function

Format:

```
i = datediff(numeric-start-date,numeric-end-date[,string-expression]);   []
```

indicates that this part is optional.

Description:
The **datediff** function returns the difference between two dates as a number. The dates must be passed in YYYYMMDD format. If no year is present then the current or next year is assumed in order to satisfy the condition that the start date is earlier than the end date. If years are present, it is possible for the start date to be later than the end date, in which case the function returns a negative difference. An optional third parameter indicates the difference requested. This function acts similarly to Microsoft Excel's "datedif" function.

Based on the optional third parameter (the default option is "d"), the function returns:

"d" days between the start and end dates
"m" months between the start and end dates
"y" years between the start and end dates
"md" days between the start and end dates ignoring both the years and months of the dates
"ym" months between the start and end dates ignoring the years of the dates
"yd" days between the start and end dates ignoring the years of the dates

Return Value:
The function returns the requested difference in dates, defaulting to the number of days between the dates if the third parameter is not specified. If the dates cannot be processed, the function returns DEFAULT. The function returns values equal to Excel's function for most parameters, but due to processing differences involving leap years, sometimes CSPro's function will return a different value than Excel's.

Example 1:

```
datediff(19790404,19820605);  returns 1158
datediff(19790404,19820605,"d");  returns 1158
datediff(19790404,19820605,"m");  returns 38
datediff(19790404,19820605,"y");  returns 3
datediff(19790404,19820605,"md");  returns 1
datediff(19790404,19820605,"ym");  returns 2
datediff(19790404,19820605,"yd");  returns 62
```

Example 2:

```
date1 = 20090120;
date2 = 20121106;
outputText = "The difference between January 20, 2009 and November 6, 2012
is %d years, %d months, and %d days";
errmsg(outputText,datediff(date1,date2,"y"),datediff(date1,date2,"ym"),dated
iff(date1,date2,"md"));

{ returns ... 3 years, 9 months, and 17 days }
```

See also: Dateadd Function, Datevalid Function, Sysdate Function

Datevalid Function

Format:

```
b = datevalid(numeric-date);
```

Description:

The **datevalid** function determines whether a date is valid. Dates must be passed in YYYYMMDD format.

Return Value:

The function returns 1 if the date is valid and 0 otherwise.

Example:

```
datevalid(20120229); // returns 1
datevalid(20130229); // returns 0
```

See also: Dateadd Function, Datediff Function, Sysdate Function

Sysdate Function

Format:

`i = sysdate([date-format]);` [] indicates that this part is optional.

Description:

The **sysdate** function returns the current system date as an integer.

The *date-format* is an alphanumeric expression composed of a combination of DD (days), MM (months), and/or YY or YYYY (years). YY returns the current year in two digits, while YYYY returns it in four digits. The strings DD, MM and YY or YYYY can be put together in any order to make up a customized format. If no date-format is specified, the **sysdate** function will return the date in the format "YYMMDD".

The current date can be returned as a string using the edit function as follows:

```
edit("99/99/99",sysdate("DDMMYY"));
```

Return value:
The function returns the system date as an integer. If the *date-format* is invalid, the function returns 0.

Example 1:
If the current date is December 17, 1999, the following calls would return:

```
x = sysdate("DDMMYYYY"); returns 17121999
x = sysdate("MMYYYY"); returns 121999
x = sysdate("DD"); returns 17
x = sysdate(); returns 991217
```

Example 2:
If the current date is March 8, 2000, the following calls would return:

```
x = sysdate("DDMMYYYY"); returns 8032000
x = sysdate("MMYYYY"); returns 32000
x = sysdate("MMYY"); returns 300
x = sysdate("DD"); returns 8
x = sysdate(); returns 308
```

See also: Dateadd Function, Datediff Function, Datevalid Function

Systime Function

Format:

```
i = systime();
```

Description:
The systime function returns the current system time as a six-digit integer in the form HHMMSS, where HH is the hour, MM are the minutes, and SS are the seconds.

The current time can be returned as a string using the edit function as follows:

```
edit("99:99:99",systime());
```

Return value:
The function returns the system time as an integer.

Example:

```
TIME = systime();
HOUR = int(TIME / 10000);
MIN = int(TIME / 100) % 100;
SEC = TIME % 100;
```

External File Functions

Clear Function

Format:

```
b = clear(ext-dict);
```

Description:
The **clear** function initializes the memory values of data items defined for an external file. Numeric items are initialized to NOTAPPL (blank) and alphanumeric items are initialized to blank. The clear function will also set the number of occurrences of records and items to 0.

Return value:
The function returns a logical value 1 (true) if successful and 0 (false) otherwise.

Example:
```
OK = clear(LOOKUP);
```

Close Function

Format:
```
b = close(ext-dict-name | file-name);
```

Description:
The **close** function closes a previously-opened external file or file declared in a **file** statement. Under most circumstances, neither an **open** nor a **close** function is necessary to manipulate the file. In data entry applications, by default, the file is opened when it is operated on with a file function, such as **loadcase**, **writecase**, **readfile**, or **writefile** and closed immediately afterward. In batch applications, by default, the file is opened at the beginning of the run and closed at the end.

If you want to control the opening and closing of the file, you can use the open and **close** functions to do this. If you code an **open** function anywhere is the application logic, then you must control all opening and closing of the file

The open function opens the specified file and leaves it open. The **close** function closes an open file.

The *ext-dict-name* or *file-name* must be supplied. *Ext-dict-name* is the name of a data dictionary defining an external data file. *File-name* is the name of a file declared in a **file** statement.

Return value:
The function returns a logical value of 1 (true) if successful and 0 (false) otherwise.

Example:
```
OK = close(LOOKUP);
```

See also: Open Function

Delcase Function

Format:
```
b = delcase(ext-dict-name[,var-list]);
```

Description:
The **delcase** function marks a case for deletion in the external file described by *ext-dict-name*. The case whose identifiers match **var-list** is the case that is marked as deleted (but not deleted!). A deleted case is marked by a tilda '~' in the first character of each record in the case.

The *ext-dict-name* must be supplied. It is the dictionary name defined in the data dictionary for the external file.

The optional *var-list* defines the case identifiers in the external file. The **delcase** function concatenates the variables specified in *var-list* to form a string whose length must be the same as the length of the case identifier in the external dictionary. All variables in the *var-list* must exist in a dictionary or working storage.

If no *var-list* is provided, the current values of the ID Items for the external file are used.

Return value:
The function returns a logical values of 1 (true) if a case is marked for deletion and 0 (false) otherwise.

Example:
```
OK = delcase(GNMR31,MCLUST,MHHNUM,MLINE);
```

Fileconcat Function

Format:
```
b = fileconcat(result-file-name, file1[, file2[, ...]]);
```

Description:
The **fileconcat** function concatenates a list of files. The list may contain individual files or wildcard file specifications.

The *result-file-name* is an alphanumeric expression giving a file name or a name declared in a **file** statement in PROC GLOBAL.

File1, *file2*, etc. are alphanumeric expressions that contain the names of specific files or a wildcard specification of a group of files.

Return value:
The function returns a logical value of 1 (true) if successful and 0 (false) otherwise.

Example:
```
fileconcat("c:\prov1\prov1.dat", "c:\prov1\01*.dat");
```

See also: File Statement, Filecopy Function, Filedelete Function, Fileexist Function

Filecopy Function

Format:
```
b = filecopy(file-name, result-file-name);
```

Description:
The **filecopy** function copies one file to another file, or a group of files to a folder.

The *file-name* and *result-file-name* are alphanumeric expressions giving the source and destination file names or names declared in a **file** statement in PROC GLOBAL.

You can use the wildcard characters "*" and "?" to specify a group of files to copy as the *file-name*. In that case you must specify the name of a folder as the *result-file-name*.

Return value:
The function returns the number of files copied.

Examples:
```
filecopy(DATA, DATACOPY);
```

DATA and DATACOPY could be the names of files declared in a **file** statement or an **alpha** variables set to contain the names of the files as a text string.

```
filecopy("*.dat", "myfolder");
```

See also: File Statement

Filecreate Function

Format:

```
b = filecreate(file-name);
```

Description:

The **filecreate** function creates a new file with the given file name. If the file already exists, it is truncated to length three (the bytes in the file will consist only of the UTF-8 byte order mark). If the file is associated with the application as an external dictionary, a new index will be created for the file.

The *file-name* is an alphanumeric expression giving a file name or a name declared in a **file** statement in PROC GLOBAL.

Return value:

The function returns a logical value of 1 (true) if successful and 0 (false) otherwise.

Example 1:

```
filecreate("c:\census data\region1\district1.dat");
```

Example 2:

```
filecreate(MYREPORT);
```

MYREPORT could be the name of a file declared in a **file** statement or an **alpha** variable set to contain the name of the file as a text string.

See also: File Statement

Fileempty Function

Format:

```
b = fileempty(file-name);
```

Description:

The **fileempty** function determines whether or not a file is empty and has no content. This function is useful for checking the status of files without needing to know the text encoding. An empty ANSI file is 0 bytes but an empty UTF-8 file is 3 bytes.

The *file-name* is an alphanumeric expression giving a file name or a name declared in a **file** statement in PROC GLOBAL.

Return value:

The function returns a logical value of 1 (true) if the file exists and is empty and 0 (false) if the file is not empty. If the file does not exist or there is a file name error, it returns –1.

Example:

```
if fileempty(customReportFile) then
    filewrite(customReportFile,"SECONDS,KEYSTROKES"); // print the header
endif;
```

See also: File Statement, Fileexist Function, Filesize Function

Fileexist Function

Format:

```
b = fileexist(file-name);
```

Description:

The **fileexist** function determines whether a file exists.

The *file-name* is an alphanumeric expression giving a file name or a name declared in a **file** statement in PROC GLOBAL.

You can use the wildcard characters "*" and "?" as the *file-name*. In that case the **fileexist** function will determine whether any file matching the specification exists.

Return value:

The function returns a logical value of 1 (true) if the file exists and 0 (false) otherwise.

Example:

```
if fileexist(MYREPORT) then
  filedelete(MYREPORT);
endif;
```

MYREPORT could be the name of a file declared in a **file** statement or an **alpha** variable set to contain the name of the file as a text string.

See also: File Statement, Fileempty Function

Filedelete Function

Format:

```
b = filedelete(file-name);
```

Description:

The **filedelete** function deletes an already existing file or group of files.

The *file-name* is an alphanumeric expression giving a file name or a name declared in a **file** statement in PROC GLOBAL.

You can use the wildcard characters "*" and "?" to specify a group of files to delete.

Return value:

The function returns the number of files deleted.

Example:

```
filedelete(MYREPORT);
```

MYREPORT could be the name of a file declared in a **file** statement or an **alpha** variable set to contain the name of the file as a text string.

```
filedelete("*.dat");
```

See also: File Statement

Filename Function

Format:
```
s = filename(dict-name | file-name);
```

Description:
The **filename** function returns the fully qualified name of a data file.

The data file may be referenced by the data dictionary *dict-name* or by *file-name* declared in a **file** statement in **PROC GLOBAL**.

Return value:
The function returns a string containing the folder and file name.

Example:
```
NAME = filename(CHILE_2000);
```

NAME might be assigned "c:\Census2000\data\09011961.dat", if the data dictionary CHILE_2000 was associated with that file.

See also: File Statement, Filesize Function, Fileexist Function, Pathname Function

Fileread Function

Format:
```
b = fileread(file-name, alpha-item-var);
```

Description:
The **fileread** function reads a text line from a file into an item or variable. After the read the file pointer is positioned to the next record in the file. This is a sequential read.

The *file-name* is a name declared in the **file** statement in PROC GLOBAL.

The *alpha-item-var* is an alphanumeric data item or variable that will receive the contents of a line from the file. If the item or variable is too long, blanks will be added at then end. If the item or variable is too short, the input will be truncated.

Return value:
The function returns a logical value of 1 (true) if successful and 0 (false) otherwise.

Example:
```
fileread(DATAFILE, TEXT);
errmsg("The text is %s", TEXT);
```

See also: File Statement, Filewrite Function, Write Function

Filerename Function

Format:
```
b = filerename(old-file-name, new-file-name);
```

Description:
The **filerename** function changes the name of a file on the disk.

The *old-file-name* and *new-file-name* are alphanumeric expressions giving a file name or names declared in a **file** statement in PROC GLOBAL.

Return value:

The function returns a logical value of 1 (true) if successful and 0 (false) otherwise.

Example:

```
filerename(MYDATAFILE, "c:\folder\my new data file.dat");
```

MYREPORT could be the name of a file declared in a **file** statement or an **alpha** variable set to contain the name of the file as a text string.

See also: File Statement

Filesize Function

Format:

```
n = filesize(file-name);
```

Description:

The **filesize** function returns the size of a file in bytes.

The *file-name* is an alphanumeric expression giving a file name or a name declared in a **file** statement in PROC GLOBAL.

Return value:

The function returns a value greater than or equal to zero if the file exists. If the file does not exist or there is a file name error, it returns –1.

Example:

```
SIZE = filesize(MYREPORT);
```

MYREPORT could be the name of a file declared in a **file** statement or an **alpha** variable set to contain the name of the file as a text string.

See also: File Statement, Filename Function, Fileempty Function, Fileexist Function

Filewrite Function

Format:

```
b = filewrite(file-name, alpha-exp[, p1[, p2[,...,pn]]]);
```

Description:

The **filewrite** function writes a line of text to a file.

The *file-name* is a name declared in the **file** statement in PROC GLOBAL.

The *alpha-exp* is an alphanumeric variable or data item or an alphanumeric expression that is written to the file.

The parameters *p1* thru *pn* are numeric or alphanumeric expressions that provide the values for the inserts described in the *alpha-exp*.

If you want a text line to begin on a new page, place **\f** at the beginning of the text string.

If you want to break a line of text into two lines, place **\n** where you want the line divided.

To output \n or \f as text instead of new line or new page, use a double backslash.

Return value:
The function returns a logical value of 1 (true) if successful and 0 (false) otherwise.

Example 1:
```
filewrite(REPORT, "Education Level = %d",EDU);
```

Example 2:
```
filewrite(REPORT, "\fThis is my page title line");
```

Example 3:
```
filewrite(REPORT, "This is the first line\nThis is the second line");
```

Example 4:
```
filewrite(PFF_FILE, "Input=C:\\file path\\name of file");
```

See also: File Statement, Fileread Function, Write Function

Find Function

Format:
```
b = find(ext-dict-name, rel-op, alpha-exp);
```

Description:
The **find** function determines the existence of a case in an external file that matches a specified condition. The find function searches the index of an external file and determines whether any case matches the specified condition. The position in the file is not changed, and no case is loaded into memory.

The *ext-dict-name* must be supplied. It is the internal name of the dictionary defined in the application for the external file.

The *rel-op* is one of the following relational operators: =, <, <=, >, or >=.

The *alpha-exp* is an alphanumeric expression which specifies a set of case identifiers or a key.

If the relational operators are < or <=, then the file is positioned at the case with the largest key which satisfies the condition. If the relational operators are > or >=, then the file is positioned at the case with the smallest key which satisfies the condition.

Return value:
The function returns a logical value of 1 (true) if a case is found and 0 (false) otherwise.

Example:
```
OK = find(CODE,>=,"10100201");
```

See also: Locate Function

Key Function

Format:
```
s = key(ext-dict-name);
```

Description:

The **key** function returns a string containing the key of the case at the current position in an external file. The *ext-dict-name* must be supplied. It is the internal name of the dictionary defined in the application for the external file.

If there has been no previous activity on the external file and no key has been established, the key function returns a null string.

Return value:
The function returns a string containing the key. If no key is present, a null string is returned.

Example:
```
THE_KEY = key(LOOKUP);
```

Loadcase Function

Format:
```
b = loadcase(ext-dict-name[, var-list]);
```

Description:
The **loadcase** function reads a specified case from an external file into memory. Once the case is loaded, all variables defined in the corresponding external dictionary are available for use.

The "ext-dict-name" must be supplied. It is the internal name of the dictionary defined in the application for the external file.

The optional "var-list" specifies the list of variables that will identify the case to load from the external file. This process is similar to matching files on the basis of key variables in statistical and database software packages. Each of the variables in "var-list" must be defined in a dictionary or working storage. The combined length of the variables in "var-list" must equal the length of the case IDs defined for the external dictionary.

The **loadcase** function concatenates the variables in the "var-list" to form a string. It then loads from the external file the case whose case identifier matches the string constructed from "var-list." If no "var-list" is provided, the next logical case in the external file will be loaded. The next logical case is defined as the case with the next sequential case identifier (in ascending order). This will not necessarily be the next case in physical sequence in the file.

Return value:
The function returns a value 1 (true) if the case was loaded successfully, 0 (false) otherwise.

Example:
```
OK = loadcase(SAMPDICT,CLUSTER,HH);
```

See also: Retrieve Function, Writecase Function

Locate Function

Format:
```
b = locate(ext-dict-name, rel-op, alpha-exp);
```

Description:
The **locate** function finds, but does not load, a case in an external file that matches a specified condition. This function searches the index of an external file and finds the first case that matches the specified condition. The file pointer is positioned to the case's location, but the case is not loaded into memory. To load the case into memory, use the retrieve function after the locate function.

The *ext-dict-name* must be supplied. It is the internal name of the dictionary defined in the application for the external file.

The *rel-op* is one of the following relational operators: =, <, <=, >, or >=.

The *alpha-exp* is an alphanumeric expression that specifies a set of case identifiers or a key. If the relational operators are < or <=, then the file is positioned at the case with the largest key which satisfies the condition. If the relational operators are > or >=, then the file is positioned at the case with the smallest key which satisfies the condition.

Return value:
The function returns a logical value of 1 (true) if a case is found and 0 (false) otherwise.

Example:
```
OK = locate(CODE,>=,"10100201");
```

See also: Find Function

Open Function

Format:
```
b = open(ext-dict-name | file-name[, update | append | create]));
```

Description:
The **open** function opens and keeps open an external file or file declared in a **file** statement.

