

CSPro

User's Guide

Version 2.6

International Programs Center
U.S. Census Bureau
Washington DC 20233-8860

Phone: 1-301-763-1451
Fax: 1-301-457-3033
E-mail: CSPro@lists.census.gov

9 September 2005

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..... | 4 |
| Cross Tabulation Applications | 4 |
| Data Dictionary | 5 |
| Forms Design | 5 |
| Tool List | 6 |
| CSPro General Concepts | 7 |
| CSPro Initial Screen Layout | 7 |
| Trees..... | 8 |
| Windows | 9 |
| How To | 9 |
| Create a CSPro Application..... | 9 |
| Change the View | 10 |
| Change Windows | 10 |
| Insert or Drop a File from an Application..... | 10 |
| Change the Print Page Setup..... | 11 |
| Print All or Part of a Document..... | 11 |
| Get Help..... | 11 |
| Save an Application | 12 |
| Close an Application | 12 |
| Copy an Application | 12 |
| Pack an Application | 13 |
| Data Dictionary Module..... | 13 |
| Introduction to Data Dictionary..... | 13 |
| Organization | 14 |
| Questionnaire and Dictionary Organization..... | 14 |
| Data File Type Structure..... | 16 |
| Dictionary Hierarchy | 17 |
| Dictionary Concepts | 18 |
| General | 18 |
| Labels..... | 18 |
| Names..... | 18 |
| Notes | 18 |
| Levels | 18 |
| Level Description..... | 18 |
| Level Properties | 19 |
| Records..... | 20 |
| Record Description..... | 20 |
| Record Properties | 21 |
| Record Type..... | 21 |
| Required Record | 22 |
| Maximum Number..... | 22 |
| Items | 23 |
| Item Description | 23 |
| Identification Items | 23 |
| Sub-Items..... | 23 |

| | |
|--|----|
| Item Properties | 24 |
| Starting Position | 25 |
| Length Function | 25 |
| Data Type | 26 |
| Occurrences | 26 |
| Decimal Places | 27 |
| Decimal Character | 27 |
| Zero Fill | 27 |
| Value Sets | 28 |
| Value Sets Description | 28 |
| Value Set Properties | 29 |
| Values | 29 |
| Value Description | 29 |
| Value Properties | 29 |
| Relations | 30 |
| Relation Description | 30 |
| Relation Properties | 30 |
| Data Dictionary Application | 31 |
| Creating a Dictionary for a New File | 31 |
| Creating a Dictionary for an Existing File | 32 |
| Data Dictionary Screen layout | 32 |
| Data Dictionary Tree | 34 |
| Relative and Absolute Mode | 34 |
| Dictionary Types | 35 |
| Reconciling Dictionary Changes | 36 |
| How to | 36 |
| Open an Existing Dictionary Application | 36 |
| Move Around a Dictionary | 36 |
| View the Dictionary Layout | 37 |
| Add Dictionary Elements | 37 |
| Modify Dictionary Elements | 38 |
| Add or Modify Levels | 38 |
| Add or Modify Records | 38 |
| Add or Modify Items | 39 |
| Add or Modify Value Sets | 39 |
| Add or Modify Values | 39 |
| Undo and Redo Changes | 40 |
| Select Several Dictionary Elements | 40 |
| Insert Dictionary Elements | 40 |
| Delete Dictionary Elements | 41 |
| Move Dictionary Elements | 41 |
| Find Dictionary Elements | 41 |
| Document Dictionary Elements | 42 |
| Convert Items to Subitems | 42 |
| Select Relative or Absolute Positioning | 42 |
| Add or Modify Relations | 42 |
| Print the Dictionary File | 42 |
| Save Dictionary As New File | 43 |
| The CSPro Language | 43 |
| Introduction to CSPro Language | 43 |
| Data Requirements | 43 |
| The CSPro Program Structure | 44 |
| Declaration Section | 45 |
| Mode | 45 |
| Files | 46 |
| Variables | 46 |

| | |
|--|----|
| Arrays | 47 |
| User-Defined Functions..... | 47 |
| Procedural Sections | 48 |
| Statements..... | 48 |
| Proc Statement..... | 48 |
| Preproc Statement..... | 49 |
| Postproc Statement..... | 49 |
| Logic | 50 |
| View Logic | 50 |
| Create and Edit Logic..... | 51 |
| Set Compiler Defaults..... | 51 |
| Compile Logic..... | 52 |
| Language Elements..... | 52 |
| Delimiters..... | 52 |
| Comments | 52 |
| Variables and Constants..... | 53 |
| Data Items..... | 53 |
| This Item (\$)..... | 53 |
| Subscripts | 53 |
| Numbers..... | 54 |
| Text Strings | 55 |
| Expressions..... | 55 |
| Expressions..... | 55 |
| Substring Expressions | 56 |
| Special Values | 57 |
| Operators | 57 |
| Operators | 57 |
| In Operator..... | 58 |
| If and Only If Operator <=> | 59 |
| Operator Precedence..... | 59 |
| And/Or Truth Table | 59 |
| Files..... | 60 |
| External Files | 60 |
| Lookup Files..... | 60 |
| Working Storage File..... | 61 |
| Program Information File | 61 |
| Data Entry Module | 62 |
| Introduction to Data Entry | 62 |
| Data Entry Application | 62 |
| General Data Entry Concepts..... | 62 |
| Data Entry Philosophies..... | 62 |
| Skip Issues..... | 63 |
| Errors at Data Entry | 63 |
| Adding Logic | 64 |
| CSPRO Data Entry Concepts | 64 |
| Operator vs. System Controlled..... | 64 |
| Data Entry Path..... | 65 |
| Data Entry Elements | 65 |
| Issues to Consider When Designing a Form | 66 |
| Create a Data Entry Application..... | 67 |
| Create a New Data Entry Application | 67 |
| Generate Default Data Entry Forms | 69 |
| The Drag Option Menu | 69 |
| Data Entry Forms Screen Layout..... | 71 |
| Data Entry Tree..... | 73 |
| Run a Data Entry Application..... | 74 |

| | |
|---|-----|
| Run Production Data Entry | 75 |
| Change Data Entry Characteristics..... | 77 |
| Change the Order of Entry..... | 77 |
| Change Data Entry Options | 77 |
| Change Default Text Font..... | 80 |
| Change Field Font..... | 81 |
| Change Error Sound | 81 |
| Forms Designer..... | 82 |
| Introduction to Forms Design..... | 82 |
| Add Things to a Form..... | 82 |
| Add a Form | 82 |
| Add Fields to a Form..... | 83 |
| Add a Roster to a Form..... | 83 |
| Add Things to a Roster | 83 |
| Add Text to a Form | 84 |
| Add Lines or Boxes to a Form | 85 |
| Modify Things in a Form | 85 |
| Selecting Items..... | 85 |
| Field Properties | 86 |
| Move Things..... | 87 |
| Align Things | 87 |
| Undo and Redo Changes | 88 |
| Cut, Copy, or Paste Things..... | 89 |
| Resize and Reposition Things in a Roster..... | 89 |
| Join and Split Roster Columns..... | 90 |
| Delete Form Elements | 90 |
| Matching the Application to the Data Dictionary | 90 |
| Change Form Properties..... | 90 |
| Change Forms File Properties | 90 |
| Change Level Properties | 91 |
| Change Form Properties..... | 91 |
| Change Field Properties | 92 |
| Change Roster Properties | 94 |
| Change Column Heading Properties | 95 |
| Change Row Heading Properties | 96 |
| Change Text Properties | 97 |
| Data Entry Editing..... | 98 |
| Introduction to Data Entry Editing..... | 98 |
| Editing Concepts | 98 |
| Type of Edits in Data Entry | 98 |
| Structure Edits..... | 99 |
| Consistency Edits..... | 100 |
| Checking errors..... | 100 |
| Writing Logic | 101 |
| Data Entry Logic Screen Layout | 101 |
| Moving Around a Logic Application..... | 103 |
| Order of Executing Data Entry Events..... | 103 |
| Sequence dictated by designer..... | 105 |
| Compile an Application | 106 |
| Run as Batch..... | 106 |
| CAPI Data Entry | 108 |
| Introduction to CAPI | 108 |
| CAPI Features | 108 |
| CAPI Strategies..... | 109 |
| Forms | 109 |
| Fields..... | 109 |

| | |
|---|-----|
| Questions | 110 |
| Values | 110 |
| Organization of the Instrument..... | 110 |
| Using Multiple Languages..... | 111 |
| Breaking Off the Interview..... | 111 |
| Coming Back Later..... | 112 |
| How to | 112 |
| Create a New CAPI Application | 112 |
| Define Languages..... | 113 |
| Organize Forms | 114 |
| Enter Question Text..... | 114 |
| Create Fills In Questions..... | 114 |
| Create Standard Forms | 115 |
| Change Formatting | 115 |
| Add Pictures..... | 116 |
| Use Multiple Language | 116 |
| Create Conditional Questions..... | 117 |
| Display Questions Without Scrolling..... | 117 |
| Structure Movement..... | 117 |
| Create Helps for Fields | 118 |
| Show Values for Selection | 118 |
| Handle Don't Know and Refused..... | 119 |
| Handle Multiple Answers | 119 |
| Choose Topic Sections | 119 |
| Create General Helps | 120 |
| Test Application..... | 121 |
| Batch Editing Application | 121 |
| Introduction to Batch Editing..... | 121 |
| Create a Batch Edit Application..... | 122 |
| Create a New Batch Edit Application | 122 |
| Batch Application Screen Layout | 123 |
| Batch Edit Tree..... | 125 |
| Run a Batch Edit Application..... | 126 |
| Order of Editing | 127 |
| Order of Executing Batch Edit Events | 127 |
| Batch Edit Order | 127 |
| Change Edit Order..... | 128 |
| Correcting Errors | 129 |
| Methods of Correcting Data..... | 129 |
| Guidelines for Correcting Data | 129 |
| Imputation..... | 130 |
| Static Imputation..... | 131 |
| Dynamic Imputation (Hot Deck)..... | 132 |
| Types of Edits in Batch Editing..... | 132 |
| How to | 134 |
| Manipulate Automatic Reports | 134 |
| Create a Specialized Report..... | 135 |
| Use Hot Decks..... | 136 |
| Hot Decks Inline | 136 |
| Hot Decks in External File | 136 |
| Interpret Reports..... | 137 |
| Run Production Batch Edits..... | 139 |
| Steps in Developing a Batch Editing Program | 141 |
| General Issues | 141 |
| Review Edit Specifications | 141 |
| Define Coding Standards | 142 |

| | |
|--|-----|
| Code Edits of Individual Data Items | 142 |
| Develop Comprehensive Test File | 143 |
| Test CSPro Program | 143 |
| Re-Test with Live Data | 143 |
| Begin Production Editing | 144 |
| Cross Tabulation Application | 144 |
| Introduction to Cross Tabulation | 144 |
| Parts of a Table | 145 |
| Total | 145 |
| Ever Married..... | 146 |
| Never Married | 146 |
| Row 1 | 146 |
| Common Uses of CrossTab..... | 146 |
| Capabilities of Cross Tab | 147 |
| Frequency Distribution..... | 147 |
| Cross Tabulations..... | 147 |
| Tabulate Counts or Percents..... | 147 |
| Tabulate Values and/or Weights | 148 |
| Restrict a Universe | 148 |
| Produce Tables by Area..... | 148 |
| Save Tabulations in Different Formats | 149 |
| Copy Table to Other Formats | 149 |
| Map Results by Geographic Area | 149 |
| Create a Cross Tab Application..... | 150 |
| Create a new Tabulation application | 150 |
| Creating a Frequency Distribution | 151 |
| Creating a Cross Tabulation..... | 152 |
| Include Percents..... | 152 |
| Handle Undefined Values..... | 154 |
| Include Values and Weights | 155 |
| Define a Universe | 156 |
| Save Tables..... | 157 |
| Print Tables..... | 158 |
| Run a Tabulation | 158 |
| Creating Cross Tabulations by Geographic Area..... | 159 |
| Area Processing | 159 |
| Create an Area Names File | 160 |
| Area IDs Dialog Box | 161 |
| Create a Thematic Map of Results | 162 |
| Using Cross Tab..... | 163 |
| Implications of Data Dictionary Value Names | 163 |
| Tabulate Items with Multiple Occurrences | 164 |
| Types of Table..... | 164 |
| Manipulating Tables | 165 |
| Change Tabulation Parameters..... | 165 |
| Change the Table Title | 165 |
| Add a Table | 166 |
| Insert a Table..... | 166 |
| Modify a Table | 167 |
| Delete a Table | 167 |
| Select Table Cells..... | 167 |
| Copy All or Part of a Table | 168 |
| CSPro Statements and Functions..... | 168 |
| Alphabetical List..... | 168 |

| | |
|---|------------|
| Statement Format Symbols | 170 |
| List of Reserved Words | 171 |
| Declaration Statements | 172 |
| Set Statement | 172 |
| File Statement | 173 |
| Numeric Statement | 173 |
| Alpha Statement | 174 |
| Array Statement | 174 |
| Relation Statement | 176 |
| Function Statement | 177 |
| Program Control Statements | 178 |
| Break Statement | 178 |
| Do Statement | 178 |
| Exit Statement | 179 |
| For Statement | 180 |
| If Statement | 181 |
| Next Statement | 181 |
| While Statement | 182 |
| Assignment Statements | 182 |
| Assignment Statement | 182 |
| Recode (Box) Statement | 183 |
| Impute Function | 185 |
| Data Entry Statements and Functions | 187 |
| Accept Function | 187 |
| Advance Statement | 187 |
| Demode Function | 188 |
| Editnote Function | 188 |
| Endlevel Statement | 189 |
| Endgroup Statement | 189 |
| Enter Statement | 190 |
| Getnote Function | 190 |
| Getoperatorid Function | 191 |
| Highlighted Function | 191 |
| Ispartial Function | 191 |
| Killfocus Statement | 192 |
| Move Statement | 192 |
| Noinput Statement | 193 |
| Onfocus Statement | 193 |
| Onkey Global Function | 194 |
| Onstop Global Function | 197 |
| Putnote Function | 198 |
| Reenter Statement | 198 |
| Savepartial Function | 199 |
| Selcase Function | 199 |
| Set Attributes Statement | 200 |
| Set Behavior Canenter Statement | 201 |
| Skip Statement | 202 |
| Visualvalue Function | 203 |
| Batch Edit Statements | 203 |
| Set Behavior Export Statement | 203 |
| Skip Case Statement | 203 |
| Export Statement | 204 |
| Numeric Functions | 205 |
| Cmcode Function | 205 |
| Exp Function | 205 |
| Int Function | 206 |

| | |
|------------------------------------|-----|
| Log Function | 206 |
| Random Function | 206 |
| Seed Function..... | 206 |
| Sqrt Function | 207 |
| String Functions..... | 207 |
| Compare Function | 207 |
| Concat Function | 208 |
| Edit Function..... | 208 |
| Getbuffer Function..... | 209 |
| Length Function | 210 |
| Maketext Function | 210 |
| Pos Function..... | 211 |
| Poschar Function..... | 212 |
| Strip Function..... | 213 |
| Tonumber Function | 213 |
| Multiple Occurrence Functions..... | 214 |
| Average Function | 214 |
| Count Function | 214 |
| Curocc Function | 215 |
| Delete Function | 215 |
| Insert Function | 216 |
| Max Function | 217 |
| Maxocc Function | 218 |
| Min Function | 218 |
| Noccurs Function..... | 219 |
| Soccurs Function..... | 219 |
| Sort Function | 220 |
| Sum Function..... | 220 |
| Totocc Function | 221 |
| General Functions | 221 |
| Clear Function | 221 |
| Errmsg (Display) Function..... | 222 |
| Execsystem Function | 223 |
| Getlabel Function | 224 |
| Getsymbol Function..... | 225 |
| Invalueset Function | 225 |
| Special Function | 226 |
| Stop Function..... | 226 |
| Sysdate Function..... | 227 |
| Sysparm Function..... | 228 |
| Systime Function | 228 |
| External File Functions | 229 |
| Close Function..... | 229 |
| Delcase Function..... | 229 |
| Fileconcat Function | 230 |
| Filecopy Function | 230 |
| Filecreate Function | 231 |
| Fileexist Function..... | 231 |
| Filedelete Function | 232 |
| Filename Function | 232 |
| Fileread Function..... | 232 |
| Filerename Function..... | 233 |
| Filesize Function..... | 233 |
| Filewrite Function | 234 |
| Find Function | 234 |
| Key Function..... | 235 |

| | |
|---|-----|
| Loadcase Function | 235 |
| Locate Function | 236 |
| Open Function | 236 |
| Retrieve Function | 237 |
| Setfile Function | 238 |
| Writecase Function | 238 |
| Write Function..... | 239 |
| Appendix | 240 |
| Appendix A - Installation | 240 |
| Hardware and Software Requirements | 240 |
| Installing CSPro | 241 |
| Uninstalling CSPro | 243 |
| Installing a Newer Version..... | 243 |
| Installing Data Entry Applications..... | 244 |
| Changing Language or Components | 244 |
| Appendix B - Keys Summary..... | 245 |
| Data Dictionary Keys | 245 |
| Data Entry Forms Designer Keys..... | 245 |
| Data Entry Application Keys | 246 |
| Batch Edit Keys | 247 |
| CrossTab Keys | 248 |
| Appendix C - Menu Summary | 249 |
| Generalized Menu | 249 |
| Initial Menu | 249 |
| Data Dictionary Menu | 249 |
| Data entry menu | 250 |
| Batch editing menu | 252 |
| Cross Tabulation Menu | 252 |
| Appendix D - Toolbar Summary | 253 |
| CSPro Toolbar | 253 |
| Data Dictionary Toolbar..... | 253 |
| Data Entry Toolbar | 254 |
| Batch Editing Toolbar | 255 |
| Cross Tabulation Toolbar | 256 |
| Appendix E - Converting Within IMPS or ISSA | 257 |
| Converting a data dictionary..... | 257 |
| Converting within IMPS | 257 |
| Converting within ISSA..... | 258 |
| Converting an IMPS Data Entry Application..... | 258 |
| Converting an ISSA Data Entry Application | 259 |
| Appendix F - Errors in Censuses and Surveys | 259 |
| The Nature of Census and Survey Data | 259 |
| Errors in Censuses and Surveys..... | 260 |
| Appendix G - File Types | 262 |
| File Types | 262 |
| Files Description..... | 263 |
| Data Dictionary File (.DCF)..... | 263 |
| Data Entry Application File (.ENT) | 263 |
| Form File (.FMF) | 263 |
| Logic File (.APP) | 264 |
| Messages File (.MGF)..... | 264 |
| Question File (.QSF) | 264 |
| Data file | 264 |
| Listing File (.LST) | 264 |
| Cross Tabulation Application File (.XTB)..... | 265 |
| Table Specifications File (.XTS)..... | 265 |

Table of Contents

| | |
|---|------------|
| Tables File (.TBW) | 265 |
| Area Names File (.ANM) | 265 |
| Batch Edit Application File (.BCH) | 265 |
| Edit Order File (.ORD)..... | 266 |
| Frequency file (.FRQ)..... | 266 |
| Map File (.MAP) | 266 |
| Map Data File (.MDF)..... | 266 |
| Program Information File (.PFF) | 266 |
| Frequency Specification file (.FQF) | 267 |
| Operator Statistics File (.LOG)..... | 267 |
| Index | 269 |

CSPro Users Guide

The CSPro System

What is CSPro

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

CSPro runs under Windows 95, 98, ME, NT 4.0, 2000, or XP. 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 ASCII 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 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], 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. This contains a tutorial that gives you an overview of CSPro's capabilities.

This section contains 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 may also 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 may be used to 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. The different needs of individual users can be catered to by including multiple value sets for variables, 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 ASCII-text format and may be viewed with any text viewer, such as CSPro's utility Text Viewer, and may also be printed. 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 which 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
- Edit_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 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

You can have 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

Cross Tabulation Applications

A **Cross 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 Cross Tabulation application, you can:

- Cross-tabulate up to four variables.

- Select the universe of tabulation.
- Tabulate values and weights.
- Tabulate counts and/or percents.
- Save tabulations in several formats.
- Copy tables to spreadsheets or word-processing documents.
- Produce tables by geographic area.
- Map results by geographic area.

See also: Create a New Cross 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 ASCII file up to a maximum of 32,000 characters 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 CrossTab 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 cross 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). The user can vary the number of intervals, size of the intervals, colors, titles and legends; change lowest geographic level shown; copy the maps to a word processor; or save a map 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 using the identification fields. Data file may be as large as 2 GB.
- **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, they must be described by the same CSPRO dictionary.

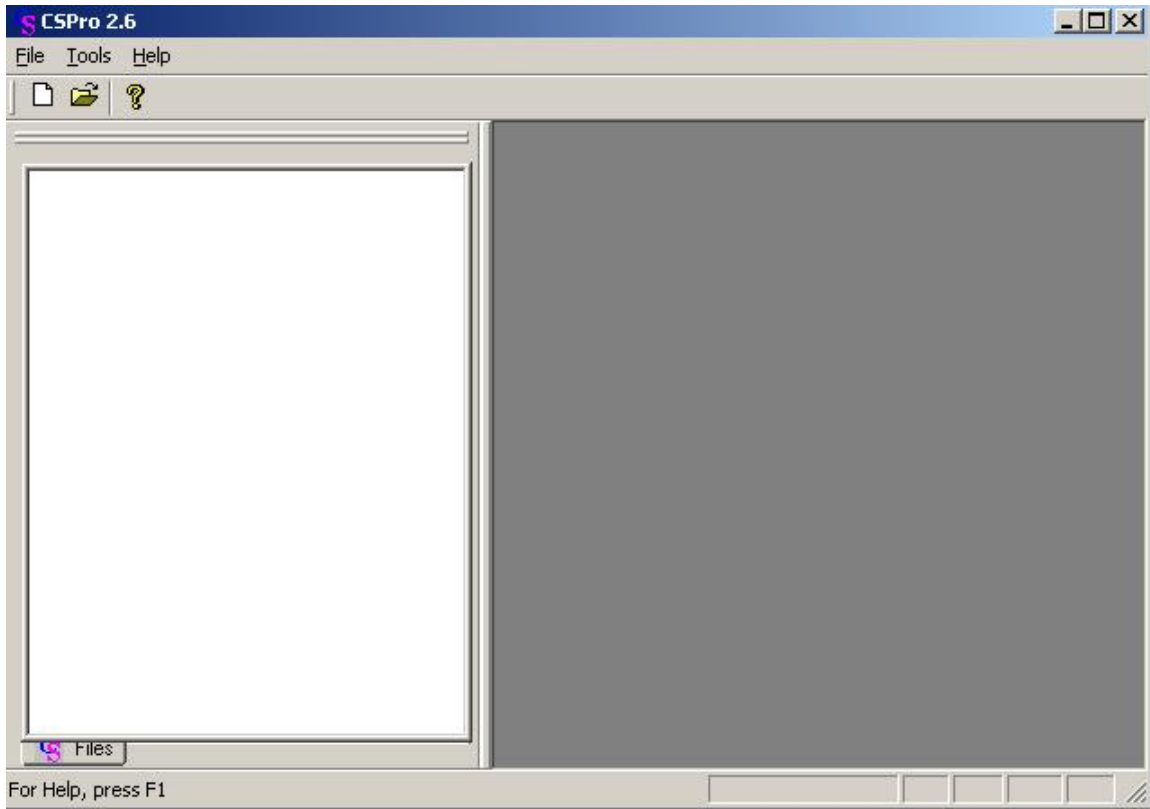
- **Concatenate Data**
Concatenate (join end-to-end) two or more CSPRO data (or other text-based) files. You do not need a dictionary for this utility, you only need to know the name and location of the files you wish to combine.
- **Table Retrieval Setup**
Create and modify a set of tables and other 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.

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.



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 Cross 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.

How To ...

Create a CSPRO Application

Click  on the toolbar; or from the **File** menu, select "New"; or press **Ctrl+N**. Select the type of application or file you want to create and in the next panel enter the names of the files you want to create, and select a folder to store the object. You can create one of the following applications:

- Data Entry Application – to create a data entry application
- **Batch Edit Application** – to create an edit application
- **CrossTab Application** – to create a cross-tabulation application
- **Data Dictionary File** – to create an external dictionary
- **Forms File** – to create a forms file outside an application

The applications and the forms file will also request the name of a data dictionary. Give the name of the primary dictionary file for this object. If the dictionary file already exists, it will be used. If the dictionary file does not exist, it will be created. The data entry application will request, in addition, the name of a forms file. Give the name of the primary form file for this application. Press the **Browse** button to locate an existing form file. If the form file already exists, it will be used. If the form file does not exist, it will be created. Press **Next**.

The summary screen displays the lists of files being created. Verify that the list of files created or used is correct. If the list is correct, you may continue with the next step. If the list is incorrect, you may return to an earlier step to make any necessary corrections before proceeding.

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+U**. 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.

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.


Insert or Drop a File from 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 Cross Tabulation applications.

To insert a file click on the **Files** tab to bring up the files tree. From the **File** menu, select "Insert File" or right-click on the application file and select "Insert File". Select the type and name of the file to be inserted.

To drop a file click on the **Files** tab to bring up the files tree. From the **File** menu, select "Drop". Select the type and name of the file to be dropped.


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 change printer font size, click  on the toolbar; or, from the **Options** menu, select "Change Font Size" and select the font size from the dialog box. CSPRO will remember your font size setting for the next time you run the Text Viewer.

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

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

Phone: 1 (301) 763-1451
Fax: 1 (301) 457-3033
E-mail: CSPro@lists.census.gov

Visit: www.census.gov/ipc/www/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

Copy an Application

Making a copy of an application, for example as a starting point for a new application, is not as easy as it may seem. This is because each CSPro application has a number of files whose names must be known to each other. You cannot simply copy and rename the files associated with the application and have the application work.

To make a copy of an application under a new name:

- **Make copies of the all files in the application, except the application file.** You can find all file names, by opening the application in CSPro and viewing the files tree. Use **Ctrl+T** to show the full file name of each file. Do not make a copy of the application file: .ENT for data entry, .BCH for batch editing, .XTB for tabulation. This file will be recreated later.

- **Rename the files.** For data entry applications, the .FMF, .APP, .MGF, and .QSF files must have the same name. For batch edit applications, the .ORD, .APP, and .MGF files must have the same name. For tabulation applications, the .XTS, .APP, and .MGF must have the same file names. Dictionary files (.DCF) can have different names.
- **Edit the .FMF, .ORD, or .XTS file.** Each of these files contains the file name of the main dictionary file. Using *Notepad*, locate the file name of the old main dictionary file and change it to the new dictionary file name. The dictionary name is found in the dictionaries section near the top of the file.

Example:

```
...
[Dictionary]
File=12/24/2003 1:28:22 PM,.\Census.dcf    { <-change this name}
...
```

- **Create a new application with CSPRO.** Open CSPRO and create a new application with same name you gave to .APP and .MGF files. CSPRO will associate these files with the new application.
- **Insert any external dictionaries or secondary form files.** To insert a file, click on the **Files** tab to bring up the files tree. From the **File** menu, select **Insert Files** or right-click on the application file and select **Insert Files**. Select the type of file, dictionary or form file, and name of the file to be inserted.

