CSPro

User's Guide

Version 2.4

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

17 December 2003

Table of Contents

CSPro Users Guide

Basic Concepts

Introduction to CSPro

CSPro is a tool for entering, editing, and tabulating data from surveys and censuses. It uses data dictionaries to provide a common description of each data file used.

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 Concepts
- Data Dictionaries
- Data Entry applications
- Batch Edit applications
- Cross Tabulation applications
- Trees and Windows

How to ...
- Create a new application
- Create a new Data Dictionary
- Create a new Data Entry application
- Create a new Batch Edit application
- Create a new Cross Tabulation application

- Open an existing application
- Move around applications
- Save an application
- Close an application

- Insert a file into an application
- Drop a file from an application
- Define dictionary type

- Print all or part of a file
- Change the print page setup
- Run a CSPro tool

- Change the view
- Change windows
- Get Help

Summaries
- Menu
- Toolbar

CSPro Concepts

Data Dictionaries

A Data Dictionary describes the organization of a data file. This description is used by all other modules of **CSPro** in order to access and use the data file for which this description applies.

A Data Dictionary allows you to give text labels for all levels, records, items, and value sets in the file. It allows you to describe how each kind of record in the file is organized, how the records are organized into questionnaires, and the characteristics of each item in the record.

You will need to a create a data dictionary for each type of data file you want to use in CSPro.

A Data Dictionary is used to give a description or picture of how data are (or will be) stored in the computer. It allows you to provide meaningful names for the data items and to define characteristics such as whether the data item is made up of numbers or letters, how many characters or digits there are in a data item, and whether a data item has an assumed decimal point. The Data Dictionary also allows you to define the overall structure of a data file.

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

The Data Dictionary facilitates communication. It provides a means of documenting a file description in a single place. Anyone needing information about any aspect of a data file can find it in the Data Dictionary.

Data Entry Applications

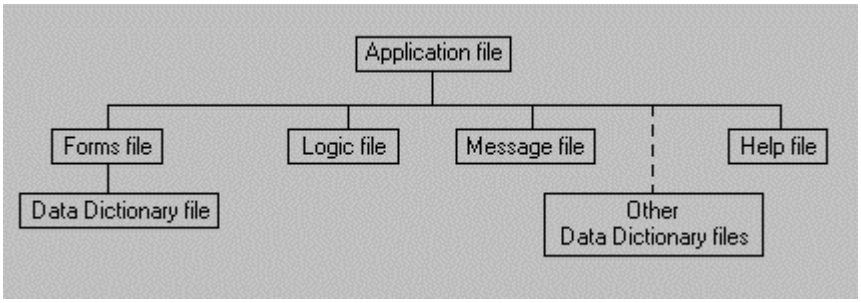
A **Data Entry** application contains a set of forms (screens) and logic which a data entry operator uses to key data to a disk file. Data entry applications can be used to add new data and to modify existing data.

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

- Add new cases (questionnaires) or retrieve and modify existing cases
- Unlimited number of forms (screens)
- 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
- 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.

Data entry applications consist of the following files:



- Application file (.ent). This specifies all other files contained in the application and includes other application information.
- Forms 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, contains text for messages displayed during data entry.
- Help file (.hpf), optional, contains text for help screens displayed during data entry. Use of this file in the data entry application is not yet supported.
- Other Data Dictionary files (.dcf), optional, represent secondary data files (such as lookup files) which are read and/or written to during data entry.

See also: Creating a New Data Entry Application

Batch Editing 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

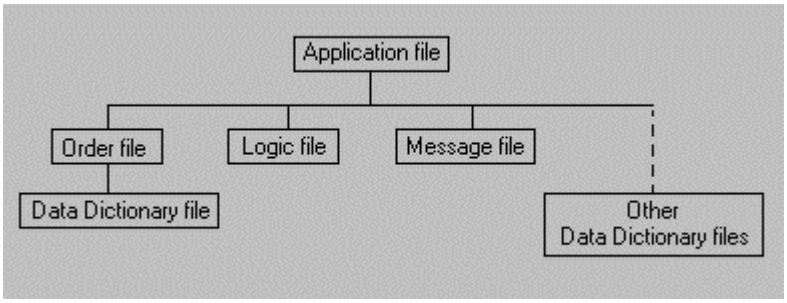
A **Data Entry** application contains a set of forms (screens) and logic which a data entry operator uses to key data to a disk file. Data entry applications can be used to add new data and to modify existing data.

You can have the following run-time features in your 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.

Batch edit applications consist of the following files:



- Application file (.bch) specifies all other files contained in the application and includes other application information.
- 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) which represents the primary data file that is being read and/or written.
- Logic file (.app) contains CSPro language statements.
- Message file (.mgf), optional, contains text for messages displayed on the output listing.
- Other Data Dictionary files (.dcf), optional, represent secondary data files (such as lookup files) which are read and/or written to during the batch run.