Under most circumstances, neither an **open** nor a **close** function is necessary to manipulate the file. In data entry applications, by default, the file is opened when it is operated on with a file function, such as **loadcase**, **writecase**, **readfile**, or **writefile** and closed immediately afterward. In batch applications, by default, the file is opened at the beginning of the run and closed at the end.

If you want to control the opening and closing of the file, you can use the **open** and close functions to do this. If you code an **open** function anywhere is the application logic, then you must control all opening and closing of the file

The *ext-dict-name* or *file-name* must be supplied. *Ext-dict-name* is the name of a data dictionary defining an external data file. *File-name* is the name of a file declared in a **file** statement.

The keywords **update**, **append** or **create** are optional. If no keyword is coded, the file is opened in **update** mode.

When an *ext-dict-name* is used, if **update** or **append** is used, the file is opened, its contents are not changed, and the file is ready to update cases. If **create** is coded, the file is opened, all previous records are removed and the file is ready to add cases.

When a *file-name* is used, if **update** is used, the file is opened and you are positioned at the beginning of the file. If **append** is coded, the file is opened, its contents are not changed, and you are positioned at the end of the file. If **create** is coded, the file is opened, all previous records are removed and you are positioned at the beginning of the file.

Return value:
The function returns a logical value of 1 (true) if file is opened and 0 (false) otherwise.

Example 1:
```
OK = open(LOOKUP);
```

Example 2:

```
OK = open(REPORT, append);
```

Retrieve Function

Format:

```
b = retrieve(ext-dict-name);
```

Description:

The retrieve function reads a case into memory from the current position of an external file. It is intended for use only after a successful execution of the locate function.

The ***ext-dict-name*** must be supplied. It is the dictionary name defined in the data dictionary for the external file.

Return value:

The function returns a logical value of 1 (true) if a case is retrieved and 0 (false) otherwise.

Example:

```
OK = retrieve(LOOKUP);
```

See also: Loadcase Function

Setfile Function

Format:

```
b = setfile(ext-dict-name | file-name, alpha-exp
            [, update | append | create]);
```

Description:

The **setfile** function assigns a new physical file to a dictionary or declared file.

The *ext-dict-name* or *file-name* must be supplied. *Ext-dict-name* is the name of a data dictionary defining an external data file. *File-name* is the name of a file declared in a **file** statement.

Alpha-exp is an alphanumeric expression containing the folder and file name of the file to be attached.

The keywords **update**, **append** or **create** are optional. If no keyword is coded, the file is opened in **update** mode.

When an *ext-dict-name* is used, if **update** or **append** is used, when the file is opened, its contents are not changed, and the file is ready to update cases. If **create** is coded, when the file is opened, all previous records are removed and the file is ready to add cases.

When a *file-name* is used, if **update** is used when the file is opened, you are positioned at the beginning of the file. If **append** is coded, when the file is opened, its contents are not changed, and you are positioned at the end of the file. If **create** is coded, when the file is opened, all previous records are removed and you are positioned at the beginning of the file.

Return value:

The function returns a logical value of 1 (true) if the new physical file is successfully assigned and 0 (false) otherwise.

Example 1:

```
OK = setfile(LOOKUP, "c:\My Lookup File.dat");
```

Example 2:

```
OK = setfile(REPORT, REPORT_FILE_NAME, create);
```

See also: File Statement, Filename Function, Setoutput Function

Writecase Function

Format:

```
b = writecase(ext-dict-name[,var-list]);
```

Description:

The **writecase** function writes a case from memory to an external data file. It can be used to update existing cases or to write new ones

The *ext-dict-name* must be supplied. It is the internal name of the dictionary defined in the application for the external file.

The optional *var-list* defines the case identifiers in the external file. The **writecase** function concatenates the variables specified in *var-list* to form a string whose length must be the same as the length of the case identifier in the external dictionary. All variables in the *var-list* must exist in a dictionary or working storage. If no *var-list* is provided, the current values of the identifiers in memory for the external file are used.

If the case identified by *var-list* already exists, the **writecase** function will overwrite the existing case if no records in the case have changed length. If any record in the case has changed length each record in the case will be marked with a tilda, '~', in the first position of the record and entire case will be appended to the end of the associated file. The **writecase** function automatically generates and updates the index file (with extension IDX) for the external data file.

After a case is written to an external file, the external dictionary variables for that case remain in memory. If the application does not assign new values to all variables in the external dictionary before the next **writecase** function is executed, then values from the previous case will be written to the external data file. Use the clear function to clear the values of these variables.

No Index Mode:

In a batch application it is possible to have the **writecase** function output cases without updating the file's index. This allows the external data file to have duplicate cases, i.e., cases sharing the same IDs. This may be useful if the batch application is a tool to reformat large sets of data, a situation in which maintaining the file's index is very time consuming. Since no check is done for duplicates you should be certain that duplicates cannot be generated, or that you want duplicates in the data file. Do not use other external file functions like loadcase or retrieve if using no index mode. To use this special mode, in the function parameters write *(noindex)* after the dictionary name.

```
OK = writecase(KIDS(noindex),CLUSNUM,HHNUM,LINE); // no index mode
```

Return value:

The function returns a logical value of 1 (true) if the write is successful and 0 (false) otherwise.

Example:

```
OK = writecase(KIDS,CLUSNUM,HHNUM,LINE);
```

See also: Loadcase Function

Write Function

Format:

```
[b =] write(alpha-exp[,p1[,p2[,...,pn]]]);
```

Description:

The **write** function writes text to a write file that can be used as a report. Each parameter (e.g., "p1") is sequentially inserted into the write string. Parameters can be numeric or alphanumeric expressions, but the type of parameter must match the type of the receiving field in the message text. If no write file is specified at run time, the write file lines are placed in the default data entry or batch error report.

In the string expression
%[n]d = Insert a number and display it as an integer
%[n.d]f = Insert a number and display it as a decimal value
%[n.d]s = Insert a text string

"n" is the size of the field and "d" is the number of decimal places to show for a number.

Numbers are never truncated. Text strings are truncated only if ".d" is used.

If "n" is positive, the insert is right-justified in the size of the field. If "n" is negative, the insert is left-justified in the size of the field. If "n" is a positive number with a leading zero, the insert is right-justified in the size of the field and zero-filled to the left. When inserting a number, if "n" is preceded by a "+", the sign of the number is always displayed.

Examples:

```
Value    %d       23456
=
23456
         %10d          23456
         %-10d    23456
         %010d    0000023456
         %+10d        +23456
         %+010d   +000023456
         %f       23456.000000


Value    %f         12.567
=
12.567
         %10.3f         12.567
         %-10.3f    12.567
         %10.2f          12.57
         %10.5f       12.56700
         %010.3f    000012.567
         %+10.3f       +12.567
         %+010.3f   +00012.567
         %d         12


Value =    %s       abcdef
"abcdef"
           %10s         abcdef
```

```
%-10s    abcdef
%10.3s          abc
%-       abc
10.3s
```

Return Value:
The function returns a logical value 1 (true) if successful and 0 (false) otherwise.

Example:

```
write("Sex = %d", SEX);
```

See also: Filewrite Function

Appendix

Appendix A - Installation

Hardware and Software Requirements

The minimal configuration:

- Pentium processor
- 512MB of RAM
- SVGA monitor
- Mouse
- 100MB of free hard drive space
- Microsoft Windows XP, Vista, 7, or 8

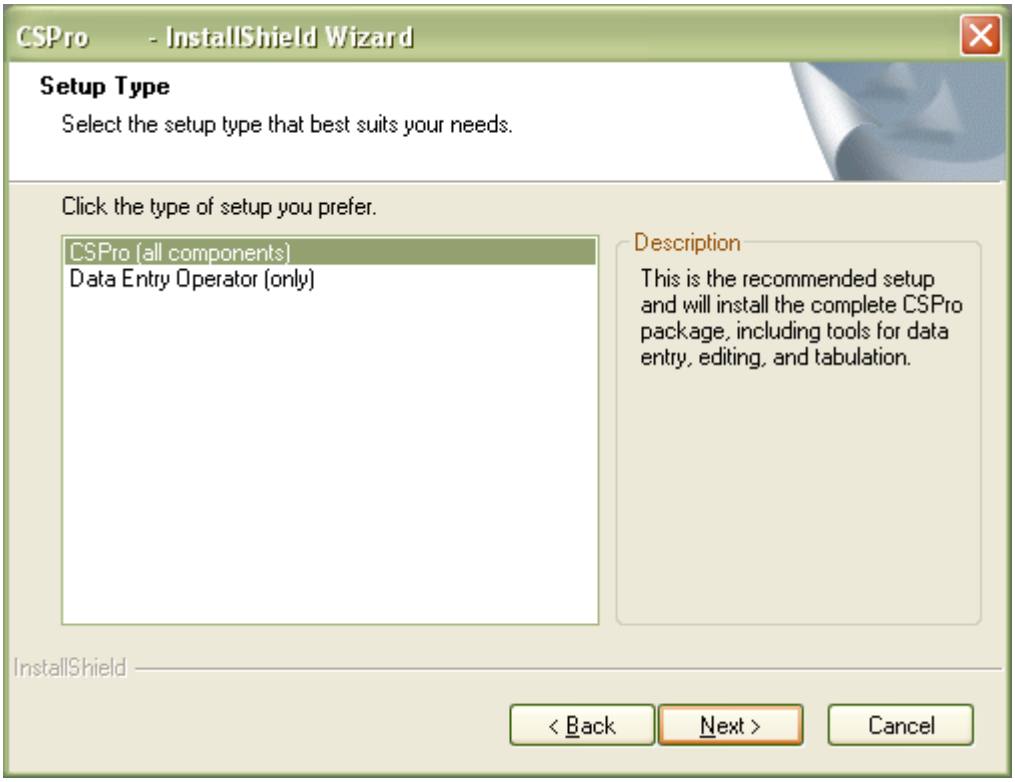
Installing CSPro

If you are installing cspro60.exe, which you downloaded from our web site:

- Double-click on the file.
- Click **OK**. The setup process takes you through a series of dialog boxes that prompt you for setup information.

Selecting components for installation:

CSPro allows you to select which components of the system you want to install. During the installation you will see the following component screen:



You have the following choices:

- **CSPro (all components)** - Select this if you plan to develop applications.
- **Data Entry Operator (only)** - Select this if you are installing a data entry application on a production machine. The operator will be able to run an already-created data entry application, but will not be able to make any changes to it.

Uninstalling CSPro

The following is based on a Windows XP setup; your steps may vary if using a different operating system.

- From the **Start** button on the taskbar, select **Settings – Control Panel**.
- Select **Add or Remove Programs**.
- From the list of currently installed programs, click on **CSPro 6.0**.
- Click the **Remove** button.
- At the prompt, click **OK** to confirm that you want to remove CSPro 6.0. The uninstall program will remove all registry entries and CSPro system files. It will not remove any applications or other files you have created.
- When the files are removed, the installation program indicates that the process is complete. Click **Finish**.

Installing a Newer Version

• Updating to CSPro 6.0 from CSPro 5.0 or earlier

If you have an old version of CSPro installed on your computer, you can install CSPro 5.0 without affecting the previous version. When you have finished your conversion of applications to CSPro 6.0, you can then uninstall the previous version.

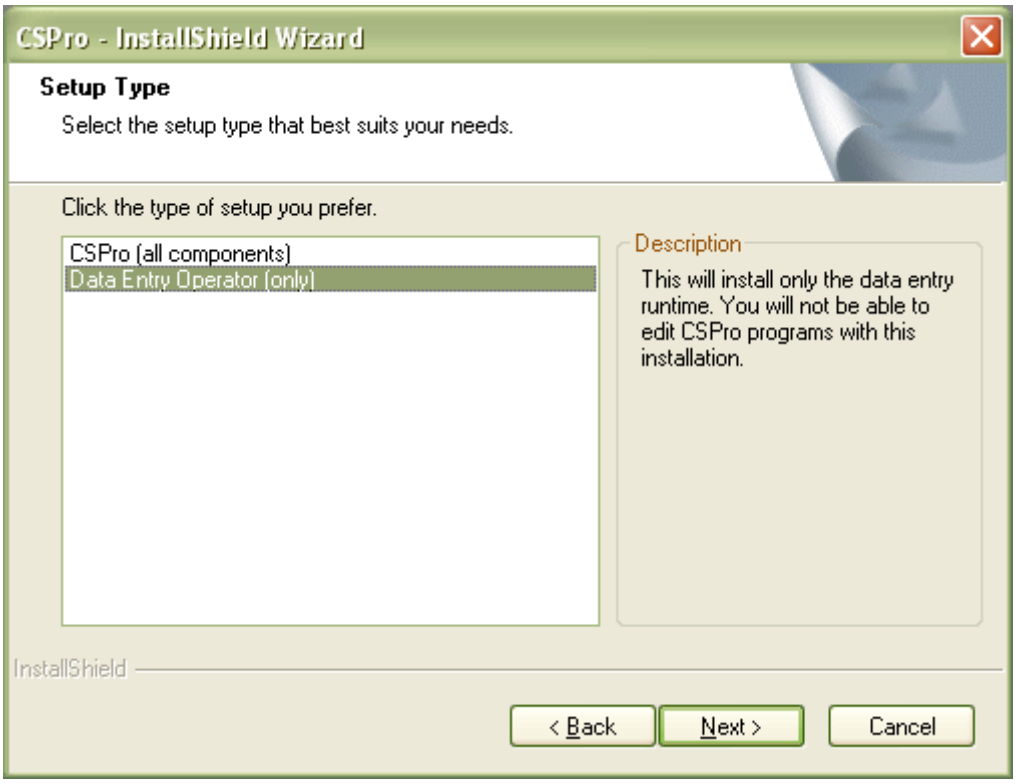
Please note that due to internal changes within CSPro 6.0, once files have been loaded in CSPro 6.0, you will no longer be able to load them in a previous version.

- **Updating a previous version of CSPro 6.0**
  If you are updating a previous version of CSPro 6.0, the new version will replace the older version. When the installation program prompts you to select the type of installation, select **Repair** and press the **Next** button. The installation program will copy the updated version of CSPro 6.0 over top the older version. It will not change any applications or other files your have created.

Installing Data Entry Applications

1) Install the CSPro Data Entry Operator Software

Run the CSPro installation program on each computer. The installation setup may be run from removable media such as a CD or USB (flash) drive, or it can be placed on a networked drive. Choose the "Data Entry Operator (only)" option during the installation. The setup program will install only the files necessary to perform data entry. The CSPro components required to modify applications will not be installed.



2) Install the CSPro Data Entry Application

Once the Data Entry Application has been tested and finalized, use the Pack Application tool to create a single file that contains all the essential application files. This file can be placed in a folder either on a network drive for multiple users or on each data entry operator's local hard drive. The application files then need to be extracted (unzipped) from the packed file.

If the application is on a network drive, multiple users can access it at the same time. Normally, only one person has access to an application on a local hard drive. The data file for an application can also be placed on a network drive or the operator's local hard drive. Two or more data entry operators cannot access the same data (or lookup) file at the same time.

Another way to distribute a data entry application is to compile it into a single combined file that contains all elements of the program. This file cannot be edited and provides security during the data entry operation. For this approach, read about Binary Data Entry Applications.

For information about data entry applications in general, see the Data Entry User's Guide.

See Also: Run Production Data Entry, Creating Limited CSPro Installations for Redistribution

Creating Limited CSPro Installations for Redistribution

At times you may wish to redistribute CSPro to others but limit them to only installing the data entry application. You can modify the standard CSPro installer so that it installs just the data entry components rather than asking the user to select the desired components to install. You can do this by calling the CSPro installation with a command line argument:

```
cspro60.exe /component=Data Entry
```

This command line argument will generally be passed to the program in a setup script. Network administrators may want to push CSPro to a group of machines without requiring a user to interact with the setup screens. To automate an installation like this, a silent command argument can be passed to the installer:

```
cspro60.exe /silent

cspro60.exe /silent /component=Data Entry
```

The first example will install the complete CSPro package. In both cases, because there is no prompt for the path of installation, CSPro will be installed to the "C:\Program Files\CSPro 6.0" folder (or "C:\Program Files (x86)\CSPro 6.0" on 64-bit machines).

To automate the removal of CSPro, use the following command:

```
cspro60.exe /silent /remove
```

See Also: Installing CSPro, Installing Data Entry Applications

Appendix B - Keys Summary

Data Dictionary Keys

Shortcuts specific to the Data Dictionary

Shortcut Key	Description
Ins	Insert level, record, item, or value at selection point.
Del	Delete level, record, item, or value.

Ctrl + A	Add level, record, item, or value to end of list.
Ctrl + D	Edit notes for this dictionary element.
Ctrl + L	Show or hide layout view.
Ctrl + M	Modify a level, record, item or value.

Shortcuts common throughout CSPro

Shortcut Key	Description
Ctrl + C	Copy the selection and put it on the Clipboard.
Ctrl + F	Find specified text.
Ctrl + J	View full screen.
Ctrl + N	Create a new document.
Ctrl + O	Open an existing document.
Ctrl + P	Print the active document.
Ctrl + S	Save the active document.
Ctrl + T	Show names instead of labels in tree.
Ctrl + U	Full screen.
Ctrl + V	Paste clipboard contents.
Ctrl + X	Cut the selection and put it on the Clipboard.
Ctrl + Y	Redo the previous undone action.
Ctrl + Z	Undo last action.
Ctrl + F4	Close the active document.
Alt + F4	Exit the application.
Alt + F6	Change focus to next window
Shift + Alt + F6	Change focus to previous window
F1	Show help contents and index.

Data Entry Keys

Shortcuts Specific to Forms Designer

Shortcut Key	Description
Del	Delete currently selected item(s).
Ctrl + Del	Delete the currently displayed form.
Ctrl + A	Add form to current level.
Ctrl + G	Generate a set of forms based on the dictionary.
Ctrl + L	View currently-selected procedure.
Ctrl + M	View currently-selected form.
Ctrl + Q	View currently-selected CAPI question.
Ctrl + R	Run the data entry application.

Shortcuts Specific to Logic View

Shortcut Key	Description
F3	Find the next occurrence of specified text.