Pack an Application

There are times when you want to collect all the files in an application to:

- Move them to another computer, for example 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. .

CSPPro 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

- **Questionnaire**

- **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, and one or more sections on different topics. Some sections in a questionnaire 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.
- **Data Dictionary**
 - **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)
2101001120109196138
2101001212105196732
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: , 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.

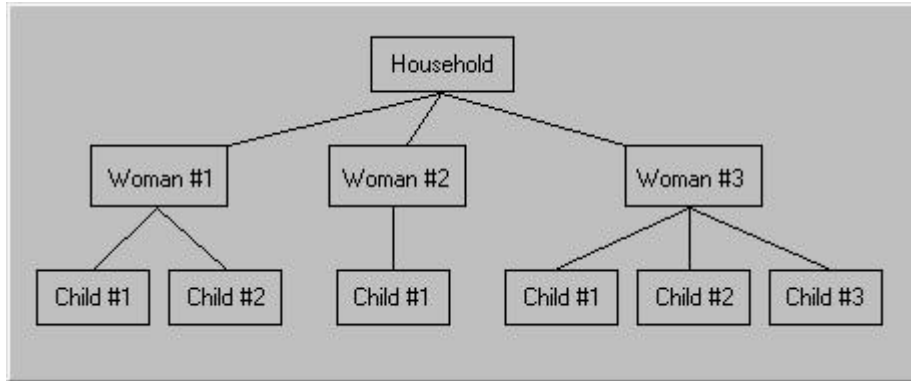
Levels

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

Level 3

Child Record

In the Forms Designer you will be required to place each record's data on different forms (as they are located in different levels). However, this will facilitate 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 (Level 2). When data entry is finished for this woman (and thus for the current level 2 occurrence), the keyer will advance to the final level, and enter information for the first recorded child of this female. After entering data for child #1 (and thereby completing a Level 3 occurrence), the Child Form will reappear, waiting for entry for the next child.

If there are no further children (or no children at all for this female), finish the level by pressing Ctrl+F12 (EndLevel) 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 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](#), [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 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 "Yes" or "No". If a record is required ["Yes"], 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 when this happens. If the record is not required ["No"], 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, and that you keep the maximum to the lowest value that will work 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 help 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.

If you open a dictionary in CSPRO and press **Ctrl+L** (L=layout), you will see a pictorial representation of this (press **Ctrl+L** again to toggle the view off).

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

Sub-Items

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 CrossTab, 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 it was very common to use subitems to redefine data items. This is more easily accomplished now 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 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 cross 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. |

| | |
|----------------------------------|--|
| <u>Occ</u> | The number of times this item will repeat within the record. The default value is "1". Identification items cannot have multiple occurrences. |
| <u>Dec</u> | The number of decimal places (if any) in the item. The default number of decimals is "0". Identification items cannot have decimals. |
| <u>Dec.Char</u> | 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 greater than zero (i.e., Dec >= 1). |
| <u>Zero Fill</u> | 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 sub-item must be within its parent item (the previous item).

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

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](#)

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 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 the CrossTab or MapViewer modules, 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 peoples' 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 CrossTab 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), 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.

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 processed, 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 be 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. |

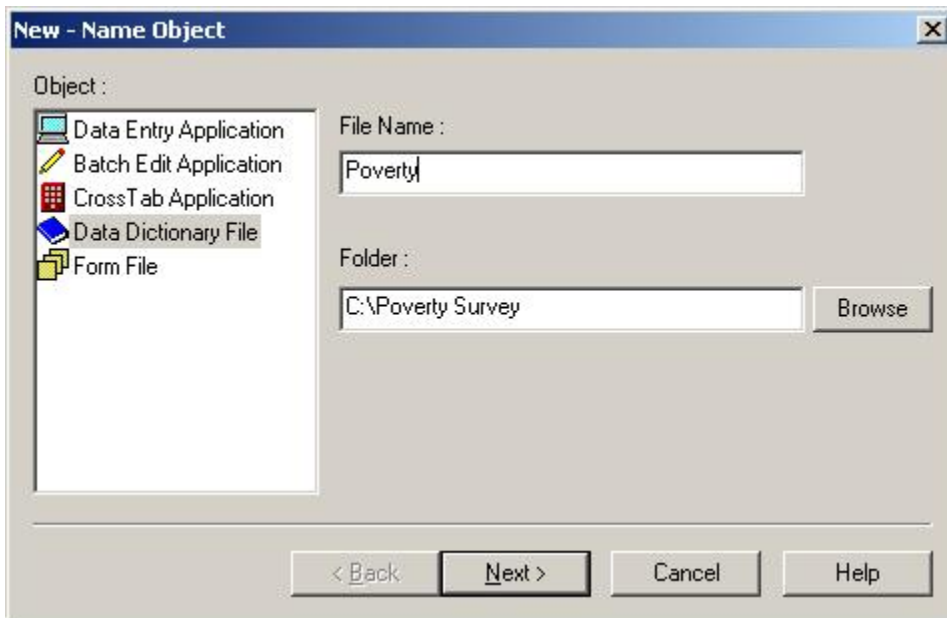
See also: Relation Description

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.

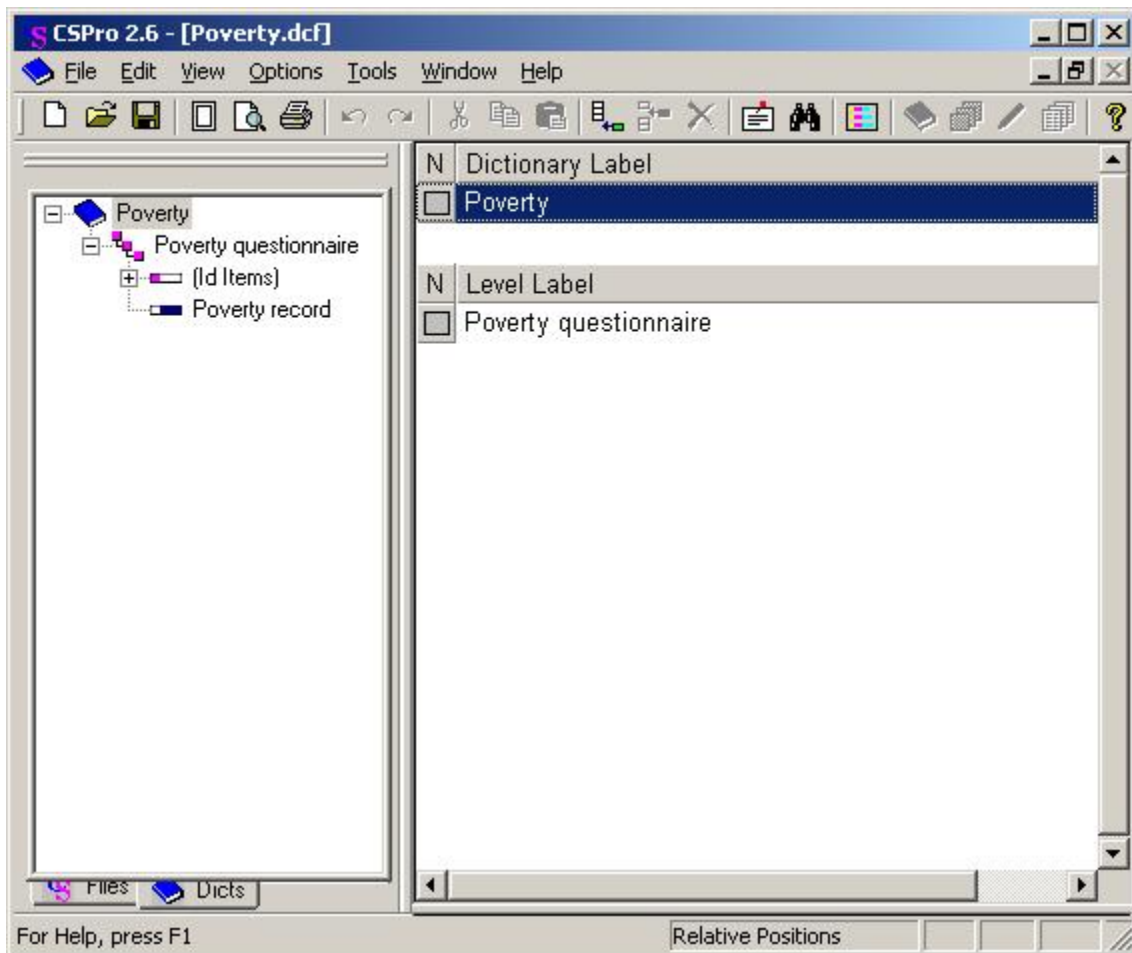
Once you have the record descriptions for the data file you are ready to create the dictionary.

- With CSPPro 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.

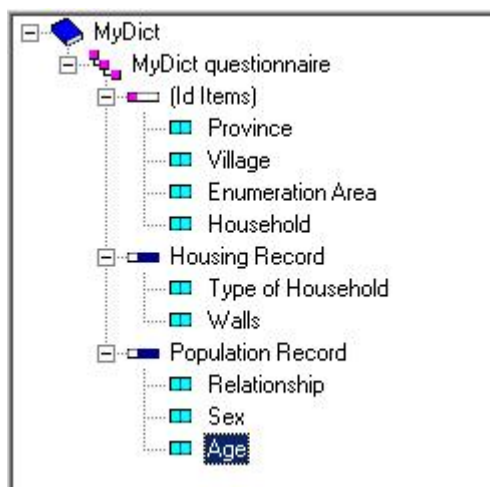
The first thing we suggest you do is to change the level properties to reflect your intended usage for them. Next, change the record properties [e.g., maximum number of records for each type, whether a given record type is required or optional, etc.] and create any needed additional

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 is 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, 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 be 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 which uses this modified data dictionary, any corresponding fields on the data entry forms will be deleted.


Move Around a Dictionary

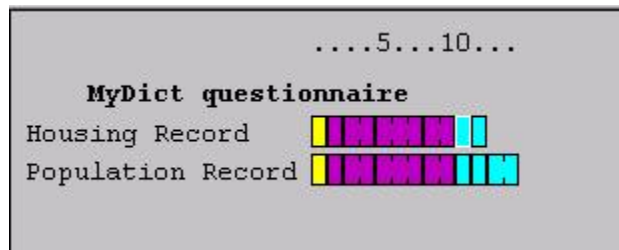
| Press | To |
|------------|--------------------------------------|
| Up Arrow | Move up one line |
| Down Arrow | Move down one line |
| Page Up | Scroll up one screen (if possible) |
| Page Down | Scroll down one screen (if possible) |

| | |
|------------------|--|
| 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 always undo a modification if you decide it was 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 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 (sub)item 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.

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 is sometimes 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 sub-items. From the **Edit** menu, select **Convert to Sub-items** or right-click on the item list in the view and select **Convert to Sub-items** from the pop-up menu. Enter information about the item that will include these sub-item(s).

To convert sub-items back to items, delete the item. When asked if you wish to "Delete sub-items 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.


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

From the **File** menu, select "**Save As.**" The dictionary in the current frame (right side of the screen) will be saved to a new file. You will be asked to enter the name of the new file.

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 CSPPro must be ASCII 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 ASCII text file before you can use it with CSPPro. Data files are limited to 2 gigabytes in overall size; the maximum length of any record in the file is 32,000 characters.

CSPPro 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. CSPPro 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.

CSPPro 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.

CSPPro 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 ASCII 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 CSPPro, 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 CSPPro Program Structure

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

- **Declaration Section**

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 }
  Relly (5) = 5; { sibling 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 of Executing Batch Edit Events

Declaration Section

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

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 statement.

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

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 the alpha statement. This is true whether you have chosen "Set explicit" or "Set implicit" for numeric variables.

See Also: Mode, Arrays, Set Statement,

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.

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 numeric variable that is used by the statements within the function. These 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. All other variables used in the function are global in scope; they can be changed anywhere in the application, including inside the function.

A user-defined function

- Returns a single numeric value.
- 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., they cannot call themselves), though they can call other functions (either user-defined or system-supplied).

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

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_ratio = 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 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:

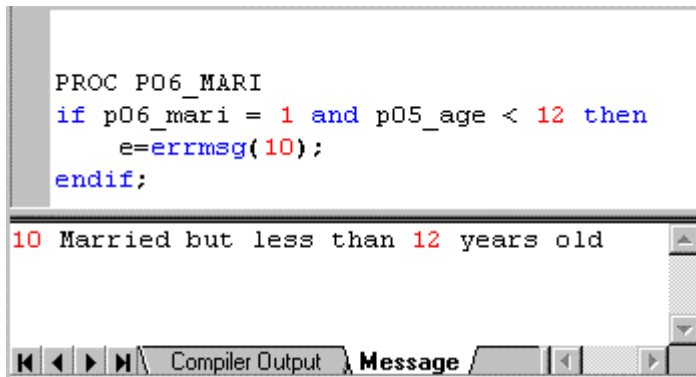
```
PROC SEX
  {preproc would go here, if desired }
postproc
  if ($ = 2 and AGE < 5) then
    reenter;
  endif;
```

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;
```

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:


| <u>System Setting</u> | <u>Program Setting</u> | <u>Result</u> |
|------------------------|------------------------|---|
| ✓ <u>S</u> et Explicit | <u>set explicit</u> ; | No affect, as program matches system setting |
| ✓ <u>S</u> et Explicit | <u>set implicit</u> ; | Program overrides system setting, variables do not need to be declared |
| <u>S</u> et Explicit | <u>set explicit</u> ; | Program overrides system setting of implicit— |

| | | |
|----------------------|----------------------------|--|
| <u>S</u> et Explicit | <code>set implicit;</code> | variables must be declared No affect, 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 cannot be nested, that is, comments within comments are not allowed.

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 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.

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 | % |
| Exponentiation | ^ |

- **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 |

- **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
```

If and Only If Operator <=>

This operator has the following truth table.

| X | if and only if [x <=> y] | |
|-------|---------------------------|---------|
| | Y true | Y 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 |
| 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.

| X | and [x and y] | |
|-------|----------------|---------|
| | Y true | Y false |
| true | true | false |
| false | false | 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 an ASCII 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 an ASCII 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 2.5\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

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 inconsistencies 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.

The screenshot displays a data entry form with three main sections:

- Identification Items:** A vertical list of fields with input boxes: Province, Village, Enumeration Area, and Household.
- Housing Unit Items:** A vertical list of fields with input boxes: Type of Household and Walls.
- Population Items:** A grid table with 6 rows and 4 columns. The columns are labeled Relationship, Sex, and Age. Each cell contains a small input box. A vertical scrollbar is on the right side of the grid.

| | Relationship | Sex | Age |
|---|----------------------|----------------------|----------------------|
| 1 | <input type="text"/> | <input type="text"/> | <input type="text"/> |
| 2 | <input type="text"/> | <input type="text"/> | <input type="text"/> |
| 3 | <input type="text"/> | <input type="text"/> | <input type="text"/> |
| 4 | <input type="text"/> | <input type="text"/> | <input type="text"/> |
| 5 | <input type="text"/> | <input type="text"/> | <input type="text"/> |
| 6 | <input type="text"/> | <input type="text"/> | <input type="text"/> |

- **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.

| | Line number | Relationship | Sex | Age |
|---|----------------------|----------------------|----------------------|----------------------|
| 1 | <input type="text"/> | <input type="text"/> | <input type="text"/> | <input type="text"/> |
| 2 | <input type="text"/> | <input type="text"/> | <input type="text"/> | <input type="text"/> |
| 3 | <input type="text"/> | <input type="text"/> | <input type="text"/> | <input type="text"/> |

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:

Housing Unit Items

| | |
|--------------------------|----------------------|
| Type of Household | <input type="text"/> |
| Walls | <input type="text"/> |

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.

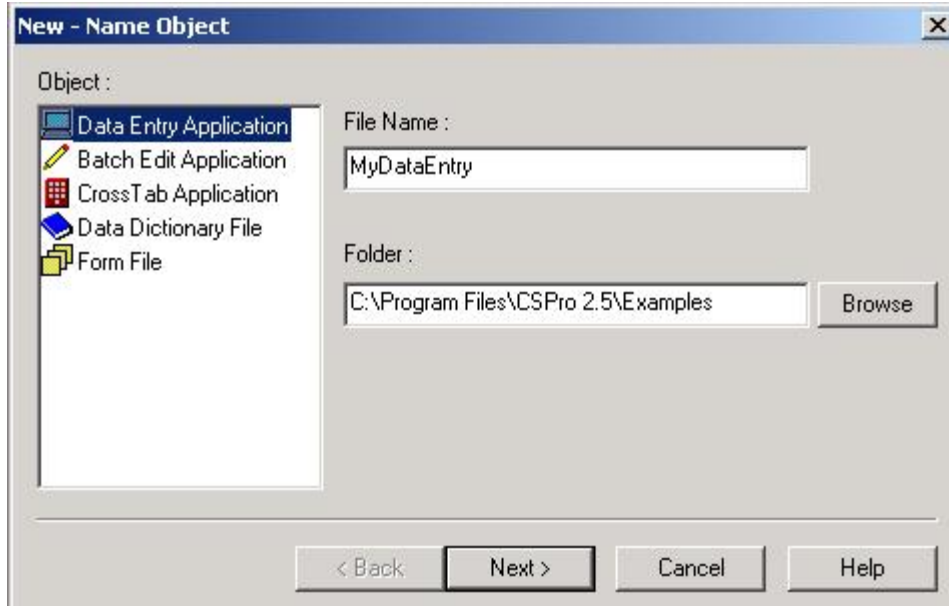
The data entry application may then contain two levels—one for the household and one for each woman in the household. The set of forms corresponding to the household make up level one. The set of forms corresponding to each woman make up level two. Each case would consist of a level one and a variable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

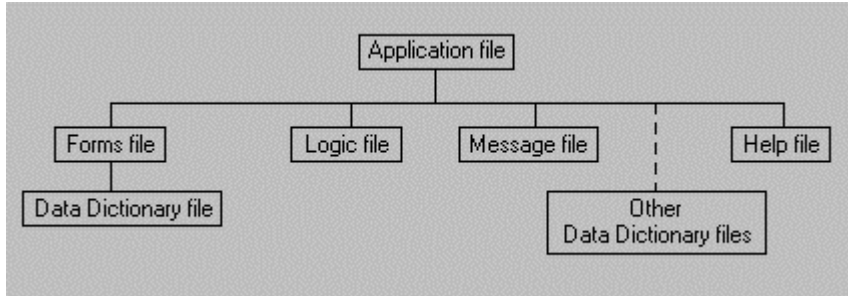
To create a new data entry application:

- Click  on the toolbar, or from the **File** menu, select **New**.



- In the Object window select **Data Entry Application**.
- Enter the file name for the application.
- Enter the name of, or select, the folder where the application will be stored. Click on **Next**.
- Enter the file name for the data entry forms, or click on **Next** to accept the default name. If you converted an ISSA dictionary to a form file, enter the name of the file you created, then click on **Next**.
- If you already have a CSPro data dictionary, select its file name. If not, enter a new name and the system will create a new dictionary for you. Click on **Next**.
- You will be given a list of the files that will be created or used. If the list is correct, press **Finish**. Otherwise, press **Back** to make changes.
- 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 (.app)**
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

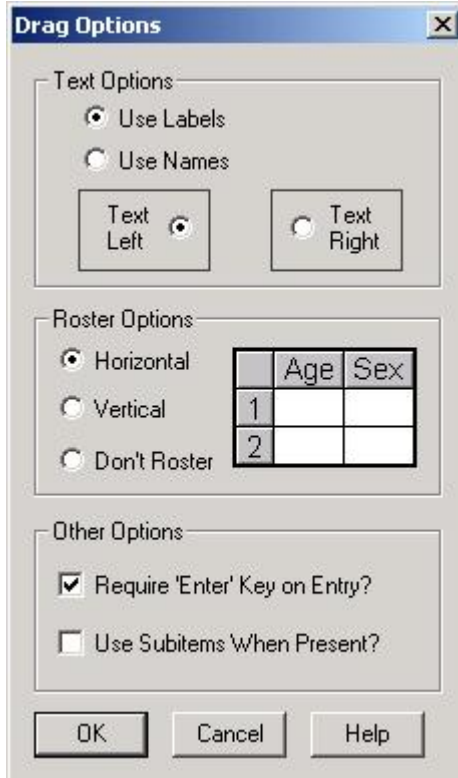
CSPro can automatically generate data entry forms that places all dictionary items onto forms. This can save time as it quickly builds up your form(s), allowing you to easily customize them to your specific needs. One form will be created for each dictionary record (identification items get their own 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.

- **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?**

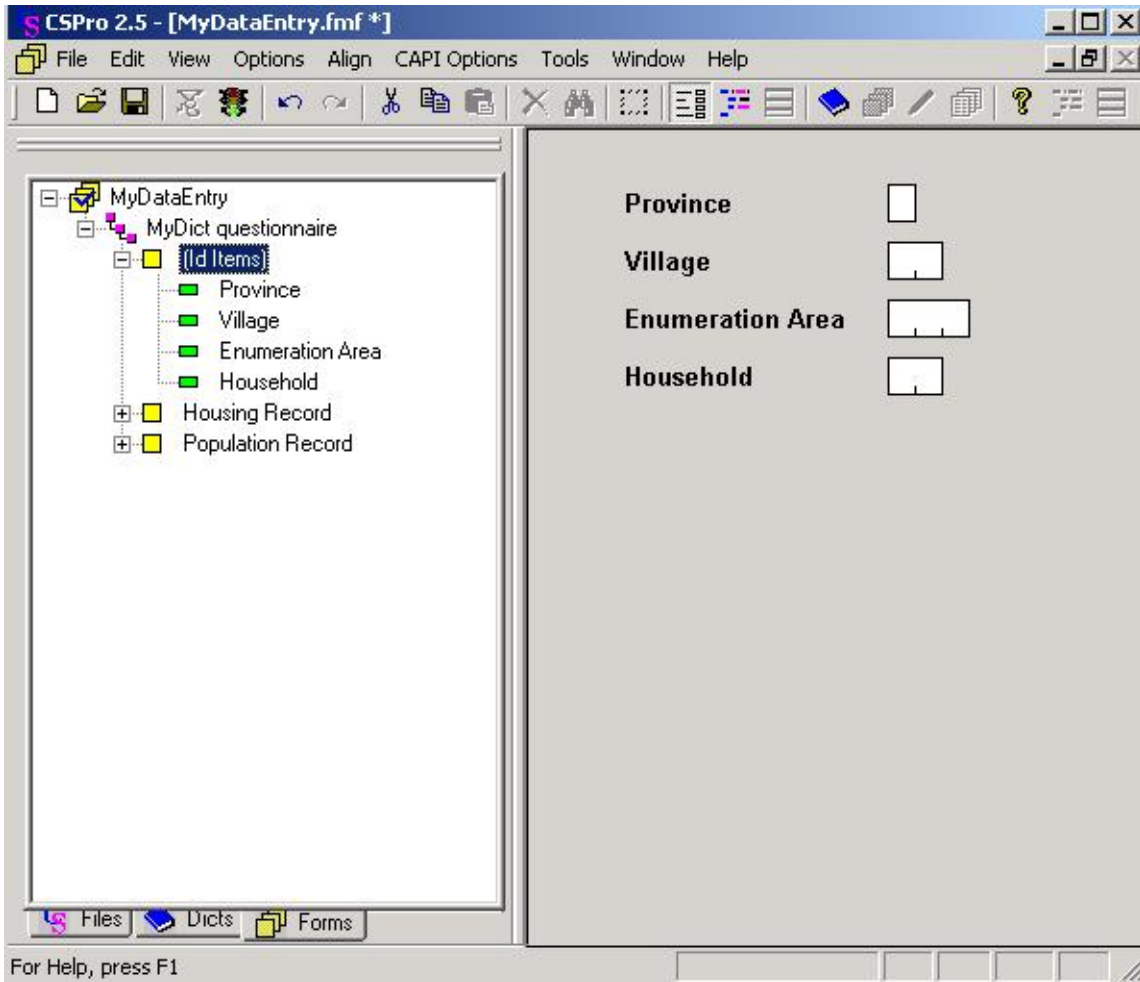
If you have items with sub-items, you may **check** this box to place the sub-items, instead of the item, on the form. For example, if you have a Date item that contained the three sub-items Day, Month, and Year, the sub-items, rather than the item Date, would be placed on the form. However, if any of 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.

Data Entry Forms Screen Layout

The CSPPro 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.

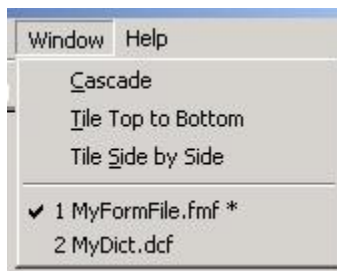
If you generated the screens by default, then the system will display the data entry tree and the Form tab on the left, and the first form created on the right window.





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.

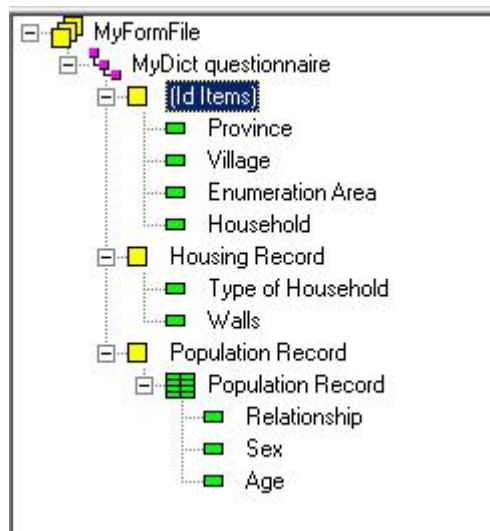







You can also switch between the forms and dictionary window by pressing either  or  in the Toolbar.

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.

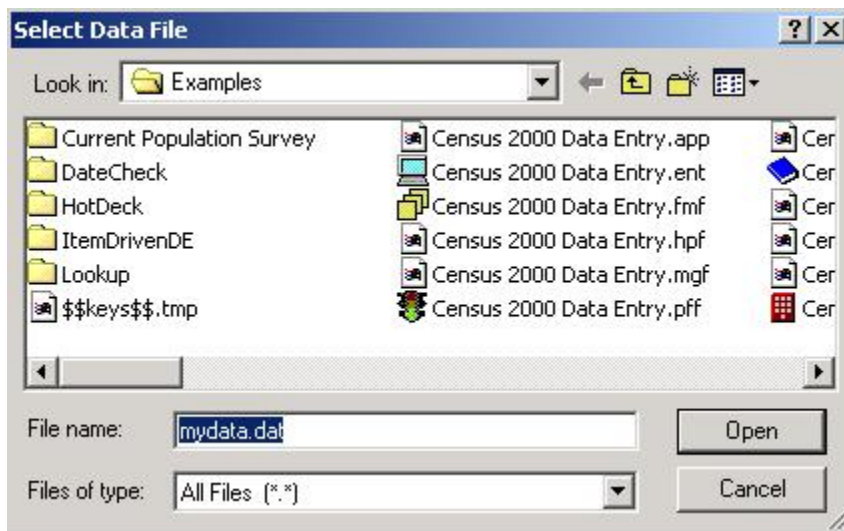
You can change the order of entry by reorganizing the forms or items within the forms by dragging them within the Data Entry tree view. When the operator enters the data, the cursor will follow the order in the tree not the order in the form. When selecting a new edit item, the contents of the logic 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 2.5\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 an ASCII 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 2.5
AppType=Entry

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

[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.