See also: Creating a New Batch Edit Application

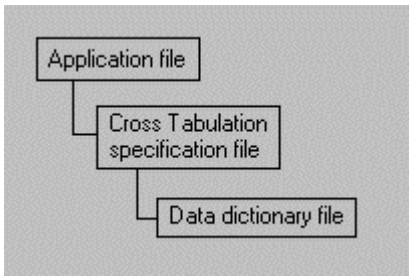
Cross Tabulation Applications

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.

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.

See also: Creating a New Cross Tabulation Application

Trees and Windows

Each time you open or create an application or a file, it is added to the **Files tree**. When you close the application or file, it is removed from the Files tree. If you have an application open, the Files tree will show you the files included in the application and their relationships with one another.

You may Insert or Drop a file from a data entry application. This is necessary if you plan to use Look-up files or Multiple forms files.

Tips
- Use **Ctrl+T** to see the full file names of the files you have open.
- Double-click on the Files tree to switch the frame on the right side of the screen.

Reconciling Dictionary Changes

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

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

To rename the item, select rename and then choose the new item name from the list presented.

To delete the item, select delete.

How to ...

Create applications or files

Create a New Application

1 Click  on the toolbar; or from the **File** menu, select **New**; or press **Ctrl+N**.
2 Select the type of application or file you want to create.
3 In the panels which follow, enter the names of the files you want to create.

Name Object

1 Select the type of application or file you want to create:

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

2 Enter the name of the file you want to create. The file name must not already exist.

3 Select the folder where the object is to be stored. You can press the **Browse** button to locate a folder.

Select Form 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.

Select 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.

Summary
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.

Create a New Data Dictionary

Follow the steps below to create a new dictionary:

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

2 Select  "Data Dictionary File" from the **Object:** listing.

3 Provide a name for the new dictionary—note you need not provide the dictionary extension (.dcf), it will be automatically appended to the name.

4 Press **Next>** to advance to the Summary Screen and review your choices.

5 If everything looks ok, press **Finish** to complete the operation.

Tips
- If you are creating a dictionary to describe an existing data file, you may want to use absolute positioning, in the event there are any "holes" in the data file(s). Or, if you only want to use 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.

Create a New Data Entry Application

To perform data entry, 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 data entry application. If you already have a CSPro data dictionary for the file, you can specify this dictionary when you create a new data entry application.

1 If you have an IMPS or ISSA data dictionary, convert it to CSPro.

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

3 Select **Data Entry Application**.

4 Enter the file name for the application.

5 Enter the name of, or select, the folder where the application will be stored. Click on **Next**.

6 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**.

7 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**.

8 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.

9 If you are using an existing CSPro data dictionary, 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.

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 data entry application. If you already have a CSPro data dictionary for the file, you can specify this dictionary when you create a new data entry application.

1 If you have an IMPS or ISSA data dictionary, convert it to CSPro.

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

3 Select **Batch Edit Application**.

4 Enter the file name for the application.

5 Enter the name of, or select, the folder where the application will be stored. Click on **Next**.

6 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**.

7 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.

8 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 forms.

Create a New Cross 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, you should convert it to a CSPro data dictionary before you create a new Cross Tabulation application. If you already have a CSPro data dictionary for the file, you can specify this dictionary when you create the new Cross Tabulation application.

1 If you have an IMPS or ISSA data dictionary, convert it to CSPro.

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

3 Select **CrossTab Application**.

4 Enter the file name for the application.

5 Enter the name of, or select, the folder where the application will be stored. Press **Next**.

6 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**.

7 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.

8 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.

Open applications or files

Open an Existing Application

1 Click  on the toolbar; or from the **File** menu, select **Open**; or press **Ctrl+O**.
2 Select from the **Files of type:** at the bottom of the dialog box. In CSPro you can open either an application or a file.
3 Select the name of the file you want to open.

Notes:

- The application or file will be added to the Files tree

- Any other files belonging to the application or file will also be opened and added to the appropriate trees.

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

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

- Any changes you make to applications and files are not made permanent until you save what you opened.

Move around applications

Moving Around Applications

CSPro shows **trees** on the left side of the screen and **windows** on the right side.

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.

Trees

The **Files** tree is always present. This tree shows you what applications are currently open. If an application is open, the Files tree shows all the files that belong to that application.

CSPro always shows a tree on the left side of the screen. There are four kinds of trees in CSPro:

- **Files** tree shows all the applications that are open, and the files they contain.
- **Dict**ionaries tree shows all the dictionaries that are open, and their contents.
- **Forms** tree shows all the form specifications that are open, and their forms and fields.
- **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.

Save and close applications

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.

1 Select the file or files you wish to close or save.
2 Click on **OK**.

Tips
- 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 Application or File, CSPro applications and files, Data Entry Application Files, Cross Tabulation Application Files

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.

1 Select the file or files you wish to close or save.
2 Click on **OK**.

Tips
- 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 Application or File, CSPro applications and files, Data Entry Application Files, Cross Tabulation Application Files

Insert or drop files from applications

Insert a File in an Application

1 Click on the **Files** tab to bring up the Files tree.
2 From the **File** menu, select **Insert File** or right click on the application file and select **Insert File**.
3 Select the type of file to be inserted.
4 Select the name of the file to be inserted.