Ctrl + K	Compile application code.
Up Arrow	Move up one line.
Down Arrow	Move down one line.
Shift + Up Arrow	Scroll up, multi-selects rows.
Shift + Down Arrow	Scroll down, multi-selects rows.
Page Up	Scroll up one screen (if possible).
Page Down	Scroll down one screen (if possible).
Shift+Page Up	Scroll up, multi-selecting pages.
Shift+Page Down	Scroll down, multi-selecting pages.
Home	Scroll to the beginning of line.
End	Scroll to the end of line.
Shift+Home	Select text from cursor to beginning of line.
Shift+End	Select text from cursor to end of line.
Ctrl+Home	Scroll to the first line of code.
Ctrl+End	Scroll to the last line of code.

CAPI Options

Shortcut Key	Description
Ctrl + B	Bold text.
Ctrl + I	Italic text.
Ctrl + U	Underline text.

Shortcuts Common Throughout CSPro

Shortcut Key	Description
Ctrl + C	Copy the selection and put it on the Clipboard.
Ctrl + F	Find specified text.
Ctrl + H	Replace the specified text with different text.
Ctrl + J	View full screen.
Ctrl + N	Create a new document.
Ctrl + O	Open an existing document.
Ctrl + S	Save the active document.
Ctrl + T	Show names instead of labels in tree.
Ctrl + V	Paste clipboard contents.
Ctrl + X	Cut the selection and put it on the Clipboard.
Ctrl + Y	Redo the previous undone action.
Ctrl + Z	Undo last action.
Ctrl + F4	Close the active document.
Alt + F4	Exit the application.
Alt + F6	Change focus to next window

Shift + Alt + F6	Change focus to previous window
F1	Show help contents and index.

Batch Edit Keys

Shortcuts specific to Batch Edit

Shortcut Key	Description
Del	Delete currently selected item(s).
F3	Find the next occurrence of specified text.
Ctrl + K	Compile code.
Ctrl + R	Run the data entry application.
Up Arrow	Move up one line.
Down Arrow	Move down one line.
Shift + Up Arrow	Scroll up, multi-selects rows.
Shift + Down Arrow	Scroll down, multi-selects rows.
Page Up	Scroll up one screen (if possible).
Page Down	Scroll down one screen (if possible).
Shift + Page Up	Scroll up, multi-selecting pages.
Shift + Page Down	Scroll down, multi-selecting pages.
Home	Scroll to the beginning of line.
End	Scroll to the end of line.
Shift + Home	Select text from cursor to beginning of line.
Shift + End	Select text from cursor to end of line.
Ctrl + Home	Scroll to the first line of code.
Ctrl + End	Scroll to the last line of code.

Shortcuts common throughout CSPro

Shortcut Key	Description
Ctrl + C	Copy the selection and put it on the Clipboard.
Ctrl + F	Find specified text.
Ctrl + H	Replace the specified text with different text.
Ctrl + J	View full screen.
Ctrl + N	Create a new document.
Ctrl + O	Open an existing document.
Ctrl + S	Save the active document.
Ctrl + T	Show names instead of labels in tree.
Ctrl + V	Paste clipboard contents.
Ctrl + X	Cut the selection and put it on the Clipboard.
Ctrl + Y	Redo the previous undone action.
Ctrl + Z	Undo last action.

Ctrl + F4	Close the active document.
Alt + F4	Exit the application.
Alt + F6	Change focus to next window
Shift + Alt + F6	Change focus to previous window
F1	Show help contents and index.

Tabulation Keys

Shortcuts specific to Tabulation

Shortcut Key	Description
Ins	Insert a new table into the table set.
Del	Delete a table from the table set.
Esc	Cancel the current selection.
Ctrl + A	Select all.
Ctrl + D	Display or hide hidden rows and columns.
Ctrl + M	Generate a thematic map. (Area only)
Ctrl + R	Run the tabulation.
Alt + A	Edit the (geographic) areas specification.
Alt+Shift+Left Arrow	Previous table.
Alt+Shift+Right Arrow	Next table.

Shortcut specific to Print View within Tabulation

Shortcut Key	Description
Ctrl + G	Go to specific page or table. (Print View only)
Ctrl + Plus (+)	Zoom in.
Ctrl + Minus(-)	Zoom out.
Alt+Home	Go to the first page of the first table.
Alt+Left Arrow	Go to the previous page.
Alt+Right Arrow	Go to the next page.
Alt+End	Go to the last page of the last table.

Shortcuts common throughout CSPro

Shortcut Key	Description
Ctrl + C	Copy the selection and put it on the Clipboard.
Ctrl + J	View full screen.
Ctrl + N	Create a new document.
Ctrl + O	Open an existing document.
Ctrl + P	Print the active document.
Ctrl + S	Save the active document.
Ctrl + T	Show names instead of labels in tree.

Ctrl + F4	Close the active document.
Alt + F4	Exit the application.
Alt + F6	Change focus to next window
Shift + Alt + F6	Change focus to previous window
F1	Show help contents and index.

Appendix C - Menu Summary

CSPro Menu

File	
New	Create a new application.
Open	Open an existing application.
Tools	
Text Viewer	View text or data files
Table Viewer	View CSPro tables.
Map Viewer	View CSPro thematic maps.
Retrieve Tables	Retrieve tables from a data set.
Tabulate Frequencies	Tabulate frequency distributions for file contents.
Sort Data	Sort cases or records.
Export Data	Export data in various formats.
Reformat Data	Reformat data using two dictionaries.
Compare Data	Compare contents of two similar data files.
Concatenate Data	Join text files one after the other.
Table Retrieval Setup	Setup table retrieval database.
Convert Dictionary	Convert an ISSA or IMPS dictionary to CSPro.
Convert Shape to Map	Convert an ESRI shape file to CSPro map file.
Pack Application	Pack entire CSPro application into a ZIP file.
Index Files	Identifies duplicate cases in a data file.
Font Preferences	Change the font preferences for the CSPro Designer.
Window	
Cascade	Arrange windows in an overlapping fashion.
Tile Top to Bottom	Arrange windows one above the other.
Tile Side by Side	Arrange windows one beside the other.
Help	
Help Topics	Get help on current application.
About	Get information about the software.

Data Dictionary Menu

File	
New	Create a new application.
Open	Open an existing application.
Close	Close an application.

Save	Save an application.
Save As	Save the current dictionary to a new file name.
Add Files	Insert a file in an existing application.
Drop Files	Drop a file from an existing application.
Page Setup	Change headers, footers, and margins for printed pages.
Print Setup	Change orientation and paper size for printed pages.
Print Preview	Preview the printed pages.
Print	Print all or part of a document.
Edit	
Undo	Undo dictionary changes.
Redo	Redo dictionary changes.
Cut	Copy selected dictionary element to clipboard and delete it.
Copy	Copy selected dictionary element to clipboard.
Paste	Paste dictionary element on clipboard to selected location.
Modify	Edit the selected dictionary element.
Add	Add a dictionary element at the end of the list.
Insert	Insert a dictionary element at the selected location.
Delete	Delete selected dictionary element.
Relations	Define relations between items and records.
Notes	Edit notes for selected dictionary element.
Find	Find a label or name with the specified text.
Convert to Subitems	Convert selected items to subitems and insert the item that contains them.
Generate Value Set	Generate a value set of intervals of a numeric data item.
View	
Names in Trees	Show names instead of labels in trees.
Full Screen	Hide the trees and show full screen view.
Layout	Show record layout of file in the window.
Options	
Relative Positions	Select whether items stay next to each other with no gaps.
ZeroFill Default "Yes"	Select whether numeric data items will have leading zeros.
DecChar Default "Yes"	Specifies whether the item should be stored in the data file with an explicit decimal character.

Data Entry Menu

File	
New	Create a new application.
Open	Open an existing application.
Close	Close an application.
Save	Save an application.
Save As	Save the active application with a new name.
Add Files	Add a dictionary or form to an existing application.
Drop Files	Drop a dictionary or form from an existing application.
Compile	Compile the logic in the application.
Run	Run the application.
Run as Batch	Run the application after finish keying data for a file.

Export Application	Generate a binary (compiled) version of the application.

Edit

Undo	Undo the most recent change.
Redo	Redo the last undo.
Cut	Copy selected element to clipboard and delete it.
Copy	Copy selected element to clipboard.
Paste	Paste element on clipboard to selected location.
Add Form	Add a form to the application.
Delete Form	Delete a form from the application.
Generate Forms	Generate forms using the Data Dictionary.
Delete	Delete selected objects.
Find	Find text in the procedures.
Find Next	Find the next occurrence of text in the procedures.
Replace	Replace text with new text in the procedures.

View

Box Toolbar	Show or hide box drawing toolbar.
Names in Trees	Show names instead of labels in trees.
Full Screen	Hide the trees and show full screen view.
Form	Shows form in right-hand window.
Logic	Show logic procedures in right-hand window.
CAPI Questions	Show screen to enter the questions.

Options

Data Entry	Change the data entry options.
Drag	Change the drag options.
Default Text Font	Change the default text font settings.
Field Font	Change the field font settings.
Set Explicit	Require declaration of all variable names in logic.

Align

Left	Position to left-most item.
Center	Center items as a group.
Right	Position to right-most item.
Top	Position to top-most item.
Mid	Align mid-points of items as a group.
Bottom	Position to bottom-most item.
Evenly Horizontal	Space evenly left to right.
Evenly Vertical	Space evenly top to bottom.

CAPI Options

Use CAPI Font 1	For CAPI text use first default font.
Use CAPI Font 2	For CAPI text use second default font.
Bold	Switch between bold and non-bold text.
Italic	Switch between italic and non-italic text.
Underline	Switch between underlined and non-underlined text.
Color	Change color of text.
Left	Align selected text to left of question window.

Center	Align selected text to center of question window.
Right	Align selected text to right of question window.
Bullets	Add bullets to selected or new text.
Insert Bitmap	Insert bitmap (BMP file) image in question window.
Help Text	Switch display between question text and help text.
Define CAPI Languages	Define the set of CAPI languages used in application.
Change CAPI Font 1	Select the first default font for a CAPI language.
Change CAPI Font 2	Select the second default font for a CAPI language.
Fit Windows to Questions	Adjust question window to display entire text.

Batch Editing Menu

File	
New	Create a new application.
Open	Open an existing application.
Close	Close an application.
Save	Save an application.
Save As	Save the active application with a new name.
Add Files	Add a dictionary to an existing application.
Drop Files	Drop a dictionary from an existing application.
Compile	Compile the logic in the application.
Run	Run the application.
Edit	
Undo	Undo latest cut/copy/paste operations.
Redo	Redo the latest undo operations.
Cut	Cut logic and place it on the clipboard.
Copy	Copy logic and place it on the clipboard.
Paste	Paste logic from the clipboard.
Find	Find text in the logic.
Find Next	Find the next occurrence of text in the logic.
Replace	Replace text with new text in the logic.
View	
Names in Trees	Show names instead of labels in trees.
Full Screen	Hide the trees and show full screen view.
Options	
Custom Order	Allow user defined order of editing.
Set Explicit	Require declaration of all variable names in logic.

Tabulation Menu

File	
New	Create a new application.
Open	Open an existing application.
Close	Close an application.
Save	Save an application.
Save As	Save the active application with a new name.
Save Tables	Save current table results in a file.
Add Files	Add a dictionary to an existing application.

Drop Files	Drop a dictionary from an existing application.
Load Preferences	Load stored table formatting preferences.
Save Preferences	Save table formatting preferences.
Run	Run the application.
Run Parts	Run Tabulate, Consolidate, or Format process.
Page Setup	Change headers, footers, and margins for printed pages.
Print Setup	Change orientation and paper size for printed pages.
Print Preview	Preview the printed pages.
Print	Print all or part of a document.
Edit	
Add Table	Add a table at the end.
Insert Table	Insert a table at the current location.
Delete Table	Delete the current table.
Tally Attributes (Table)	Edit the universe or filter of tabulation.
Area	Enable or change tabulation by geographic area.
Format (Table)	Edit formats that apply to the entire table.
Format (Application)	Edit formats that apply to the entire application.
Format Print (Table)	Edit formats for the printed tables.
Go to	Go to a particular page, table, or area.
Copy	Copy selected tables cells and headings to the clipboard.
Copy Cells Only	Copy only the selected table cells to the clipboard.
Copy Table Spec	Copy table specification to the clipboard
Paste Table Spec	Paste table specification into application
Select All	Select the entire table.
Cancel Selection	Cancel the currently selected cells.
Preferences	Change application format and tally preference.
View	
Names in Trees	Show names instead of labels in trees.
Full Screen	Hide the trees and show full screen view.
Hidden Parts	Show or hide hidden parts of a table.
Previous Table	Show the previous table.
Next Table	Show the next table.
Zoom In	Zoom in on a print preview.
Zoom Out	Zoom out on a print preview.
Facing Pages	Show facing pages (two) on a print preview.
Multiple Pages	Show selected number of pages in print preview.
First Page	Show first print page of first table.
Previous Page	Show previous print page.
Next Page	Show next print page.
Last Page	Show last print page of last table.
Map	Create thematic map of selected cells.

Appendix D - Toolbar Summary

CSPro Toolbar

The CSPro toolbar is displayed across the top of the window, below the menu bar. It provides quick mouse access to many features used in CSPro.

Click To

Create a new application.
Open an existing application
Get help.

The CSPro toolbar only appears when CSPro is opened without specifying an application or file. When applications or files are open, the toolbar corresponding to the contents of the right-hand screen appears.

Data Dictionary Toolbar

The Data Dictionary toolbar is displayed across the top of the window, below the menu bar. It provides quick mouse access to many features used in the Data Dictionary. It is available whenever the right-hand screen is displaying dictionary items

Click To

Create a new dictionary.
Open a dictionary.
Save a dictionary.

Set up page margins and headings for printing.
Preview contents of the dictionary.
Print contents of the dictionary.

Undo the last change to dictionary.
Redo last undo.
Cut the selected records, items, or values to the clipboard.
Copy the selected records, items, or values to the clipboard.
Paste the contents of the clipboard to the current position.

Add levels, records, items, values sets, or values.
Insert levels, records, items, values sets, or values.
Delete levels, records, items, value sets, or values.

Edit Notes for dictionary, level, record, item, value set, or value.
Find a label or a name in the dictionary.
Show the Layout window.

Show last Dictionary window.
Show last Forms window.
Show last Batch Edit window.
Show last Tabulation window.

Get Help.

Data Entry Toolbar

The Forms Designer toolbar is displayed across the top of the window, immediately below the menu bar. The toolbar provides quick mouse access to many of the often-used features found in the Forms Designer.

Click To

Create a New application.
Open an application.
Save an application.
Compile the logic (code) of your data entry application.
Run the current data entry application (i.e., start up CSEntry).

Undo the latest changes.
Redo the latest changes.

Cut the selected elements to the clipboard.
Copy the selected elements to clipboard.
Paste the contents of the clipboard to the form.

Delete the currently selected item(s).
Find text in logic.

Toggle between selecting item(s) or drawing boxes .
View the forms
View the logic
View the CAPI question

Show last Dictionary window.
Show last Forms window.
Show last Batch Edit window.

Show last Tabulation window.

Get Help.

Batch Editing Toolbar

The Edit Designer toolbar is displayed across the top of the window, immediately below the menu bar. The toolbar provides quick mouse access to many of the often-used features found in the Edit Designer.

Click To

Create a New application
Open an application
Save an application

Compile the logic (code) of your application
Run the current batch edit application

Undo the latest text changes.
Redo the latest text changes.

Cut the selected logic to the clipboard.
Copy the selected logic to clipboard.
Paste the contents of the clipboard to the logic.
Find text in the logic

Launch Text Viewer

Show last Dictionary window.
Show last Forms window.
Show last Batch Edit window.
Show last Tabulation window.

Get Help.

Tabulation Toolbar

The Tabulation toolbar is displayed across the top of the window, immediately below the menu bar. The toolbar provides quick mouse access to many of the more frequently-used features found in Tabulation.

Click To

Create a New application.

Open an existing application.
Save an application.
Save tables.

Setup page margins and headings for printing.
Preview printing of tables.
Print tables.

Run a tabulation.

Add a table.
Insert a table.
Delete the current table.

Specify Tally Attributes for the table.
Define Area Processing variables.

Copy a selection from a table.

Show Previous Table.
Show Next Table.

Use the MapViewer

Show last Dictionary window.
Show last Forms window.
Show last Batch Edit window.
Show last Tabulation window.

Get Help.

Appendix E - Converting Within IMPS or ISSA

Converting a data dictionary

CSPro will convert your existing IMPS 3.1, IMPS 4.1, or ISSA dictionary (any version) to the CSPro dictionary format. You can also save your CSPro dictionary file out as an IMPS 3.1 or ISSA dictionary. Either way, just do the following:

• Click the **Tools** menu, and then click **Convert Dictionary**.

• Choose whether you are converting between CSPro and IMPS, or CSPro and ISSA, then press **Next**.

• Choose the type conversion you are performing and press **Next**.

• Choose the file name of the dictionary you are converting.

• Specify the file name of the dictionary you are generating.

• Press **Finish** when ready.

Converting within IMPS

• **Convert an IMPS 3.1 (DOS) data dictionary to CSPro**

- Hypens (dashes) within record and item names are changed to underlines.

- All common items in IMPS are changed to ID items in CSPro.

- As CSPro does not allow the use of subitems in the ID section, any subitems found in the IMPS common section will be converted to ID items. This could increase the length of the ID section. Therefore we advise you to check your CSPro dictionary after conversion if you receive a message about "subitems found in common, promoted to ID items". Similarly, if you receive any error messages during the conversion about overlapping items, this is most likely due to the use of subitems in the common block of IMPS.

- If you have used "NR" as a value name in your IMPS dictionary, it will be converted to the special value "missing."

- If you have used "NA" as a value name in your IMPS dictionary, it will be converted to the special value "notappl."

• **Convert an IMPS 4.1 (Windows) data dictionary to CSPro.**

- All hierarchy information is lost.

- All other comments as listed under IMPS 3.1 to CSPro apply.

• **Convert a CSPro data dictionary to IMPS 3.1 (DOS).**

- Underlines within record and item names are changed to hyphens (dashes).