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.

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>
```

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 **Option** menu to change any of the following:

Data Entry Options

Type

Operator controlled
 System controlled

Require Enter Key

All fields
 No fields
 Some fields

Upper Case (alpha only)

All fields
 No fields
 Some fields

Verify

All fields
 No fields
 Some fields

Force Out-of-range

All fields
 No fields
 Some fields

Verify Every Nth Case

Frequency: 1 Start: 1

Random Start

Ask for operator ID:
Confirm end-of-case:
Allow partial save:
Show case tree:
CAPI Mode:

OK Cancel Help

- **Type**
This choice is very important and will have a large effect at data entry time. Please 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 add 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 contents.
- **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 require**

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 require**
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.

- **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 require**
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.

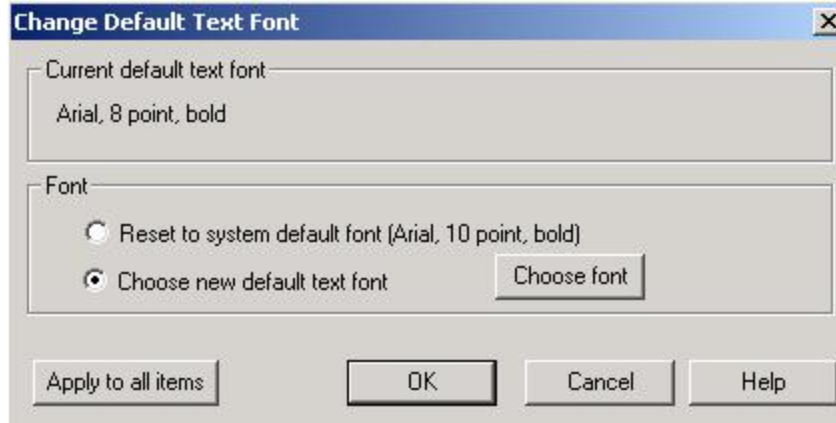
- **Verify Every Nth Case**
During verification, you may choose to verify only a subset of the cases in the data file, instead of verifying all the cases.
 - **Frequency**
This is the interval between cases that CSEntry will use for verification. For example, if this value is 10, every 10th case will 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**.



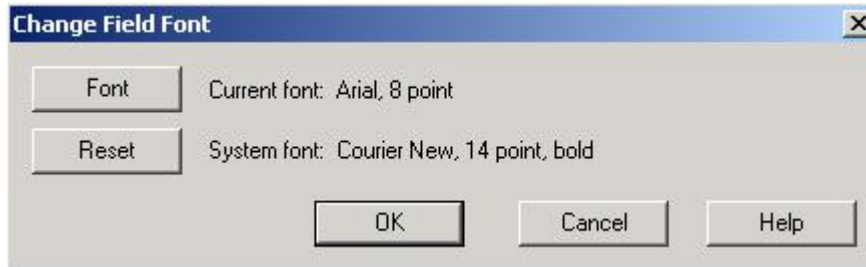
You can change this by using the **Font** radio buttons. If you select **Choose new default text font**, you can then click on the **Choose font** button to customize your own font

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 on 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

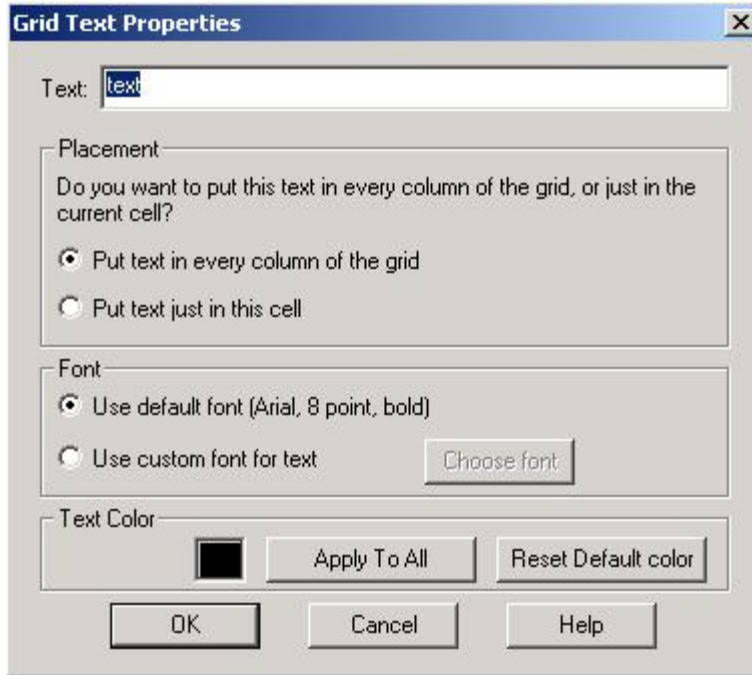
Add Things to a Roster

- **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 |
|---|--|
| {bmc BOXARROW.BMP} | 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 |
| {bmc BOXTHICK.BMP} | 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) 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

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

CSPPro 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**. CSPPro 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, CSPPro 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.

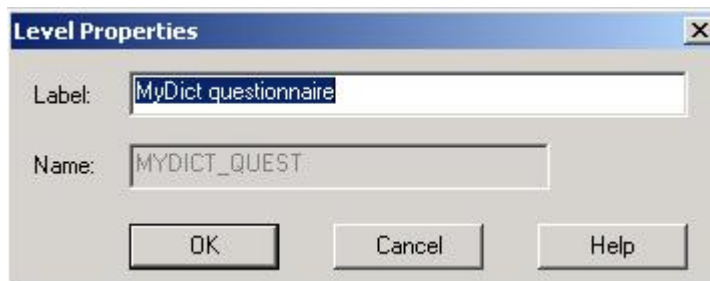


- **Label**
This is descriptive text that helps you identify the current forms file. It may contain any characters (including blanks) and be up to 120 characters long.
- **Name**
This is the name of the forms file which you would use when writing programming logic. It may be up to 32 characters long, 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."

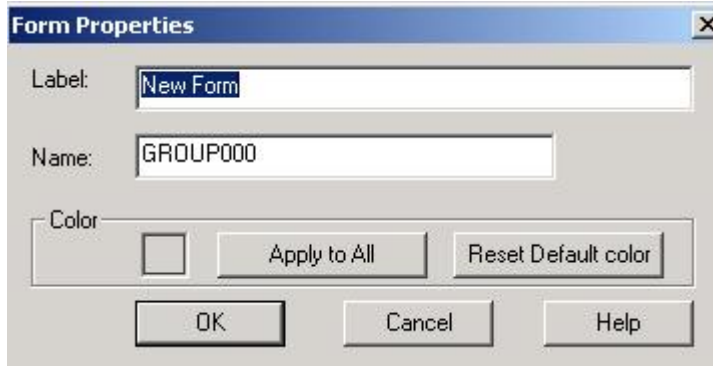


- **Label**
This is descriptive text that helps you identify the current level. It may contain any characters (including blanks) and be up to 120 characters long.
- **Name**
This is the name of the level which you would use when writing programming logic. It may be up to 32 characters long, and must 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.
- **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.

You can see either labels or names on the forms tree. Press **Ctrl+T** to switch back and forth between them.

Change Field Properties

Right click on the field on the tree or on the form and choose "Properties."

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 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.
- **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.

Change Roster Properties

Right click on the gray space in any roster cell and choose "Properties."

Roster Properties

Roster Label: Population Record

Roster Name: POP000

Occurrence control Field name

Orientation

Horizontal (row-oriented)

Vertical (column-oriented)

| | Age | Sex |
|---|-----|-----|
| 1 | | |
| 2 | | |

Free Movement

Data Entry Traversal :

Horizontal

Vertical

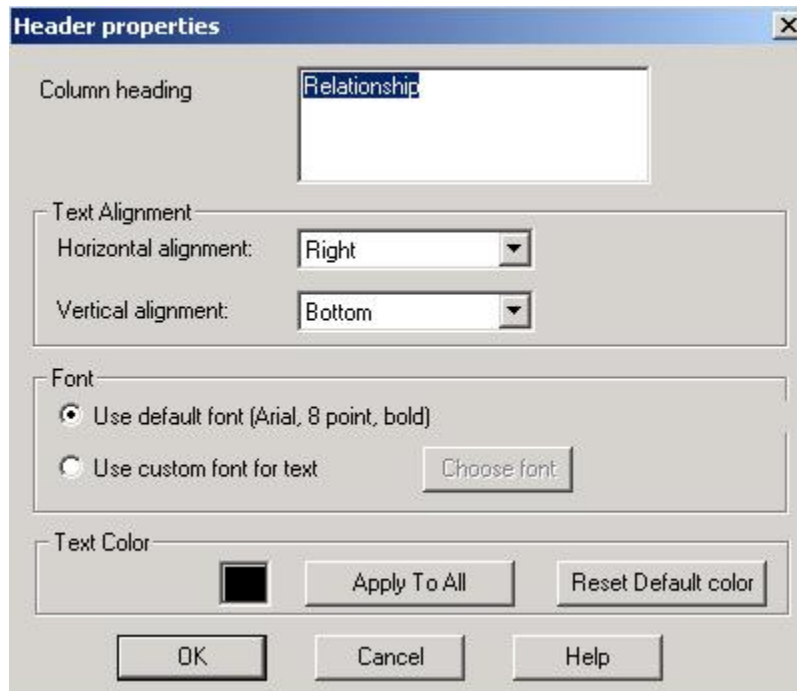
OK Cancel Help

- 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**.

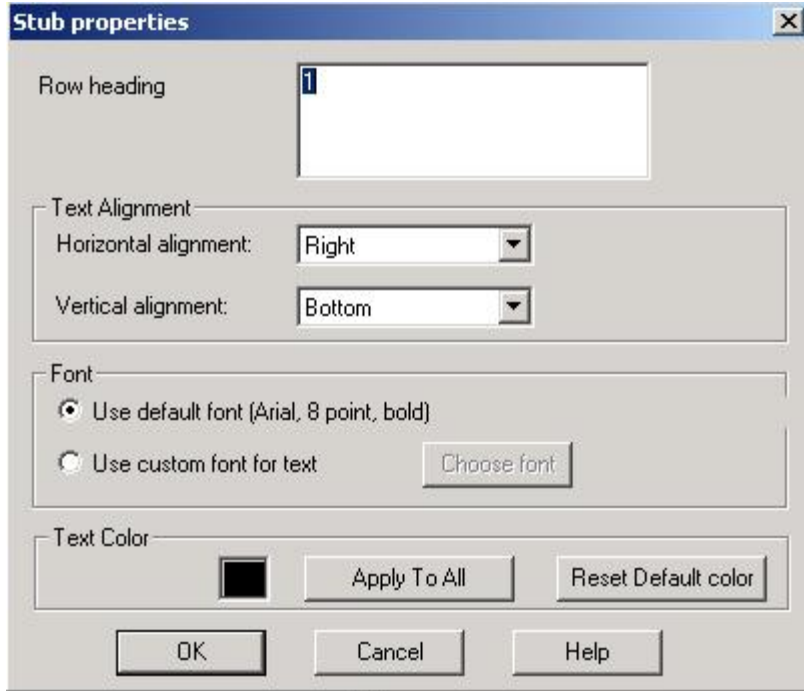


- **Column Heading**
This is the text that shows in the heading. CSEntry may automatically wrap text to make two or more lines, if the column width is small. You can force your own multi-line text by using **Ctrl+Enter** at the end of each line. For example if you type "Age of", then **Ctrl+Enter**, 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 Default 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."



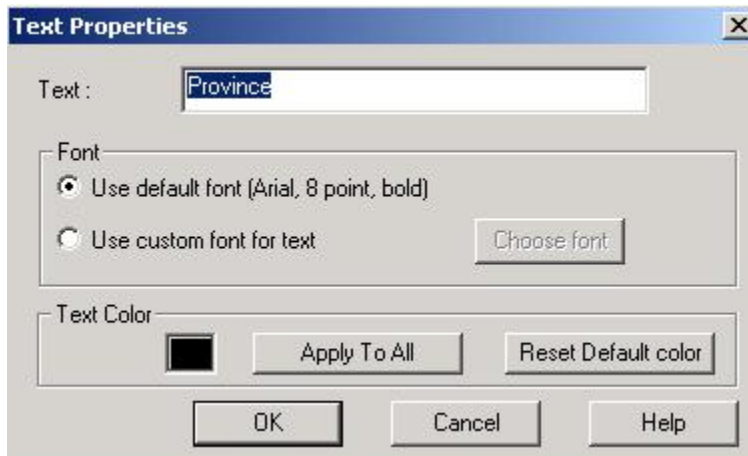
- **Row Heading**

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 CSPPro 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 CSPPro to develop the data entry application. You use CSEntry to run the data entry application, and use the CSPPro 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 complete 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 cross-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:

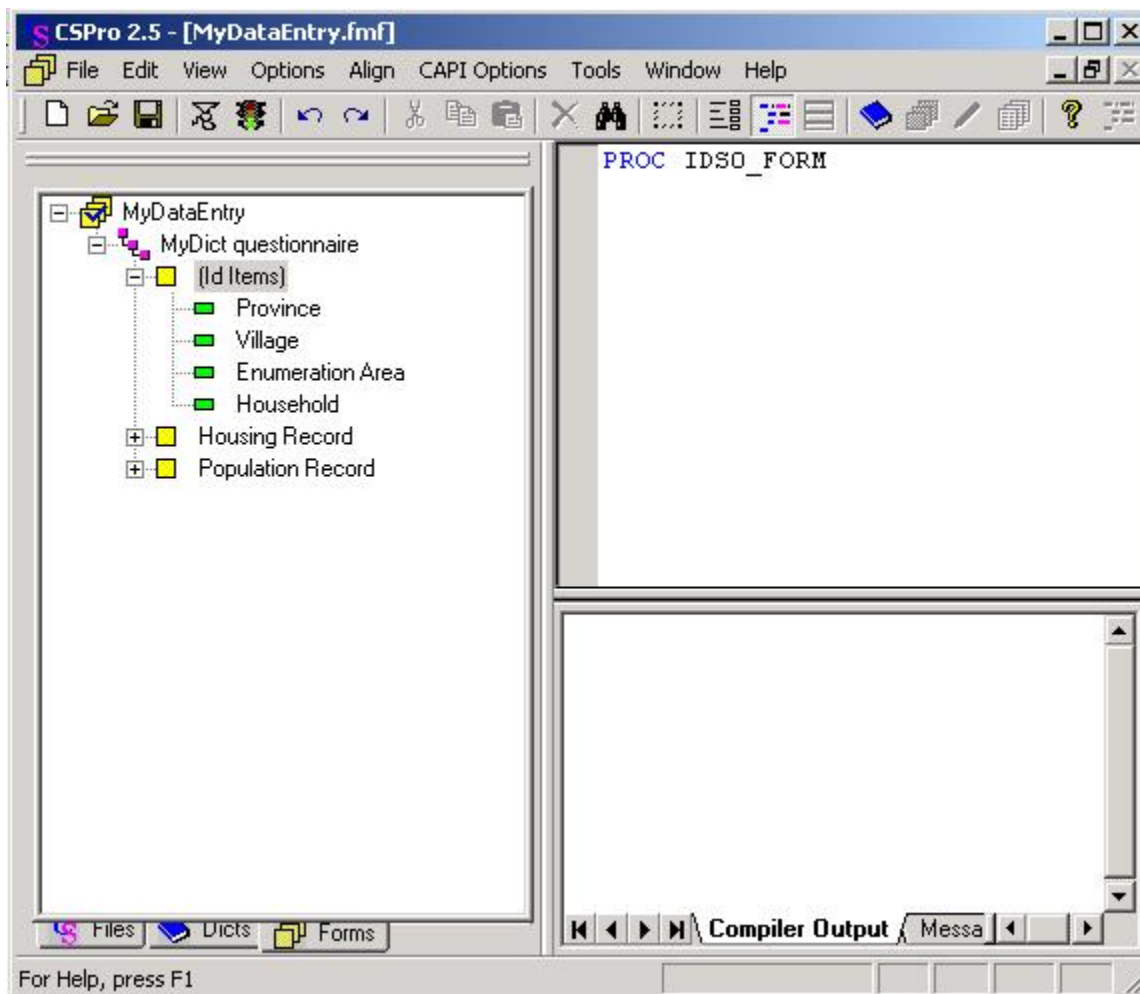
```
PROC FERTILITY
if FERTILITY in 0:20 then { there are 0-20 children }
  if sex = 1 then {if sex = male}
    errmsg ("male has fertility info present"); {message
      displayed}
    reenter; {operator must enter valid value}
  else {sex is not male}
    if sex = 2 then {if sex = female, check woman's age }
      if age < 15 then { 15 = minimum age for
        fertility }
        errmsg ("woman is too young (%d) to have children", age);
      endif;
    endif;
  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. These 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 .app 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, 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**.

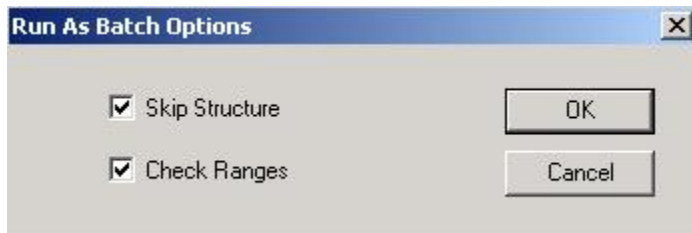
CSPro always compiles your application when you run the application. If there are errors, you cannot proceed until the errors are corrected. During code development, you should compile only the logic for an individual data entry or batch edit entity. This saves time, because the system does not have 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

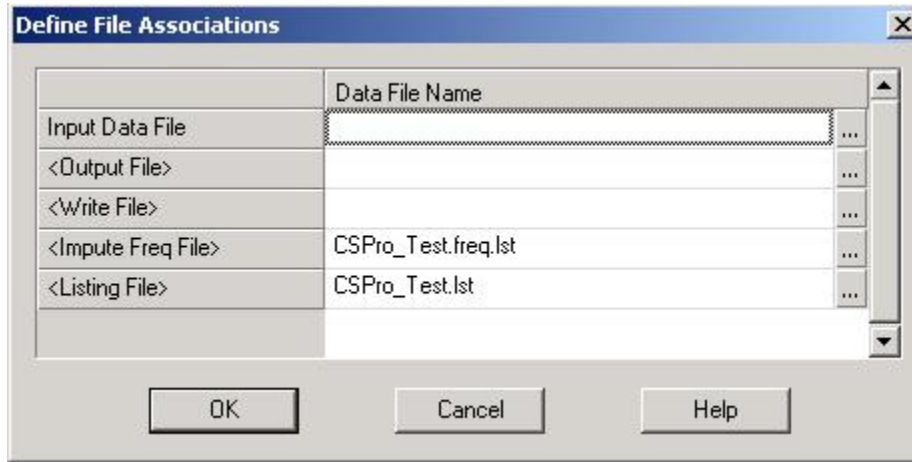
Run as Batch

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 .freq, 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 computer 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.

On the other hand, deploying a CAPI survey could be a costly undertaking, since each interviewer needs a laptop computer, and possibly more training as well. Creating good instruments 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
- 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. |

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 than 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

CSPPro 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. CSPPro 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 CSPPro'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.

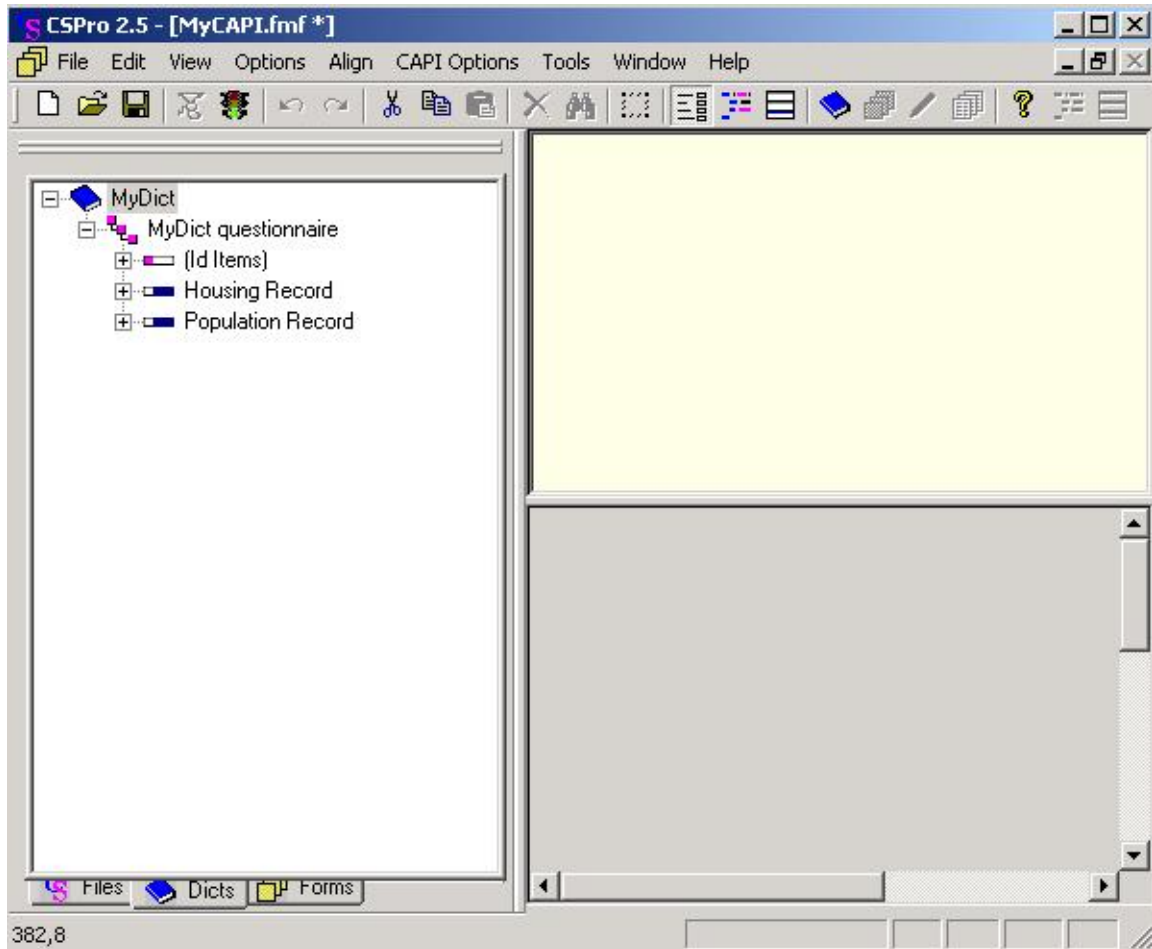
Coming Back Later

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

- Create a New Data Entry Application
- Don't generate forms
- From the options menu, select Data Entry Options.
- Check the box "CAPI Mode"
- 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 as 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:

- a. 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 ("%").

3. 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.

4. 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?

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 CAPI font 2
 bold
 italic

use the font specifications from CAPI font 1
 use the font specifications from CAPI font 2
 make text **bold**
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 be 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.

5. Select an item for which you want to add a condition (from the left-hand tree).
6. 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.
7. Right-click on the blue highlight bar in the condition window
8. Choose Add Condition
9. Enter a condition.
10. **Enter text** for this question in the window above.
11. 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 | <u>Effect</u> |
|---------|---------|-----------|--|
| 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) |

Notes:

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.

12. Select Resize Question Windows from the CAPI Options menu.
13. 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).

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 application require even more testing than data entry applications from paper forms because of the enumerators need to be able to more easily and quickly around the questionnaire.

Here are some suggestions on how to test an application.

- 1 Go through the case in intended sequence to test skip patterns, range checks, and edit checks.
- 2 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?
 - 3 Test moving around within the application, moving backward and forward.
- 4 Test moving between sections, if that is allowed.
- 5 Test field helps and general helps.

Batch Editing Application

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

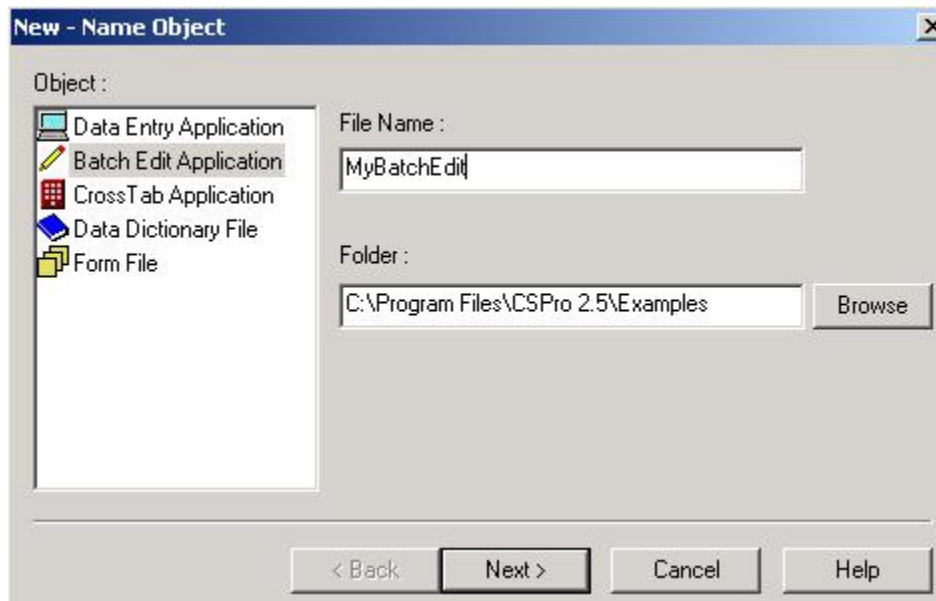
Create a Batch Edit Application

Create a New Batch Edit Application

To perform batch editing, you will need a data dictionary to describe the data file. If you have an IMPS or ISSA data dictionary, you should convert it to a CSPro data dictionary before you create a new batch edit application. If you already have a CSPro 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**.

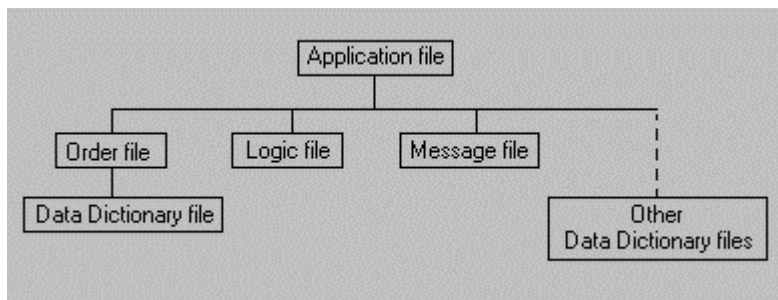


- In the Object window select **Batch Edit Application**.