Note: 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.

Drop a File from an Application

1 Click on the **Files** tab to bring up the Files tree.
2 From the **File** menu, select **Drop**.
3 Select the type of file to be dropped.
4 Select the name of the file to be dropped.

Define Dictionary Type

Every dictionary associated with an application has a type value which 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 can not 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.

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

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.

Print

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 …

1 Select the text you want to print.
2 Click  on the toolbar; or from the **File** menu, select **Print**; or press **Ctrl+P**.

To change printer font size …

1 Click  on the toolbar; from the **Options** menu, select **Printer Font Size**.
2 Select the font size from the dialog box.

Tip

- 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**.

Change the Print Page Setup

To change the page headers, footers, or margins …

1 Click  on the toolbar or from the **File** menu, select **Page Setup**.
2 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.

Tip

- The headers, footers, and margins you specify will remain in effect until you change them.

To change the page orientation or size …

3 From the **File** menu, select **Print Setup**.
4 In the print setup dialog box make changes to orientation (portrait or landscape) and paper size.

Run CSPro tools

Run a CSPro Tool

From the **Tools** menu, select one of the tools listed below you want to run.

Text Viewer
This tool allows you to examine, but not change, any text file. The file can be of any length and may contain individual records up to 32,000 characters wide.

Table Viewer
This tool allows you to examine, but not change, the contents of any CSPro tables file. A tables file (extension **.tbw**) contains the results of CSPro tabulations. You can also copy, save, or print all or parts of the tables in the file.

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.

Retrieve Tables
This tool allows you retrieve tables, maps, and other documents from a database of files. 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.

Sort Data
This tool allows you to sort a data file by questionnaires using the identification fields.

Export Data
This tool allows you to export 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
This tool allows you to reformat data 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
This tool allows you to 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
This tool allows you to concatenate (that is, 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.

Convert Dictionary
This tool converts ISSA or IMPS data dictionaries to CSPro data dictionaries. It converts both IMPS 3.1 and IMPS 4.1 data dictionaries. It also converts ISSA data dictionaries to CSPro data dictionaries and data entry form files.

Convert Shape to Map
This tools converts 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.

Change view and windows

Change the View

Names in Tree

To toggle between labels and names in trees …

From 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.

Full Screen

To toggle between trees on left and full screen …

From 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.

Get help

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

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**.

Summaries

CSPro Menu Summary

The CSPro menu is displayed across the top of the window. It appears only when CSPro is opened without any applications or files open. When applications are open, the menu corresponding to the contents of the right-hand window appears.

Files

New	Create a new application.

Open	Open an existing application.
Tools	
Text Viewer	View text or data files.
Table Viewer	View CSPro tables.
Map Viewer	View CSPro thematic maps.
Retrieve Tables	Retrieve tables from a data set.
Tabulate Frequencies	Tabulate frequency distributions for file contents.
Sort Data	Sort cases 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.
Help	
Help Topics	Get help on current application.
About	Get information about the software.

CSPro Toolbar Summary

The CSPro toolbar is displayed across the top of the window, below the menu bar. It provides quick mouse access to many features used in CSPro.

Click	To
	Create a new application.
	Open an existing application.
	Get help.

The CSPro toolbar only appears when CSPro is opened without specifying an application or file. When applications or files are open, the toolbar corresponding to the contents of the right-hand screen appears.

Data Dictionary

Introduction to Data Dictionaries

The Data Dictionary module allows you to describe the structure of data files that are used by other parts of **CSPro**. It also describes your census or survey questionnaire.

This section contains the following information:

Data Dictionary Concepts
- Dictionaries
- Questionnaires and File Organization
- Levels
- Records
- Items

Dictionary Concepts

Dictionary

Dictionaries

A Data Dictionary gives a description of how data are stored in a file. It allows you to define:

- The overall structure of a data file.
- Meaningful names for records, items, and values.
- Position in the data record.
- The type of data in a item (numbers or text).
- Length of an item.
- The number decimal places.
- Valid values or ranges of values.
- Other documentation.

This description is used by other modules of CSPro to correctly read and write data to files.

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.
- Labels can be up to 255 characters long.

Tips

- 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 must be a letter. The last character cannot be an underline (_).
- Names can be 1 to 32 characters long.
- Names cannot be CSPro reserved words.
- Names cannot be duplicated within a dictionary. However, the same name can be used in different dictionaries.

Examples:
SEX
RELATIONSHIP
MOTHER_ALIVE
Q102_AGE_CHILD

Tips

- 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 dictionary and any of its elements can have notes.

Notes can contain any printable character and spaces.
Notes can be up to 65,000 characters long.

To add, modify, or delete notes …

Select the dictionary element in the dictionary window that contains or will contain the note.

Press  , or from the **Edit** menu, select **Notes**, or press **Ctrl+D**.

Tips

- 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.

Questionnaires and File Organization

Questionnaires

A questionnaire is a set of questions relating to the same unit of observation (such as a household, person, or factory). The body of a questionnaire is often divided into sections, with each section asking a related set of questions. In the dictionary these sections would normally be grouped into a record. The following are sample questionnaire structures.