- All level record, item, and value set labels are lost.

- All value labels are truncated to 16 characters.

- Value sets, after the first, are converted to subitems.

- All notes are lost.

- All level information is lost.

- All special values (Missing, NotAppl, and Default) are lost.

- CSPro allows the use of 32 characters as a unique name—however, IMPS only allows 16 characters. Therefore, if any of your unique names are longer than 16 characters, you will receive a message for each variable that its name will be truncated to 16.

- CSPro does not support the use of short names, as described in IMPS. Therefore, if the first 8 characters of any item name in CSPro are not unique, the converted IMPS dictionary will have

duplicate shorts names. You should check the converted IMPS dictionary to ensure that there are no duplicate short names.

Converting within ISSA

In an ISSA data entry application, a dictionary file contains both descriptive information about each data item (item type, item length, etc), as well as information about the form layout. However, in CSPro, we break this information into two separate files. Therefore, if you:

- Convert an ISSA data dictionary to CSPro data dictionary only, you will receive a CSPro dictionary (.dcf) only. You can choose to convert either a regular data dictionary (.dic) or a working storage dictionary (.wst).

- Convert an ISSA data dictionary to CSPro form and data dictionary, you will receive both a CSPro dictionary (.dcf) and a CSPro form file (.fmf). You can choose to convert either a regular data dictionary (.dic) or a working storage dictionary (.wst).

Converting an IMPS Data Entry Application

CSPro provides a utility to convert existing IMPS data dictionaries to CSPro. However, there is no automated tool to convert CENTRY application files. You must create the forms again in CSPro.

At any time in CSPro (you needn't have anything open), you can go to the **Tools** menu option. Select **Convert Dictionary** from the drop-down menu, then choose to convert **Between CSPro and IMPS** in the opening dialog box.

CSPro is much less constrained than CENTRY in the relationship between dictionary records and data entry forms. In CSPro you can mix dictionary items from different records on the same form. See "Issues to Consider When Designing a Form" for more details.

In CSPro you can make the equivalent of the CENTRY "Batch", "Questionnaire", and "Record" screens. If you use this approach, you must be sure to make all the fields on the "Batch" screen persistent.

Converting an ISSA Data Entry Application

If you have an existing ISSA Dictionary that contains forms, CSPro provides a utility to convert it to a CSPro Form File (a dictionary will be generated as well).

At any time in CSPro (you needn't have anything open), you can go to the **Tools** menu option. Select **Convert Dictionary** from the drop-down menu, then choose to convert **Between CSPro and ISSA** in the opening dialog box.

Next, state that you'd like to convert from **ISSA to a CSPro Forms and Data Dictionary**. Provide the name of the original ISSA dictionary file, and the name you would like to call the CSPro form file to be generated (a CSPro dictionary file will also be created, and its name will be based on the form file name). Press **OK** when ready and the files will be created for you. You are then ready to fine-tune the layout of the forms as desired.

Note that this creates a stand-alone dictionary and form file; it does **not** create a data entry application. Until there exists a data entry application, you cannot write logic for the form variables, nor can you enter data based on this form file. If you would like to generate a data entry application, proceed as you would for creating a new data entry application. When you are asked to provide the name of the form file, simply use the same form file name that was used during the conversion, and CSPro will complete the task.

Appendix F - Errors in Censuses and Surveys

The Nature of Census and Survey Data

A census is a complete count of a given entity, whether population, housing, agricultural holdings, businesses, or other, in a given geographic area at a given time. Because of financial and time constraints, the information collected in a census is usually basic and limited. A survey is admin istered to a subset of a population to obtain detailed information about certain groups, such as school-age children or mothers, or to obtain other information, on a sample basis, about the entity in question (households, businesses, agricultural establishments, etc.).

A survey can provide valuable infor ma tion about the population being studied, but because of the limitations of samples, the results can be generalized only for relatively high-level geographic areas, such as the country as a whole, or (depending on the way in which the sample was selected) for specific regions or areas of the country (e.g., urban vs. rural). A census, however, attempts to cover the entire geographic area of the subject population, so it can provide reliable information at very low levels of geography. In addition to counts (e.g., number of people, number of housing units, number of farms, number of businesses, etc.), a census usually provides a profile of additional related characteristics such as fertility, housing quality, acreage, number of employees, and so on.

Census and survey data are used to plan for education, health facilities, administration, and other needs. In order to implement programs and activities, statistics are needed by government administrators and by private users, including businesses, industries, research organizations, and the general public. These statistics also may serve as measures of existing conditions for small areas, providing a basis for planning development programs, and perhaps establishing a basis for action.

In order to obtain an accurate census or survey, the data must be as free as possible from errors and inconsistencies. Statistics derived from 'dirty data' (that is, data which still contain errors) may produce an inaccurate profile of the country or geographic area. Therefore, before any tabulation programs are run, the data should be checked for errors and changed so that important data items are valid and consistent. This is not to say that correction of data after they are collected can compensate for poorly collected data. It is not practical (if not impossible) to produce a data file which is 100 percent error free. Every effort at accuracy should be made in all stages of the census or survey.

Errors in Censuses and Surveys

- **Type of Errors**

  Editing is the process of maximizing the quality of data, in the shortest time possible, while minimizing the introduction of new errors. The process involves a number of sequential, interrelated activities as shown below. During each activity errors may occur.

  - Enumeration Respondent Errors, Enumerator Errors
  - Field Editing Field Checking, Office Checking
  - Office Coding Miscodes
  - Data Capture Miskeys, Incorrectly Scanned
  - Computer Editing Logic Errors, Misallocation, Miscorrection
  - Tabulation Distribution of Unknowns
  - Publication Misprints

- **Errors in Enumeration**

  Two types of errors occur at the enumeration stage: the respondent sometimes errs when giving information to the enumerator either by offering what the respondent believes to be the "proper response" [as opposed to a truthful response] to the questions; or by misunderstanding the question; and the enumerator, in asking the questions, recording the responses, and reviewing entries at the conclusion of the interview, also may add errors to the data.

  Little can be done to improve the quality of responses from individuals, except through publicity for the census and well-trained enumerators who explain the purpose of the census and reasons for asking

the various questions. The quality of enumerators and enumerator training often can be the crucial factors in census processing. Enumerators must be properly trained in all relevant aspects of census procedures and made to understand why their part of the census is important, and how the enumeration fits in with the other stages of the census. Pretesting should be used to eliminate problems in the questionnaires and materials, and to help enumerators obtain the data and complete the enumeration in the allotted time. Also, since enumerators come from many different backgrounds and have varying levels of education and training, training must be developed to make certain the enumerators know how to ask the questions to obtain an unbiased response.

- **Errors in Field Editing**

  A general rule for editing could be: the closer that corrections are made to the source of the data the higher the quality of the final statistics. Field editing is, therefore, perhaps the most important stage in the census processing. Supervisors, in addition to training enumerators, must also be able to collect the data themselves and to correct enumerator errors while the forms, the respondents, and the enumerators are still available. Questions that arise can then be answered before the questionnaires are sent to the central census office.

  Once the forms leave the enumeration areas, changes can no longer easily be made with the knowledge and help of the respondents and enumerators, so other procedures must be used to cope with inaccurate, incomplete, or inconsistent data. During preliminary census office editing, checks of crucial entries must be carried out quickly to determine completeness and consistency in the collected data. Highly-aberrant forms may be sent back to the field, if time and money permit. Place codes must be checked for validity, and relationships between numbers of expected individuals as recorded on the household form, and the actual numbers of individual forms (if individual forms are used) must agree.

- **Errors in Coding**

  Precise, detailed instructions for coding in preparation for manual and computer editing must be determined after the tabulation plans are developed, but before the enumeration is actually undertaken. Back in the census office, it is no longer possible to make corrections while in contact with the respondents, so editing must be determined on the basis of assumptions about what the most probable response would be. If computer editing is possible, manual office editing should be minimized to checking for completeness.

  If census data are collected on precoded questionnaires (coded either by respondents or enumerators), and machines are used to convert the information to computer-readable data, then except for the introduction of errors due to stray marks or physical problems with the questionnaires, the errors found should be minimal.

  Errors may occur when the data are coded, since the coder may miscode some piece of information. If the miscode is invalid, it should be caught during the computer editing; if the code is valid but incorrect (for example, if two digits are reversed for the entry for birthplace), the computer will not note the errors, and the information will remain incorrect for the tabulations. Coders must be trained to edit according to the edit specifications, and efforts must be made to obtain and maintain quality of coders, 'weeding out' inefficient inaccurate coders. Spot checks and verification of samples from each coder can help to identify persistent coding errors.

- **Errors in Data Capture**

  Data capture is the process of converting data to a form which the computer can use. The most common method used to convert the data is keying; scanning technologies [optical character readers (OCR) and optical mark readers (OMR)] are increasingly used, and while these methodologies appear to offer lower costs and reduced time when compared with operator-based entry, in fact each presents a separate set of difficulties, different from those associated with manual keying, that must be overcome, and which can nullify any perceived gains.

  With keyer-based entry, errors are introduced into the data through miskeying. Verification (rekeying or double-keying) can reduce these errors. A system called "intelligent data entry" [IDE] may be used to

prevent invalid entries from ever getting into the system. An IDE system ensures that the value for each field or data item is within the permissible range of values for that item. Such a system increases the chance that the data entry operator will key in reasonable data and relieves some of the burden on later stages of the data preparation process.

With scanning technologies, errors can be more insidious, and proper verification is more time-consuming, because it will require either manual comparison between the information captured by the scanner and the forms, or the establishment of a separate keying operation so that keyed output can be compared with scanned output. It is extremely important not to assume that the use of "advanced" technology reduces or eliminates the need for verification; errors can and will occur, so they must be caught early in the cycle of capture so that corrective measures (technological or manual) may be applied. If this is not done, systemic error can corrupt the data beyond repair.

- **Errors in Computer Editing**
  The high degree of accuracy and uniformity obtainable with computer editing cannot be achieved through manual editing. In computer editing, range checks and within-record consistency checks can easily be made; between-record edits can also be done if the computer programs have this capability; and unknown information can be allocated automatically. If an allocation method is used, as much of the original information as possible must be retained.

  Computer edit checks have been used in almost all censuses carried out since the 1980 round. For proper implementation of this tool, there must be good communication between the subject-matter specialists and the programmers. Subject-matter specialists should write complete and clear edit specifications. Programmers should review these specifications and work closely with subject-matter specialists to resolve questions or difficulties in implementing the specifications. Programmers also should make sure that subject-matter specialists are involved in testing the edit programs, that is, in providing test data and reviewing the outputs to insure that all the necessary edits were included in the speci fi ca tions. It is the programmers' responsibility to produce an edit program free of errors. If these programs are inadequately thought out or not completely tested, existing errors in the data may not be corrected and even more errors may be introduced.

- **Errors in Tabulation**
  Errors can occur at the tabulation stage due to improper programming or use of unknown information. Errors at this stage are difficult to correct without introducing new errors.

- **Errors in Publication**
  Errors can occur at the publication stage through lack of inter-tabulation checking, or through printing errors. If errors are carried through all stages of the process to publication, they will be apparent and the results will be of questionable value. Most importantly, obvious errors at this stage diminish the credibility of the organization presenting the data. Finally, it is very important that error analysis be done to help in interpreting the extent and kind of errors in the census and to aid in preparing for future censuses and surveys.

Appendix G - File Types

File Types

Data Entry Application

ENT - Data Entry application
ENT.APC - Logic
ENT.MGF - Message
ENT.QSF - Question
FMF - Forms
DCF - Data Dictionary

IDX - Data file index (binary format)
LST - Listing of data entry activity for each data file
LOG - Operator statistics
NOT - Notes for data (putnote, getnote, editnote functions)
STS - Status of data (partial save, verification)

Batch Edit Application
BCH - Batch Edit application
BCH.APC - Logic
BCH.MGF - Message
ORD - Edit Order
DCF - Data Dictionary

LST - Listing (errmsg function output)
FRQ - Impute frequencies (impute statement)
BCH.SVA - Saved arrays

Tabulation Application
XTB - Tabulation Application
XTB.APC - Logic
XTB.MGF - Message
XTS - Tabulation specification
DCF - Data Dictionary

ANM - Area names
LST - Listing (errors in tabulation)
TBW - Output tables (for Table Viewer)
MPC - Map file (compressed)
MDF - Map Data file

TAB - Table Matrices File
TAI - Table Matrices Index File

Other Application Files
PFF - Program Information (used to launch applications)

Importing Data to CSPro Format

There are occasions, such as when using lookup files, when you will want to convert data created in another application to a format that CSPro can read. This is a two-step process:

1. Convert the data to a flat file with one record per line and all records for a case organized together. (Read more about Data Requirements or the File Structure.)
2. Create a CSPro dictionary that describes this data file.

There are many formats from which data can be converted, but converting data to CSPro format from Microsoft Excel will be highlighted due to the popularity of that software package. To convert data to CSPro format:

1. Open a file in Excel. We will demonstrate this conversion using the following file:

	A	B	C	D	E
1	**Date**	**Establishment**	**Amount**	**Exchange Rate**	**Dollar Amount**
2					
3	11/10/2009	Hotel Intercontinental	KES 1,500.00	0.013387	$20.08
4	11/12/2009	China Anhui Huaren Restaurant	KES 1,000.00	0.013387	$13.39
5	11/13/2009	Alliance Française	KES 100.00	0.013387	$1.34
6	11/15/2009	Kenya Forest Service	KES 150.00	0.013387	$2.01
7	11/19/2009	Waamo Pizza & Fries	KES 750.00	0.013388	$10.04

2. Remove any header information in the file. CSPro data files should contain only data and thus any metadata (e.g., column headings) should be removed:

	A	B	C	D	E
1	11/10/2009	Hotel Intercontinental	KES 1,500.00	0.013387	$20.08
2	11/12/2009	China Anhui Huaren Restaurant	KES 1,000.00	0.013387	$13.39
3	11/13/2009	Alliance Française	KES 100.00	0.013387	$1.34
4	11/15/2009	Kenya Forest Service	KES 150.00	0.013387	$2.01
5	11/19/2009	Waamo Pizza & Fries	KES 750.00	0.013388	$10.04

3. All fields must be right-justified:

	A	B	C	D	E
1	11/10/2009	Hotel Intercontinental	KES 1,500.00	0.013387	$20.08
2	11/12/2009	China Anhui Huaren Restaurant	KES 1,000.00	0.013387	$13.39
3	11/13/2009	Alliance Française	KES 100.00	0.013387	$1.34
4	11/15/2009	Kenya Forest Service	KES 150.00	0.013387	$2.01
5	11/19/2009	Waamo Pizza & Fries	KES 750.00	0.013388	$10.04

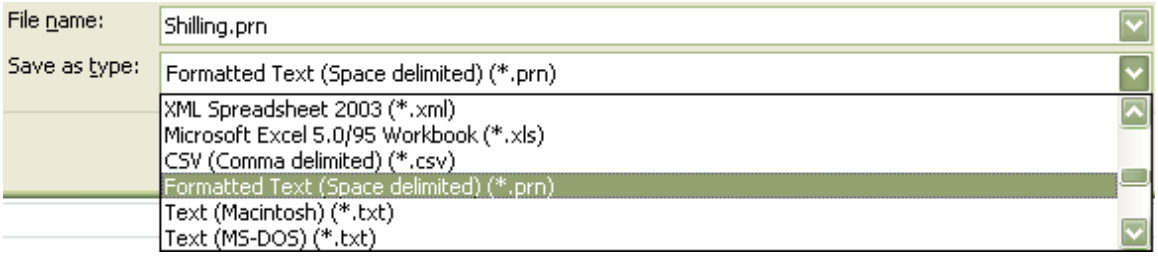
4. Remove any specialized formatting attached to cells (unless that formatting is desired in the final output):

	A	B	C	D	E
1	11/10/2009	Hotel Intercontinental	1500.00	0.013387	20.08
2	11/12/2009	China Anhui Huaren Restaurant	1000.00	0.013387	13.39
3	11/13/2009	Alliance Française	100.00	0.013387	1.34
4	11/15/2009	Kenya Forest Service	150.00	0.013387	2.01
5	11/19/2009	Waamo Pizza & Fries	750.00	0.013388	10.04

5. Set the column widths to appropriate values. When this information is finally saved, the width of each column determines the length of each item in your CSPro dictionary.

	A	B	C	D	E
1	11/10/2009	Hotel Intercontinental	1500.00	0.013387	20.08
2	11/12/2009	China Anhui Huaren Restaurant	1000.00	0.013387	13.39
3	11/13/2009	Alliance Française	100.00	0.013387	1.34
4	11/15/2009	Kenya Forest Service	150.00	0.013387	2.01
5	11/19/2009	Waamo Pizza & Fries	750.00	0.013388	10.04

6. Save As, choosing "Formatted Text (Space delimited) (*.prn)" as the output format:



Using the above example, the resulting file looks like this:

```
11/10/2009       Hotel Intercontinental  1500.00  0.013387 20.08
11/12/2009China Anhui Huaren Restaurant  1000.00  0.013387 13.39
11/13/2009           Alliance Française   100.00  0.013387  1.34
11/15/2009         Kenya Forest Service   150.00  0.013387  2.01
11/19/2009          Waamo Pizza & Fries   750.00  0.013388 10.04
```

7. Create a CSPro dictionary that describes this data. The length of each item is determined by the column width specified in Excel. Other options, such as whether a decimal character should exist, or whether numeric values should be zero-filled, can also be specified in Excel.

It is a good idea to run frequencies on each item to make sure that the dictionary you created properly describes the .prn file.

The above example is a rather simple one, but more complex imports, including importing cases with multiple records (and thus a record type), are also possible.