- Enter the file name for the application
- Enter the name of, or select, the folder where the application will be stored. Click on **Next**.
- If you already have a CSPRO data dictionary, select its file name. If not, enter a new name and the system will create a new dictionary for you. Click on **Next**.
- You will be given a list of the files that will be created or used. If the list is correct, press **Finish**. Otherwise, press **Back** to make changes.

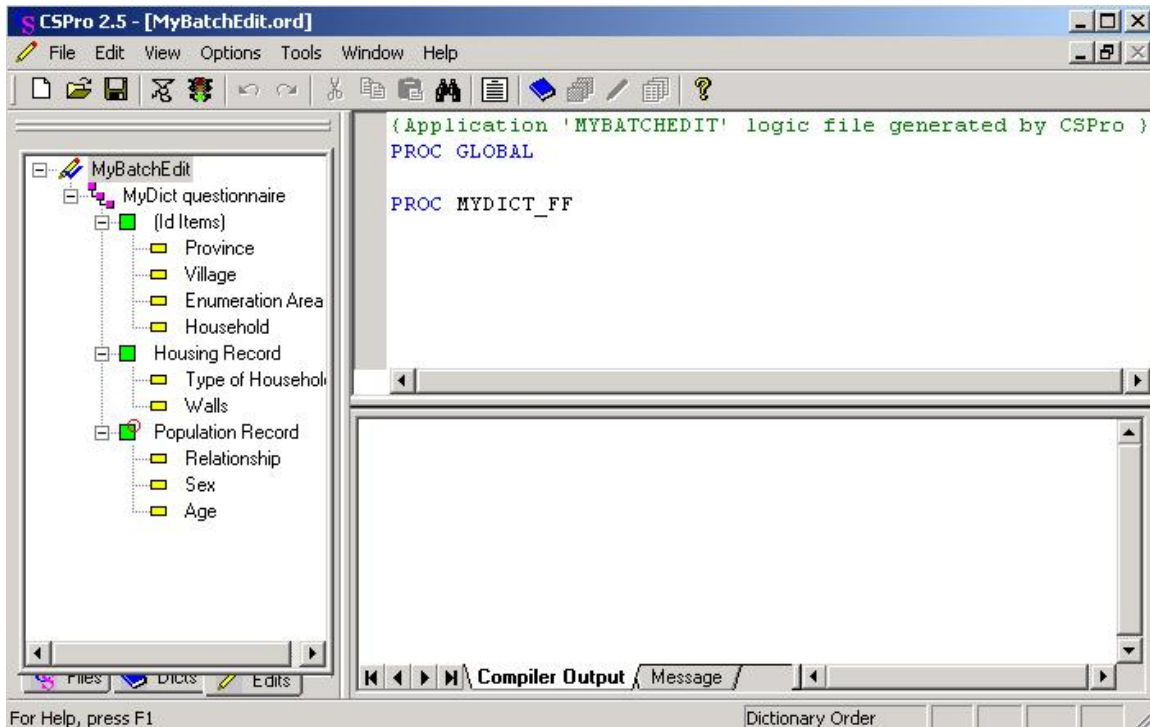
If you are using an existing CSPRO data dictionary, you may begin creating batch edit procedures. If you are creating a new CSPRO data dictionary, you will need to enter information into the dictionary about records, items, and values before you can create edits.

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 (.app)**
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

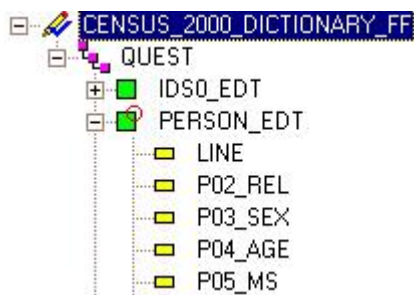
compiled successfully, it will state "Compile Successful." The **Message** tab is used to type in error messages that will be used in the execution of the program.





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.

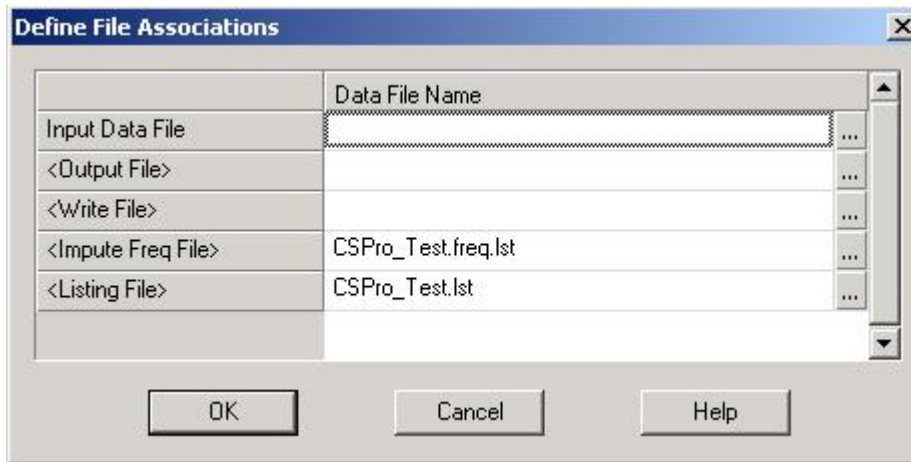
You can never delete edit levels, records, or items (i.e., the entries shown on the **edit** tab). However, you can change the order of the logic execution by dragging the items within the Batch

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'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 .freq, 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 **preprocs** and **postprocs** 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 the dictionary or items are 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 introducing 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. Editing instructions should be written by a subject-matter specialist in collaboration with the computer specialist. 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 recently revised 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 hot-deck example folder or to the United Nations Handbook on Population and Housing Census Edits.

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 CSPPro 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 CSPPro, 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. CSPPro 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 CSPPro's edit statistics reports.
- **Generate edit reports automatically:**
CSPPro 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 CSPPro during the editing process.
- **Generate reports**
You can write customize reports to a file.
- **Match files:**
CSPPro 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 may 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

There are two ways to initialize and maintain arrays in CsPro: inline, or using external files. In both cases you must first declare the array under the PROC GLOBAL section.

If you wish to use hot-decks in your application, refer to the example provided in the *Examples\HotDeck* directory. For a more detailed explanation of what hot decks are, refer to the United Nations Handbook on Population and Housing Census Edits.

Hot Decks Inline

This technique is the simplest to implement. 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

| | |
|---|---|
| <pre>If AGE = notappl then AGE = AgeSRDeck (sex, relationship) Else AgeSRDeck (sex,relationship)= AGE Endif;</pre> | <pre>{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}</pre> |
|---|---|

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.

Hot Decks in External File

Using an ASCII editor you must first create the external hot deck data file and its corresponding data dictionary. The same file used in the example above will look like the following:

| | | | | | | |
|---|----|----|---|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 27 | 30 | 6 | 58 | 11 | 24 |
| 2 | 32 | 25 | 8 | 60 | 10 | 28 |

The first line indicates the column number. "Zero" (0) is used to set up the first line as a comment.

Before processing the data files, the program using the loadcase command, reads in the existing values from the external file containing the hot deck for age. The values are then used and refreshed during processing. When finished processing the data file(s), the updated hot deck values are saved to the external file using the writecase command.

| | |
|--|--|
| <pre>PROC GLOBAL Numeric I, J; Array AgeSRDeck (2,6);</pre> | {declare array in the program} |
| <pre>PROC HOTDECK01 Preproc Do varying I = 1 until I > 3 Loadcase (AgeDeck_dict, sex); Do varying J = 1 until J > 7 AgeSRDeck(sex,J) = Age(J); Enddo; Enddo;</pre> | {read in one row of values of external file} {transfer those values to array defined in program} |
| <pre>Postproc Do varying I = 1 until > 3 Do varying J = 1 until J > 7 Age(J) = AgeSRDeck(sex,J) Enddo; Writecase (AgeDeck_dict, sex); Enddo;</pre> | {transfer values to array defined in program to external file} {update hot deck file} |
| <pre>PROC AGE If AGE = notappl then AGE = AgeSRDeck(sex,relationship) Else AgeSRDeck(sex,relationship)= AGE Endif;</pre> | {if the value for age is not valid} {assign the value from the hot deck based on the sex and relationship} {if the value for age is valid} {update the value of the hot deck} |

There are many variations to this program using external files. If you wish to use hot-decks in your application, refer to the examples provided in the Examples\HotDeck directory.

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 2.5\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 an ASCII 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 2.5
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
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.

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 dictionary 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 of 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 sex 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 programmers. 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 specifications 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.

Cross Tabulation Application

Introduction to Cross Tabulation

Cross Tabulation, more commonly known as CrossTab, is the module of CSPro that allows you to tabulate data quickly and easily. CrossTab is a versatile tool that enables the user to quickly and easily produce basic frequency distributions and cross-tabulations from one or more data files. The only requirements are that:

- The data to be tabulated be in ASCII 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 data base, spreadsheet, or other package), they must be exported to ASCII-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 CrossTab application.

When you create your application, you can use an existing data dictionary or you may create one as you create the application. There is no programming language to learn. The system is entirely menu-driven.

This section contains the following information:

- Parts of a Table
- Common Uses of CrossTab
- Capabilities of CrossTab
- Create a CrossTab Application
- Creating Cross Tabulations by Geographic Area
- Using CrossTab
- Manipulating Tables

Parts of a Table

The content of the different parts of the table will depend on the way in which data items have been defined in the data dictionary on which the tabulation is based. If, for example, the variable "age" is used in a tabulation, it is the value set selected (age in 5-year groups, age in 10-year groups, age by single year, etc.) that will determine the content of the title and of the row or column category headings (depending on whether "age" is used as a row or column variable).

- Heading**
 The heading is that part of a table which precedes the first row of data [numbers]. It consists of a table title and of headings for (on the left) the row variable(s) and (on the right) the categories of the column variable(s). By default, the title is created from the names of the value sets or variables selected for tabulation. The title may be modified by the user; column and row headings may not be modified
- Rows**
 Rows are variable categories, and are determined either by the value set chosen, or, if no value set exists, by the range of potential values established by the variable's type and size. CrossTab will create default categories if no value set exists for the selected variable. A maximum of two variables may be selected as row categories. If more than one variable is selected, the first variable is considered the independent variable and the second variable is the dependent variable.
- Column**
 Columns are variable categories, and are determined either by the value set chosen, or, if no value set exists, by the range of potential values established by the variable's type and size. CrossTab will create default categories if no value set exists for the selected variable. A maximum of two variables may be selected as column categories. If more than one variable is selected, the first variable is considered the independent variable and the second variable is the dependent variable.
- Cells**
 A cell is the intersection of a row and a column. It displays the count [weighted or unweighted] of the number of cases found displaying the characteristics of the intersecting row and column categories. Where a value (rather than a count) is tallied, the cell will display the accumulated total of values from cases displaying the characteristics of the intersecting row and column categories.
- Footnote**
 Text below the rows and can be used to qualify or explain information presented in, or omitted from, a table.

| {HEADING} | Table 1. Marital Status and Age Group by Sex for Population 12+ Years Old | | | |
|-----------|---|-------------|---------------|-----------------|
| | | Total | Male | Female |
| | ----- | -{COLUMN} - | -- {COLUMN}-- | ---{COLUMN}---- |
| {ROW} | | 376 | 196 | 180 |
| {ROW} | Total | 102 | 54 | 48 |
| {ROW} | 12 - 19 years | 171 | 90 | 81 |
| : | 20 - 39 years | 75 | 49 | 26 |
| : | 40 - 59 years | 28 | 3 | 25 |
| : | over 59 years | | | |

| | | | | |
|------------|--|-----|-----------|-----|
| ⋮ | Ever Married 12 - 19 years 20 - 39 years 40 - 59 years over 59 years | 272 | 132 | 140 |
| ⋮ | | 32 | {CELL} 11 | 21 |
| ⋮ | | 140 | 70 | 70 |
| ⋮ | | 73 | 48 | 25 |
| ⋮ | | 27 | 3 | 24 |
| ⋮ | Never Married 12 - 19 years 20 - 39 years 40 - 59 years over 59 years | 104 | 64 | 40 |
| {ROW} | | 70 | 43 | 27 |
| {ROW} | | 31 | 20 | 11 |
| {ROW} | | 2 | 1 | 1 |
| | | 1 | 0 | 1 |
| {FOOTNOTE} | Note: Not real data | | | |

Cells in a table

| | Column 1 | Column 2 | Column 3 | ... | Column n |
|--------------|------------|------------|------------|-----|------------|
| Row 1 | Cell (1,1) | Cell (1,2) | Cell (1,3) | ... | Cell (1,n) |
| Row 2 | Cell (2,1) | Cell (2,2) | Cell (2,3) | ... | Cell (2,n) |
| Row 3 | Cell (3,1) | Cell (3,2) | Cell (3,3) | ... | Cell (3,n) |
| ... | ... | ... | ... | ... | ... |
| Row m | Cell (m,1) | Cell (m,2) | Cell (m,3) | ... | Cell (m,n) |

Common Uses of CrossTab

CrossTab 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 Cross Tab 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, however, may require that a choice be made (e.g., age, occupation, educational levels, etc.) in grouping the available values. CrossTab may be used to examine the effects of various groupings before 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 CrossTab 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 CrossTab 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 CrossTab 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. CrossTab 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. While CrossTab is not currently designed for use as a custom tabulation package, it can be useful as a means of comparing the results produced by other tabulation software. In this case, the relative rigidity and transparency of CrossTab specifications means that there will be virtually no errors attributable to programming faults; if differences are encountered between the tables produced by CrossTab and tables produced by other software, the tables produced by CSPRO are likely to be closer to the "true" counts. In any case, CrossTab can be an effective tool for cross-checking numbers.

Capabilities of Cross Tab

Frequency Distribution

A frequency distribution is a table which shows the number of occurrences of each of the defined values for a data item and for the total of undefined values, which helps you identify errors in your unedited data files. Frequencies may be displayed as numeric values [default], as percentages, or as both. User 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.

You can create a frequency tabulation using CrossTab or use the module "Tabulate Frequencies" under Tools.

Cross Tabulations

Cross tabulations display relationships between a minimum of two, or as many as four, data items. There may be one independent and one dependent variable in each dimension [row and column]. This is in addition to any geographic subtotals which may be desired; geographic levels which contribute to the identification of the data are not counted as tabulation variables, per se. Like frequency distributions, cross-tabulation results may be displayed as numeric values [default], as percentages, or as both numbers and percentages.

The user can elect to display cross-tabulations in terms of actual counts or as percentages of the total, and has the option of displaying and/or dumping the undefined values. As with frequency distributions, the counts in cross-tabulations may be unweighted or weighted. User 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: Creating a Cross Tabulation

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

Both frequency distributions and cross-tabulations allow the use of a data item 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 with or without decimal positions. When a value is specified 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.

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. CrossTab 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. It is not possible to assign a weight globally (that is, to all the tables in an application); the weight must be applied to each table individually.

See also: Include Values and Weights

Restrict a Universe

When you define a universe, CrossTab 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

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 you choose to use for your area, these area identification codes must be based on the questionnaire identification

codes for the records you wish to tabulate. 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.
- **ASCII tab delimited (.other):**
Lets you save your tables so they can be used later by **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

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: Copy All or Part of a Table

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..

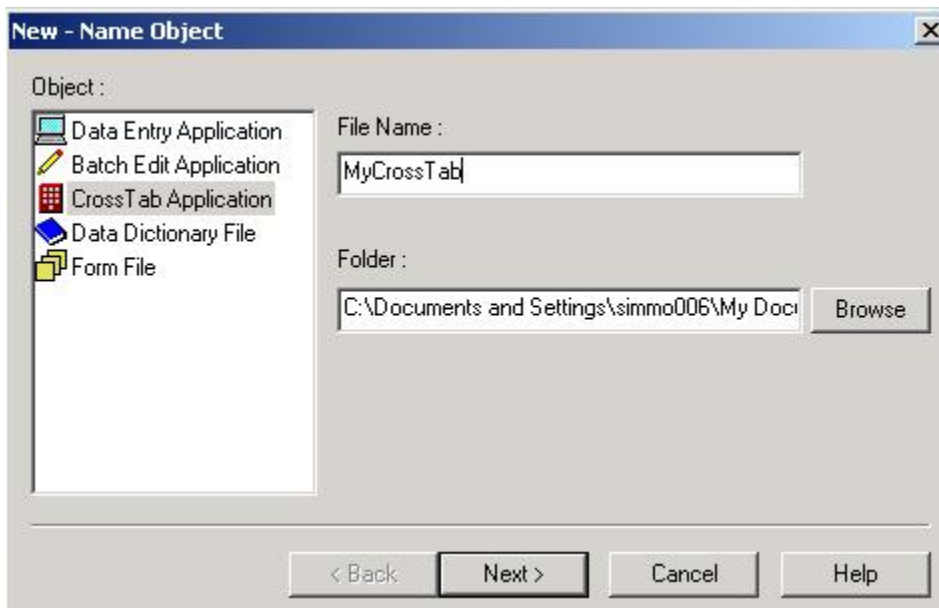
See also: Create a Thematic Map of Results

Create a Cross Tab Application

Create a new Tabulation application

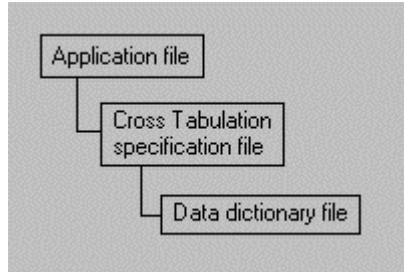
To perform cross-tabulations, you will need a data file, and a data dictionary to describe the file. If you do not have a data dictionary, you will need a written description of the data file structure and organization. You can create the CSPro data dictionary as you create the new Cross Tabulation application.

- If you have an IMPS or ISSA data dictionary, convert it to CSPro. Click  on the toolbar, or from the **File** menu, select **New** then select **CrossTab Application**.



- Enter the file name for the application, then enter the name of, or select, the folder where the application will be stored. Press **Next**.
- If you already have a CSPro data dictionary for the file, you can specify this dictionary when you create the new Cross Tabulation application. If not, enter a new name and the system will create a new dictionary for you. Click on **Next**. You will be given a list of the files that will be created or used. If the list is correct, press **Finish**. Otherwise, press **Back** to make changes.
- 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.

Cross Tabulation applications consist of the following files:



- **Cross 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.

Creating a Frequency Distribution

To create a frequency distribution in CrossTab do the following:

- Select the Dictionaries [**Dicts**] tab to make the dictionary file structure visible.
- Expand the tree until the item(s) you wish to use for a row or column variable appears on the tree.
- Drag the desired dictionary item and drop it on the table. Where you drop it on the table will determine whether it is used as a row or column variable. Imagine a diagonal line drawn from the top left corner of your table to the bottom right corner (forming a lower left and upper right triangle). For a frequency distribution you need to drop the item in the lower triangle.
- To delete the row, right-click in the side area and delete the variable. The table will remain open if you wish to select another variable. To delete the table right-click on any part of the table and choose "Delete Table".

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 you want to tabulate.
- Select the weight Item.
- Modify the table title if desired, or use the one CrossTab automatically generates for you.

See also: Frequency Distribution, Tabulate Frequencies

Creating a Cross Tabulation

To create a Cross Tabulation do the following:

- Select the Dictionaries [**Dicts**] tab to make the dictionary file structure visible.
- Expand the tree until the item(s) you wish to use for a row or column variable appears on the tree.
- Drag the desired dictionary item and drop it on the table. Where you drop it on the table will determine whether it is used as a row or column variable. Imagine a diagonal line drawn from the top left corner of your table to the bottom right corner (forming a lower left and upper right triangle). If you drop the item in the lower triangle, the item will become a row item. If you drop it in the upper triangle, it will become a column item.
- You can repeat Step 3 for a total of two items (or subitems or value sets) per row and two items per column.
- To delete a row/column variable, right-click in the side/top heading area and choose the variable to delete. If you have more than one row/column variable, you must right-click over the variable itself.


Optional Definitions

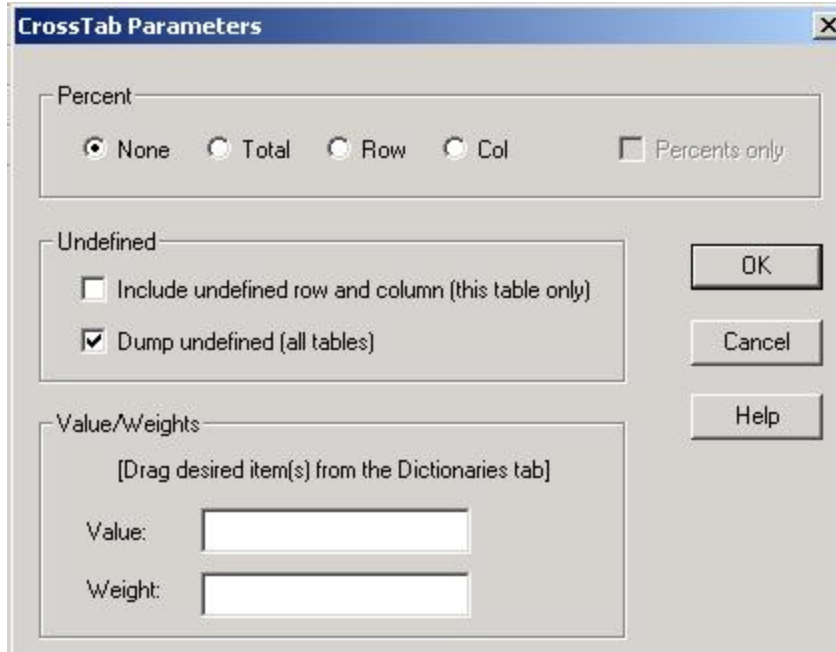
- Define the universe for the table.
- Define the area for the table (required if you plan to create a thematic map of results from your table).
- Select the value Item you want to tabulate.
- Select the weight Item.
- Modify the table title if desired, or use the one CrossTab automatically generates for you.

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: Cross Tabulations

Include Percents

Click on , or press Alt+P, or select "Parameters" in the Edit menu to open the CrossTab Parameters Dialog Box.




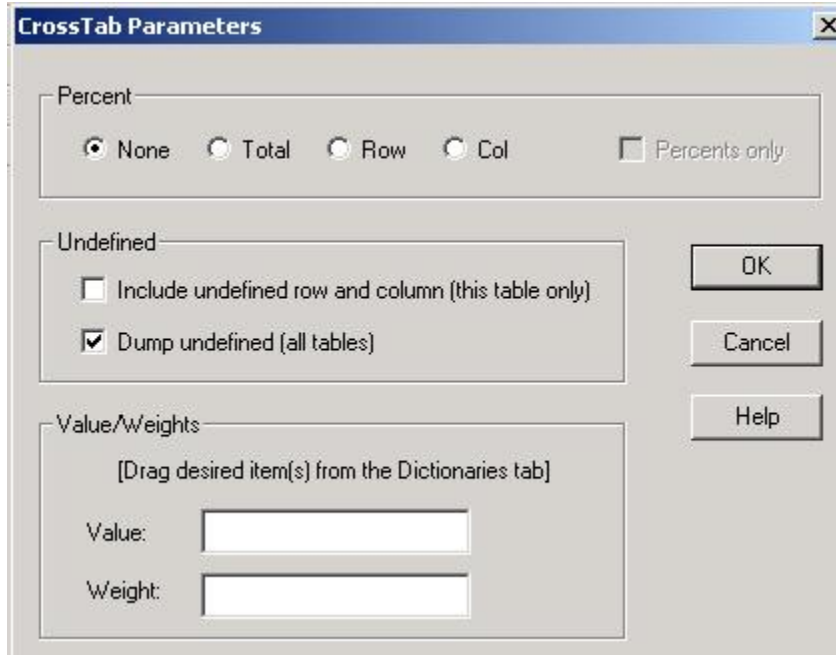
Choosing any of the options except **None**, CrossTab will automatically generate percent columns in your table. Select one of the radio buttons in the box labeled **Percent**. If the radio button is anything other than **None**, you may also check the **Percents only** box. The following options are available:

- **None**
No percentages will be generated for this table.
- **Total**
CrossTab will add an extra column next to each counts column. The value in each cell of these columns is calculated as the corresponding count divided by the grand total of the table. The percent cells corresponding to all non-total counts in the entire table will add up to 100.0 (though the sum may not equal 100 due to the effects of rounding).
- **Row**
CrossTab will add an extra column next to each counts column. The value in each cell of these columns is calculated as the corresponding count divided by the total for that row of the table. The percent cells **across** each row will add up to 100.0 (though the sum may not equal 100 due to the effects of rounding). If there are no column items selected, this setting has no effect.
- **Col**
CrossTab will add an extra column next to each counts column. The value in each cell of these columns is calculated as the corresponding count divided by the total for that column of the table. The percent cells **down** each column will add up to 100.0 (though the sum may not equal 100 due to the effects of rounding).
- **Percents only**
The table generated from this selection will depend on whether you chose the "**Total**", "**Row**", or "**Column**" box. No numbers will appear, only percentages.

See also: Tabulate Counts or Percents

Handle Undefined Values

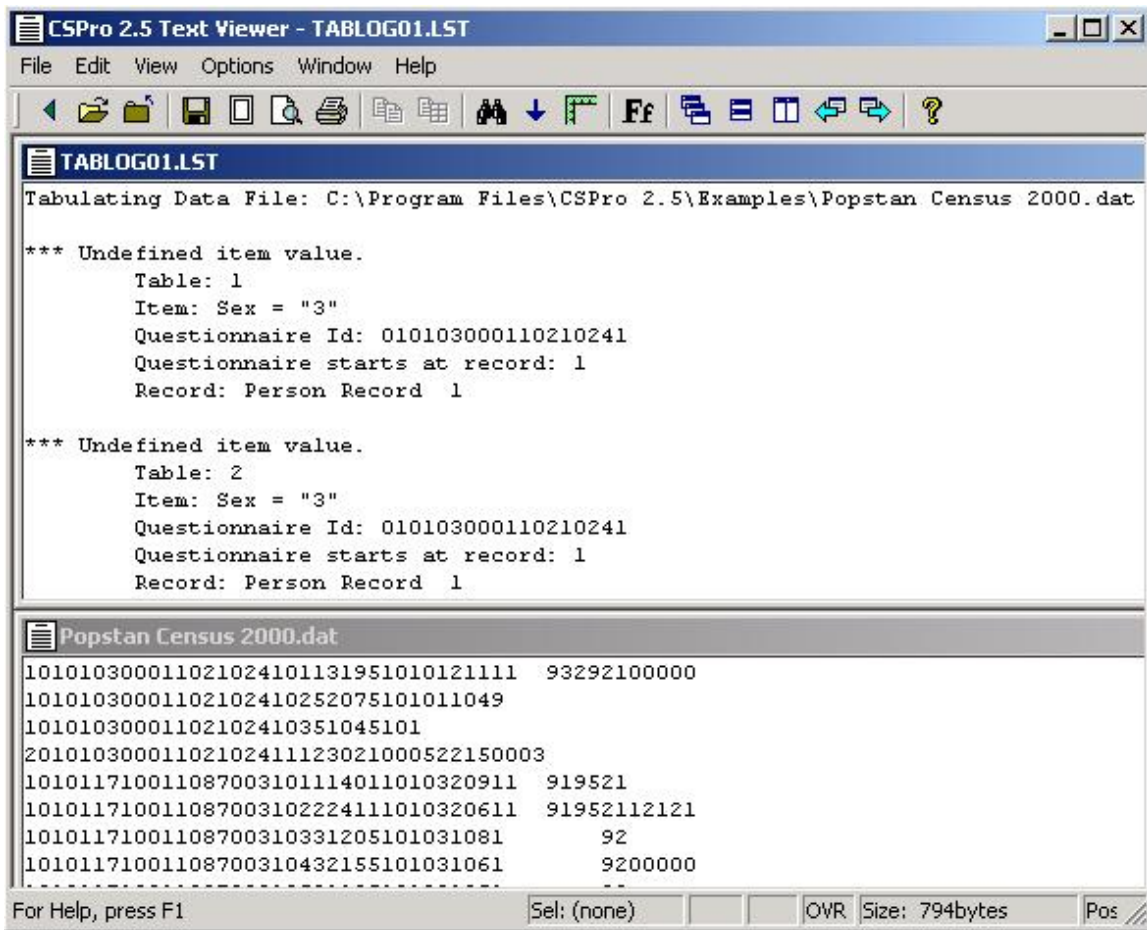
Click on , or press Alt+P, or select "Parameters" in the Edit menu to open the Parameters Dialog Box.