Example 1: In a typical housing and population census, a questionnaire would contain the following records:

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

Normally there would be one or more population records, 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 records.

Example 2: 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

Example 3: Finally, 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

In the Data Dictionary, records with the same questionnaire identification codes (i.e., Questionnaire Ids) constitute a questionnaire. In a data file, if a group of questionnaire Ids uniquely identify the unit under observation, then those records make up one questionnaire.

Note that in some cases, one record constitutes a questionnaire. For example, a student roster might consist of a record for each student. The student identification number could serve as the questionnaire identification. The type of data file produced from this dictionary is known as a flat data file.

Data File Organization

All data files used by **CSPro** 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 out as an ASCII text file before you can use it with CSPro. Data files are limited to 2 gigabytes in length.

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.

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**.

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 would contain the following records:

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

Therefore for a given questionnaire there would be one or more population records for one household record. A sample (and recommended) file structure could be as follows (not all fields are defined for this example):

```
110100011122122
210100112010919613 8
210100121210519673 2
21010013117071992 07
11010002111112 1
21010021107161930 69
21010022208121928 71
```

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.

Levels

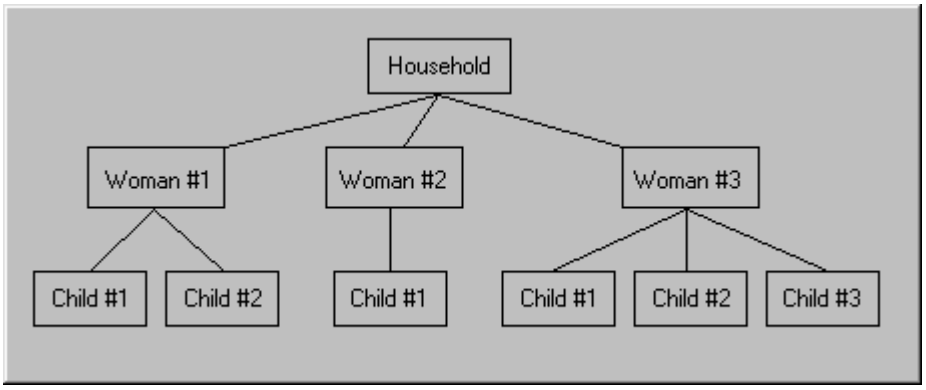
Levels

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 (note that the maximum number of levels allowed in a dictionary 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 therefore the level), the keyer will advance to the final level, and enter information for a child (if any). After entering data for child #1 (and thereby completing Level 3), the Child Form will reappear, waiting for entry for the next child.

If there are none, 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 there are implications when using more than one level, with respect to the order of executing logic in a data entry application or in a batch edit application.

Identification Items

Identification items (i.e., ID items) are those data items that uniquely identify the questionnaire. These data will appear on every record in a data file, as they are "common" to all of the records. Quite often such items are identification or geographical items, such as **Province**, **District**, or **Survey ID Number**. If you open a dictionary in CSPro and press **Ctrl+L** (L=layout), you will see a nice pictorial representation of this (press **Ctrl+L** again to toggle the view off).

If you are using absolute positioning, you will typically want to structure your dictionary such that the Id items begin in column 2, so that they will precede a record's data items (column 1 is usually reserved for the record type identifier). If you are using relative positioning, you have no choice but to place the Id Items first.

See also: Item Properties

Level Properties

Level properties are visible when the dictionary has been selected in the tree tab. To modify any of the following properties, select the level you want to modify in the view and press **Ctrl+M**.

Property	Meaning
Label	A descriptive text label to identify this level.
Name	The name of this level for use in the CSPro language procedures.

Tips
- You can press the **Esc** key to abandon modification. No changes will be saved.
- You can always undo a modification if you decide it was incorrect.

Records

Records

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.