Locking Application Files

If distributing a CSPro application to users, one simple, though not rigorous, way of preventing the users from modifying the applications is to add the command [NoEdit] at the top of either of these files:

- .ent
- .bch
- .xtb
- .dcf
- .fmf

For instance:

```
[NoEdit]
[CSPro Application]
Version=CSPro 6.0
```

When the user tries to open any such data file, the CSPro Designer will give an error message. However, the user can open the files in the context of running an application, whether that is running a batch

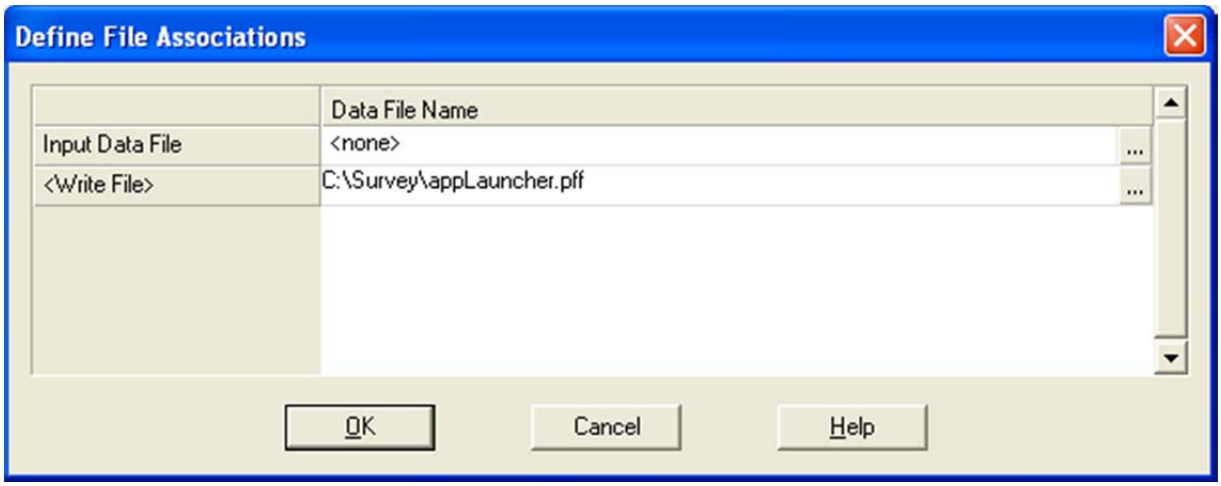
program or exporting data. This setting only affects the CSPro Designer. It is not particularly robust protection, as a knowledgeable user can remove [NoEdit] from the file, but it will protect against most users modifying application files. With data entry applications, .pen files can be distributed for more protection against editing.

Temporary Data File

Most CSPro applications sensibly require data files to operate but there may be times when the contents of a data file are unimportant. Examples of this include a menu program that launches another program based on a user's selection. Such a program generally has no output and thus the contents of the file are meaningless. For batch programs it may be the case that you want to use CSPro functions to perform some task, such as creating content using the write statement. CSPro can create a temporary data file in the Windows temporary folder that will eventually be removed by the operating system. This file will satisfy CSPro's file requirements but will not persist in any working folders.

In the Define File Associations box write "<none>" for the Input Data File. If your data entry application application does not use an external file or a write file you will not be presented with a Define File Associations box and must edit the PFF file manually. To do this, remove the "InputData=..." line from the files section of the PFF.

Data entry applications run as they would with a file specified. Batch applications, however, do not run any logic associated with elements of the dictionary. Instead only the application file preprec and postproc are executed.



Batch Edit Example

```
PROC GLOBAL

// This program concatenates all the *.dat files in a directory by order of
date.
// The program can be run with any dictionary, it is only necessary that the
name
// of the application file procedure is valid.

alpha (300) str;
alpha (50) listingName = "files.txt";
alpha (50) pffName = "CSConcat.pff";
alpha (50) outputName = "concatenated.dat";
```

```
file inputFile;
file pffFile;


PROC CONCATENATE_FF

    setfile(pffFile,strip(pffName));

    filewrite(pffFile,"[Run Information]");
    filewrite(pffFile,"Version=CSPro 6.0");
    filewrite(pffFile,"AppType=Concatenate");
    filewrite(pffFile,"[Files]");

    str = maketext('cmd /c "dir /b /od *.dat > %s"',strip(listingName));
    execsystem(str,wait);

    setfile(inputFile,strip(listingName));

    while fileread(inputFile,str) do
        filewrite(pffFile,"InputData=.\%s",strip(str));
    enddo;

    close(inputFile);
    filedelete(strip(listingName));

    filewrite(pffFile,"OutputData=%s",outputName);
    filewrite(pffFile,"Listing=.\CSConcat.lst");
    filewrite(pffFile,"[Parameters]");
    filewrite(pffFile,"ViewListing=No");
    filewrite(pffFile,"ViewResults=No");

    close(pffFile);

    execpff(strip(pffName));
```

Files Description

Data Dictionary File (.DCF)

- Each data file manipulated by CSPro must be described by a data dictionary. The data dictionary file contains information defining the layout of a data file, including levels, records, items, value sets and values.

- CSPro allows you to explicitly open, close and save data dictionary files independently of other application files. You must be careful when you do so if more than one application uses the data dictionary.

- CSPro applications may optionally contain data dictionaries which represent secondary files, such as Look-up files, which are opened during data entry.

- The data dictionary file is a text file which may be viewed with any text editor, such as CSPro's Text Viewer or the Windows Notepad. It is not recommended to make changes to this file outside the CSPro environment.

Binary Data Entry Application File (.PEN)

• A Binary data entry application file is an alternative to the normal set of text data entry application files. It is one file that contains the same information as the set of text files, including the data dictionaries, forms, logic and CAPI question text. It cannot be viewed with any text editor.

• This file can be generated in the CSPro designer and opened by CSEntry, on Windows machines or Android devices. It cannot be opened by the CSPro designer.

• Binary data entry application files provide security during a data entry operation. If you distribute this file to the data entry workstations, the operators cannot change the application, even if they have the full CSPro installed on their machines.

• Binary data entry applications created in versions of CSPro prior to 6.0 had the extension .ENC, not .PEN. These files cannot be read by this version of CSPro.

Data Entry Application File (.ENT)

• The Data Entry Application file is the master file for a data entry application. This file specifies all other files contained in the application, along with other information.

• CSPro allows you to open, close and save data entry application files. When you do so, all other files associated with the application are also opened, closed or saved.

• The application file is a text file which may be viewed with any text editor, such as CSPro's Text Viewer or the Windows Notepad. It is not recommended to make changes to this file outside the CSPro environment. Advanced users might do so, however, to change the names of associated files from the CSPro assigned defaults.

Form File (.FMF)

• The forms file contains information about forms, their fields, text and rosters. The forms file also contains the name of the associated data dictionary file. Fields and rosters have links into the data dictionary.

• The flow during data entry, that is, the order in which forms and fields are entered, is defined in the forms file, not in the data dictionary.

• CSPro allows you to explicitly open, close and save forms files independently of the application file. When you do so, the associated data dictionary file is also opened, closed or saved.

• Note that if you open a forms file you will not have access to its application's logic. Generally, only advanced users open forms files explicitly.

• The forms file is a text file which may be viewed with any text editor, such as CSPro's Text Viewer or the Windows Notepad. It is not recommended to make changes to this file outside the CSPro environment. Advanced users might do so, however, to change the name of the associated data dictionary file.

Logic File (.APC)

- The logic file contains all the CSPro language statements which control the application. There is one logic file associated with each application.

- CSPro does not allow you to explicitly open the logic file. It is opened only when you open its associated application.

- By default, the logic file has the same name as the application file, just a different extension. This is not a requirement, however. Advanced users who change the name of this file must also remember to change the corresponding name in the application file.

- The logic file is a text file which may be viewed with any text editor, such as CSPro's Text Viewer or the Windows Notepad. While you may make changes to this file outside the CSPro environment, CSPro provides a powerful text editor which is integrated with the CSPro compiler.

- Prior to CSPro 6.0, logic files had the extension .app. Though the default extension changed, CSPro can still read applications that have logic files with the .app extension.

Messages File (.MGF)

The message file is a text file where you can store message text and an associated message number. Each line in the message file contains one message. A message consists of a message number followed by text which can be up to 240 characters long. It is displayed when an "errmsg" function with the message number is executed in a data entry application. A message may contain parameters.

Question File (.QSF)

- The question file contains information related to **CAPI** (**C**omputer **A**ssisted **P**ersonal **I**nterviewing) data entry applications. Such information includes question text to appear on the screen with each field and help screens to appear when the operator presses the help (**F2**) key.

Data File

The data file is a text file encoded using UTF-8. Data files are limited to 2 gigabytes in length. A data file can have any extension and CSPro does not assign an extension by default. Read the Unicode Primer for more information about how data is stored in the data file.

Data File Index (.IDX)

The data file index is a binary file that contains an index of case ids used by the CSPro Data Entry application to insure that each case has a unique set of case ids. It is also used by all applications for external files to locate cases with the file.

Notes File (.NOT)

The notes file is a text file that contains all notes processed by the data entry application. The file contains the case ids, name of the data item to which each note belongs, and the note text.

The notes file stores all the notes entered by data entry operators for its corresponding data file. Each note is identified by its case ids and field name. Each record in the NOT file contains note text for one of

the fields. There may be multiple note records for a particular filed. All the note text for a particular field follow one another in order in the NOT file.

The format of each record is as follows:

- The first n characters are the id fields of the case.
- The next 32 characters are the field name as given in the dictionary (left justified).
- The next 12 characters are blank (reserved for future use).
- If field has record occurrences then the next 5 characters are the record occurrence number (right justified, blank filled). If the field has only one record occurrence then the next file characters are blank.
- The remaining characters are note text.

Note text can span multiple NOT file records. Each record will have the same initial fields (as described above) followed by the continued note text. Note text is always broken at word (space boundaries. There is an implied blank character between the note text in each note record for a field.

Note text can contain "\n" characters indicating a new line (Enter key). If the note text is automatically wrapped in the note text box, there will be no new line characters.

The NOT file could be processed by another CSPro application by creating data dictionary for it.

Notes can be created, editing, and deleted using the getnote, putnote, and editnote functions of CSPro.

Data File Status (.STS)

The data file status file is a text file that contains information about the partial save and verification status of the data file.

If the data file contains any partial save cases, the STS file identifies the ids of the cases were partially saved and under which mode (add, modify, or verify) they were partially saved.

If the data file was verified, the STS file identifies the ids of the last case verified.

Listing File (.LST)

Almost any time that a CSPro application or tool is run a listing file is generated. It contains run information about data files and results. The listing file associated with a Batch Edit application is the usually the primary output of the applications. For most other applications and tools the listing file is a secondary output.

Tabulation Application File (.XTB)

• The Tabulation Application file is the master file for a Tabulation application. This file specifies all other files contained in the application, along with other information.

• CSPro allows you to open, close and save Tabulation application files. When you do so, all other files associated with the application are also opened, closed or saved.

• The application file is a text file which may be viewed with any text editor, such as CSPro's Text Viewer or the Windows Notepad. It is not recommended to make changes to this file outside the CSPro

environment. Advanced users might do so, however, to change the names of associated files from the CSPro assigned defaults.

Table Specifications File (.XTS)

- The Tabulation specification file contains tables, dictionary items/value sets and other information which defines a set of tables. The file also contains the name of the associated data dictionary file. Items and value sets have links into the data dictionary.
- The Tabulation specification file is a text file that may be viewed with any text editor, such as CSPro's Text Viewer or the Windows Notepad. It is not recommended to make changes to this file outside the CSPro environment.

Tables File (.TBW)

The tables file (.TBW) is a text file that contains information about a table layout such as stubs, the headers, column size, etc.; and the table data. This file is read by the Table Viewer to produce a "published" table.

See Also: Save Tables for the Table Viewer

Area Names File (.ANM)

The Area Names File (.anm) is a text file that you can create using any text editor or word processor. Be sure you save this file with extension .anm.

The Area Names File defines the hierarchical levels of geography and assigns text names to the numeric codes for each geographic unit. The items must be defined in the common part of the data dictionary and should be listed in order from major to minor division.

See also: Create an Area Names File

Table Matrices File (.TAB)

A Table Matrices File (.tab) contains the output of tabulated numbers from the Tabulate process of CSPro. They are row and column contents of the tabulated tables, without any surrounding text.

A Table Matrices File may also contain the output of the Consolidate process of CSPro, where table matrices for one geographic level are added together to produce table matrices for a higher geographic level.

These files are used as output and input when tabulations are run in parts.

See also: Table Matrices Index File (.TAI), Introduction to Run in Parts

Table Matrices Index File (.TAI)

A Table Matrices Index File (.tai) contains indices for a Table Matrices File with the same name. While it is not necessary to save these files, because they can be recreated, having them around makes tabulation processing go faster.

See also: Table Matrices File (.TAB), Introduction to Run in Parts

Batch Edit Application File (.BCH)

- The Batch Edit Application file is the master file for a batch edit application. This file specifies all other files contained in the application, along with other information.
- CSPro allows you to open, close and save batch edit application files. When you do so, all other files associated with the application are also opened, closed or saved.
- The application file is a text file that may be viewed with any text editor, such as CSPro's Text Viewer or the Windows Notepad. It is not recommended to make changes to this file outside the CSPro environment. Advanced users might do so, however, to change the names of associated files from the CSPro assigned defaults.

Edit Order File (.ORD)

The Edit Order File is a text file that contains information about the sequence in which fields defined in the associated data dictionary are edited during batch editing. Each item in the associated CSPro Dictionary is listed in the Edit Order File in the sequence in which the procedures for that item will be executed.

Frequency file (.FRQ)

If your program contains any impute statements, the results of this command will be written to this file. The default file extension is .frq, but you can use whatever extension you prefer. This field is optional, so if your program contains impute commands but you forget to specify a frequency file, no file will be generated. Similarly, if you indicate a frequency file but your program does not contain any impute commands, no file will be generated.

Map Data File (.MDF)

The map data file (.MDF) is a tab delimited text file that contains the statistical data associated with the map and give the map location associated with the data.

For details on the format of the Map Data File see the Map Viewer User's Guide.

Map File (.MPC)

The compressed map file (.MAP) is a binary file that contains the map names, codes, and polygons for each geographic area.

Program Information File (.PFF)

- Program information files (PFF) are used to run applications (data entry and batch edit) or tools (tabulate frequencies, sort data, export data, reformat data, compare data, and concatenate data) in production mode.

- The PFF file stores the name of the application or tool, the data file(s) to be used, and any runtime parameters specific to the application or tool.

- You can use a PFF file as a command line parameter for CSEntry, CSBatch, CSTab, CSFreq, CSSort, CSExport, CSReFmt, CSDiff, CSConcat, CSPack, or CSIndex.

See also: Run Production Data Entry, Run Production Batch Edits, Run Production Frequencies, Run Production Sorts, Run Production Exports, Run Production Reformats, Run Production Compares, Run Production Concatenates, Run Production Packs, Run Production Indexing

Frequency Specification file (.FQF)

Tabulate Frequencies is a CSPro tool that allows you to display a frequency distribution of a dictionary variable(s) based on its value set(s). For example, you could have defined several value sets such as "age in 5 year groups," "level of education," or "type of occupation." Tabulate Frequencies gives you both the numeric count and percentage distribution for the selected variables. It also gives the cumulative distributions.

Operator Statistics File (.LOG)

The **LOG** file stores operator statistics generated by the Data Entry module for the corresponding data file. The Data Entry module creates a **LOG** file when it creates a new data file. When the Data Entry module opens, it look for the corresponding **LOG** file. If it doesn't find one, it creates a new one.

The **LOG** file is a comma delimited text file with a fixed format. It is easily imported into other software packages for custom processing. The LOG file can be processed by a CSPro application by creating a data dictionary for it.

Each record in the **LOG** file represents one data entry session. The record layout is as follows:

Position contents

1 – 3	Mode ('**ADD**' or '**MOD**' or '**VER**')
4	<comma>
5 – 36	Operator ID (as entered)
37	<comma>
38 – 47	Start date (mm/dd/yyyy)
48	<comma>
49 – 56	Start time (hh:mm:ss)
57	<comma>
58 – 65	End time (hh:mm:ss)
66	<comma>
67 – 74	Total time (End time – Start time) (seconds)
75	<comma>
76 – 83	Pause time (seconds)
84	<comma>
85 – 92	Number of cases written
93	<comma>
94 – 101	Number of records written
102	<comma>
103 – 110	Number of keystrokes
111	<comma>

112 – 119	Number of bad keystrokes
120	<comma>
121 – 128	Number of fields with errors attributed to keyer
129	<comma>
130 – 137	Number of fields with errors attributed to verifyer
138	<comma>
139 – 146	Total number of fields verified

Saved Arrays File (.SVA)

The saved array file stores values from arrays in a batch edit application marked with the save keyword. At the end of program execution, the current values of any saved arrays are written to the file. When the program is run, the saved array file is read and the array is initialized with the values from the saved array file.

CSPro creates the saved array file automatically when one or more arrays in a batch edit application are marked as a saved array. The name of the saved array file is always the same as that of the application with the additional .sva file extension added (for example myapplication.bch.sva).

The saved array file is a text file that may be viewed with any text editor, such as CSPro's Text Viewer or the Windows Notepad. It is not recommended to make changes to this file outside the CSPro environment. Advanced users might do so, however, to set initial values for arrays.

Synchronization File (.PNC)

Overview

CSPro synchronization routines, with the file extension .pnc, allow you to transfer data and programs back and forth between Android devices and remote machines. Currently, you can transfer data via a Dropbox account or by using an FTP server. Multiple synchronization routines can exist on a device, and you will have to determine the paradigm under which you would like to have your interviewers synchronize data and programs. For example, one synchronization routine could update all survey instruments on the Android device and another routine could send collected data to the server. Alternatively, one routine could be used for both tasks.

Once retrieved from a server, synchronization files are stored in the CSEntry "sandbox," meaning that they are safe from other applications and unable to be viewed by your interviewers (with the rare exception of interviewers using rooted devices). If you make changes to the synchronization routine, you must have your interviewers retrieve the synchronization file again. Because of this, and although synchronization makes updating applications and data easy, it is important to ensure that your synchronization routine is comprehensive and complete before distributing it.

There is no tool that comes with CSPro that you can use to edit your routines, but you can create and edit them in a text editor such as Notepad.