- Include Undefined Row and Column Values (this table only)**
 When this setting is used [i.e., when the box is checked], CrossTab will add an extra row (if there are row items selected) and an extra column (if there are column items selected) entitled "Undefined". CrossTab tallies [counts] into this row and/or column whenever it finds a value in the data file that is not one of the values listed for that item in the selected value set. For example, suppose you are using "Sex" as a column variable. "Sex" has a valid range of "1" (male) or "2" (female). If, during tabulation, a value of "3" is found for this item, the "Undefined" column for this variable would be incremented.

You can use this option to identify errors in unedited data files. If these counts are very high, it may indicate an error in the data dictionary. To help you find the error, use the **Dump Undefined** option

- Dump Undefined (all tables)**
 This feature takes the "Include undefined row and column" option one step further. Rather than merely showing the number of times an undefined data value was encountered, **Dump Undefined** will point directly to the record(s) containing these undefined values by creating a text window listing the record number and value of each item in question, allowing you to easily locate the entry in error. When selected, this option will be valid for all tables in the tabulation.




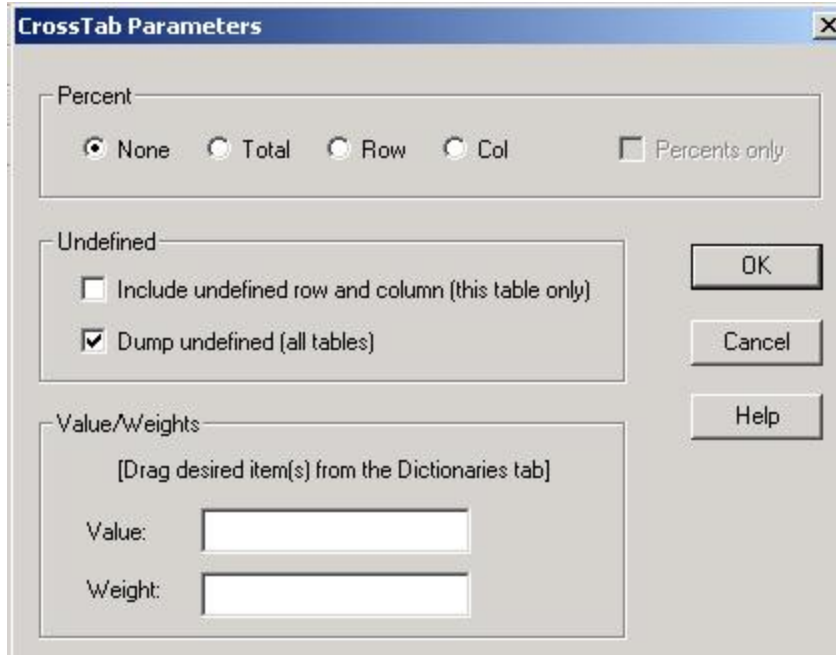
When tabulating data, an "undefined" value is any value not included in the value set that was chosen for the table in question. For example, assume the data dictionary specified for tabulation includes the following value set for the field "Sex":

| Value | Label | From |
|-------|--------|------|
| 1 | Male | 1 |
| 2 | Female | 2 |

If (for whatever reason) any value other than "1" or "2" appears in this field in any record in the data file, it will be trapped as an error when the **Dump Undefined** option is active. The record number (sequential position of this record within the data file) and questionnaire identification (fields selected as **Questionnaire IDs** in the data dictionary) will be displayed to help the user locate the item in question.

Include Values and Weights

Click on , or press Alt+P, or select "Parameters" in the Edit menu to open the Parameters Dialog Box.



- **To Tabulate a Value**

To use an item as a tabulation value, drag the desired item from the dictionary tree to the **Value** entry in the dialog box.

For example, if you wish to tabulate farm acreage by **Type of Farm** and **Region**, you might choose **Region** as the row item, **Type of Farm** as the column item, and **Number of Acres** as the value item. This is useful for obtaining counts. In the above example, if you wish to tabulate the number of farms by **Type of Farm** and **Region**, leave the value item blank.


- **Weighted Tabulations**

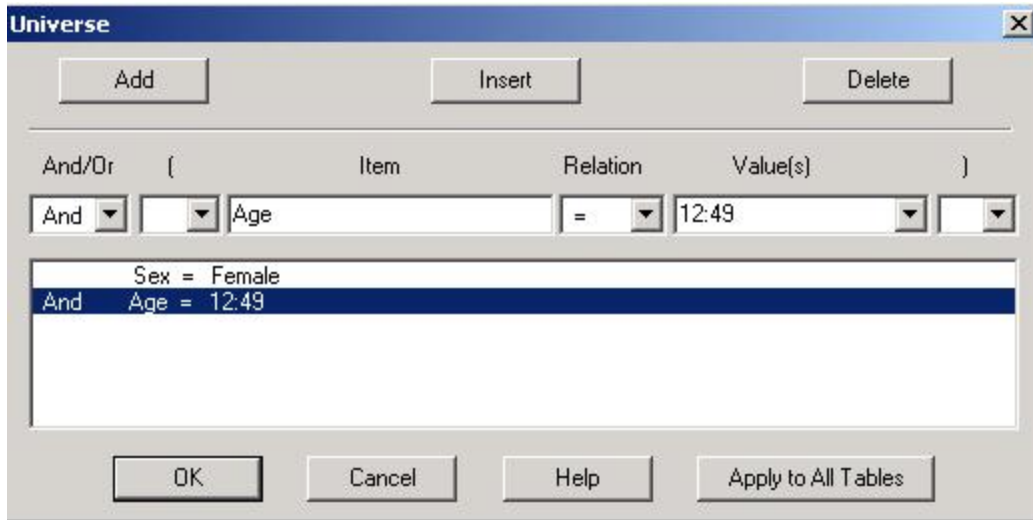
To use an item as a tabulation weight, drag the desired item from the dictionary tree to the **Weight** entry in the dialog box.

See also: [Tabulate Values and/or Weights](#)

Define a Universe

To define a universe do the following:

- Select the Dictionaries [**Dicts**] tab to make the dictionary file structure visible.
- Expand the tree until the item(s) you want to use are visible in the tree.
- Click the  button to launch the **Universe** dialog box. (You can also press **Alt+U** or select the **Edit** menu item, then **Universe**.)



- The first time the universe is launched for a table, the **Item** cell will be empty. From the dictionary tree, drag the desired item (e.g., Age) and drop it in to the cell marked **Item**.
- Now select a relationship (=, <>, >, >=, <, <=) value.
- Select a value from the drop-down box, or type in a value. If you want a range of values, rather than a single value, type in the lower and upper bounds separated by a colon (e.g., 12 : 49).
- You may enter several conditions using the **and / or**. You can also add parentheses to modify the order of evaluation of the conditions.
- You can **Add**, **Insert**, or **Delete** a condition by using the buttons with the same names.

Examples:

- To restrict your table to females of reproductive age, you might state:

```
Sex = Female
AND Age = 12:49
```


- To restrict your table to heads of households who are economically active, you might state:

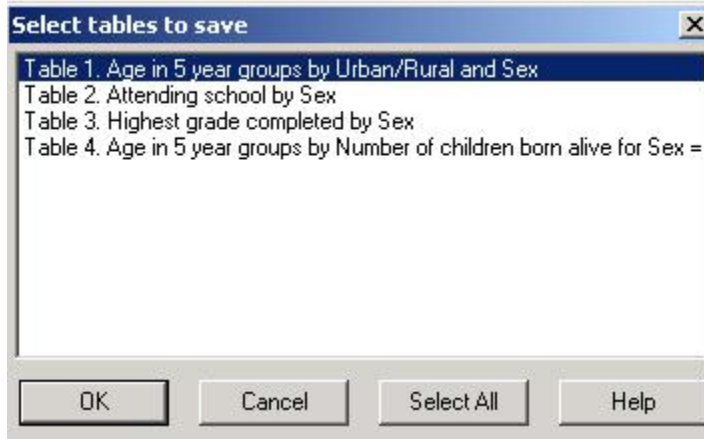
```
Relation = Head
AND Econ_Active = YES
```

You can apply a universe to all the tables, by pressing the **Apply to All Tables** button.

See also: Restrict a Universe


Save Tables

- **To save entire tables**
Make sure that none of the table's cells are currently selected (press **Esc** if they are). Click  on the toolbar; or from the **File** menu, select **Save As**; or press **Ctrl+S**.



If your tables file has multiple tables defined within it, a dialog box will appear which lets you select the tables to be saved. Select all the tables that you would like to save in one file. A **Save As** dialog box lets you enter the file directory (folder), file name and file type.

- **To save part of a table**


Select table cells you want to save. The corresponding boxheads and stubs of selected cells are also selected. Click  on the toolbar; or from the **File** menu, select "**Save As**"; or press **Ctrl+S**.

A dialog box will appear asking if you want to save the whole table or just the selected parts. Choose "**Selection**" and press **OK**. A **Save As** dialog box lets you enter the file directory (folder), file name and file type. Press **Esc** to deselect.


See also: Save Tabulations in Different Formats

Print Tables


- **To preview the printing of tables**

Click  on the tool bar; or from the **File** menu, select **Print Preview**.

- **To print entire tables**

Make sure that none of the table's cells are currently selected (press **Esc** if they are). Click  on the toolbar; or from the **File** menu, select **Print**; or press **Ctrl+P**. If your tables file has multiple tables defined within it, a dialog box will appear which lets you pick which tables to print. Select all the tables that you would like to print at once.

- **To print part of a table:**

Select table cells you want to print. The corresponding boxheads and stubs of selected cells are also selected. Click  on the toolbar; or from the **File** menu, select **Print**; or press **Ctrl+P**. A dialog box will appear asking if you want to print the whole table or just the selected parts. Choose **Selection** and press **OK**.

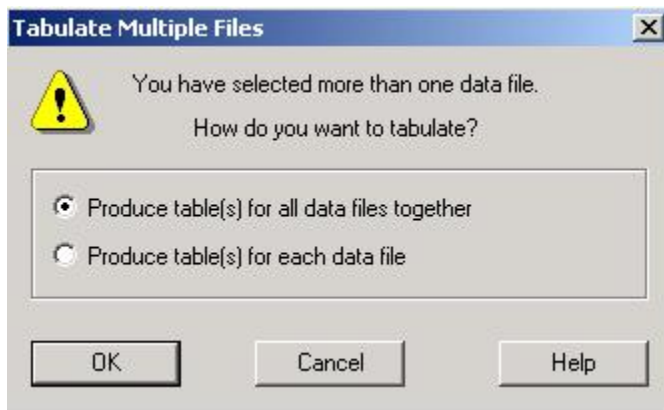
A **Print** dialog box lets you select the printer, the page range, and the number of copies.

Run a Tabulation

Once you have finished defining your table(s), you are ready to run them against your data files to produce the real tables (i.e., a file with extension `.tbw`). You can speed up identification of errors in your data file by using the dump undefined option. You can run your tables using a single or multiple data files.

Press the **Run Tabulation**  button or type **Ctrl+R**. You may be asked to save tables from a previous run. Then choose one option:

- **To Tabulate a Single Data File**
Navigate to the subdirectory that contains your data files and select the desired data file. Press the **Open** key and the tables will be produced
- **To Tabulate Multiple Data Files**
Navigate to the subdirectory that contains your data files. Hold down the **Ctrl** key while selecting your data files (they must all be in the same directory [folder]). Choose one of two options:



- Produce table(s) for all data files together:
This is equivalent to first concatenating all the data files, then running the tabulation on this single (concatenated) file. (**Note:** The data files will **not** actually be concatenated on your disk!) One single table set (i.e., file with extension `.tbw`) will be produced. This is useful when the data you wish to tabulate are split among several physical files.
 - Produce table(s) for each data file:
This is equivalent to running the same tabulations separately on each data file. One table set (i.e., file with extension `.tbw`) will be produced for each data file. This option avoids having to run each data file one by one. You can tabulate up to 30 files at once using this method. As more than one table set is generated with this selection, Table Viewer will be launched to display the tabulation results.
- Press **OK** when ready to tabulate.

Creating Cross Tabulations by Geographic Area

Area Processing

The Area Processing is the generation of tables by a defined hierarchical structure of area codes. If the geographic hierarchy of 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 cross-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 cross-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)
  :
  :
```

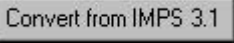
Area Processing will apply to **all** tables in your application.

After producing the tables using area processing you can select a cell to display in a thematic map and view it spatially.

See also: Area Names File

Create an Area Names File

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. This file defines the levels of geography and assigns text names to the numeric codes for each geographic unit. It is identical to the IMPS area names file (.ANM). You can also convert an area file created in IMPS with extension .ara to .anm in the area IDs dialog box.

The following is an excerpt from the area names file for Popstan (popstan.anm). As you can see, the first section identifies the type of file, i.e., an [Area Names] file, and then the CSPro version number is given. (**Note** for IMPS CrossTab users: You can use the IMPS area name file [* .anm] as is and convert it to CSPro .anm file with the  option.)

Beneath that, the [Levels] section provides the names of the geographic areas (levels) that are defined under the [Areas] section. The number of levels named must agree with the number of areas defined. Following this example is an explanation of the [Areas] section.

```
[Area Names]
Version=CSPro 2.5

[Levels]
Name=Province
```

```

Name=District

[Areas]
0 0 = Popstan
1 0 =   Artesia
1 1 =     Dongo
1 2 =     Idfu
:
2 0 =     Copal
2 1 =     Baja
2 2 =     Bassac
:
3 0 =     Dari
3 1 =     Argentina
3 2 =     Benlata
3 3 =     Bristol
:

```

The very first line following the [Areas] section is the name of the country. It is considered the 'zero'th level, and hence, has 0 0 = **Popstan**, where the first column represents the Province level, and the second column represents the District level. This will be the only line in the entire file that will have a code of 0 0.

Now begin to list the geography for the first named level (i.e., Province) in Popstan, which in our example is *Artesia*. If you were only defining one level, you would immediately proceed to define the next province (*Copal*). However, in our example we want to define a second level, [District]; therefore, immediately after defining the first province (*Artesia*), we must name all districts in that province.

Note that each line begins with 1, as this is *Artesia*'s province code (as defined in Popstan's data dictionary); the second column lists the district code, again as defined in Popstan's data dictionary. If a third level had been named (e.g., *Tract*), then all tracts would follow each district to which they belonged.

For illustrative purposes, we have written the names indented according to their level. You can choose to do this in your own file as you wish—it will make no difference in the processing of the file.


If you want to use the tables generated from CrossTab to create thematic maps, the Area Processing feature **must** be used. Further, the number of levels defined in the .anm file must **not** be greater than the number of polygon levels defined in your .map file.

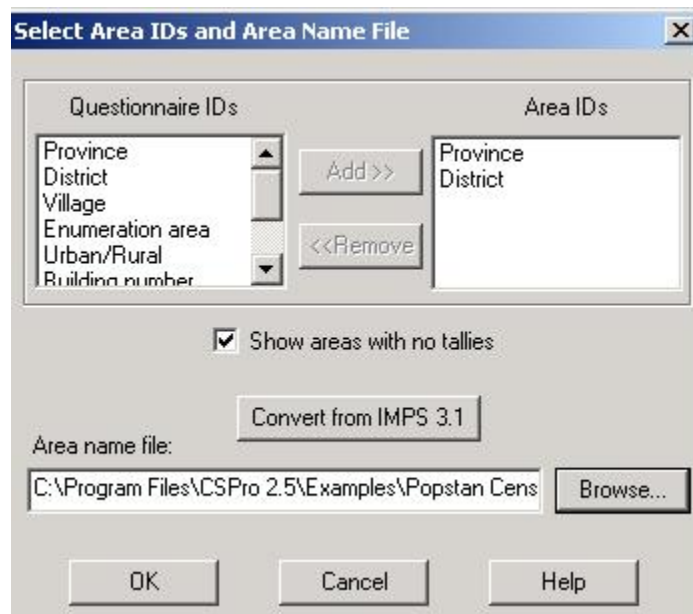
Finally, when creating your .anm file, separate each line item within the [Areas] section by either commas, spaces, or a combination of both. Hence, any of the following would be acceptable to define an item at the district level:

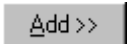
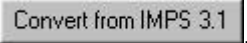
- 3 15 = Sharif
- 3, 15 = Sharif
- 3,15 = Sharif


See also: Area Processing Area Names File

Area IDs Dialog Box

Click on , or press Alt+A, or select "Area" in the Edit menu to open the Area IDs Dialog Box. This dialog box allows you to select the Questionnaire IDs to be used for area processing, as well as to specify the area names 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  to add the Questionnaire IDs to the Area IDs list.
- Decide whether to show or not show areas where no data are tallied.
- Specify the name of the Area Names file for these area levels. For CSPro, this file must have the extension `.anm`. If you have IMPS 3.1 or an earlier version (the file will have an `.ara` extension), then use the convert feature by clicking on .

If you no longer need a specific Area ID, select the ID and press  to delete it from the Area IDs list. You can launch the Area IDs dialog by pressing **Ctrl+A**. Alternatively, you can also launch the Area IDs dialog by selecting **Edit** from the menu bar, then "R"

See also: Area Processing, Create an Area Names File

Create a Thematic Map of Results

- **To Generate a Map from CrossTab:**
 - Produce a table using area processing from CrossTab. When the finished table appears, click on the cell that represents the variable you wish to map.
 - Click on  [the MapViewer icon, located on the toolbar] and enter a Map Viewer Variable Name that will appear as title in your map, or accept the default title. Select the

corresponding map file from the dialog box. You can choose between an `.mdf`, `.mpc`, or `.map` file.

- **MapView** will be launched and a thematic map, representing the data you selected, will appear.
- **To Create a Map Data File (.mdf) from CrossTab:**
You can import several tabulated variables into MapViewer and then save them all in the same 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` or `.map` file).
 - Switch back to CrossTab (i.e., select it from the Windows 95 task bar or use **Alt+Tab**). Do **not** close down MapViewer.
 - Generate the next variable from CrossTab as in steps 1-5 above. 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, as long as they share the same `.mpc` file.
 - 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

Using Cross Tab

Implications of Data Dictionary Value Names

Within the data dictionary, the user can define, for each data item, individual values and value ranges as they should be displayed in CrossTab tables. In other words, if one table requires that "Age" be displayed in five-year intervals, and in another table persons are counted by single year of age, both requirements can be fulfilled with two value set definitions. Moreover, you can define the same item several times, with a different set of value ranges for each definition. Redefinition is done by defining an item as a sub-item. Selecting the appropriate value set will ensure the correct display of data in the table.

When selecting a value, you must take into account the following considerations:

- If names are given for all value sets, the names and ranges appear in the tables at the intervals defined.
- If names are given for value sets, and some names define split ranges, then the items for all parts of the split range will be totaled for the table.
- If names are not given for value sets but there are multiple value ranges, the ranges themselves will appear as the row, column, or layer heading.

- If the user has not defined any value sets for a numeric item, CrossTab will create default value sets based on the size of the item. If the item being tabulated is one digit in size, CrossTab will tabulate each of the possible values [0 to 9] individually. If the item is two or three digits in size, then CrossTab will create default groupings: 0-9, 10-19, 20-29, ..., 990-999. If the item is defined as four digits long, the groups will be in ranges of 100 units. For example, if the range is 4731 to 6342, then groups will be 4731-4799, 4800-4899, ... and 6300-6342.
- If an alphanumeric item does not have at least one value set defined, it cannot be selected for tabulation. CrossTab will issue an error message indicating that the user must define at least one value set before using the data item in a tabulation.

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 over 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`, `MyItem(1)` and `MyItem(2)` would both 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. If no occurrence number is specified by the user, the first occurrence will be used.

Types of Table

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 table 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 frequency distribution, with the tabulation categories in the rows of the table, the frequency counts in the first column of the table, and the cumulative counts in the second column of the table.

- **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 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. Totals will always appear in the left-most column and in the top-most row.

- **2 Row/Column Variables**

When a table 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.

Manipulating Tables

Change Tabulation Parameters

In addition to using area processing and universe statements, there are several other ways to customize your tabulation results. Most of these options apply only to the current table. The three option areas are the following:

- **Percent**

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.

- **Undefined**

This section deals with values in the data file. Two options will help you identify errors in your unedited data files. A third option, when selected, changes the order in which rounding and summing take place when decimal weights are used in a tabulation.

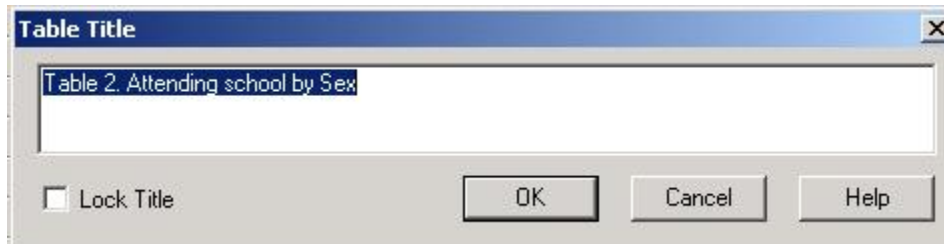
- **Value/Weights**

With this option you can specify a data item to be used as a value and/or a weighting factor during tabulation.

Change the Table Title

CrossTab generates a title for each table based on the **Row Items**, **Column Items**, **Value**, **Weight**, and **Universe** statements in use, as well as the **Percents only** option. Every time you change any of these, CrossTab modifies the title accordingly. If, however, you would like to use a title of your own making, rather than the one generated by CrossTab, you can choose to **Lock** the title. Do this by checking the box found in the **Table Title** dialog box.

You can unlock the title at any time. However, when you do, be aware that if you have modified the table definition since you locked it, CSPro will ask you if you want to reset the title—that is, if you want to use the CSPro-generated title, answer **Yes**; if you want to retain the current title, answer **No**. If you don't reset the title but later modify the table definitions, be aware that the title will change (since it is no longer locked).



To modify the table title, either right-click on the table cell containing the title, or select **Edit** from the menubar and then select **Modify Title**.

Add a Table


Any table added to an existing table set 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 tab 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 by selecting the **Edit** menu, then typing **"A"** to add.

Insert a Table

Any table inserted into the existing table set will always be placed **before** the currently-displayed table. You'll notice a new tab 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"). If you're already on the first table, it will become the new Table 1 and move the rest down. Finish the definition of the inserted table by adding dictionary items and specifying any universe definitions or other tabulation parameters desired.

You can also insert a table by right-clicking anywhere in a table and select **"Insert Table"** from the pop-up menu. Alternately, you can insert a table by clicking on the  icon, or selecting the **Edit** menu, then typing **"I"** to insert.

Modify a Table


You can modify a table in a number of ways, such as:

- Changing the table title
- Changing the universe definition
- Changing the tabulation parameters of an existing table.
- Shifting row/column variables: Use the drag-and-drop method. For example, if you want to make the row heading `Sex` (with its value set of `Male` and `Female`) a column heading instead, drag one of its value set items and drop it in the column heading area.
- Removing row/column variables: Drag any of the item's value categories back on to the dictionary tree—drop it anywhere—and it will be removed from the table. You can also right-click over any of the value categories and select **Delete** from the pop-up menu.

To make modifications to a table, select the desired table (from either the left-hand view table tree tab, or from the right-hand view table tabs along the bottom) and proceed as desired. When finished, make sure you save the table specification file so the changes will be permanently recorded.

Delete a Table

The table that is currently on view in the right-hand portion of your screen is always the one affected when you choose to delete. Select the table you would like to remove by either: choosing it from the left-hand view table tree tab; or using the right-hand view table tabs along the bottom.

Delete the table by either: choosing **Edit** from the menubar, and then select "**Delete**"; or pressing the toolbar's delete  button. Answer "**Yes**" to the delete prompt if you wish to proceed.

Select Table Cells


- **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.
 - Then release the mouse button. The selected cells and their heads and stubs are highlighted.
- **To select additional cells**

Hold the **Ctrl** key down as you make additional selections. If you are selecting several pages of material, you can press the **Page Down** or **Ctrl-End** keys while holding the mouse button down. If you drag the mouse outside of the cell area, the table will automatically scroll and continue the blocking action.
- **To select ALL cells from the Edit menu**

Select All; or press Ctrl+A.
- **To deselect cells**

Press the **Esc** key; or from the **Edit** menu, select "**Cancel Selection**".

Copy All or Part of a Table

- Select the table cells you want to copy. To copy the entire table, select from **Edit** menu, **Select All**. The corresponding boxheads and stubs of selected cells are also selected.
- Click  on the toolbar; or from the **Edit** menu, select **Copy**; or press **Ctrl+C**.
- Paste the cells you copied directly into a word processor or spreadsheet. They appear in tabular format.

See also: [Save tables](#)

CSPro Statements and Functions

Alphabetical List

| | |
|--------------|---|
| 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. |
| 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. |
| 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. |
| curocc | Returns the current occurrence number for a repeating form, roster, or record. |
| 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. |
| 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. |
| execsystem | Starts another windows 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. |
| fileexist | Determines whether a file exists. |
| filedelete | Delete an already existing file. |
| 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. |
| highlighted | Returns whether field is on path or reached during data entry. |
| getbuffer | Returns a string containing the contents of a data item. |
| getlabel | Returns the label of a dictionary symbol or text associated to symbol's value |
| getnote | Gets a data entry field note and assigns it to a string. |
| getoperatorid | |
| getsymbol | Returns the name of the current procedure being executed |
| if | Executes statements conditionally. |
| impute | Assigns a value to a data item and logs the frequency of assignments. |
| 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. |
| 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 | |
| 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. |
| onfocus | Declares that following statements are executed object becomes active. |

| | |
|-------------|--|
| onkey | Allows user 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. |
| 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. |
| putnote | Puts the contents of string to a data entry field note. |
| random | Returns a pseudo-random integer in a given range. |
| 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 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. |
| selcase | Allows a data entry operator to select and load a case from an external file. |
| set | Switches the values of various system parameters. |
| setfile | Assigns a new physical file to a dictionary or declared file. |
| skip | Jumps 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 | The sort function sort 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. |
| strip | Removes leading and trailing blanks from a string. |
| sum | Returns the sum of an item that occurs multiple times. |
| sysdate | Returns the current system date as an integer. |
| sysparm | Returns a parameter from your data entry or batch edit pff file. |
| systeme | Returns the current system time as an integer. |
| tonumber | Converts a string to a number. |
| totocc | Returns the total occurrences for a repeating form, roster, or record. |
| visualvalue | Returns the value of a data item prior to it's input. |
| while | Executes one or more statements repeatedly while a logical condition remains true. |
| write | Write to a text file. |
| writcase | 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.

| | | | |
|--------------------------|----------------------------|---------------------------|--------------------------|
| accept | endif | ioerror | relation |
| add | endgroup | ispartial | retrieve |
| advance | endlevel | item | savepartial |
| all | enter | key | seed |
| alpha | errmsg | killfocus | selcase |
| and | exec | length | set |
| array | execsystem | level | savepartial |
| average | exit | linked | setfile |
| box | exp | loadcase | setlb |
| break | export | locate | setub |
| by | file | log | skip |
| case | fileconcat | maketext | soccurs |
| clear | filecopy | max | sort |
| close | filecreate | maxocc | special |
| cmcode | fileexist | min | specific |
| compare | filedelete | missing | sqrt |
| concat | filename | move | stat |
| count | fileread | next | stop |
| crosstab | filerename | noccurs | strip |
| curocc | filesize | noinput | sum |
| default | filewrite | not | summary |
| delcase | find | notappl | sysdate |
| delete | float | numeric | sysparm |
| demenu | for | onfocus | systeme |
| demode | function | onkey | then |
| denom | getbuffer | onstop | title |
| disjoint | getlabel | open | to |
| display | getnote | or | tonumber |
| do | getoperatorid | pos | totocc |
| edit | getsymbol | poschar | until |
| editnote | if | postproc | update |
| else | impute | preproc | visualvalue |

| | | | |
|----------------------------------|-----------------------------------|--|--|
| elseif end endbox enddo | in insert int invalueset | proc putnote random recode reenter | vset where 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:

| <u>System Setting</u> | <u>Program Setting</u> | <u>Result</u> |
|-----------------------|------------------------|---|
| Set Explicit | set explicit; | No effect |
| Set Explicit | set implicit; | Program overrides system setting, variables do not need to be declared |
| Set Implicit | set explicit; | Program overrides system setting of implicit—variables must be declared |
| Set Implicit | set implicit; | No effect |

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, Alpha Statement, Array Statement

Alpha Statement

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, if 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 a,b,c;
  alpha(10) x,y;

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: Numeric Statement, Array Statement

Array Statement

Format:

```
array [alpha[(len)]] array-name(dim-1[,dim-2[,dim-3]]);
```

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.

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.

See also: Arrays, Numeric Statement, Alpha 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 express 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 or alpha expressions can be passed to a user-defined function as parameters. To specify an alphanumeric parameter, you must place the keyword **alpha** before the parameter. By default, the length of the alphanumeric the parameter is 16 characters. To specify a different length, place **alpha (length)** before the parameter name.

Functions always return 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.

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. Variables mentioned in the parameter list will by default be considered numeric; if the user needs to pass a string variable to a function, the **alpha** specification must be explicit in the parameter list. See Example 3.

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

An example of a user-defined function can be found in the **DateCheck** folder, under the *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** phrase 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** phrase 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 effect 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: Skip Case Statement, Stop Function

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

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 remember (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";
```

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.

Data Entry Statements and Functions

Accept Function

Format:

```
i = accept(heading, opt-1, opt-2[,...opt-n]);
```

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.

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:

```
PROC UR
preproc
i = 0;
do until i in 1:2
  i = accept("Area Designation?", "Urban", "Rural");
enddo;
$ = i;
noinput;
```

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

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;
```

Editnote Function

Format:

```
[s =] editnote();
```

[] indicates that this part is optional.

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 **endsect** 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\ItemDrivenDE* 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;
```

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

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 himself 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());
```

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 data and false (0) if the field is highlighted.

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 Batch Edit Events, Reenter Statement, If Statement

Move Statement

Format 1:

```
move [to] field-name [adv|skp]
```

Format 2:

```
move [to] alpha-variable [adv|skp]
```

[] 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 an 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 **skp** or **adv**. If no keyword or **skp** is coded, forward movement acts exactly like a skip statement. If **adv** is coded, forward movement acts exactly like an advance statement.

Example 1:

```
move SEX;
```

Example 2:

```
move to SEX adv;
```

Example 3:

```
move to LAST_FIELD adv;
```

where SEX is a field and LAST_FIELD is an alpha variable.

See also: Renter 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;
```

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 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 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.

Returned value:

Like any global function, the onkey global function returns an integer value. The value should be either value of the key pressed (the same as the value passed to the function), or 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.

| <u>Key</u> | <u>Add</u> |
|----------------|------------|
| 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

| <u>Key Combination</u> | <u>Code</u> |
|------------------------|-------------|
| 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:

| <u>Key</u> | <u>Code</u> |
|------------|-------------|
| 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.

KeyCode

| | |
|---|----|
| 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

| <u>Key</u> | <u>Code</u> |
|------------|-------------|
| 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:

| <u>Key</u> | <u>Code</u> |
|------------|--------------------------------|
| 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:

| <u>Key</u> | <u>Code</u> |
|------------|------------------|
| SysReq | no code returned |
| Bksp | 8 |

| | |
|-------------|-------------------------------|
| Tab | 9 |
| Enter | 13 (with or without num lock) |
| Break | 19 |
| Caps | 20 |
| Escape | 27 |
| 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 (back quote) |
| [| 219 |
|] | 221 |
| \ | 220 |
| ' | 222 (single quote) |

Control Keys:

| <u>Key</u> | <u>Code</u> |
|------------|-------------|
| Shift | 1016 |
| Ctrl | 2017 |
| Alt | 4018 |

See also: Global Procedure, User Defined Functions


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.

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 a logical value of 1 (true) if a case is found and 0 (false) otherwise.

Example:

```
PROC COOKING
if $ <> 9 then
  putnote("");
endif;
```

See also: Getnote Function, Editnote 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 in any procedure except the preproc and postproc of the run. 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]);
```

[] 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 key, that should be suppressed upon presentation.

Return value:

The function returns a logical value of **true** (1) if a key is selected and **false** (0) otherwise.

Example:

```
OK = selcase(LOOKUP, concat(PROV, DIST));
```

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 (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 field 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:

| | |
|----------------|--|
| display | If a field is hidden, its value will now be visible; if it was already visible, the setting has no effect. |
| visible | If a field is hidden, its value will now be visible; if it was already visible, the |

| | |
|-----------------------|---|
| | setting has no effect |
| <code>autoskip</code> | 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. |
| <code>return</code> | 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. |
| <code>protect</code> | 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. |
| <code>hidden</code> | If a field is visible, its value will now be hidden from view; if it was already hidden, the setting has no effect. |
| <code>native</code> | 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. |

Description of Format 2:

The set attributes statement 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.

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.

Example1:

```
set attributes (total_HH_income) protect;
```

Example 2:

```
set attributes(REL, SEX, EDUC) assisted on (responses);
```

Example 3:

```
set attributes(MYDICT) assisted on (responses);
```

See also: Change Field Properties, Change Data Entry Options

Set Behavior Canenter Statement

Format:

```
set behavior() canenter(notappl) on | off
(confirm | noconfirm);
```

Description:

The **set behavior canenter** statement allows the entry of blanks, notappl, for numeric data items during data entry. This behavior affects all numeric data items from the point where it is executed. To limit its scope, it must be turned on and off at appropriate times.

If when confirm is used, the operator must confirm whether they want blank to be entered. If noconfirm is used, blank is accepted without any message being given.

Example:

```
set behavior() canenter(notappl) on (noconfirm);
```

See also: Special Values

Skip Statement

Format 1:

```
skip [to [next]] field-name;
```

Format 2:

```
skip [to [next]] alpha-variable;
```

[] 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.

"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

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

Set Behavior Export Statement

Format:

```
set behavior() export (model [itemtype]);
```

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**, **All**, **CSPRO**, **TabDelim**, **CommaDelim**, or **SemiColonDelim** indicating the type of file being exported .

The *itemtype* is optional. It is one of the keywords **ItemOnly**, **SubitemOnly**, and **ItemSubitem** indicating how subitems and their parent item are handled when entire records are exported. If *itemtype* is not coded, **SubitemOnly** is assumed.

See also: Export 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: Stop Function, Exit Statement , Errmsg Function, If Statement, Totocc Function

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** phrase 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.

See also: Set Behavior Export Statement

Numeric Functions

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 $(81 \times 12) + 6 = 978$.

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.

Exp Function

Format:

```
d = exp(numeric-expression);
```

Description:

The **exp** function raises the value of **e** to the power given by "numeric-expression". The value of **e** is the Napier constant, the basis of natural logarithms.

Return value:

The function returns a decimal number power. If the value of "numeric-expression" is a special value (MISSING, NOTAPPL, or DEFAULT), the function returns that value.

Example:

```
X = exp(Y);
```

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);
```

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 which 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);
```

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.

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);
```

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 which return strings.

Return value:

The function returns the concatenated string.

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));
```

Results in the following values:

```
FIRST_NAME = "John"           "
LAST_NAME  = "Henry"         "
FULL_NAME  = "John Henry"    "
```

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., "p1") 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 = 23456 | %d | 23456 |
| | %10d | 23456 |
| | %-10d | 23456 |
| | %010d | 0000023456 |
| | %+10d | +23456 |
| | %+010d | +000023456 |
| | %f | 23456.000000 |

| | | |
|----------------|----------|------------|
| Value = 12.567 | %f | 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 = "abcdef" | %s | abcdef |
| | %10s | abcdef |
| | %-10s | abcdef |
| | %10.3s | abc |
| | %-10.3s | abc |

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-string" 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).

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

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

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 noccurr 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: Noccurr Function, Soccurs Function, Totocc Function, Curocc Function

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. The **curocc** of a record will always be the total number of occurrences found. 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, Noccurr 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):

```
i = 1;
NumPeople = totocc (PERSON_REC);
while i <= NumPeople do
  if rel(i) = notappl and
    sex(i) = notappl and
    age(i) = notappl then
    delete (PERSON_REC(i)); { remove "blank" population records }
  else
    i = i + 1; { increment i only if we DON'T delete someone }
  endif;
enddo;
```

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, While 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 can not 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:

```
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. If the form does not repeat, **noccurs** will return 1 (a roster must always repeat). If the **noccurs** function is executed prior to the roster or form it names, it returns 0. If it is executed from a field within the roster or form, 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, **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

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. If the record does not repeat, **soccurs** will return 1. If the **soccurs** function is executed prior to the record it names, it returns 0. If it is executed from a field within the record, it returns the current occurrence number. If it is executed after the entry of the record has completed, it returns the total number of occurrences of the record. When used in a data entry application, this function is equivalent to the noccurs function.

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);
```

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.

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:

```
Sort(PERSON using LINE_NUM);
```

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);
```


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 after the **postproc** of the first field within a repeating form or roster is executed. 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 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, **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

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);
```

Errmsg (Display) Function

Format 1:

```
[b =] errmsg(alpha-exp[, p1[, p2[, . . . , pn]]]) [denom=var] [case|summary];
```

Format 2:

```
[b =] errmsg(msg-num[, p1[, p2[, . . . , pn]]]) [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       12.567
                 %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 = "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.

Return Value:

The function returns a logical value 1 (true) if successful and 0 (false) otherwise.

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;
```

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

Format:

```
b = execsystem(alpha-exp, [maximized | normal | minimized],
               [focus | nofocus], [wait | nowait]);
```

Description:

The **execsystem** function starts another windows application or process.

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 apostrophies (').

Coding one of 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 parameters **focus** or **nofocus** determines whether the new application will receive focus, that is 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**.

Return value:

The function returns a logical value of 1 (true) if the new application is started successfully and 0 (false) otherwise.

Example 1:

```
Execsystem("c:\windows\calc.exe");
```

Example 2:

```
Execsystem('textview "c:\program files\cspiro 2.6\readme.txt" ',maximized,wait);
```

See also:

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 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

Invalueset Function

Format:

```
b = invalueset(item-name [, valueset-name]);
```

Description:

The **invalueset** function determines whether a data item's value is within the items 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 invaluset(P03_SEX) then
  errmsg("Sex is out of range. Value is %d", P03_SEX);
endif;
```

Example 2:

```
if invaluset(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

Special Function

Format:

```
b = special(numeric-exp);
```

Description:

The **special** function determines whether a variable's 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, If Statement, Invaluset 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 stopped. If the parameter is empty or 0, entry of the current case is stopped (the same as 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.

When the stop function is executed in data entry, any data entered for the current case is lost. If you want to save data entered for the current case, you must include a `savepartial` function just before the stop function.

If the stop function is used in a batch edit application, the argument has no effect. 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

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
```

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. A parameter can be used in a data entry or batch edit program; merely add in the desired string to your data entry pff file or batch edit pff file.

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();
```

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 % 10000;
```


External File Functions

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** or 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**, **writcase**, **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 in 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 who is marked for deletion (but not deleted; a compact application is needed to actually delete cases marked for deletion).

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 identifiers in memory 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.

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.

Return value:

The function returns a logical value of 1 (true) if successful and 0 (false) otherwise.

Example:

```
filecopy(DATA, DATACOPY);
```

DATA and DATACOPY could be the names of files declared in a **file** statement or a variables declared in an **alpha** statement.

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 zero length.

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 a variable declared in an **alpha** statement.

See also: File Statement

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.

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 a variable declared in an **alpha** statement.

See also: File Statement

Filedelete Function

Format:

```
b = filedelete(file-name);
```

Description:

The **filedelete** function deletes an already existing 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:

```
filedelete(MYREPORT);
```

MYREPORT could be the name of a file declared in a **file** statement or a variable declared in an **alpha** statement.

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

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.

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

Filename Function

Format:

```
b = filename(old-file-name, new-file-name);
```

Description:

The **filename** function changes the name of a file.

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:

```
filename(MYDATAFILE, "c:\folder\my new data file.dat");
```

MYREPORT could be the name of a file declared in a **file** statement or a variable declared in an **alpha** statement.

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:

```
filesize(MYREPORT);
```

MYREPORT could be the name of a file declared in a **file** statement or a variable declared in an **alpha** statement.

See also: File Statement, Filename 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 the characters backslash and then the letter f (\f) at the beginning of the text string.

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);
```

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-ex" 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 = 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** or 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 in 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](#)

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. 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.

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 = 23456 | %d | 23456 |
| | %10d | 23456 |
| | %-10d | 23456 |
| | %010d | 0000023456 |
| | %+10d | +23456 |
| | %+010d | +000023456 |

| | | |
|------------------|----------|--------------|
| | %f | 23456.000000 |
| Value = 12.567 | %f | 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 = "abcdef" | %s | abcdef |
| | %10s | abcdef |
| | %-10s | abcdef |
| | %10.3s | abc |
| | %-10.3s | abc |

Return Value:

The function returns a logical value 1 (true) if successful and 0 (false) otherwise.

Example:

```
write("Sex = %d", SEX);
```

Appendix

Appendix A - Installation

Hardware and Software Requirements

A minimum configuration

- 33MHz 486 processor
- 16MB of RAM
- VGA monitor
- Mouse
- 45 MB of free hard drive
- Microsoft Windows 95

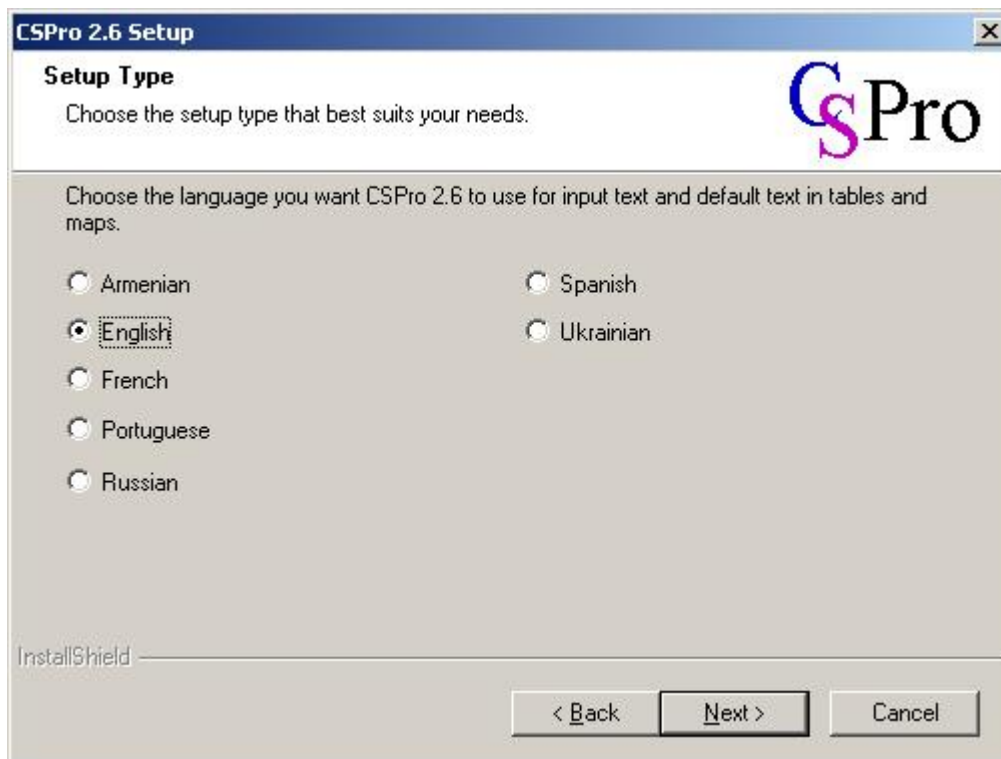
A recommended configuration

- Pentium processor
- 256MB of RAM
- SVGA monitor
- Mouse
- 45 MB of free hard drive space
- Microsoft Windows 98, Me, NT 4.0, 2000, or XP

Installing CSPRO

- **Installing from our CD**
 - Place the CD in your CD-ROM drive. Wait for a few seconds – the installation program will automatically launch
 - Proceed through the setup process. This will take you through a series of dialog boxes, which will prompt you for setup information
- **Installing from floppy diskettes:**
 - From the Start button on the taskbar, select Run.
 - Place CSPRO Disk 1 in your diskette drive and enter **a:\setup**
 - Click OK. The setup process takes you through a series of dialog boxes that prompt you for setup information.
- **Installing from a file downloaded from our web site:**
 - From the Start button on the taskbar, select Run.
 - Enter **drive:\folder\cspro25.exe** (For example: c:\download\cspro25.exe)
 - Click OK. The setup process takes you through a series of dialog boxes, which will prompt you for setup information.
- **Selecting languages for installation**

CSPRO allow you to select a language that will be used in some parts of the system. During installation you will be presented with the following language screen:



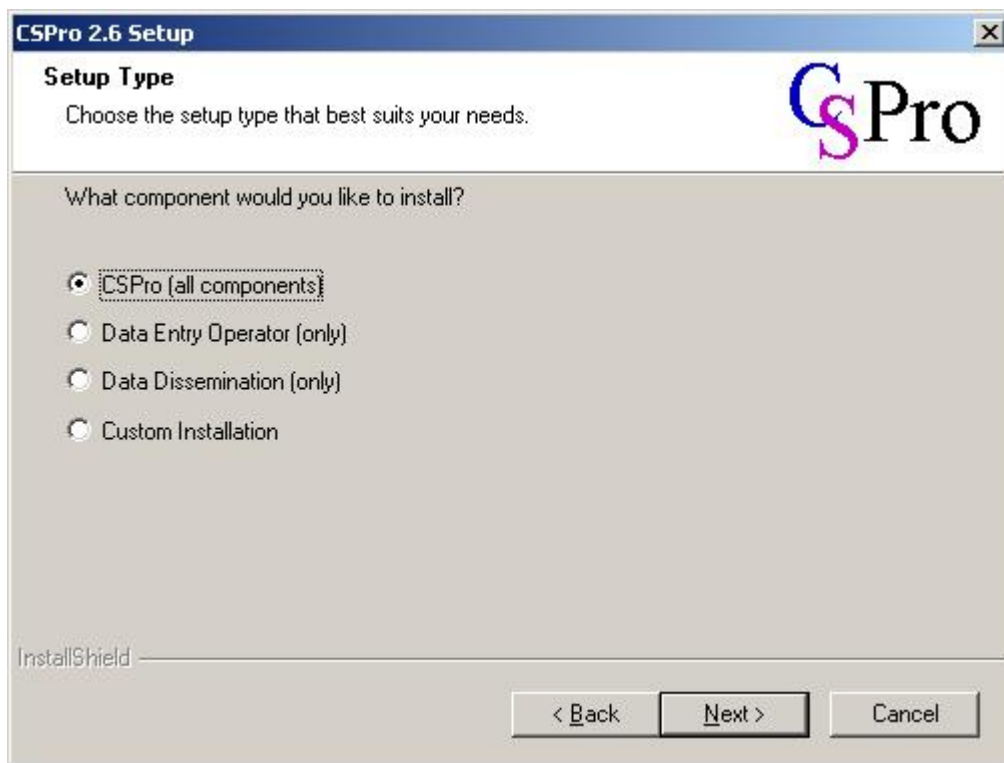
The language setting determines:

- the small words (Table, and, by, Legend, etc.) generated by tables and maps
- the font style, in order to support languages such as Russian, Ukrainian, or Armenian on data entry screens and in tabulation output.

Note that CSPro's menu options, dialog boxes, and the help system are only available in English. You can always change the language setting by rerunning the installation program and choosing "Modify."

• **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. The Data Entry, Compare Data, Text Viewer, and Table Viewer components are installed.
- Data Dissemination (only). Select this if you are installing CSPro data dissemination tools for the user to access statistical data you have produced using CSPro. The Map Viewer, Table Retrieval, Table Viewer, Text Viewer, and Compare Data components are installed.

- Custom Installation. Advanced users can select any particular set of components, such as a particular tool.

At any later time, you can change the components installed by rerunning the installation.

Uninstalling CSPRO

The following is based on a Windows 2000 setup. Your steps may vary if using a different operating system.

- From the **Start** button on the taskbar, select **Settings – Control Panel**.
- Select **Add/Remove Programs**.
- From the list of currently installed programs, click on **CSPRO 2.6**.
- Click the **Change/Remove** button.
- The installation program prompts you to select the type of installation. Select **Remove** and press the **Next** button.
- At the prompt, click **OK** to confirm that you want remove CSPRO 2.6. The uninstall program will remove all registry entries and CSPRO system files (that is, all files within the CSPRO 2.6 folder its Examples subfolder). 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 2.6 from CSPRO 2.5**

If you have CSPRO 2.5 installed on your computer, you can install CSPRO 2.6 without affecting the CSPRO 2.5 installation. When you have finished your conversion of applications to CSPRO 2.6, you can then uninstall CSPRO 2.5.

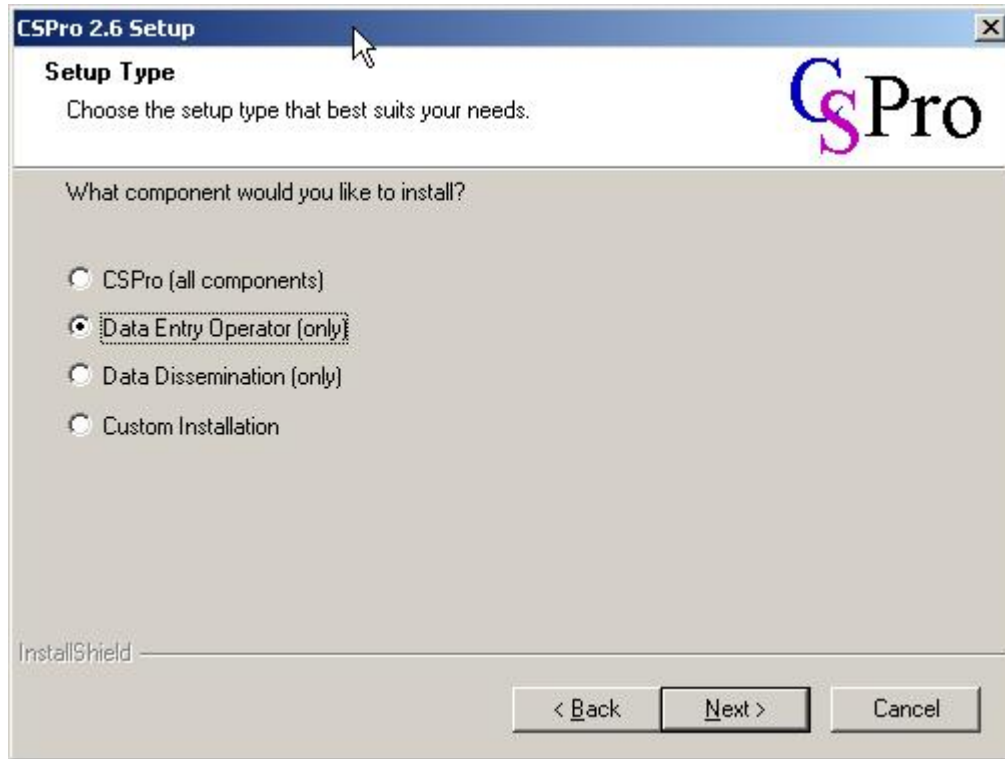
Please note that due to internal changes within CSPRO 2.6, once files have been loaded in CSPRO 2.6, you may no longer be able to load them in CSPRO 2.5.

- **Updating a previous version of CSPRO 2.6**

If you are updating a previous version of CSPRO 2.6, 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 2.6 over top the older version. It will not change any applications or other files your have created.

Installing Data Entry Applications

- **By Itself**
To install data entry applications on each data entry computer you must:
- **Install the CSPPro Data Entry Operator Software**



Run the CSPPro installation program (setup) on each computer. The installation setup may be run from diskettes, CDROM, or your network. Choose the "Data Entry Operator (only)" option during the installation. The setup program will only install the files necessary to perform data entry. The CSPPro components necessary to modify an application will NOT be installed.

- **Install the CSPPro Data Entry Application**
Make a folder (directory) on each data entry computer and copy the application files into it. You can determine which files are in your application by opening the application in CSPPro and clicking on the "Files" tab. You may have to use **Ctrl-T** to see the physical file names and their locations as opposed to their descriptive labels. For information about each of these files and about data entry applications in general, see the Data Entry User's Guide.

Changing Language or Components

You can change the language or components that are installed on your computer by running the installation again. When the installation program prompts you to select the type of installation, select **Modify** and press the **Next** button.

Make any changes you wish to make and press the **Next** button. Your installation will be changed. New components will be added and unwanted components will be removed. Application and data files will NOT be affected.

Appendix B - Keys Summary

Data Dictionary Keys

Shortcuts specific to the Data Dictionary

| | |
|-----------------|--|
| 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 CSPPro

| | |
|------------------|---|
| 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. |
| F1 | Show help contents and index. |

Data Entry Forms Designer Keys

Shortcuts specific to Data Entry Designer

| | |
|-------------------------|--|
| 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 + M | Show or hide form |
| Up Arrow | Move up one line. |
| Down Arrow | Move down one line. |
| Shift+Up Arrow | Scrolls up, multi-selects rows. |
| Shift+Down Arrow | Scrolls down, multi-selects rows. |

| | |
|------------------------|--|
| Page Up | Scroll up one screen (if possible). |
| Page Down | Scroll down one screen (if possible). |
| Shift+Page Up | Scrolls up, multi-selecting pages. |
| Shift+Page Down | Scrolls down, multi-selecting pages. |
| Home | Scrolls to the beginning of line. |
| End | Scrolls to the end of line. |
| Shift+Home | Selects text from cursor to beginning of line. |
| Shift+End | Selects text from cursor to end of line. |

Shortcuts common throughout CSPRO

| | |
|------------------|---|
| 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 + 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 | Quit the application. |
| F1 | Show help contents and index. |

Data Entry Application Keys

Shortcuts specific to Data Entry Designer

| | |
|-------------------------|---|
| Del | Delete currently selected item(s). |
| Ctrl + Del | Delete the currently displayed form. |
| Ctrl + A | Add form to current level. |
| Ctrl + D | Change options for dragging things from dictionary. |
| Ctrl + E | Change options for data entry application. |
| Ctrl + G | Generate a set of forms based on the dictionary. |
| Ctrl + K | Compile code. |
| Ctrl + L | Show or hide logic |
| Ctrl + M | Show or hide form |
| Ctrl + R | Run the data entry application. |
| Ctrl + Q | CAPi questions |
| F3 | Find the next occurrence of specified text. |
| Up Arrow | Move up one line. |
| Down Arrow | Move down one line. |
| Shift+Up Arrow | Scrolls up, multi-selects rows. |
| Shift+Down Arrow | Scrolls down, multi-selects rows. |

| | |
|------------------------|--|
| Page Up | Scroll up one screen (if possible). |
| Page Down | Scroll down one screen (if possible). |
| Shift+Page Up | Scrolls up, multi-selecting pages. |
| Shift+Page Down | Scrolls down, multi-selecting pages. |
| Home | Scrolls to the beginning of line. |
| End | Scrolls to the end of line. |
| Shift+Home | Selects text from cursor to beginning of line. |
| Shift+End | Selects text from cursor to end of line. |
| Ctrl+Home | Scrolls to the first line of code. |
| Ctrl+End | Scrolls to the last line of code. |

CAPI Options

| | |
|-----------------|----------------|
| Ctrl + B | Bold text |
| Ctrl + I | Italic text |
| Ctrl + U | Underline text |

Shortcuts common throughout CSPro

| | |
|------------------|---|
| 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 + 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 | Quit the application. |
| F1 | Show help contents and index. |

Batch Edit Keys

Shortcuts specific to Batch Edit Designer

| | |
|-------------------------|---|
| Del | Delete currently selected item(s). |
| Ctrl + K | Compile code. |
| Ctrl + R | Run the data entry application. |
| F3 | Find the next occurrence of specified text. |
| Up Arrow | Move up one line. |
| Down Arrow | Move down one line. |
| Shift+Up Arrow | Scrolls up, multi-selects rows. |
| Shift+Down Arrow | Scrolls down, multi-selects rows. |
| Page Up | Scroll up one screen (if possible). |
| Page Down | Scroll down one screen (if possible). |

| | |
|------------------------|--|
| Shift+Page Up | Scrolls up, multi-selecting pages. |
| Shift+Page Down | Scrolls down, multi-selecting pages. |
| Home | Scrolls to the beginning of line. |
| End | Scrolls to the end of line. |
| Shift+Home | Selects text from cursor to beginning of line. |
| Shift+End | Selects text from cursor to end of line. |
| Ctrl+Home | Scrolls to the first line of code. |
| Ctrl+End | Scrolls to the last line of code. |

Shortcuts common throughout CSPPro

| | |
|------------------|---|
| 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 + 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 | Quit the application. |
| F1 | Show help contents and index. |

CrossTab Keys

Shortcuts specific to Cross Tabulation

| | |
|-----------------|--|
| Ins | Insert a new table into the table set. |
| Del | Delete a table from the table set. |
| Esc | Cancel the current selection. |
| Ctrl + A | Add a new table to the table set. |
| Ctrl + M | Generate a thematic map. |
| Ctrl + R | Run the tabulation. |
| Alt + A | Edit the (geographic) areas of tabulation. |
| Alt + M | Modify the current table's title. |
| Alt + P | Modify the current table's parameters. |
| Alt + U | Modify the current table's universe. |

Shortcuts common throughout CSPPro

| | |
|-----------------|---|
| Ctrl + C | Copy the selection and put it on the Clipboard. |
| 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 + F4** Close the active document.
- Alt + F4** Quit the application.
- F1** Show help contents and index.

Appendix C - Menu Summary

Generalized Menu

The CSPro menu is displayed across the top of the window. Every application includes the generalized menu and menus specific to the application. When applications are open, the menu corresponding to the contents of the right-hand window appears.

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 based on ids.
- 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.
- Convert Dictionary Convert an ISSA or IMPS dictionary to CSPro.
- Convert Shape to Map Convert an ESRI shape file to CSPro map file.

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

Initial Menu

It appears only when CSPro is opened without any applications or files open. It does not include a window menu.

Files

- New Create a new application.
- Open Open an existing application.

Data Dictionary Menu

Files

- 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. |
| Insert 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. |
| 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 which contains them. |

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

Files

| | |
|--------------|---|
| | Create a new application. |
| Open | Open an existing application. |
| Close | Close an application. |
| Save | Save an application. |
| Insert Files | Insert a file in an existing application. |
| Drop Files | Drop a file 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 |

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 |
| Use CAPI Font 2 |
| Bold |
| Italic |
| Underline |
| Color |
| Left |
| Center |
| Right |
| Bullets |
| Insert Bitmap |
| Help Text |
| Change CAPI Languages |
| Change CAPI Font 1 |
| Change CAPI Font 2 |
| Fit Windows to Questions |

Batch editing menu

Files

| | |
|--------------|---|
| New | Create a new application. |
| Open | Open an existing application. |
| Close | Close an application. |
| Save | Save an application. |
| Insert Files | Insert a file in an existing application. |
| Drop Files | Drop a file 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. |
| Properties | Show and modify properties of items in the order tree. |
| 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. |

Cross Tabulation Menu

Files

| | |
|---------------|---|
| New | Create a new application. |
| Open | Open an existing application. |
| Close | Close an application. |
| Save | Save an application. |
| Save Tables | Save current table results in a file. |
| Insert Files | Insert a file in an existing application. |
| Drop Files | Drop a file from an existing application |
| Run | Run the 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

| | |
|--------------|--|
| Add Table | Add a table at the end. |
| Insert Table | Insert a table at the current location. |
| Delete Table | Delete the current table. |
| Copy | Copy selected parts of the table to the clipboard. |
| Select All | Select the entire table. |

| | |
|------------------|---|
| Cancel Selection | Cancel the currently selected cells. |
| Modify Title | Edit the table title. |
| Universe | Edit the universe or filter of tabulation. |
| Parameters | Select tabulation by value, weight, percents and undefined value. |
| Area | Enable or change tabulation by geographic area. |

View

| | |
|----------------|---|
| Names in Trees | Show names instead of labels in trees. |
| Full Screen | Hide the trees and show full screen view. |
| 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



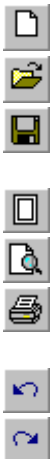
- Create a new application.
- an existing application
- Get help.

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 Cross 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 Cross 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.
-  Removes the currently selected text
-  Copies the current selection to the clipboard
-  Pastes the current contents of the clipboard to the cursor position
-  Find text in the logic
-  Launch Text Viewer




-  Show last Dictionary window.
-  Show last Forms window.
-  Show last Batch Edit window.
-  Show last Cross Tabulation window.
-  Get Help.


Cross Tabulation Toolbar





The CrossTab 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 CrossTab.





Click To




-  Create a New application
-  Open an application
-  Save an application
-  Save tables

-  Setup page margins and headings for printing.
-  Preview contents of the dictionary.
-  Print tables.

-  Run a tabulation

-  Add a table
-  Insert a table
-  Delete the current table
-  Copy table selection

-  Specify a Universe for tabulation
-  Specify tabulation Parameters
-  Define Area Processing variables
-  Use the MapViewer

-  Show last Dictionary window.
-  Show last Forms window.
-  Show last Batch Edit window.



Show last Cross 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.
 - CSPRO does not currently support the existence of multiply-occurring items within a multiply-occurring record, if any are found, the record will be made singly-occurring.
 - If you have used "NR" as a value name in your IMPS dictionary, it will be converted to have a special value of "missing."
- **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 16 characters of any item name in CSPro is not unique, you should check the converted IMPS dictionary to ensure the names are correct.

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, CSPPro provides a utility to convert it to a CSPPro Form File (a dictionary will be generated as well).

At any time in CSPPro (you needn't have anything open), you can go to the **T**ools menu option. Select **Convert Dictionary** from the drop-down menu, then choose to convert **Between CSPPro and ISSA** in the opening dialog box.

Next, state that you'd like to convert from **ISSA to a CSPPro Forms and Data Dictionary**. Provide the name of the original ISSA dictionary file, and the name you would like to call the CSPPro form file to be generated (a CSPPro 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 CSPPro 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 administered 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 information 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 | Miskeyes, 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 specifications. 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**

- DCF - Data Dictionary file
- ENT - Data Entry application file
- FMF - Forms file
- APP - Logic file
- MGF - Message file
- QSF - Question file

- **Batch Edits**

- DCF - Data Dictionary file
- BCH - Batch Edit application file
- ORD - Edit Order file
- APP - Logic file
- MGF - Message file
- LST - Listing file
- FRQ - Frequency file

- **Cross Tabulations**

- DCF - Data Dictionary file
- XTB - Cross Tab Application file
- XTS - Cross Tabulation Specification file
- TBW - Tables file
- ANM - Area Name file

- APP - Logic file
- MGF - Message file

- **Map Viewer**
 - MAP - Map file
 - MDF - Map Data file

- **Others**
 - PFF - Program Information file
 - FQF - Frequency specification file
 - LOG - Operator Statistics file

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 an ASCII 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.

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 an ASCII 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.

- CSPPro 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 an ASCII text file which may be viewed with any text editor, such as CSPPro's Text Viewer or the Windows Notepad. It is not recommended to make changes to this file outside the CSPPro environment. Advanced users might do so, however, to change the name of the associated data dictionary file.

Logic File (.APP)

- The logic file contains all the CSPPro language statements which control the application. There is one logic file associated with each application.
- CSPPro 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 an ASCII text file which may be viewed with any text editor, such as CSPPro's Text Viewer or the Windows Notepad. While you may make changes to this file outside the CSPPro environment, CSPPro provides a powerful text editor which is integrated with the CSPPro compiler.

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.

See also: Message File.

Question File (.QSF)

- The question file contains information related to **CAPI (Computer Assisted Personal Interviewing)** 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 (**F1**) key.

Data file

The data file is an ASCII text file. Data files are limited to 2 gigabytes in length. A data file can have any extension, CSPPro does not assign an extension by default.

Listing File (.LST)

This file is always generated in a batch application, regardless of whether your program includes `errmsg` commands. It contains the results of the run.

Cross Tabulation Application File (.XTB)

- The Cross Tabulation Application file is the master file for a cross tabulation application. This file specifies all other files contained in the application, along with other information.
- CSPro allows you to open, close and save cross tabulation application files. When you do so, all other files associated with the application are also opened, closed or saved.
- The application file is an ASCII 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 Cross Tabulation specification file contains tables, dictionary items/value sets and other information which defines a set of cross-tabulations. The file also contains the name of the associated data dictionary file. Items and value sets have links into the data dictionary.
- CSPro allows you to explicitly open, close and save Cross Tabulation specification files independently of the application file. When you do so, the associated data dictionary file is also opened, closed or saved.
- The Cross Tabulation specification file is an ASCII 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.

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.

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. You can also convert an area file created in IMPS with extension .ara to .anm in the area IDs dialog box.

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

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 an ASCII 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.

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 you would 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.

Map File (.MAP)

The map file (.MAP) is a text file that contains information about the map and contains the map polygon data points.

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.

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, 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

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 age 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 open, 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 designed to be easily imported into other software packages for custom processing. The LOG file could be processed by a CSPro application by creating 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 verifier |
| 138 | <comma> |
| 139 – 146 | Total number of fields verified |

Index

| | | |
|-------|---------------------------------------|--|
| | i | |
| - | Operator67 | Batch Edit Application 142 |
| | Operator Precedence.....70 | Batch Edit Application File..... 313 |
| | | Batch Edit Applications..... 4 |
| | | .DCF |
| | | Data Dictionary 6, 36, 37 |
| | | Data Dictionary File 310 |
| | | In a Batch Edit Application 142 |
| | | In a Cross Tabulation Application ... 176 |
| | | In a Data Entry Application..... 79 |
| | | .ENT |
| | | Data Entry Application File 310 |
| | | Data Entry Applications 4, 79 |
| | | .FMF |
| | | Data Entry Applications 4, 79 |
| | | Form File 6, 311 |
| | | .FQF |
| | | Frequency Specification File 314 |
| | | .FRQ |
| | | Frequency File..... 314 |
| | | .LOG |
| | | Operator Statistics File 315 |
| | | .LST |
| | | Listing File 312 |
| | | .MAP |
| | | Map File..... 314 |
| | | .MDF |
| | | Map Data File 314 |
| | | .MGF |
| | | In a Batch Edit Application 142 |
| | | In a Data Entry Application..... 79 |
| | | Message File 312 |
| | | .MPC |
| | | 190 |
| | | .ORD |
| | | Batch Edit Application 142 |
| | | Order File..... 313 |
| | | .PFF |
| | | Program Information File..... 314 |
| | | Run Production Batch Edits 164 |
| | | Run Production Data Entry..... 87 |
| | | .QSF |
| | | In a Data Entry Application..... 79 |
| | | Question File 312 |
| | | .TBW |
| | | Tables File 313 |
| | | .WRT |
| | | Create a Specialized Report 158 |
| | | .XTB |
| | | Cross Tabulation Application..... 176 |
| | | Cross Tabulation Application File..... 312 |
| | | Cross Tabulation Applications..... 5 |
| | | .XTS |
| | | Cross Tabulation Application..... 176 |
| | | Cross Tabulation Applications..... 5 |
| | ! | |
| ! | Operator67 | |
| | Operator Precedence.....70 | |
| | | |
| | \$ | |
| \$ 62 | | |
| | | |
| | % | |
| % | Operator67 | |
| | Operator Precedence.....70 | |
| %d | errmsg function262 | |
| | write function283 | |
| %f | errmsg function262 | |
| | write function283 | |
| %s | errmsg function262 | |
| | write function283 | |
| | | |
| | & | |
| & | Operator67 | |
| | Operator Precedence.....70 | |
| | Truth Table.....70 | |
| | | |
| | * | |
| * | Operator67 | |
| | Operator Precedence.....70 | |
| | | |
| | . | |
| .ANM | Area Names File313 | |
| .APP | In a Batch Edit Application 142 | |
| | In a Data Entry Application 79 | |
| | Logic File311 | |
| .BCH | | |

| | |
|---------------------------------|-----|
| Table Specifications File | 312 |
| / | |
| / | |
| Operator | 67 |
| Operator Precedence..... | 70 |
| ^ | |
| ^ | |
| Operator | 67 |
| Operator Precedence..... | 70 |
| | |
| | |
| Operator | 67 |
| Operator Precedence..... | 70 |
| Truth Table..... | 70 |
| + | |
| + | |
| Operator | 67 |
| Operator Precedence..... | 70 |
| < | |
| < | |
| Operator | 67 |
| Operator Precedence..... | 70 |
| <= | |
| <= | |
| Operator | 67 |
| Operator Precedence..... | 70 |
| <=> | |
| <=> | 69 |
| If and Only If..... | 69 |
| Operator | 67 |
| Operator Precedence..... | 70 |
| <> | |
| <> | |
| Operator | 67 |
| Operator Precedence..... | 70 |
| = | |
| = | |
| Operator | 67 |
| Operator Precedence..... | 70 |
| > | |
| > | |
| Operator | 67 |
| Operator Precedence..... | 70 |
| >= | |
| >= | |
| Operator | 67 |

| | |
|---------------------------|----|
| Operator Precedence | 70 |
|---------------------------|----|

A

| | |
|--|--------|
| Absolute | |
| Mode..... | 41 |
| Positioning | 49 |
| Accept Function | 220 |
| Add | |
| Dictionary Elements | 44 |
| Fields to a Form | 97 |
| Form | 96 |
| Identification items..... | 27 |
| Items in Dictionary..... | 44, 46 |
| Levels in Dictionary | 44, 45 |
| Lines or Boxes to a Form | 99 |
| Logic in Data Entry | 75 |
| Pictures in CAPI Question Text..... | 135 |
| Records in Dictionary | 44, 45 |
| Relations in Dictionary..... | 50 |
| Roster to a Form | 97 |
| Table..... | 195 |
| Text to a Form | 99 |
| Things to a Roster | 98 |
| Value Sets in Dictionary | 44, 46 |
| Values in Dictionary..... | 44, 46 |
| Advance Statement..... | 221 |
| Align Text and Fields | 102 |
| Bottom | 102 |
| Center..... | 102 |
| Horizontal | 102 |
| Left..... | 102 |
| Mid..... | 102 |
| Right | 102 |
| Top | 102 |
| Vertical..... | 102 |
| Allow Partial Save | 90 |
| Alpha Statement | 204 |
| Alphabetical List of Statements and Functions..... | 197 |
| Alphanumeric | |
| Array | 205 |
| Data Type | 30 |
| Variables..... | 54 |
| And..... | 67, 70 |
| Operator | 67 |
| Operator Precedence | 70 |
| Truth Table | 70 |
| ANM | |
| Area Names File..... | 313 |
| APP | |
| In a Batch Edit Application | 142 |
| In a Data Entry Application..... | 79 |
| Logic File | 311 |
| Applications..... | 86 |
| Batch Edit | 4 |

- Closing 14
 Compile 123
 Create a CSPro 10
 Create a New Batch Edit 142
 Create a New Data Entry 79
 Create a New Tabulation 176
 Cross Tabulation 5
 Insert or Drop a File 12
 Installing Data Entry 287
 Matching to the Data Dictionary 106
 Move Around a Logic Application 120
 Open an Existing Dictionary Application
 43
 Run a Batch Edit 147
 Run a Data Entry 86
 Save 13
- Area**
 Create a Names File 188
 IDs Dialog Box 190
 Processing 187
- Area Names File**
 .ANM 313
 ANM 313
- Arithmetic operators 67
- Array**
 Alphanumeric 205
 Numeric 205
- Arrays 55
 Description 55
- Ask for Operator ID 90
- Assignment Statement 215
- Assignment Statements 56
- Automatic**
 Correction 150
 Edit Report 157
 Skip 74
- Automatically Generate Data Entry Forms 81
- Average Function 253
- B**
- Batch**
 Application Screen Layout 144
 Compile 123
 Logic View 144
 Message View 144
 Run an Edit Application 147
 Tree View 144
- Batch Edit**
 Applications 4
 Introduction 142
 Keyboard Summary 292
 Menu Summary 297
 Order 149
 Order or Executing Events 148
 Run Production Batch Edits 164
- Toolbar Summary 300
- Tree 9, 146
- Window 10
- BCH**
 Batch Edit Application 142
 Batch Edit Application File 313
 Batch Edit Applications 4
- Bell**
 Change Error Sound 95
- Blanks**
 Strip Function 251
- Box**
 Adding to a Form 99
 Change Field Box Size 95
 Statement 215
- Break Statement 209
- Breaking Off the Interview in CAPI 130
- By**
 Do Statement 210
- C**
- CAPI**
 Add Pictures in Question Text 135