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

Therefore, 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
```

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, you need to be able to identify one record from another in the data file. Record Type provides the method to do this. For example, a census data file would most likely have a housing record (describing details of the home) 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:

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

6 In the view on the right, you'll notice the first line is (record type). Only three values are used, **Starting_Position**, **Len**gth, 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.

Tips
- Upper- and lowercase letters are distinct Record Type values (i.e., 'A' is not the same as 'a').
- Each record **must** have a unique identifying alphanumeric symbol.
- Blank is a valid Record Type value.

Record Properties

You can view a record's properties by selecting the questionnaire to which it belongs (via the dictionary tree tab). To modify any of the following properties, select the record in the view and press **Ctrl+M**.

Attribute	Meaning
Label	A descriptive text label to identify this record.
Name	The name of 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.

Tips

- You can press the **Esc** key to quit modifying without making changes.
- Use undo if you completed the modification incorrectly.

See also: Records, Record Type

Record Type Value

If a record type is required for data file, each different type of record must have a unique record type value. The record type value is always alphanumeric. For a typical housing and population census, there are usually two types of records, a housing record and population record. The housing records might have a record type value 1 while population records have a record type value of 2. This value tells what kind of information is contained on the record.

See also: Record Type, Record Properties, Label, Name, Max

Required

One of the Record properties is whether or not the record is required. This means for a given questionnaire, must there be at least one occurrence of the given record (in which case it's required), or can you have a complete questionnaire with no occurrences of the given record (in which case it's not required).

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 the persons in that household. You can have two scenarios:

- If you allow vacant housing units (i.e., you collect information on empty 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.

The Record is either Required (**Yes**) or not (**No**).

See also: Label, Name, Type Value, Max.

Maximum Number

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

For example, 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 the persons in that household. There should be only one occurrence of the household record, but for the person record you'll of course need more than one occurrence, as there will likely be more than one person in a household. So your 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 a record can have is 9,999, though we would not recommend that you ever have this many.

See also: Record Type, Record Label, Record Name, Type Value, Required.

Items

Record Items

A data item describes 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).

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.

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 in 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

Tip

- The use of subitems to provide multiple groups of values (for example, "Age by 10 years" and "Age by 5 years") isn't necessary in CSPro. Value sets allow you to have as many value definitions for each item as you need.

See Also: Item Properties

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:

Attibute	Meaning
Label	A descriptive text label for the item. Used as default field text in data entry forms and in default titles in cross tabulation.
Name	The name of this item for use in CSPro procedures.
Start	The starting position of the item within the record.
Len	The length of the data item (i.e., the number of characters necessary to represent the item).
Data Type	The data type of the item, either numeric or alphanumeric.
Item Type	Whether it is a "regular" item or a subitem (i.e., part of a larger item).
Occ	The number of times this item will repeat within the record.
Dec	The number of decimal places (if any) in the item.
Dec Char	Should the item be stored in the data file with a decimal character? (Yes/No)
Zero Fill	Should the item be stored in the data file with leading zeros? (Yes/No)

Tips

- 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.

Starting Position

This item property indicates the starting location of a data item. In conjunction with the length property, it specifies the location of the item in a record.

- In absolute positioning mode, you cannot give a starting position that will cause the item to overlap with another item.
- The start position of a subitem must be within its parent item (the previous item).

See also: Relative vs. Absolute positioning. Other item properties Label, Name, Length, Data Type, Item Type, Occ, Dec, Dec Char, Zero Fill.

Length

This item property indicates the total length of a data item. In conjunction with the **start** property, it specifies the location of the item in a record.

- In absolute position mode, you can not give a length that will cause the item to overlap with another item.
- The maximum length of a **Numeric** item is 15 digits.
- The maximum length of an **Alpha** item is 255 characters.

See also: Other item properties Label, Name, Start, Data Type, Item Type, Occ, Dec, Dec Char, Zero Fill.

Data Type

This item property specifies what type of data will be expected during data entry for the dictionary item. The possible data types are **Num**eric or **Alpha**numeric.

- **Num**eric items can contain numbers or blanks. Numeric values will be right-justified and, if requested, zero-filled.

- **Alpha**numeric items can contain any character, letter, or number. These values will be left-justified and are blank-filled, whether or not zero-fill has been selected.

Some responses are quantitative, such as size of farm, and some are qualitative, such as relationship to head of household. Responses can be numeric or alphanumeric. 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.

Thus, 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. A discrete value may be used to designate a quantity category. 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.

An alphanumeric value consists of alphabetic and numeric characters, blanks, and special characters. For example, 'M' or 'F' for gender is an alphanumeric value.

See also: Other item properties Item Label, Item Name, Start, Len, Item type, Occ, Dec, Dec Char.

Item Type

This item property specifies whether the data is an **Item** or a **Subitem** (i.e., a redefinition of a portion of an item). **Item** will be your most common choice for a data item.

Example:

Item Label	Item Type	Start	Len
Date of birth	Item	19	8
Day of birth	Subitem	19	2
Month of birth	Subitem	21	2
Year of birth	Subitem	23	4

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.

Tip

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

See also: Other item properties Item Label, Item Name, Start, Len, Data type, Dec, Dec Char, Zero Fill.

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 its number of occurrences.

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 wide for this item.

Realize 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 select the **occurrence** size 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.