File Format

```
[SyncFile]
Version=CSPro 6.0
AppName=application name; optional
```

```
[Server]
Type=FTP or Dropbox
Host=ftp://...
Username=FTP user name
Password=FTP password

[Sync]
CreateClientPath=path name to be created on the Android device
ClientPath=path name on the Android device to move to

Put=file name to upload from the Android device to the server

CreateServerPath=path name to be created on the server
ServerPath=path name on the server to move to

Get=file name to download from the server to the Android device
```

When processing a synchronization file, CSEntry first attempts to connect to the server. If successful, the steps of the synchronization routine are executed sequentially, in a similar fashion to a batch file or shell script.

The CreateClientPath and CreateServerPath statements are similar to a mkdir (make directory) command. If the folder already exists, then no folder will be created.

The ClientPath and ServerPath statements are similar to a cd (change directory) statement. Synchronization will fail if a folder does not exist, so if you are not sure that a folder exists, be sure to create it before moving to that directory.

In the Get and Put statements, you can use the ? and * wildcards. The Get and Put statements will always download or upload files, regardless of whether the files have been updated on the server or on the device.

In any statement, you can use a few replacement parameters:

- **%AppName%**: the application name optionally defined in the synchronization header
- **%Date%**: the date, in "year-month-day" (without hyphens) format, that the synchronization routine occurred, using the Android device's clock
- **%DateTime%**: the date and time, in "year-month-day-hour-minute-second" (without hyphens) format, that the synchronization routine occurred, using the Android device's clock
- **%UID%**: the device's user ID, which corresponds to the value that is returned by the getusername function

Synchronization File Example 1

The following synchronization routine downloads a CSEntry application from a preexisting folder on Dropbox called "Granite Geological Survey." It also uploads any collected data, assuming that collected data had the filename "Kaymoor.dat."

```
[SyncFile]
Version=CSPro 6.0

[Server]
Type=Dropbox
```

```
[Sync]
ServerPath=/Granite Geological Survey

CreateClientPath=/Granite Geological Survey
ClientPath=/Granite Geological Survey

Get=Granite Geological Survey.pen
Get=Granite Geological Survey.pff

Put=Kaymoor.dat
```

Synchronization File Example 2

Using replacement parameters, it is possible to simplify the synchronization routine and make it easily modifiable for other applications. The following synchronization routine, building off Example 1, downloads the application but then uploads all data, as well as the notes files associated with the data files. The assumption here is that the data is stored in a subfolder underneath the data entry application called "Data," and that data files have the extension ".dat."

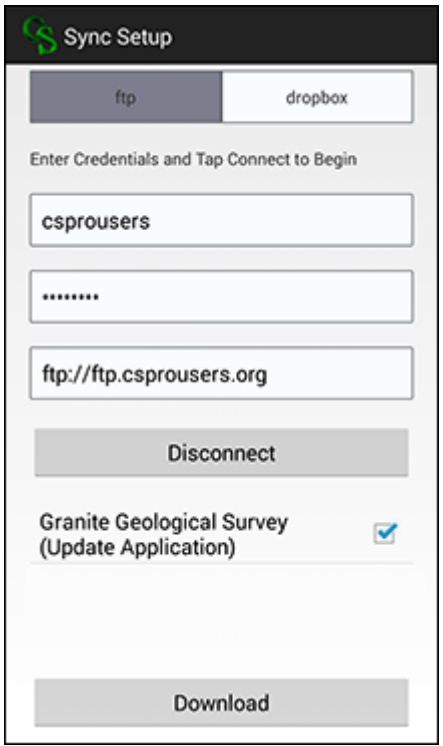
```
[SyncFile]
Version=CSPro 6.0
AppName=Granite Geological Survey

[Server]
Type=Dropbox

[Sync]
ServerPath=/%AppName%

CreateClientPath=/%AppName%
ClientPath=/%AppName%

Get=*.*

CreateServerPath=Data
ServerPath=Data

CreateClientPath=Data
ClientPath=Data

Put=*.dat
Put=*.dat.not
```

Synchronization File Example 3

If you plan to frequently update your data entry application, you may want to always get the latest version as part of the synchronization routine. Often, however, your interest in synchronization may only be to upload data. In this example, there are two synchronization files. The first, "Granite Geological Survey (Update Application).pnc," is used to update the data entry application. The second, "Granite Geological Survey (Upload Data).pnc," is used to upload data to the server. The uploaded data goes into a data folder, as above, but this time based on the Android device's user ID as well as the date. This allows you to track the data entry process by interviewer over time. This time the synchronization will interact with an FTP server.

```
[SyncFile]
Version=CSPro 6.0
AppName=Granite Geological Survey

[Server]
Type=FTP
Host=ftp://ftp.example.org/
Username=bituminous
Password=coal

[Sync]
ServerPath=/%AppName%

CreateClientPath=/%AppName%
CreateClientPath=/%AppName%/Data
ClientPath=/%AppName%

Get=*.*

---------------------------------

[SyncFile]
Version=CSPro 6.0
AppName=Granite Geological Survey

[Server]
Type=FTP
Host=ftp://ftp.example.org/
Username=bituminous
Password=coal

[Sync]
CreateServerPath=/%AppName%/Data
CreateServerPath=/%AppName%/Data/%UID%
CreateServerPath=/%AppName%/Data/%UID%/%Date%
ServerPath=/%AppName%/Data/%UID%/%Date%

ClientPath=/%AppName%/Data

Put=*.dat
Put=*.dat.not
```

Synchronization Files on Android Devices



To download a synchronization file to an Android device, select the Sync Setup option on the menu when in the Entry Applications screen. After entering your credentials, click on the connect button.

- When entering FTP credentials, you must enter a host, username, and, optionally, a password. The host, in addition to the server name, can include a path indicating which remote directory stores the synchronization files.
- When entering Dropbox credentials, you only need to enter a path indicating which remote directory stores the synchronization files, if applicable. After clicking on the

connect button, you will be asked to validate your Dropbox credentials.

CSEntry will connect to the remote server and, if successful, present a list of available synchronization files. After selecting the applicable files, clicking Download will retrieve the files and place them in the CSEntry sandbox. This process only downloads synchronization files; it does not execute the routines detailed in the files. After the download, you will be asked if you want to execute the synchronization.

From the Entry Application screen, the Synchronize option, if enabled, will also allow you to synchronize your programs and data. After a user selects a synchronization routine, the script will execute, with any errors during this process reported to the user. As soon as one error occurs, the entire routine will terminate.

Index

- 64
  - Operator 64
  - Operator Precedence 66

! 64
  - Operator 64
  - Operator Precedence 66

$ 60

% 64
  - Operator 64
  - Operator Precedence 66

%d 404
  - errmsg function 404
  - write function 428

%f 404
  - errmsg function 404
  - write function 428

%s 404
  - errmsg function 404
  - write function 428

& 64
  - Operator 64
  - Operator Precedence 66
  - Truth Table 67

* 64
  - Operator 64
  - Operator Precedence 66
  - Rounded to Zero Mask 224

.ANM 459
  - Area Names File 459

.APC
  - Logic File 456

.APP 154
  - Logic File 456

.BCH 154
  - Batch Edit Application 154
  - Batch Edit Application File 460
  - Batch Edit Applications 3

.DCF 34
  - Data Dictionary 5, 34, 35
  - Data Dictionary File 455
  - In a Batch Edit Application 154
  - In a Data Entry Application 76
  - In aTabulation Application 191

.ENC 456
  - Binary Data Entry Application File 456

.ENT 456
Data Entry Application File 456
Data Entry Applications..... 3, 76
.FMF 76
Data Entry Applications..... 3, 76
Form File..... 5, 456
.FQF 461
Frequency Specification File 461
.FRQ..... 460
Frequency File 460
.LOG 461
Operator Statistics File..... 461
.LST 458
Listing File 458
.MAP 255
Map File 460
.MDF 255
Map Data File 460
.MGF 154
In a Batch Edit Application..... 154
In a Data Entry Application 76
Message File 457
.MPC 255
.NOT 457
.ORD 154
Batch Edit Application..... 154
Order File 460
.PEN 456
Binary Data Entry Application File 456
.PFF 460
Program Information File 460
Run Production Batch Edits..... 175
Run Production Data Entry..... 82
.PNC..... 462
Synchronization File 462
.QSF 76
In a Data Entry Application 76
Question File 457
.STS..... 458
Data File Status 458
.SVA..... 462
Saved Arrays File..... 462
.TAB..... 459
Table Matrices File .TAB 459
.TAI 459
Table Matrices Index File .TAI 459

.TBW 459
Tables File 459
.WRT 171
Create a Specialized Report 171
.XTB 191
Tabulation Application 191
Tabulation Application File 458
Tabulation Applications 4
.XTS 191
Table Specifications File 459
Tabulation Application 191
Tabulation Applications 4
/ 64
Operator 64
Operator Precedence 66
[NoEdit] 453
^ 64
Operator 64
Operator Precedence 66
| 64
Operator 64
Operator Precedence 66
Truth Table 67
+ 64
Operator 64
Operator Precedence 66
< 64
Operator 64
Operator Precedence 66
<= 64
Operator 64
Operator Precedence 66
<=> 66
If and Only If 66
Operator 64
Operator Precedence 66
<> 64
Operator 64
Operator Precedence 66
= 64
Operator 64
Operator Precedence 66
> 64
Operator 64
Operator Precedence 66
>= 64

Operator 64
Operator Precedence 66
Abs Function 380
Absolute 47
Mode 38
Positioning 47
Accept Function 336
ActiveSync 150
Add 41
Dictionary Elements 41
Fields to a Form 91
Form 91
Identification items 27
Items in Dictionary 41, 42
Levels in Dictionary 41
Lines or Boxes to a Form 93
Logic in Data Entry 72
Pictures in CAPI Question Text 133
Records in Dictionary 41
Relations in Dictionary 48
Roster to a Form 92
Row or Column for Post Calculation .. 275
Table 206
Text to a Form 93
Things to a Roster 92
Value Sets in Dictionary 41, 42
Values in Dictionary 41, 44
Variable to a Tabulation 209
Add a Pagenote or Footnote to a Table ... 234
Add a Subtitle to a Table 236
Add a Variable to a Tabulation 209
Add Borders to a Table 237
Add Borders to Cells 242
Add Header/Footer Text to a Table 235
Add or Modify Relations 48
Add Reader Breaks to a Table 238
Add Stub Leadering to a Table 236
Add Subtotals to a Table 300
Add Summary Statistics to a Table 216
Advance Statement 337
Alias Statement 53
Align Text and Fields 98
Bottom 98
Center 98
Horizontal 98
Left 98

Mid 98
Right 98
Top 98
Vertical 98
Alignment in Tables 242
Allow Partial Save 86
Alpha Statement 320
Alphabetical List of Statements and Functions 312
Alphanumeric 321
Array 321
Data Type 29
Variables 52
And 64, 66, 67
Operator 64
Operator Precedence 66
Truth Table 67
Android Development 138
Android Navigation 139
ANM 459
Area Names File 459
APC
Logic File 456
APP 456
Logic File 456
Application 14
Add a File 14
Save As 16
Applications 82
Batch Edit 3
Closing 16
Compile 118
Create a CSPro 11
Create a New Batch Edit 154
Create a New Data Entry 76
Create a New Tabulation 191
Insert or Drop a File 14
Installing Data Entry 431
Matching to the Data Dictionary 101
Move Around a Logic Application 116
Open an Existing Dictionary Application 39
Run a Batch Edit 159
Run a Data Entry 82
Save 15
Tabulation 4
Apply Changes Across Panels 260
Area 248

Create a Names File 248
IDs Dialog Box 250
Processing 247
Tabulate Only Certain Levels of Geography 257
Area Captions..... 251
Area Names File 459
.ANM 459
ANM 459
Arithmetic operators 64
Array 321
Alphanumeric..... 321
Numeric..... 321
Saved Array File 462
Arrays..... 52
Description..... 52
Ask for Operator ID..... 86
Assignment Statement 331
Assignment Statements..... 54
Automatic..... 161
Correction 161
Edit Report..... 170
Generation of Text in Tables 224
Skip 71
Automatically Generate Data Entry Forms77
Average 216
Add Summary Statistics to a Table..... 216
Produce Summary Statistics 186
Average Function..... 394
Batch 156
Application Screen Layout 156
Compile..... 118
Logic View..... 156
Message View..... 156
Run an Edit Application..... 159
Tree View..... 156
Batch Edit..... 3
Applications 3
Introduction..... 154
Keyboard Summary 435
Menu Summary..... 440
Order 160
Order or Executing Events..... 160
Run Production Batch Edits..... 175
Toolbar Summary 444
Tree 8, 158
Window..... 8

BCH 154
Batch Edit Application..... 154
Batch Edit Application File 460
Batch Edit Applications 3
Bell 90
Change Error Sound..... 90
Binary application 85
Binary data entry application 85
Binary Data Entry Application File (.PEN) 456
Blanks 392
Strip Function..... 392
Border 237
Add Borders to a Table 237
Add Borders to Cells..... 242
Box 93
Adding to a Form 93
Change Field Box Size..... 90
Statement..... 331
Boxhead 245
Change the Repeating of Boxheads 245
Formats for Printing..... 226
Parts of a Table 181
Break 326
Break Statement 326
Reader Breaks 238
Breaking Off the Interview in CAPI..... 129
By 327
Do Statement..... 327
CAPI 120
Add Pictures in Question Text..... 133
Breaking Off the Interview 129
Change Formatting..... 132
Coming Back Later 129
Create a New Application 129
Create Conditional Questions 133
Create Fills In Questions..... 131
Create Standard Forms..... 132
Display Questions Without Scrolling . 134
Enter Question Text 131
Features 121
Full Screen Value Sets 135
Images in Value Sets..... 135
Organization of the Instrument 128
Use Multiple Language..... 133
Using Multiple Languages 128
CAPI Mode 86

Caption 251
Area Caption 251
Formats for a Part of a Table 219
Parts of a Table 181
Case 404
Errmsg Function 404
Skip Case Function 379
Case ID 27
Identification Items 27
Case tree 86
Show 86
Cases and Levels 75
Cells 219
Formats for a Part of a Table 219
Formatting Row
Column and Cell Data 221
Parts of a Table 181
Select in Table 271
Center Text and Fields 98
Change 86
Data Entry Options 86
Default Text Font 89
Edit Order 161
Error Sound 90
Field Box Size 90
Field Font 90
Field Properties 103
Fonts and Colors in a Table 241
Form Properties 102
Formatting in CAPI 132
Forms File Properties 101
Level Properties 101
Order of Entry 85
Print Page Setup 14
Roster Column Heading Properties 108
Roster Occurrence Labels 109
Roster Properties 107
Row Heading Properties 109
Table Title 230
Text Properties 110
Unit of Tabulation 293
View 13
Windows 14
Change Indentation and Alignment of Items 242
Change Stub Column Position 246

Change the Automatically Generated Text 240

Change the number of Decimal Places Displayed 233

Change the Repeating of Boxheads 245

Change the Way Numbers are Displayed 239

Changekeyboard Function 338

Character 30

Decimal 30

Check box 95

Checking errors 113

Choose Topic Sections 137

Clear Function 416

Close 16

An Application 16

Function 417

Cmcode Function 380

Code Edits of Individual Data Items 179

Coding Standards 178

Define 178

Cold Deck 3

Batch Edit Applications 3

Static Imputation 163

Color 241

Change Colors in a Table 241

Form 102

Text 110

Column 275

Add for Post Calculation 275

Percent 214

Column Head 221

Format Column data in a Table 221

Formats for a Part of a Table 219

Hide/Show Columns in a Table 231

Parts of a Table 181

Columns 221

Format Column data in a Table 221

Hide/Show Columns in a Table 231

Join and Split in a Roster 100

Spacing 260

Variables 192

Combo box 95

Coming Back Later [CAPI] 129

Comments 59

About 59

In Data Dictionary 22

Common Uses of Tabulation Applications 184

Compare 5
Data 5
Function 386
Compile Logic 58
Compiler 58
Set Defaults 58
Concat Function 387
Concatenate 5
Data 5
Conditions 62
Confirm End-of-Case 86
Consistency Checks 3
Consistency Edits 113
Consolidation 253
Custom 253
Run Consolidate in Batch 289
Run Consolidate Interactively 288
Run in Parts 285
Controlling Program Flow 341
Endgroup Statement 341
Endlevel Statement 340
Convert 5
Dictionary 5
Items to Subitems 47
Number to string 388
Shape to Map 5
String to number 393
Converting 445
A data dictionary 445
An IMPS Data Entry Application 447
An ISSA Data Entry Application 447
Within IMPS 446
Within ISSA 447
Copy 16
Application 16
Copy and Paste Table Specification ... 188
Copy Table Data to a Spreadsheet or Word Processor 274
Cut or Paste Things 99
Dictionary 48
Select and Copy Table Data to Other Applications 271
Table to Other Formats 188
Copy Table Data to a Spreadsheet or Word Processor 274
Correction 163
Count Function 395
Countnonspecial Function 381

Countvalid Function..... 381
Create 192
A CSPro Application 11
A Dictionary for a New File 34
A Dictionary for an Existing File..... 35
A New Application 11
A New Batch Edit Application 154
A New CAPI Application 129
A New Data Entry Application..... 76
A New Tabulation Application..... 191
A Roster 92
A Specialized Report 171
A Table..... 192
A Table With Multiple Variables 194
A Thematic Map of Results 255
An Area Names File..... 248
Conditional Questions in CAPI 133
Logic 57
Standard Forms in CAPI..... 132
Thematic Maps..... 2
Create Dictionary with No Record Types. 47
Create Fills In Questions..... 131
Create General Helps 138
Create Helps for Fields 134
Create Multiple Subtables..... 189
Cross Tabulation 192
Applications 4
Create an Application 191
Creating..... 192
Creating a Table..... 194
Definition 185
Introduction..... 180
Keyboard Summary 436
Using Relations..... 298
CSBatch 175
CSPro 2
Capabilities 2
Data Requirements..... 49
General Menu Summary 437
Initial Screen Layout..... 7
Introduction..... 1
Toolbar Summary 441
Tools 5
What is 1
CSPro Application 11
Create 11