 Breaking Off the Interview 130
 Change Formatting 134
 Coming Back Later 131
 Create a New Application 131
 Create Conditional Questions 136
 Create Standard Forms 134
 Display Questions Without Scrolling 137
 Enter Question Text 133
 Features 126
 Organization of the Instrument 129
 Use Multiple Language 136
 Using Multiple Languages 129
- CAPI Mode 90
- Case**
 Errmsg Function 262
 Skip Case Function 240
- Case ID**
 Identification Items 27
- Case tree**
 Show 90
- Cases and Levels 78
- Cells**
 Select in Table 196
- Center Text and Fields 102
- Change**
 Data Entry Options 90
 Default Text Font 94
 Edit Order 150
 Error Sound 95
 Field Box Size 95
 Field Font 95
 Field Properties 108

| | | | |
|---|----------|---|-----|
| Form Properties | 107 | Confirm End-of-Case | 90 |
| Forms File Properties..... | 106 | Consistency Checks | 4 |
| Language or Components | 288 | Consistency Edits | 116 |
| Level Properties | 107 | Controlling Program Flow | |
| Order of Entry..... | 90 | Endgroup Statement | 223 |
| Print Page Setup | 12 | Endlevel Statement | 223 |
| Roster Column Heading Properties .. | 111 | Convert | |
| Roster Occurrence Labels | 112 | Dictionary | 7 |
| Roster Properties | 110 | Items to Subitems..... | 49 |
| Row Heading Properties | 112 | Number to string..... | 246 |
| Table Title | 194 | Shape to Map | 7 |
| Tabulation Parameters..... | 194 | String to number..... | 252 |
| Text Properties..... | 113 | Converting | |
| View..... | 11 | A data dictionary..... | 302 |
| Windows..... | 11 | An IMPS Data Entry Application | 304 |
| Change Formatting in CAPI..... | 134 | An ISSA Data Entry Application | 305 |
| Changing Form File Properties..... | 111, 112 | Within IMPS..... | 303 |
| Changing Level Properties | 107 | Within ISSA | 304 |
| Changing Roster Properties | 110 | Copy | |
| Character | | Cells..... | 197 |
| Decimal | 31 | Cut, or Paste Things..... | 104 |
| Checking errors | 117 | Table..... | 197 |
| Choose Topic Sections..... | 140 | Table to Other Formats | 176 |
| Clear Function | 261 | Copy an Application..... | 14 |
| Close | | Correction | 152 |
| An Application | 14 | Count Function..... | 253 |
| Function | 270 | Create | |
| Cmcode Function..... | 241 | A Cross Tabulation | 179 |
| Code Edits of Individual Data Items | 168 | A CSPPro Application | 10 |
| Coding Standards | | A Dictionary for a New File..... | 36 |
| Define | 167 | A Dictionary for an Existing File | 37 |
| Cold Deck | 4 | A Frequency Distribution | 178 |
| Batch Edit Applications | 4 | A New Application | 10 |
| Static Imputation | 153 | A New Batch Edit Application..... | 142 |
| Color | | A New CAPI Application..... | 131 |
| Form..... | 107 | A New Cross Tabulation Application | 176 |
| Text | 113 | A New Data Entry Application | 79 |
| Column Variables | 179 | A Roster..... | 97 |
| Columns | | A Specialized Report..... | 158 |
| Join and Split in a Roster | 105 | A Table | 193 |
| Coming Back Later [CAPI]..... | 131 | A Thematic Map of Results | 190 |
| Comments | | An Area Names File | 188 |
| About..... | 62 | Conditional Questions in CAPI..... | 136 |
| In Data Dictionary..... | 21 | Logic..... | 59 |
| Common Uses of Cross Tab | 172 | Standard Forms in CAPI | 134 |
| Compare | | Thematic Maps..... | 2 |
| Data..... | 7 | Create Fills In Questions | 133 |
| Function | 245 | Create General Helps | 141 |
| Compile Logic..... | 61 | Create Helps for Fields | 138 |
| Compiler | | Creating | |
| Set Defaults..... | 60 | A Cross Tabulation | 179 |
| Concat Function..... | 245 | A data dictionary application | 36 |
| Concatenate | | A dictionary for a new file | 36 |
| Data..... | 7 | A Dictionary for an Existing File | 37 |
| CONCOR..... | 75 | Cross Tabulation..... | 179 |
| Conditions..... | 65 | Applications | 5 |

| | | | |
|----------------------------|-----|---------------------------------------|-----------|
| Create an Application..... | 176 | Designer Keyboard Summary | 289 |
| Creating..... | 179 | Elements..... | 77 |
| Creating a Table..... | 193 | Errors..... | 75 |
| Definition | 173 | Forms Screen Layout..... | 83 |
| Introduction | 169 | Installation | 287 |
| Keyboard Summary | 293 | Logic Screen Layout..... | 118 |
| Toolbar Summary..... | 301 | Menu Summary | 295 |
| CrossTab Keys | 293 | Order of Executing Events | 121 |
| CSBatch..... | 164 | Path | 76 |
| CSPro | | Run Production..... | 87 |
| Capabilities..... | 2 | Toolbar Summary..... | 299 |
| Data Requirements | 51 | Tree | 9, 85 |
| General Menu Summary | 293 | Window..... | 10 |
| Initial Screen Layout..... | 8 | Data Entry Methodogies | 76 |
| Introduction | 1 | Data Entry Operator Mode..... | 221 |
| Toolbar Summary..... | 298 | Add | 221 |
| Tools..... | 7 | Modify | 221 |
| What is | 1 | Verify | 221 |
| CSPro Application | | Data Entry Philosopies | |
| Create..... | 10 | Heads-Down Keying..... | 73 |
| CSPro Program Structure | | Heads-Up Keying | 73 |
| Declaration Section | 52 | Data file | 312 |
| Procedural Section..... | 52 | Data File | |
| Curocc Function..... | 254 | Size..... | 18 |
| Cut | | Type Structure..... | 18 |
| Copy, or Paste Things..... | 104 | Data File Organization | |
| | | About | 18 |
| | | Records | 23 |
| | | Data Items..... | 62 |
| | | Data Organization | 15 |
| | | Data Records | 23 |
| | | Data Type | 30 |
| | | Data Validation..... | 4 |
| | | DataEntryIDs..... | 87 |
| | | Date | |
| | | Sysdate Function..... | 268 |
| | | DCF | |
| | | Data Dictionary | 6, 36, 37 |
| | | In a Batch Edit Application | 142 |
| | | In a Cross Tabulation Application ... | 176 |
| | | In a Data Entry Application..... | 79 |
| | | Decimal | |
| | | Character..... | 31 |
| | | Places..... | 31 |
| | | Declaration Section..... | 52 |
| | | Default | |
| | | Special Values..... | 67 |
| | | Text Font | 94 |
| | | Define | |
| | | A Universe | 183 |
| | | Coding Standards..... | 167 |
| | | Dictionary Type | 42 |
| | | Languages in CAPI | 132 |
| | | Delcase Function | 271 |
| | | Delete | |
| | | A Table | 196 |

D

| | |
|------------------------------------|-----|
| Data | |
| Compare | 7 |
| Concatenate..... | 7 |
| CSPro Requirements | 51 |
| Export..... | 7 |
| Reformat | 7 |
| Sort..... | 7 |
| Data Dictionary | 6 |
| Creating for a New File | 36 |
| Creating for an Existing File..... | 37 |
| Introduction | 15 |
| Keyboard Summary | 289 |
| Labels..... | 21 |
| Levels..... | 21 |
| Menu Summary..... | 294 |
| Names..... | 21 |
| Screen layout | 38 |
| Toolbar Summary..... | 298 |
| Tree | 40 |
| Data Edit Application | |
| Creating..... | 142 |
| Data Entry | |
| Add Logic | 75 |
| Application File [.ENT]..... | 310 |
| Application Keys..... | 290 |
| Applications..... | 4 |
| Create a New Application..... | 79 |

| | |
|--|----------|
| Editing Results | 2 |
| Execsystem Function | 264 |
| Executable Statements..... | 56 |
| Exit Statement | 211 |
| Exp..... | 242 |
| Explicit Mode | 53 |
| Export | |
| Data..... | 7 |
| Export Statement..... | 240 |
| Expressions | 65 |
| External Dictionary..... | 42 |
| External Files | |
| About..... | 70 |
| Sharing..... | 70 |
| F | |
| Field | |
| Change Box Size | 95 |
| Change Font | 95 |
| Change Properties | 108 |
| Name..... | 21 |
| Field Properties..... | 100 |
| Mirror | 100 |
| Persistent | 100 |
| Protected | 100 |
| Sequential | 100 |
| Upper Case | 100 |
| Fields | 101, 128 |
| Add to a Form | 97 |
| Align | 102 |
| Description | 77 |
| File | 146 |
| Data Dictionary..... | 16 |
| Data Items | 16 |
| Identification | 16 |
| Inserting in an Application | 12 |
| Organization..... | 6, 16 |
| Program Information (.PFF) | 164 |
| Records..... | 16 |
| Types..... | 309 |
| Value Sets | 16 |
| File Statement..... | 203 |
| Fileconcat Function | 271 |
| Filecopy Function..... | 272 |
| Filecreate Function | 272 |
| Filedelete Function | 274 |
| Fileexist Function..... | 273 |
| Filename Function | 274 |
| Fileread Function..... | 275 |
| Filerename Function..... | 275 |
| Files | 54 |
| Area Names File .ANM | 313 |
| Batch Edit Application File .BCH..... | 313 |
| Cross Tabulation Application File .XTB | |
| | 312 |
| Data Dictionary File .DCF..... | 310 |
| Data Entry Application File .ENT..... | 310 |
| Edit Order File .ORD | 313 |
| Form File .FMF..... | 311 |
| Frequency File .FRQ..... | 314 |
| Frequency Specification File .FQF... | 314 |
| Listing File .LST..... | 312 |
| Logic File .APP..... | 311 |
| Map Data File .MDF | 314 |
| Map File .MAP..... | 314 |
| Message File .MGF | 312 |
| Operator Statistics File .LOG | 315 |
| Program Information File .PFF..... | 314 |
| Question File .QSF..... | 312 |
| Table Specifications File .XTS | 312 |
| Tables File .TBW | 313 |
| Filesize Function..... | 276 |
| Filewrite Function..... | 276 |
| Find | |
| Dictionary Elements | 48 |
| Find Function | 277 |
| Finding | |
| Errors..... | 150 |
| Flow of Program | |
| endgroup statement | 223 |
| endlevel statement | 223 |
| FMF | |
| Data Entry Applications | 4, 79 |
| Form File | 6, 311 |
| Font | |
| Change Field..... | 95 |
| Default Text | 94 |
| Field..... | 95 |
| Printer Font..... | 13 |
| Footer..... | 12 |
| Change Print Page Setup..... | 12 |
| For Statement | 212 |
| Force out-of-range | 90 |
| Form..... | 6 |
| Adding | 96 |
| Adding Fields..... | 97 |
| Adding Lines or Boxes | 99 |
| Adding Text | 99 |
| Delete Elements | 105 |
| File [.FMF] | 311 |
| Form Design Issues..... | 78 |
| Forms..... | 127 |
| Description | 77 |
| Design | 6 |
| Tree | 9, 85 |
| Window..... | 10 |
| FQF | |
| Frequency Specification File | 314 |
| Frequencies | 7 |
| Frequency Distribution..... | 173, 178 |
| FRQ | |

Frequency File314
 Full Screen..... 11
 FullScreen.....87
 Function
 Getbuffer247
 Getnote224
 Length29, 248
 Open.....279
 Statement.....208
 Writecase282
 Functions55
 Alphabetical Listing197
 Numeric.....241
 User Defined55

G

General Issues.....166
 General Menu Summary293
 Generate Default Data Entry Forms.....81
 Geographic
 Area Tabulation190
 Map Results by Areas176
 Processing187
 Get Help.....13
 Getbuffer Function247
 Getlabel Function.....265
 Getnote Function224
 GetOperatorId Function.....224
 Getsymbol Function.....266
 Global
 Arrays55
 Mode53
 Numeric Statement204
 Procedure.....52
 Set Statment202
 User Defined Functions.....55
 Variables54
 Guidelines for Correcting Data152

H

Handle Don t Know and Refused139
 Handle Multiple Answers139
 Handle Undefined Values.....181
 Hardware and Software Requirements ...284
 Header12
 Heads-Down Keying73
 Heads-Up Keying.....73
 Help13
 Support.....13
 Hierarchy6
 Dictionary20
 Highlighted Function225
 Hot Deck
 Batch Edit Applications4

Dynamic Imputation [Hot Deck]..... 154
 In External File 160
 Inline159
 Using 159

I

Identification 16
 Data Dictionary 16
 File 16
 Items..... 27
 Questionnaire 16
 IDs Dialog Box 190
 If and Only If
 Operator 67, 69
 Operator precedence 70
 If Statement..... 213
 Implications of Data Dictionary Value Names
 191
 Implicit Mode..... 53
 IMPS
 Converting a Data Dictionary 303
 Converting a Data Entry Application 304
 Imputation 153
 About 152
 Cold Deck 153
 Correcting Errors 152
 Dynamic, Hot Deck..... 154
 Hard-Coded 153
 In Batch Edit Applications..... 4
 Static..... 153
 Using Hot Decks..... 159
 Impute
 Freq File 147
 Function..... 218
 In
 If Statement 213
 Operator 68
 Operator Precedence 70
 Include Percents 179
 Inconsistencies 150
 Insert
 Data Item 48
 Dictionary Elements 48
 File in an Application 12
 Function..... 255
 Level 48
 Record..... 48
 Table..... 195
 Value 48
 Value Set..... 48
 Installing
 CSPro 284
 Data Entry Applications 287
 Newer Version..... 287
 Uninstalling CSPro 286

| | | | |
|--|--------|----------------------------------|---------|
| Int..... | 243 | Cross Tabulation | 293 |
| Interactive | 87 | Data Dictionary..... | 289 |
| Interactive Editing | 2 | Data Entry | 290 |
| Interpret Reports..... | 162 | Data Entry Designer | 289 |
| Introduction to... | | Keying | |
| Batch Editing | 142 | Heads-Down..... | 73 |
| CAPI..... | 126 | Heads-Up | 73 |
| Cross Tabulation | 169 | Skips Issues | 74 |
| CSPPro..... | 1 | Killfocus Statement | 226 |
| CSPPro Language..... | 50 | | |
| Data Dictionary..... | 15 | L | |
| Data Entry | 73 | Labels | |
| Data Entry Editing | 113 | Data Dictionary..... | 21 |
| Forms Design..... | 96 | Languages | |
| Invalueset Function | 266 | Define in CAPI..... | 132 |
| Ispartial Function | 225 | Layout | |
| ISSA | | Initial Screen..... | 8 |
| Converting a Data Dictionary | 304 | Screen | 144 |
| Converting a Data Entry Application | 305 | View Dictionary..... | 43 |
| Issues to Consider When Designing a Form | 78 | Length | |
| Item | | Function..... | 29, 248 |
| Adding | 44 | Level | |
| Convert to Subitem | 49 | Adding | 44 |
| Data Type..... | 28, 30 | Delete | 48 |
| Decimal Character | 28, 31 | Description | 21 |
| Decimal Places | 28, 31 | Edit Tree | 146 |
| Delete | 48 | Find..... | 48 |
| Description | 26 | Insert..... | 48 |
| Edit Tree..... | 146 | Label..... | 23 |
| Find | 48 | Modify..... | 44 |
| Insert | 48 | Name..... | 23 |
| Label..... | 21, 28 | Notes | 49 |
| Length | 28 | Properties | 23 |
| Modify..... | 44 | Search | 48 |
| Name..... | 21, 28 | Levels and Cases | 78 |
| Notes..... | 49 | Lines | |
| Occurrences..... | 28, 31 | Adding to a Form | 99 |
| Properties..... | 28 | List of reserved words..... | 201 |
| Search..... | 48 | Listing File..... | 147 |
| Select | 100 | Loadcase Function..... | 278 |
| Starting Position | 28, 29 | Locate Function | 279 |
| Type | 27, 28 | Lock | 87 |
| Zero Fill | 28, 32 | Log | 243 |
| Item with Multiple Occurrences | | LOG | |
| Tabulate | 192 | Operator Statistics File | 315 |
| | | Logic | 311 |
| J | | Batch Edit Applications..... | 4 |
| Join and Split Roster Columns | 105 | Editing..... | 148 |
| | | File .APP..... | 311 |
| K | | In Batch Edit Applications..... | 4 |
| Key Function..... | 277 | In Data Entry Applications | 4 |
| Keyboard Summary | | View..... | 59, 144 |
| Batch Edit..... | 292 | Logical | |
| | | Expressions | 65 |
| | | Operators | 67 |

Lookup Files71
 Using71
 LST
 Listing File312

M

Main Dictionary
 Type42
 Maketext Function248
 Manipulate Automatic Reports157
 Manipulate Data Files2
 Manual
 Correction.....150
 Skip74
 Map
 Create a Thematic Map.....190
 Data File .MDF314
 Map File314
 Results by Geographic Area176
 Viewer7
 Margins12
 Matching the Application to the Data
 Dictionary106
 Mathematical operators67
 Max Function256
 Maximum
 Number of Record.....26
 Record Properties24
 Maxocc Function.....257
 MDF
 Map Data File.....314
 Menu Summary
 Batch Editing297
 Cross Tabulation297
 Data Dictionary.....294
 Data Entry295
 General.....293
 Initial.....294
 Message File
 .MGF312
 In Batch Edit Applications4
 In Data Entry Applications.....4
 Message View144
 Methods of Correcting Data.....150
 MGF
 In a Batch Edit Application142
 In a Data Entry Application79
 Message File.....312
 Min Function258
 Mirror Fields.....100
 Missing
 Special Values67
 Mode53
 Absolute and Relative41
 Explicit53

Implicit 53
 Modify
 Demode Function 221
 Dictionary Elements 44
 Items..... 44, 46
 Levels 44, 45
 Records 44, 45
 Relations in Dictionary..... 50
 Table..... 196
 Value Sets 44
 Value Sets in Dictionary 46
 Values..... 44
 Values in Dictionary..... 46
 Modify or Add Dictionary Elements..... 44
 Move
 Around a Dictionary..... 43
 Around a Logic Application 120
 Dictionary Elements 48
 Things in a Form 101
 Move Statement..... 226
 Moving Around a Logic Application 120
 Multiple
 Occurrences, Tabulate..... 192
 Record Types 18
 Selection..... 47
 Tabulate Items..... 192
 Value Ranges..... 34
 Multiple records..... 78

N

Name of File
 Filename function 274
 Names
 Data Dictionary 21
 In Tree 11
 Navigating a Dictionary 43
 Next
 Skip Statement 238
 Statement 213
 Noccurs Function 258
 NoFileOpen..... 87
 Noinput Statement 227
 Not
 Applicable Value..... 67
 Operator 67
 Operator Precedence 70
 Notappl
 Special Values..... 67
 Notes
 Adding 21
 Document Dictionary Elements..... 49
 Number
 Convert to string 246
 Decimal Places..... 31
 Numbers..... 64

| | | |
|---|---------|--|
| Numeric | | |
| Array..... | 205 | |
| Expressions..... | 65 | |
| Item | 30 | |
| Statement..... | 204 | |
| Variables | 54 | |
| O | | |
| Occurrences | 78 | |
| Item | 31 | |
| Onfocus Event..... | 228 | |
| Onfocus Statement..... | 228 | |
| Onkey Global Function..... | 229 | |
| Onstop Global Function..... | 232 | |
| Open | | |
| An Existing Dictionary Application | 43 | |
| Function | 279 | |
| Operator | | |
| Ask for Operator ID | 90 | |
| Statistics File .LOG | 315 | |
| Vs System Controlled..... | 76 | |
| Operators..... | 67 | |
| And/Or Truth Table | 70 | |
| Arithmetic | 67 | |
| If and Only If..... | 69 | |
| In Expression | 68 | |
| Logical..... | 67 | |
| Precedence | 70 | |
| Relational | 67 | |
| Option | | |
| Set Explicit | 60 | |
| Set Implicit..... | 60 | |
| Or | | |
| Operator | 67 | |
| Operator Precedence..... | 70 | |
| Truth Table..... | 70 | |
| ORD | | |
| Batch Edit Application | 142 | |
| Order File | 313 | |
| Order | | |
| Batch Edit..... | 149 | |
| Executing Batch Edit Events..... | 148 | |
| Executing Data Entry Events | 121 | |
| File .ORD | 313 | |
| Organization | | |
| Data Dictionary..... | 6 | |
| Data File | 18 | |
| Data File Type Structure | 18 | |
| Questionnaire and Dictionary..... | 16 | |
| Organization of the Instrument in CAPI... | 129 | |
| Organize Forms..... | 133 | |
| Output File | 147 | |
| P | | |
| Pack an Application | 15 | |
| Parameter | 164 | |
| PFF | 164 | |
| Run | 87 | |
| Sysparm Function | 269 | |
| Parameters | | |
| Change Tabulation | 194 | |
| Partial Save | | |
| Allow | 90 | |
| Parts of a Table..... | 170 | |
| Paste | | |
| Copy or Cut Things | 104 | |
| Path..... | 76 | |
| Off..... | 76 | |
| On..... | 76 | |
| Percents | | |
| Change Tabulation Parameters | 194 | |
| Including in Tables | 179 | |
| Persistent Fields | 100 | |
| PFF | | |
| Parameter..... | 87, 164 | |
| Program Information File..... | 314 | |
| Run Production Batch Edits | 164 | |
| Run Production Data Entry..... | 87 | |
| Sysparm Function | 269 | |
| Pictures | | |
| Add in CAPI Question Text | 135 | |
| Pos function | 250 | |
| Poschar Function..... | 251 | |
| Position | | |
| Item..... | 29 | |
| Within a String..... | 250 | |
| Positioning | | |
| Relative or Absolute | 49 | |
| Postproc Statement | 58 | |
| Preproc Statement..... | 57 | |
| Preview | | |
| Printing | 13 | |
| Table..... | 185 | |
| Print | | |
| All or Part of a Document | 13 | |
| Change Font..... | 13 | |
| Change Page Setup | 12 | |
| Dictionary file | 50 | |
| Preview..... | 185 | |
| Tables..... | 185 | |
| Printer | | |
| Footer | 12 | |
| Header..... | 12 | |
| Margins..... | 12 | |
| Proc Statement | 57 | |
| Procedure Section | 52 | |
| Process Census or Survey Data..... | 2 | |
| Produce Tables by Geographic Area..... | 175 | |

| | | | |
|------------------------------------|-----|---|--------|
| Production | | Label..... | 21 |
| Begin Production Editing..... | 169 | Max..... | 24 |
| Run Production Batch Edits | 164 | Maximum Number | 26 |
| Run Production Data Entry | 87 | Modifying | 45 |
| Program Flow | 223 | Multiple Record Types..... | 18 |
| Program Information File | 72 | Name | 21 |
| Properties | | Notes | 49 |
| Change Field..... | 108 | Properties | 24 |
| Change Form | 107 | Required..... | 25 |
| Change Forms File..... | 106 | Single Record Type..... | 18 |
| Change Level..... | 107 | Type..... | 18, 25 |
| Change Roster | 110 | Type Value | 24 |
| Change Roster Column Heading | 111 | Redo and Undo Changes | 104 |
| Change Text..... | 113 | Reenter Statement..... | 234 |
| Field..... | 100 | Reformat | |
| Item | 28 | Data | 7 |
| Level..... | 23 | Relation Statement | 207 |
| Record..... | 24 | Relational operators..... | 67 |
| Value | 34 | Relations | |
| Value Set..... | 33 | Add and Modify | 50 |
| Protected Fields..... | 100 | Description | 35 |
| Putnote | 233 | Properties | 36 |
| | | Relative | |
| | | Mode..... | 41 |
| | | Positioning..... | 49 |
| | | Relocate | |
| | | Dictionary Elements | 48 |
| | | Remove | |
| | | Insert or Drop a File from an Application | 12 |
| | | Trailing blanks | 251 |
| | | Reorder Editing | 149 |
| | | Report | |
| | | Automatic | 162 |
| | | Required | |
| | | Enter Key..... | 90 |
| | | Hardware and Software..... | 284 |
| | | Record..... | 25 |
| | | Reserved | |
| | | Commands | 201 |
| | | Words..... | 201 |
| | | Resize and Reposition Things in a Roster | 104 |
| | | Column Order..... | 104 |
| | | Column Width..... | 104 |
| | | Roster Size..... | 104 |
| | | Row Height..... | 104 |
| | | Restrict a Universe..... | 175 |
| | | Re-Test with Live Data | 169 |
| | | Retrieve | |
| | | Function..... | 280 |
| | | Tables..... | 7 |
| | | Review Edit Specifications..... | 166 |
| | | Right Side of Screen | 10 |
| | | Roster | |
| | | Add to a Form..... | 97 |

Q

| | |
|---|-----|
| QSF | |
| In a Data Entry Application | 79 |
| Question File | 312 |
| Question | |
| File .QSF | 312 |
| Questionnaire | |
| Form..... | 16 |
| Identification | 16 |
| Identification Items | 27 |
| Organization..... | 16 |
| Questions | 16 |
| Responses | 16 |
| Section | 16 |
| Questionnaire and Dictionary Organization | 16 |
| Questions..... | 128 |

R

| | |
|-------------------------------------|-----|
| Random | 243 |
| Ranges | |
| Multiple Value..... | 34 |
| Rearrange Things in a Form..... | 101 |
| Recode Statement..... | 215 |
| Reconciling Dictionary Changes..... | 42 |
| Record | 146 |
| Adding | 45 |
| Delete..... | 48 |
| Description | 23 |
| Find | 48 |
| Insert | 48 |

| | |
|--|---------|
| Adding Things | 98 |
| Change Column Heading Properties | 111 |
| Change Occurrence Labels | 112 |
| Change Properties | 110 |
| Create..... | 97 |
| Description | 77 |
| Join and Split Columns | 105 |
| Resize and Reposition Things | 104 |
| Row Variables..... | 179 |
| Run | 124 |
| A Batch Edit Application..... | 147 |
| A CSPro Tool | 7 |
| A Data Entry Application | 86 |
| A Tabulation | 186 |
| As Batch..... | 124 |
| Data Entry Application as Batch | 124 |
| Parameter | 87, 164 |
| Production Batch Edits..... | 164 |
| Production Data Entry..... | 87 |
| Sysparm Function | 269 |
| S | |
| Save | |
| An application..... | 13 |
| Dictionary As New File | 50 |
| Tables..... | 185 |
| Tabulations in Different Formats..... | 175 |
| Savepartial Function | 235 |
| Screen | |
| Full..... | 11 |
| Screen Layout | |
| Batch Application | 144 |
| Data Dictionary..... | 38 |
| Data Entry Forms | 83 |
| Data Entry Logic..... | 118 |
| Initial CSPro Application..... | 8 |
| Screens..... | 6 |
| Search Dictionary Elements | 48 |
| Seed Function..... | 244 |
| Selcase Function | 235 |
| Select | |
| Dictionary Elements | 47 |
| Items in a Form | 100 |
| Relative or Absolute Positioning | 49 |
| Table Cells | 196 |
| Sequence | |
| Dictated by CSPro..... | 121 |
| Dictated by Designer..... | 123 |
| Sequential Fields | 100 |
| Set | |
| Attributes Statement..... | 236 |
| Compiler Defaults..... | 60 |
| Explicit | 202 |
| Implicit | 202 |
| Statement..... | 202 |
| System Settings | 60 |
| Set Behavior Export Statement | 240 |
| Set Behavior Statement..... | 237 |
| Setfile Function | 281 |
| Setup..... | 287 |
| Sharing External Files..... | 70 |
| Show Case Tree | 90 |
| Show Values for Selection..... | 138 |
| Single Record Types | 18 |
| Size | |
| Change Field Box..... | 95 |
| Size of Data Files..... | 18 |
| skip | |
| Skip Case Statement..... | 240 |
| Skip | |
| Automatic | 74 |
| Issues | 74 |
| Manual..... | 74 |
| Manual Skip To | 108 |
| Skip Statement | 238 |
| Soccurs Function | 259 |
| Sort | |
| Function..... | 259 |
| Sort Data | 7 |
| Sound | |
| Change | 95 |
| Special | |
| Create a Specialized Report | 158 |
| Function..... | 267 |
| Output Dictionary..... | 42 |
| Values..... | 67 |
| Specific | |
| Impute Function..... | 218 |
| Sqrt | 244 |
| Starting Position of Item..... | 29 |
| StartMode | 87 |
| Stat | |
| Impute Function..... | 218 |
| Statement Format Symbols | 200 |
| Statements..... | 56 |
| Alphabetical Listing | 197 |
| Assignment..... | 56 |
| Executable..... | 56 |
| Static Imputation | 4 |
| Stop Function..... | 267 |
| Strategies | |
| CONCOR..... | 75 |
| String | |
| Expressions | 65 |
| String to Number | 252 |
| Substring Expressions..... | 66 |
| Text..... | 64 |
| Variables..... | 54 |
| Strip function | 251 |
| Structure Edits | 4, 115 |
| Structure Movement..... | 137 |

Subitems27
 Convert to Item49
 Item Type27
 Sub-Items27
 Subscripted Variables63
 Substring Expressions66
 Sum Function260
 Summary
 Batch Edit Keyboard292
 Batch Editing Menu297
 Batch Editing Toolbar300
 Cross Tabulation Keyboard293
 Cross Tabulation Menu297
 Cross Tabulation Toolbar301
 CSPro Toolbar298
 Data Dictionary Keyboard289
 Data Dictionary Menu294
 Data Dictionary Toolbar298
 Data Entry Menu295
 Data Entry Designer Keyboard289
 Data Entry Forms Designer Toolbar299
 Data Entry Keyboard290
 Errmsg Function262
 Generalized Menu293
 Initial Menu294
 Support13
 Switching Between Data Entry Forms and Dictionary83
 Sysdate Function268
 Sysparm Function269
 System
 Vs Operator Controlled76
 Systime Function270

T

Table
 Add195
 Change Title194
 Delete196
 Insert195
 Modify196
 Parts170
 Preview185
 Print185
 Retrieval7
 Save185
 Select Cells196
 Tree9
 Viewer7
 Window10
 Tabulate
 By Geographic Area190
 Counts or Percents174
 Data2
 Frequencies7

Items with Multiple Occurrences 192
 Percents 179, 194
 Tool List 7
 Undefined Values 181, 194
 Values and/or Weights 174
 Values, About 174
 Values, Changing 194
 Weights, About 174
 Weights, Changing 194
 Tabulation
 Change Parameters 194
 Creating a Cross Tabulation 179
 Cross Tabulation Application File .XTB 312
 Cross Tabulation Keyboard Summary 293
 Cross Tabulation Menu Summary 297
 Cross Tabulation Toolbar Summary 301
 Cross Tabulations 173, 193
 Frequency Distribution 178
 Introduction to CrossTabs 169
 Run 186
 Save in Different Formats 175
 Universe 183
 Tabulation Application
 About 5
 Creating 176
 TBW
 Tables File 313
 Test
 Application 141
 CSPro Program 168
 Text
 Adding to a Form 99
 Align 102
 Change Properties 113
 Default Font 94
 Strings 64
 Tool List 7
 Viewer 7
 The Nature of Census and Survey Data. 305
 Thematic Map
 Create 190
 Then
 If Statement 213
 This Item (\$) 62
 Tile
 Side by Side 11
 Top to Bottom 11
 Windows 11
 Time
 Systime Function 270
 Title
 Impute Function 218
 To
 Advance Statement 221

| | |
|---------------------------|-----|
| Change..... | 11 |
| Delete a Table..... | 196 |
| Dictionary Layout..... | 43 |
| Full Screen..... | 11 |
| Logic..... | 59 |
| Modify a Table..... | 196 |
| Names in Tree..... | 11 |
| Print Tables..... | 185 |
| Visualvalue Function..... | 239 |
| VSet | |
| Impute Function..... | 218 |

W

| | |
|-------------------------------------|-----|
| Weights | |
| Change Tabulation Parameters..... | 194 |
| Include Values and Weights..... | 183 |
| Tabulate Values and/or Weights..... | 174 |
| Tabulation..... | 178 |
| What is CSPro..... | 1 |
| Where | |
| Average Function..... | 253 |
| Count Function..... | 253 |
| Max Function..... | 256 |
| Min Function..... | 258 |
| Sum Function..... | 260 |
| While Statement..... | 214 |
| Window..... | 83 |
| Windows..... | 10 |

| | |
|----------------------------------|-----|
| Change..... | 11 |
| Tile Side by Side..... | 11 |
| Tile Top to Bottom..... | 11 |
| Working Dictionary..... | 42 |
| Working Storage File..... | 72 |
| Write..... | 147 |
| File..... | 147 |
| Function..... | 283 |
| Run an Application..... | 147 |
| Specialized Edit Report..... | 158 |
| Writecase Function..... | 282 |
| WRT..... | 158 |
| Create a Specialized Report..... | 158 |

X

| | |
|--|-----|
| XTB | |
| Cross Tabulation Application..... | 176 |
| Cross Tabulation Application File..... | 312 |
| Cross Tabulation Applications..... | 5 |
| XTS | |
| Cross Tabulation Application..... | 176 |
| Cross Tabulation Applications..... | 5 |
| Table Specifications File..... | 312 |

Z

| | |
|----------------|----|
| Zero Fill..... | 32 |
|----------------|----|