See also: Other item properties Item Label, Item Name, Start, Data type, Item type, Dec, Dec Char, Zero Fill.

Decimal Places

This item property lets you specify how many digits of the numeric item represent a 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.

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")

See also: Other item properties such as Item Label, Item Name, Start, Length, Item type, Occ, Zero Fill.

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. Therefore 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 data type of numeric, the Data Dictionary will not allow any value other than **No.**

See also: Other item properties such as Item Label, Item Name, Start, Len, Item type, Occ, Zero Fill.

Zero Fill

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

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'`

See also: Other item properties such as Item Label, Item Name, Start, Item type, Occ, Dec, Dec Char.

Values

Value Sets

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 (which is the 5th item in the person record and the reason why we have prefaced the names with "P05_") with the following properties:

Value Set Label	Value Set Name	Value Label	From	To
Age	P05_AGE_V1			
			0	98
		Not Reported	99	
Age by 5 years	P05_AGE_V2			
		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	P05_AGE_V3			
		Infant	0	0
		Child	1	12
		Teenager	13	19
		Adult	20	59
		Senior	60	98

The AGE item now has three defined value sets: P05_AGE_V1, P05_AGE_V2, and P05_AGE_V3. 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.

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.

Attibute	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.

Values

A single value set can contain one or more values. The following properties are available to describe these values.

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	Value or starting value of a range.
To	Ending value of a range.

Special What type of special value this is (blank if there is no special value; otherwise, you may choose among Missing, NotAppl(icable), and Default).

Multiple ranges

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.

Strategies

Creating a Dictionary for a New File

To begin, you need to create a new data dictionary. When finished, **CSPro** will have generated a new dictionary (named **MyDict** here) with the following structure:

So what is this? CSPro created a dictionary ("MyDict") with one level ("Questionnaire"), and that level contains a set of ID Items ("(Id Items)") and one record ("Record").

The first thing we suggest you do is change the level properties (i.e., the label and/or name) to reflect your intended usage for them. Next, change the record properties; for example, if this is an agricultural survey, you might want to call your record 'Crop'. Note that you **cannot** change the label or name of the (Id Items) set.

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

However, if you need additional records for this level, you should create them first. To do so, select the Crop record and press **Ctrl+A**. Provide the required record properties and continue as desired. To terminate the 'add record' mode, press <Esc> when you receive a new (blank) record entry.

If you need to add additional levels (recommended only for more complex censuses and surveys), you can do this by selecting the 'Questionnaire' level and pressing **Ctrl+A**. After

entering the second level, the 'add level' mode will continue for one additional level (you are allowed a maximum of three levels). 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 are now ready to begin using your dictionary to design forms, run cross tabulations, and more!

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 tell what are the different kinds of records, what fields are on each record, what is the starting position and length of each field, what type of data is contained in each field, what values can appear in each field and what do they mean.

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

1 With CSPro create a new Data Dictionary file.
2 Turn OFF the **Option** for **Relative Positioning** so that you can position each data item according to the written specification.
3 Define in the Data Dictionary, the records, items, values from the record description.

See also: Select Relative or Absolute Positioning

Converting ISSA or IMPS Dictionaries

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:

1 Click the **Tools** menu, and then click **Convert Dictionary**.

2 Choose whether you are converting between CSPro and IMPS, or CSPro and ISSA, then press **Next**.

3 Choose the type conversion you are performing and press **Next**.

4 Choose the file name of the dictionary you are converting.

5 Specify the file name of the dictionary you are generating.

6 Press **Finish** when ready.

How to ...

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, Delete, and Modify Dictionary Elements. Toolbar Summary

View the Dictionary Layout

Pressing  from the toolbar will display the current dictionary's file layout. It 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 (possible when the file's status is absolute).

- denotes the Record Type
- denotes Id Items
- denotes record Items
- denotes Subitems

Tips

- You can also launch the viewer by pressing **Ctrl+L** or, from the **View** menu option, select **Layout**
- 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

You can add a level, record, item, value set, or value to a dictionary. You can add from either the tree or view—in either case, the dictionary's menu bar and popup 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 add one of the following:

To add a level …

1 From the dictionary tree, select any level within the dictionary.

2 Right-click to get the pop-up menu and select "Add Level" (or press **Ctrl+A**).

3 Complete the level properties requested.

4 When you are finished entering the level(s) desired and wish to terminate data entry, press the **<Esc>** key.

Note: There is a maximum of three levels for a dictionary.

Tips

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

To add a record …

1 From the dictionary tree, select any record (or (Id Items) set) within the level you wish to add.

2 Right-click to get the pop-up menu and select "Add Record" (or press **Ctrl+A**).

3 Complete the record properties requested.

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

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

Tips

- After selecting a record in the tree, you can press  to initiate add mode.
- The item will always be added at the end of the record.
- If you add to the wrong place, press the **Esc** key to stop the add.
- Use undo if you added at the wrong place.

To add a data item …

1 From the dictionary tree, select the item within the (Id Items) set or Record you wish to add an item to.

2 Right-click to get the pop-up menu and select "Add Item" (or press **Ctrl+A**).

3 Complete the item properties requested.

4 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.

Tips

- After selecting an item in the tree, you can press  to initiate add mode.
- The item will always be added at the end of the record.
- If you add to the wrong place, press the **<Esc>** key to stop the add.
- Use undo if you added at the wrong place.

To add a value set …

1 From the dictionary tree, select the (sub)item you wish to add a value set to.

2 Right-click to get the pop-up menu and select "Add Value Set".

3 Provide the Label and Name for the Value Set.

4 Complete the value set properties requested.

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

Tips
- 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.

To add a value …

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

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