CSPro Program Structure 50
Declaration Section 50
Procedural Section 50
Curocc Function 395
Custom Consolidation 253
Custom Special Values 213
Custom Text 230
Cut 99
Copy or Paste Things 99
Data 5
Compare 5
Concatenate 5
CSPro Requirements 49
Export 5
Reformat 5
Sort 5
Synchronization 462
Data Dictionary 5
Creating for a New File 34
Creating for an Existing File 35
Introduction 18
Keyboard Summary 432
Labels 22
Levels 22
Menu Summary 437
Names 22
Screen layout 36
Toolbar Summary 442
Tree 37
Data Edit Application 154
Creating 154
Data Entry 72
Add Logic 72
Application File [.ENT] 456
Application Keys 433
Applications 3
Create a New Application 76
Elements 73
Errors 72
Forms Screen Layout 79
Installation 431
Logic Screen Layout 114
Menu Summary 438
Order of Executing Events 116
Path 73
Run Production 82

Toolbar Summary 443
Tree 8, 81
Window 8
Data Entry Operator Mode..... 339
Add..... 339
Modify..... 339
Verify 339
Data Entry Philosophies..... 71
Heads-Down Keying..... 71
Heads-Up Keying..... 71
Data file..... 457
Size..... 20
Type Structure..... 20
Data File Index .IDX..... 457
Data File Organization..... 20
About..... 20
Records 24
Data File Status 458
Data Items 59
Data Organization 18
Data Records..... 24
Data Type..... 29
Data Validation 3
DataEntryIDs 82
Date 415
Dateadd Function 413
Datediff Function 414
Datevalid Function..... 415
Sysdate Function 415
Date capture type 95
DCF 34
Data Dictionary 5, 34, 35
In a Batch Edit Application..... 154
In a Data Entry Application 76
In a Tabulation Application 191
Debug 51
CSPro Programs..... 51
Table totals..... 217
Decimal 30
Character 30
Decimal Places in a Table..... 233
Places 30
Symbol in Tables 224
DeckArrays 165
Getdeck Function 377
Leftover Rows..... 168

Putdeck Function 378
Declaration Section..... 50
Default..... 266
Preferences and Default Formats..... 266
Reset Format of Table Item to Default 245
Special Values..... 63
Text Font..... 89
Define..... 210
A Universe 210
Coding Standards..... 178
Dictionary Type 38
Languages in CAPI..... 130
Delcase Function..... 417
Delete 45
Dictionary Elements..... 45
Form Elements 100
Item 45
Level 45
Record..... 45
Table 206
Value..... 45
Value Set..... 45
Delete Function..... 396
Delimiters..... 59
Demode Function..... 339
Denom..... 404
Errmsg Function..... 404
Deploy PPC Application..... 150
Description..... 73
Fields..... 73
Forms 73
Rosters..... 73
Design a Form..... 75
Issues..... 75
Develop Comprehensive Test File..... 179
Dictionary 38
About..... 5
Adding Elements..... 41
Conversion 5
Converting..... 445
Creating..... 35
Delete 45
External..... 38
Hierarchy..... 21
Inserting Elements 45
Introduction to Data Dictionary..... 18

Main 38
Modify..... 41
Moving Around..... 39
Moving Elements 45
Notes 46
Print..... 48
Save As 48
Select Elements..... 45
Special Output..... 38
Tree 8
Types..... 38
Viewing Layout 40
Window..... 8
Working 38
Disjoint Categories..... 302
In Value Sets..... 302
Display Function..... 404
Display Orientation..... 339
Display Questions Without Scrolling in CAPI 134
Display Results for One Geographic Area 256
Distribute Finished Tables to Other Users 273
Do Statement..... 327
Document Dictionary Elements..... 46
Drag Option Menu 78
Require Enter Key..... 78
Roster Options 78
Text Options..... 78
Use Sub-Items..... 78
Draw Boxes on a Form..... 93
Drop down 95
Drop Files from an Application 14
Dump..... 213
Undefined Values..... 213
Dynamic Imputation (Hot Deck) 164
Edit..... 158
Batch Tree..... 158
Logic 57
Reports 3
Specifications..... 162
Edit Function..... 388
Edit Order File [.ORD] 460
Edit Report..... 170
Automatic..... 170
Specialized 171
Editing..... 3

Editnote 340
Else 329
If Statement 329
Elseif 329
If Statement 329
ENC 456
Binary Data Entry Application File 456
End 324
Function Statement 324
Endcase Statement 376
Enddo 327
Do Statement 327
For Statement 328
While Statement 330
Endgroup 341
Endif 329
If Statement 329
Endlevel Statement 340
Endnote 234
Add a Pagenote or Footnote 234
Formats for a Part of a Table 219
Parts of a Table 181
EndRecode 331
Recode Statement 331
Endsect Statement 341
ENT 456
Data Entry Application File 456
Data Entry Applications 3, 76
Enter 2
Modify and Verify Data 2
Question Text in CAPI 131
Enter Key 86
Required 86
Enter Statement 341
Errmsg Function 404
Error Sound 90
Change 90
Errors 448
In Censuses and Surveys 448
In Data Entry 72
Events 50
Order of Executing Batch Edit 160
Order of Executing Data Entry ... 116, 160
Examine 2
Data Files 2
Editing Results 2

Exclude 213
Special Values in a variable 213
Tables from Run 216
ExecPFF Function 407
Execsystem Function 406
Android 406
Windows 406
Executable Statements 54
Existing Application 12
Open 12
Exit Statement 328
Exp 382
Explicit Mode 51
Export 5
Data 5
Export Statement 376
Expressions 62
Extended Controls 121
External Dictionary 38
External Files 67
About 67
Sharing 67
Facing Pages 259
Field 90
Change Box Size 90
Change Font 90
Change Properties 103
Change Properties (for Multiple Fields) 105
Name 22
Field Capture Type 95
Check Box 95
Combo Box 95
Date 95
Drop Down 95
Radio Button 95
Text Box 95
Field Properties 95
Mirror 95
Persistent 95
Protected 95
Sequential 95
Upper Case 95
Fields 127
Add to a Form 91
Align 98
Description 73

File 158
Add to an Application 14
Data Dictionary 18
Data Items 18
Identification 18
Organization 5, 18
Program Information (.PFF) 175
Records 18
Types 450
Value Sets 18
File Statement 319
Fileconcat Function 418
Filecopy Function 418
Filecreate Function 419
Filedelete Function 420
Fileempty Function 419
Fileexist Function 420
Filename Function 421
Fileread Function 421
Filerename Function 421
Files 53
Area Names File .ANM 459
Batch Edit Application File .BCH 460
Data Dictionary File .DCF 455
Data Entry Application File (Binary) .ENC 456
Data Entry Application File .ENT 456
Edit Order File .ORD 460
Form File .FMF 456
Frequency File .FRQ 460
Frequency Specification File .FQF 461
Listing File .LST 458
Logic File .APC 456
Map Data File .MDF 460
Map File .MAP 460
Message File .MGF 457
Notes [.NOT] 457
Operator Statistics File .LOG 461
Program Information File .PFF 460
Question File .QSF 457
Saved Arrays File .SVA 462
Table Specifications File .XTS 459
Tables File .TBW 459
Tabulation Application File .XTB 458
Filesize Function 422
Filewrite Function 422
Find and Replace Logic 58

Find Dictionary Elements 46
Find Function 423
Finding 161
Errors..... 161
Fit Columns..... 260
Flow of Program 341
endgroup statement 341
endlevel statement..... 340
FMF..... 76
Data Entry Applications..... 3, 76
Form File..... 5, 456
Folio Text..... 226
Font 90
Change Field 90
Change Fonts in a Table 241
Default Text 89
Field 90
Printer Font 15
Footer 14
Change Print Page Setup..... 14
Footnote 234
For Statement..... 328
Force out-of-range 86
Form 5
Adding 91
Adding Fields 91
Adding Lines or Boxes 93
Adding Text 93
Delete Elements 100
File [.FMF]..... 456
Form Design Issues..... 75
Format 224
Application..... 224
For a Table 223
For a Table Object 219
Row, Column, or Cell Data 221
Rows/Columns in Subgroupings..... 303
Run Format in Batch..... 291
Run Format Interactively 290
Tables for Printing 226
Forms 126
Description..... 73
Design 5
Tree 8, 81
Window..... 8
FQF 461

Frequency Specification File 461
Frequencies 5
FRQ..... 460
Frequency File 460
Full Screen 13
Full Screen Value Sets (CAPI) 135
FullScreen 82
Function Statement 324
Functions..... 53
Alphabetical Listing..... 312
Numeric..... 380
User Defined 53
General Issues 177
General Menu Summary 437
Generate Binary Data Entry Application.. 85
Generate Default Data Entry Forms 77
Generate Numeric Value Sets..... 43
Generated Text..... 224
In Tables..... 224
Geographic..... 250
Area Tabulation 250
Map Results by Areas 189
Processing 247
Get Help 15
Getbuffer Function..... 389
Getcapturetype Function..... 342
Getdeck Function..... 377
Getlabel Function..... 408
Getlanguage Function 342
Getnote Function..... 342
Getocclabel Function 343
GetOperatorId Function 343
Getorientation Function 339
Getos Function 344
Getrecord Function 344
Getsymbol Function..... 408
Getusername Function 345
Getvalue Function..... 336
Getvaluealpha Function 336
Getvaluenumeric Function..... 336
Global..... 52
Arrays..... 52
Mode 51
Numeric Statement..... 320
Procedure 50
Set Statment 318

User Defined Functions 53
Variables 52
Goto..... 260
Print Preview Options..... 260
GPS Function..... 345
Guidelines for Correcting Data..... 162
Handle Don t Know and Refused 136
Handle Multiple Answers 136
Handle Undefined Values..... 213
Hardware and Software Requirements ... 429
Has Operator 65
Header 14
Heads-Down Keying..... 71
Heads-Up Keying..... 71
Help..... 15
Support..... 15
Hidden Parts..... 229
Hide or Change the Position of the Total 214
Hide or Show a Row or Column..... 231
Hide Rows Containing All Zeros..... 233
Hiding Rows and Columns in Subgroupings 303
Hierarchy..... 5
Dictionary 21
High Function 382
Highlighted Function 347
Hot Deck 3
Batch Edit Applications 3
DeckArrays 165
Dynamic Imputation [Hot Deck] 164
Initialize From Saved Array..... 173
Initialize In Program Logic..... 172
Using..... 172
HTML 274
Prepare Tables for Posting to the Web 274
Identification..... 18
Data Dictionary..... 18
File 18
Items..... 27
Questionnaire 18
IDs Dialog Box 250
If and Only If 66
Operator 64, 66
Operator precedence 66
If Statement..... 329
Images in Value Sets (CAPI)..... 135
Implications of Data Dictionary Value Names 196

Implicit Mode 51
Importing Data 451
IMPS 446
Converting a Data Dictionary 446
Converting a Data Entry Application . 447
Imputation 163
About 163
Cold Deck 163
Correcting Errors 163
DeckArrays 165
Dynamic
Hot Deck 164
Hard-Coded 163
In Batch Edit Applications 3
Static 163
Using Hot Decks 172
Impute 159
Freq File 159
Function 334
In
Operator 65
Operator Precedence 66
Inc Function 382
Include Percents 214
Inconsistencies 161
Indent 242
Change Indentation or Alignment 242
Insert 14
Add File to an Application 14
Data Item 45
Dictionary Elements 45
File in an Application 14
Function 397
Level 45
Record 45
Table 206
Value 45
Value Set 45
Installing 429
CSPro 429
Data Entry Applications 431
Limiting the Installation to Data Entry 432
Newer Version 430
Uninstalling CSPro 430
Int Function 383
Interactive 82

Interactive Editing..... 2

Interpret Reports 174

Introduction to Production Tabulations .. 283

Introduction to Run in Parts..... 285

Introduction to..... 154

Batch Editing 154

CAPI 120

CSPro 1

CSPro Language 49

Data Dictionary..... 18

Data Entry 70

Data Entry Editing 111

Forms Design..... 90

Table Post Calculation 274

Tabulation 180

Invalueset Function..... 409

Ispartial Function 347

ISSA..... 447

Converting a Data Dictionary 447

Converting a Data Entry Application . 447

Issues to Consider When Designing a Form 75

Item 41

Adding..... 41

Convert to Subitem 47

Data Type..... 28, 29

Decimal Character 28, 30

Decimal Places..... 28, 30

Delete 45

Description..... 26

Edit Tree..... 158

Find 46

Insert 45

Label 22, 28

Length 28, 29

Modify..... 41

Name 22, 28

Notes 46

Occurrences..... 28, 30

Properties 28

Search..... 46

Select 94

Starting Position..... 28, 29

Type 27, 28

Zero Fill 28, 31

Item with Multiple Occurrences 197

Tabulate..... 197

Join and Split Roster Columns..... 100
Key Function..... 424, 433
Key Override Character Map..... 355
Keyboard Input 106
Keyboard Summary 435
Batch Edit..... 435
Data Dictionary..... 432
Data Entry 433
Tabulation 436
Keying..... 71
Heads-Down 71
Heads-Up 71
Skips Issues..... 71
Killfocus Statement..... 348
Labels..... 22
Data Dictionary..... 22
Languages 130
Define in CAPI 130
Getlanguage Function 342
Setlanguage Function..... 366
Layout 7
Initial Screen 7
Screen..... 156
View Dictionary..... 40
Leadering 236
Stub 236
Length 29
Function 389
Item 29
Level 41
Adding..... 41
Delete 45
Description..... 22
Edit Tree..... 158
Find 46
Insert 45
Label 23
Modify..... 41
Name 23
Notes 46
Properties 23
Search..... 46
Levels and Cases..... 75
Lines 93
Adding to a Form..... 93
Adding to a Table..... 242

Linked Value Sets 42
List of Reserved Words 317
Listing File 159
ListingWidth 175
Load and Save Formatting Preferences .. 187
Loadcase Function 424
Loading and Saving Preferences..... 268
Locate Function 424
Lock 82
Locking Application Files..... 453
Log 383
Operator Statistics File..... 461
Logic 456
Batch Edit Applications 3
Editing..... 160
File .APC..... 456
In Batch Edit Applications..... 3
In Data Entry Applications 3
View 56, 156
Logical 62
Expressions 62
Operators..... 64
Lookup Files 68
Importing Data to CSPro Format..... 451
Using 68
Low Function 383
Lowest Break Level 203
LST 458
Listing File 458
Main Dictionary 38
Type 38
Make Captions Span Data Cells 244
Maketext Function 390
Manipulate Automatic Reports 170
Manipulate Data Files 2
Manual 161
Correction 161
Skip 71
Map 255
Create a Thematic Map..... 255
Data File .MDF 460
Map File 460
Results by Geographic Area 189
Viewer..... 5
Margins 14
Matching the Application to the Data Dictionary..... 101

Mathematical operators..... 64
Max Function..... 397
Maximum..... 216
Add Summary Statistics to a Table..... 216
Number of Record..... 26
Produce Summary Statistics 186
Record Properties..... 25
Maxocc Function 398
MDF..... 460
Map Data File 460
Mean 216
Add Summary Statistics to a Table..... 216
Produce Summary Statistics 186
Median 216
Add Summary Statistics to a Table..... 216
Produce Summary Statistics 186
Menu Summary..... 440
Batch Editing 440
Data Dictionary..... 437
Data Entry 438
General..... 437
Tabulation 440
Merge Cells..... 244
Make Captions Span Data Cells 244
Message File 457
.MGF..... 457
About..... 69
In Batch Edit Applications..... 3
In Data Entry Applications 3
Message View..... 156
MessageWrap..... 175
Methods of Correcting Data..... 161
MGF..... 154
In a Batch Edit Application..... 154
In a Data Entry Application 76
Message File 457
Min Function..... 398
Minimum..... 216
Add Summary Statistics to a Table..... 216
Produce Summary Statistics 186
Mirror Fields 95
Missing..... 63
Special Values..... 63
Mode 51
Absolute and Relative 38
Add Summary Statistics to a Table..... 216

Compiler 51
Explicit..... 51
Implicit..... 51
Produce Summary Statistics 186
Modify..... 339
Demode Function..... 339
Dictionary Elements..... 41
Items..... 41, 42
Levels..... 41
Records 41
Relations in Dictionary 48
Row and Column Spacing for Printing260
Tabulation Preferences..... 267
Value Sets 41
Value Sets in Dictionary 42
Values 41
Values in Dictionary 44
Modify or Add Dictionary Elements 41
Modulo Operator..... 64
Move 39
Around a Dictionary 39
Around a Logic Application 116
Between Tables..... 207
Dictionary Elements..... 45
Things in a Form..... 97
Move Statement 348
Moving Around a Logic Application..... 116
Multiple..... 197
Occurrences
Tabulate..... 197
Record Types 20
Selection..... 45
Tabulate Items..... 197
Value Ranges 33
Multiple records..... 75
Name of File 421
Filename function 421
Names 22
Data Dictionary..... 22
In Tree..... 13
Navigating a Dictionary..... 39
Navigating Between Pages Tables and Areas 258
Next..... 370
Skip Statement 370
Statement..... 330
Noccurs Function..... 399

NoFileOpen 82
Noinput Statement 349
Not 457
Applicable Value 63
Operator 64
Operator Precedence 66
Notappl 63
Special Values 63
Notes 22
Adding 22
Document Dictionary Elements 46
Notes File [.NOT] 457
N-tiles 216
Add Summary Statistics to a Table 216
Produce Summary Statistics 186
Number 388
Convert to string 388
Decimal Places 30
Numbers 61
Numeric 321
Array 321
Expressions 62
Item 29
Statement 320
Variables 52
Occurrences 75
Getocclabel Function 343
Item 30
Occurrence Labels 47
Tabulate Items with Occurrences 197
Onchangelanguage Global Function 349
Onchar Global Function 350
Onfocus Event 351
Onfocus Statement 351
Onkey Global Function 351
Onstop Global Function 356
Open 12
An Existing Application 12
An Existing Dictionary Application 39
Function 425
Operator 86
Ask for Operator ID 86
Statistics File .LOG 461
Vs System Controlled 72
Operators 64
And/Or Truth Table 67