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

4 Complete the value properties requested.

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

Tips
- 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.

Insert Dictionary Elements

You can insert a level, record, item, value set, or value into a dictionary. You can insert from either the tree or view—in either case, the dictionary's menu bar and popup 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 of insertion will change. In general, the steps to insert an object are as follows:

1 In the view, move the cursor to the location where you want to insert the dictionary element.
2 Click  on the toolbar or press the **Insert (Ins)** key.
3 Enter the information requested.

Tips
- 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.

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 popup 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

Delete Dictionary Elements

1 Select the element you want to delete.

2 Click  on the toolbar; or from the **Edit** menu, select **Delete**; or press **Delete/Del**.

Tips

- 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.
- You can also select multiple lines by selecting the first item, then pressing down on the **Shift** key while you use the up or down arrow to adjust the selection.

Undo and Redo Changes

Press  on the toolbar; or from the **Edit** menu, select **Undo**; or press **Ctrl+Z**.

Tip

- To undo the next-to-last change, press the **Undo** button again.

Select Several Dictionary Elements

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

Using the mouse:

1 Click on the line where you want to start selecting.
2 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.
3 Release the mouse button.

Using the keyboard:

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

Tips

- 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.

Move Dictionary Elements Around

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 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 …

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

To copy things …

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

To paste things …

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

Tips
- You can paste cut or copied material to more than one location.
- Use undo if you paste to the wrong place.

See also: undo

Convert Items to Subitems

1 Select the items you want to convert to subitems.
2 From the **Edit** menu, select **Convert to Subitems**.
3 Enter information about the item that will include these subitem(s).

Tips
- To Convert to Subitems, you can also right-click on the item list in the view and select **Convert to Subitems** from the popup menu.
- To convert subitems back to items, delete the item. When asked if you wish to "Delete subitems too?", answer **No**.

Select Relative or Absolute Positioning

This feature primarily refers to the start positions of your data items. In **Absolute** positioning, the start position of each data item you define is kept intact—so if data items are later deleted or moved, "gaps" that will invariably be introduced will not be automatically removed by the Data

Dictionary. If you don't want to introduce holes in your data file, keep the Dictionary's positioning **Relative**.

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.

Here are the characteristics of a dictionary (and therefore of its data file) when using the different types of positioning:

In Relative Positioning

- 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 record's items will be placed after **all** defined Id items (even those defined at a higher level than 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 other items to give the implied relative arrangement.

In Absolute Positioning

- The record type, id items and record 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.

Tips

- Use **Relative** positions when designing a new data file—you do not normally want holes in your data files, as this will increase the size of the file.

- Use **Absolute** positions when describing an existing data file—in this way you won't have to define the holes in your data file(s). Further, if you only want to use a subset of the data file's information, using absolute positioning allows you to define only those data items of interest to you.

Find Dictionary Elements

Search for a given text string using the Find Dialog Box. Names and labels of all dictionary entities (for example, levels, record, items, value sets) will be searched.

1 Press  on the toolbar or **Ctrl+F**.
2 In the find dialog box enter the name of the record, item or value to find.
3 Press the Next button to search.

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.

Tip

- You can also launch **find** from the menu bar. Choose **Edit**, then **Find**.

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.

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

Match Case
If checked, the search will match only if the letters are the same case (upper or lower) as you entered them. If not checked, the search will ignore case.

Close
Close the dialog box.

Document Dictionary Elements

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

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.

Summaries

Data Dictionary Menu Summary

The Data Dictionary menu is displayed across the top of the window. It provides access to most features used in Data Dictionary. The following menu options are available whenever the right-hand screen is displaying dictionary items.

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 File	Insert a file into an existing application.

Drop File	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
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.

Data Dictionary Toolbar Summary

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 Dictionary Keys Summary

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 CSPro

Ctrl + C Copy the selection and put it on the Clipboard.
Ctrl + F Find specified text.
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 Insert Clipboard contents clipboard.
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 Designer

Introduction to Data Entry Designer

The Data Entry Designer allows you to create, using a single dictionary, one or more forms for data entry.

This is the design stage of the data entry process. This tool allows you to create data entry forms (screens) and to specify how the data entry application will behave. 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.

Data Entry Concepts
- Data Entry Methodologies
- Operator vs System Controlled
- Data Entry Path
- Forms
- Fields
- Rosters

Strategies
- Creating a New Data Entry Application
- Deciding What Forms and Rosters to Use
- Converting an ISSA Data Entry Application
- Converting an IMPS Data Entry Application

How to ...
- Generate a Default Application
- Add Things

Use Rosters
Rearrange Things
Modify Things
Change Entry Characteristics
Add and Modify Procedures
Test and Run Applications
Setup a Production System

Summaries

Menu
Toolbar
Keys

Data Entry Concepts

Data Entry Methodologies

Heads-Down Keying

This methodology is common in census keying 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 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 methodology is 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.

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. These applications generally allow the data entry operator more flexibility during data entry. This type 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 consistent with 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 consistent with the heads up methodology.
- Logic in the application is strictly enforced; operator cannot bypass.