Arithmetic 64
Has Operator 65
If and Only If 66
In Expression 65
Logical 64
Precedence 66
Relational 64
Option 58
Set Explicit 58
Set Implicit 58
Or 64
Operator 64
Operator Precedence 66
Truth Table 67
ORD 154
Batch Edit Application 154
Order File 460
Order 160
Batch Edit 160
Executing Batch Edit Events 160
Executing Data Entry Events 116
Fields and Forms 85
File .ORD 460
Organization 5
Data Dictionary 5
Data File 20
Data File Type Structure 20
Questionnaire and Dictionary 18
Organization of the Instrument in CAPI . 128
Organize Forms 131
Output File 159
Overlapping Value Sets 196
Pack an Application 17
Page Break 260
Print Preview Options 260
Page Numbering 187
Pagenote 234
Add a Pagenote or Footnote 234
Formats for a Part of a Table 219
Parts of a Table 181
Parameter 175
PFF 175
Run 82
Sysparm Function 411
Partial Save 86
Allow 86

Parts 285
Run in Parts 285
Parts of a Table 181
Paste 99
Copy or Cut Things 99
Paste Table Specification 188
Path 73
Off 73
On 73
Pathname Function 409
PDA 145
PEN 456
Binary Data Entry Application File 456
Percentile 216
Add Summary Statistics to a Table 216
Produce Summary Statistics 186
Percents 214
Including in Tables 214
Only 214
Proportions 198
Persistent Fields 95
PFF 175
Parameter 82, 175
Program Information File 460
Run Production Batch Edits 175
Run Production Data Entry 82
Run Production Tabulation 283
Sysparm Function 411
Pictures 133
Add in CAPI Question Text 133
PNC 462
Synchronization File 462
Pocket PC 145
Deploy application 150
Install CSPro 148
Introduction 145
Requirements 145
Retrieve data files 154
Run application 152
Write application 146
Write CSPro application 146
Pos function 391
Poschar Function 392
Position 214
Change the Position of the Total 214
Item 29

Within a String 391
Positioning 47
Relative or Absolute 47
Post Calculation 275
Adding Rows and Columns For..... 275
For Individual Cells 276
For Rows Columns and Ranges..... 279
Introduction to Table Post Calculation 274
Row and Column Indexing 281
Postcalc 274
Table Post Calculation 274
Postproc Statement..... 56
PPC 145
Preferences..... 266
Default Formats 266
Designer Font Preferences 13
Load and Save..... 187
Modifying 267
Sharing 269
Prepare Tables for Posting to the Web ... 274
Preproc Statement 55
Preview 15
Printing..... 15
Table 264
Print..... 15
All or Part of a Document..... 15
Change Font..... 15
Change Page Setup 14
Dictionary file 48
Preview 264
Tables..... 264
Print Only Selected Tables or Pages..... 266
Print Preview Options..... 260
Print Setup..... 263
Printer..... 14
Footer 14
Header 14
Margins 14
Proc Statement 55
Procedure Section 50
Process Census or Survey Data..... 2
Produce Summary Statistics 186
Produce Tables by Geographic Area 188
Production..... 180
Begin Production Editing..... 180
Run Production Batch Edits..... 175

Run Production Data Entry 82
Run Production Tabulation 283
Program Flow 340, 341
Program Information File 70
Properties 103
Change Field 103
Change Field (Multiple) 105
Change Form 102
Change Forms File 101
Change Level 101
Change Roster 107
Change Roster Column Heading ... 108
Change Text 110
Field 95
Item 28
Level 23
Record 25
Value 33
Value Set 32
Proportion 198
Protected Fields 95
Putdeck Function 378
Putnote 357
QSF 76
In a Data Entry Application 76
Question File 457
Question 457
File .QSF 457
Questionnaire 18
Form 18
Identification 18
Identification Items 27
Organization 18
Questions 18
Responses 18
Section 18
Questionnaire and Dictionary Organization 18
Questions 127
Radio button 95
Random 384
Randomin Function 384
Randomizevs Function 357
Ranges 33
Multiple Value 33
Ratio 198
Proportions 198

Table Post Calculation 274

Reader Breaks 238

Rearrange Things in a Form 97

Recode Statement..... 331

Recodes 307

In Tables Using Value Sets and Subtables 307

Using Table Logic..... 294

Reconciling Dictionary Changes 39

Record 158

Adding..... 41

Delete 45

Description..... 24

Find 46

Insert 45

Label 22

Max 25

Maximum Number..... 26

Modifying 41

Multiple Record Types 20

Name 22

Notes 46

Properties 25

Required..... 26

Single Record Type..... 20

Type 20, 25

Type Value..... 25

Redo and Undo Changes 99

Reenter Statement 358

Reformat 5

Data 5

Relation Statement 323

Relational operators 64

Relations 48

Add and Modify 48

Description..... 33

In Tabulation 298

Properties 34

Relative 38

Mode 38

Positioning 47

Relocate..... 45

Dictionary Elements..... 45

Remainder 64

Remove 14

Drop a File from an Application..... 14

Remove a Variable from a Tabulation 210

Trailing blanks 392

Renaming Tables and Table Applications 208

Reorder Editing 160

Report 174

Automatic 174

Required 86

Enter Key 86

Hardware and Software 429

Record 26

Reserved 317

Commands 317

Words 317

Reset Format of Table Item to Default ... 245

Resize and Reposition Things in a Roster 100

Column Order 100

Column Width 100

Roster Size 100

Row Height 100

Restrict a Universe 187

Re-Test with Live Data 179

Retrieve 426

Function 426

Tables 5

Retrieve Data Files from the Pocket PC . 154

Review Edit Specifications 178

Rich Text Format 270

Save Tables as Text

Rich Text or HTML 270

Right Side of Screen 8

Roster 92

Add to a Form 92

Adding Things 92

Change Column Heading Properties 108

Change Occurrence Labels 109

Change Properties 107

Create 92

Description 73

Join and Split Columns 100

Resize and Reposition Things 100

Rounded To Zero Mask 224

Row 214

Add for Post Calculation 275

Format 221

Hide/Show 231

Percent 214

Spacing 260

Variables 192

Row and Column Indexing for Post Calculation 281

Run 119

A Batch Edit Application 159

A CSPro Tool 5

A Data Entry Application 82

A Tabulation 207

As Batch 119

Data Entry Application as Batch 119

In Parts 285

Parameter 82, 175

Production Batch Edits 175

Production Data Entry 82

Sysparm Function 411

Run All in Batch 284

Run Consolidate in Batch 289

Run Consolidate Interactively 288

Run Format in Batch 291

Run Format Interactively 290

Run PPC Application on the Pocket PC . 152

Run Tabulate in Batch 286

Run Tabulate Interactively 286

Save 15

An application 15

Save Keyword for Arrays 321

Tables 269

Tabulations in Different Formats 188

Save As 16

Application 16

Dictionary 48

Saved Arrays 321

Savepartial Function 358

Saving Tables as Text HTML or Rich Text 270

Screen 13

Full 13

Screen Layout 156

Batch Application 156

Data Dictionary 36

Data Entry Forms 79

Data Entry Logic 114

Initial CSPro Application 7

Screens 5

Search Dictionary Elements 46

Secondary Stub Head 219

Formats for a Part of a Table 219

Parts of a Table 181
Seed Function..... 385
Seek Function..... 400
Seekmax Function..... 400
Seekmin Function 401
Selcase Function 359
Select..... 45
Dictionary Elements..... 45
Errmsg Function..... 404
Items in a Form..... 94
Relative or Absolute Positioning 47
Table Cells 271
Sequence 116
Dictated by CSPro..... 116
Dictated by Designer..... 118
Sequential Fields..... 95
Set 360
Attributes Statement..... 360
Compiler Defaults..... 58
Explicit..... 318
Implicit..... 318
Statement..... 318
System Settings..... 58
Set Behavior Canenter Statement 362
Set Behavior Export Statement..... 379
Set Behavior SpecialValues Statement... 385
Set Errmsg Statement..... 364
Setcapturepos Function..... 362
Setcapturetype Function..... 363
Setfile Function..... 426
Setfont Function..... 364
Setlanguage Function..... 366
Setorientation Function..... 339
Setoutput Function..... 379
Setup 431
Setvalue Function..... 335
Setvalueset Function 366
Setvaluesets Function..... 367
Share the Same Format on Multiple Computers 269
Sharing External Files..... 67
Show Case Tree 86
Show Function 368
Show Values for Selection..... 135
Showarray Function 369
Single Record Types 20
Size..... 90

Change Field Box 90

Size of Data Files 20

skip 71

Automatic 71

Issues 71

Manual 71

Manual Skip To 103

Skip Case Statement 379

Skip Statement 370

Skip 370

Soccurs Function 401

Sort 402

Function 402

Sort Data 5

Sound 90

Change 90

Spacing 260

Modify Row and Column Spacing for Printing 260

Span 244

Make Captions Span Data Cells 244

Spanner 219

Formats for a Part of a Table 219

Parts of a Table 181

Special 410

Create a Specialized Report 171

Function 410

Output Dictionary 38

Special Values 63

Defined 63

Include/Exclude Special Values in a Tabulation 213

Specific 334

Impute Function 334

Specify Application File Names 16

Sqrt 385

Standard Deviation 216

Add Summary Statists to a Table 216

Produce Summary Statistics 186

Start Page 187

Starting Position of Item 29

StartMode 82

Stat 334

Impute Function 334

Statement Format Symbols 317

Statements 54

Alphabetical Listing 312

Assignment 54

Executable 54
Static Imputation 163
Stop Function 410
String 62
Expressions 62
String to Number 393
Substring Expressions 62
Text 61
Variables 52
String Statement 321
Strip function 392
Structure Edits 3, 112
Structure Movement 134
STS 458
Data File Status 458
Stub 246
Change Stub Column Position 246
Format Tables for Printing 187
Formats for a Part of a Table 219
Leadering 236
Parts of a Table 181
Stub head 219
Formats for a Part of a Table 219
Parts of a Table 181
Subgroupings 209
Add a Variable to a Tabulation 209
Create a Table 192
Format/Hide Rows/Columns in 303
Subitems 27
Convert to Item 47
Item Type 27
Sub-Items 27
Subscriped Variables 60
Substring Expressions 62
Subtable 189
Create Multiple Subtables 189
Parts of a Table 181
Recodes in Tables Using Subtables 307
Tally Attributes for a Subtable 203
Using Subtables 292
Subtitle 236
Adding a Subtitle 236
Formats for a Part of a Table 219
Parts of a Table 181
Subtotals 300
Sum Function 402

Summary 435
Batch Edit Keyboard..... 435
Batch Editing Menu 440
Batch Editing Toolbar..... 444
CSPro Toolbar 441
Data Dictionary Keyboard 432
Data Dictionary Menu..... 437
Data Dictionary Toolbar 442
Data Entry Keyboard 433
Data Entry Menu..... 438
Data Entry Toolbar 443
Errmsg Function..... 404
Generalized Menu..... 437
Tabulation Keyboard 436
Tabulation Menu..... 440
Tabulation Toolbar..... 444
Summary Statistics..... 216
Add Summary Statistics to a Table..... 216
Produce Summary Statistics 186
Support..... 15
SVA..... 462
Saved Arrays File..... 462
Swap Function 403
Switching Between Data Entry Forms and Dictionary 79
Synchronization 462
Sysdate Function..... 415
Sysparm Function 411
System..... 72
Vs Operator Controlled..... 72
Systime Function 416
TAB..... 459
Table Matrices File .TAB 459
Table 206
Add..... 206
Change Title..... 230
Parts..... 181
Preview 264
Print..... 264
Retrieval..... 5
Save..... 269
Select Cells..... 271
Tree 8
Viewer..... 5
Window..... 8
Table Logic 294
Table Matrices File .TAB 459

Table Matrices Index File .TAI 459
tablogic..... 294
Table Logic 294
Tabulate..... 250
By Geographic Area 250
Categories with disjoint values 302
Counts or Percents 186
Data 2
Frequencies 5
Items with Multiple Occurrences..... 197
Only Certain Levels of Geography 257
Percents 214
Run Tabulate in Batch 286
Run Tabulate Interactively..... 286
Tool List..... 5
Undefined Values..... 213
Values
About..... 186
Values and/or Weights..... 186
Values Instead of Frequencies 213
Weights
About..... 186
Tabulate Items in Relations 299
Tabulation 192
Creating a Tabulation..... 192
Introduction to Tabulation 180
Run 207
Save in Different Formats..... 188
Tabulation Application File .XTB 458
Tabulation Keyboard Summary..... 436
Tabulation Menu Summary 440
Tabulations..... 185, 194
Toolbar 444
Universe 210
Tabulation Application 4
About..... 4
Creating..... 191
Tabulation keys..... 436
TAI 459
Table Matrices Index File .TAI 459
Tally Attributes 203
For a Table/Subtable 203
For a Variable 198
Tally Items from Related Records 191
TBW..... 459
Tables File..... 459

Temporary Data File 454
Test 138
Application 138
CSPro Program 179
Text 93
Adding to a Form 93
Align 98
Change Properties 110
Customize Table Text 230
Default Font 89
Save Tables as Text
Rich Text or HTML 269
Strings 61
Tool List 5
Viewer 5
Text Box 95
Text Box Options 125
The Nature of Census and Survey Data .. 448
Thematic Map 255
Create 255
Then 329
If Statement 329
This Item ($) 60
Tile 14
Side by Side 14
Top to Bottom 14
Windows 14
Time 416
Systime Function 416
Title 219, 368
Formats for a Part of a Table 219
Impute Function 334
Parts of a Table 181
Show Function 368
Table Title Template 224
To 337
Advance Statement 337
Skip Statement 370
Tolower Function 393
Tone 90
Error 90
Tonumber Function 393
Tool List 5
Compare Data 5
Concatenate Data 5
Convert Dictionary 5

Convert Shape to Map 5
Export Data 5
Map Viewer 5
Reformat Data 5
Sort Data 5
Table Retrieval 5
Table Viewer 5
Tabulate Frequencies 5
Text Viewer 5
Toolbar Summary 444
Batch Edit 444
CSPro 441
Data Dictionary 442
Data Entry 443
Tabulation 444
Total 217
Debug Table Totals 217
Hide or Change the Postion of the Total 214
Percent 214
Totocc Function 403
Toupper Function 394
Trace Function 411
Trailing blanks 392
Remove 392
Tree 79
Batch Edit 158
Data Dictionary 37
Data Entry 81
Initial Screen Layout 7
Names in 13
Restore 13
Trees 8
View 156
Truth Table 67
And/Or 67
If and Only If 66
Type 27
Item 27
Record Type 25
Subitem 27
Type of... 72
Data Entry Application 72
Dictionary 38
Edits in Batch Editing 169
Edits in Data Entry 111
Errors in Censuses and Surveys 448

Table 194

Undefined Values..... 213

Dump..... 213

Tabulate..... 213

Undo and Redo Changes 99

Undo or Reset Changes in Print Preview 265

Unicode Primer 9

Uninstalling CSPro 430

Unit of Tabulation..... 191

Changing the Unit of Tabulation 191

Tally Attributes for a Table..... 203

Universe 210

Defining in Tabulations 210

Universe Statement 330

Using expressions in 311

Until 327

Do Statement..... 327

Upper..... 86

Case..... 86

Case Fields 95

Use 172

Hot Decks..... 172

Multiple Language in CAPI..... 133

Use and Shared External Files 2

Use Expressions in Universe and Value Tallied 311

User Support 15

Userbar Function..... 371

User-Defined Functions 53

Using 68

Lookup Files 68

Multiple Languages in CAPI 128

Sort Function..... 402

Using Print Preview 257

Using Subtables 292

Using Table Data in the Map Viewer 272

UTF-8..... 9

Validating Data 3

Value 45

Adding..... 41

Delete 45

Description..... 32

Find 46

Insert 45

Label 22

Modify..... 41

Notes 46

Properties 33
Record Type..... 25
Search..... 46
Undefined..... 213
Value Set..... 31
Adding..... 41
Delete 45
Description..... 31
Disjoint..... 302
Dynamic modification 366
Find 46
Implications in Tabulation 196
Insert 45
Label 32
Linked Value Sets 42
Modify..... 41
Name 22
Notes 46
Properties 32
Recodes in Tables Using Value Sets .. 307
Search..... 46
Value Tallied..... 186
Tabulate Values and/or Weights..... 186
Tabulate Values Instead of Frequencies 186
Using expressions in 311
Values 128
Variables 209
Adding to a Table..... 209
In CSPro Language 52
Variance 216
Add Summary Statistics to a Table..... 216
Produce Summary Statistics 186
Varying 327
Do Statement..... 327
Verify 86
Demode Function..... 339
Every nth case 86
View 13
Change 13
Dictionary Layout 40
Full Screen 13
Logic 56
Names in Tree 13
Print Tables 264
Viewing Multiple and Facing Pages..... 259
Views of Table..... 229

Visualvalue Function 375
VSet..... 334
Impute Function 334
Web 274
Prepare Tables for Posting to the Web 274
Weights 212
Add Weights to Table 212
Tabulate Values and/or Weights..... 186
What is CSPro..... 1
Where 394
Average Function..... 394
Count Function..... 395
Max Function 397
Min Function..... 398
Show Function 368
Sum Function 402
While Statement..... 330
Width of listing file..... 175
Window..... 79
Windows 8
Change 14
Tile Side by Side..... 14
Tile Top to Bottom..... 14
Working Dictionary 38
Working Storage Dictionary 294
Table Logic 294
Working Storage File..... 69
Wrap messages..... 175
Write 159
File 159
Function 428
Run an Application 159
Specialized Edit Report..... 171
Writecase Function 427
WRT..... 171
Create a Specialized Report 171
XTB..... 191
Tabulation Application 191
Tabulation Application File 458
Tabulation Applications..... 4
XTS 191
Table Specifications File..... 459
Tabulation Application 191
Tabulation Applications..... 4
Zero Fill 31
Zero Mask 224