Note: You set the application type on 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 off, depending on the 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.

Forms

A form is a collection of fields, text and, optionally, rosters which appears on the screen at the same time during data entry.

A form may be larger than the actual screen. In this case, the form will scroll as necessary during data entry. A form may repeat if it contains fields from a dictionary record which has more than one occurrence.

See also: Adding a Form, Adding Fields to a Form

Fields

Fields are areas of a data entry form that may be keyed or may show values. Fields belong to either forms or to rosters. 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. You may define the following kinds of special kinds 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. Right-click on the field and select **Properties** to get to the Field Properties Dialog Box.

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. Right-click on the field and select **Properties** to get to the Field Properties Dialog Box.

Note: 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 which 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. Right-click on the field and select **Properties** to get to the Field Properties Dialog Box.

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

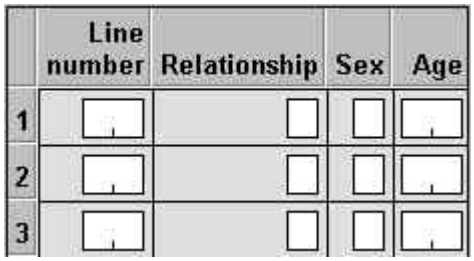
Mirror fields show the value of a previously-entered field on the screen. The cursor never goes to a mirror field during data entry. Mirror fields are useful to display values from one screen on another screen. Any field from a single-occurrence group can be a mirror field.

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: Add Fields to a Form, Change Field Properties

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 above. Rosters can have a vertical orientation, in which case the rows and columns would be reversed.

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

The darker gray area at the top of each column is called a **column heading**. In the example above, the column headings contain the text "Line number", "Relationship", "Sex", and "Age".

The text in the darker gray area to the left of each row is called the **occurrence label.** In the example above, the occurrence labels are "1", "2", "3". These are the default values.

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

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

Strategies

Creating a New Data Entry Application

When you create a data entry application (**File**/**New**/**Data Entry Application**), you may select an existing dictionary on which to base the forms, or you may create a new dictionary and add variables to it as you create your forms.

If you use an existing dictionary, CSPro will create, by default, one form for each dictionary level, placing the ID Items from that level on the form. No record items will be placed on the form(s). You are now ready to modify the form(s) as desired.

If you are creating a new dictionary, CSPro will create one blank form by default.

Tip
- Save time by automatically generating data entry forms.

Deciding What Forms and Rosters to Use

CSPro provides flexibility in the way you define how many forms to use and what fields go on what forms. If you plan to key data from paper questionnaires you generally try to make the forms match the pages in the questionnaire.

There are limitations imposed by the structure of the data dictionary. Some limitations 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 subitem 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 subitems 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 what goes on what form.

Converting an ISSA Data Entry Application

If you have an existing ISSA Dictionary that contains forms, CSPro provides a utility to convert it to a CSPro Form File (a dictionary will be generated as well). See ISSA Conversions for additional information on how the conversion will work.

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

Next, state that you'd like to convert from **ISSA to a CSPro Forms and Data Dictionary**. Provide the name of the original ISSA dictionary file, and the name you would like to call the

CSPro form file to be generated (a CSPro dictionary file will also be created, and its name will be based on the form file name). Press **OK** when ready and the files will be created for you. You are then ready to fine-tune the layout of the forms as desired.

Note that this creates a stand-alone dictionary and form file; it does **not** create a data entry application. Until there exists a data entry application, you cannot write logic for the form variables, nor can you enter data based on this form file. If you would like to generate a data entry application, proceed as you would for creating a new data entry application. When you are asked to provide the name of the form file, simply use the same form file name that was used during the conversion, and CSPro will complete the task.

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. See IMPS Conversions for details on how the conversion works.

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 Deciding What Forms and Rosters to Use 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.

How to ...

Generate a Default Data Entry Application

CSPro can automatically generate a data entry application 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 (ID 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 a data entry application, either press **Ctrl+G** 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, an options dialog box 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 subitems dropped instead of the item, etc. See the Drag Options help for more information.

Add Things

Add a Form

There are three basic ways to add a new (blank) form. Each method will present you with the form property dialog box.

Method 1: 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.

Method 2: From the Form Designer's Menubar

Select **Edit**, then the **Add Form** option.

Method 3: 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 edit tree.

Add Fields to a Form

The Forms Designer Window Layout

Notice that the CSPro window is split in half. The left side contains one or more tabs; the two tabs of interest when designing forms are the **Dict** [Dictionary] and **Form** tabs. Immediately after opening or creating a data entry application, CSPro will display the **Dict** tab on the left, and the first form for the first (and perhaps only) level will be displayed on the right. You are now ready to start dragging items and/or records from the dictionary to the form(s).

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 subitems 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.

Change Drag Options

Whenever you automatically generate a data entry application, 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 **Edit** menu and select **Drag Options**.]

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 **checked**, the operator must always press the **Enter** key to advance to the next field.

Use Subitems When Present?

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

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

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). To do so:

1 Right-click on the form at the point where you want the text to start.
2 Select **Add Text** from the pop-up menu.
3 Type in the text, and press **Enter**.

Draw Boxes on 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 move or delete them. If one item is selected, you can also review its individual (field/roster/text) properties.

Tips

- 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.

Drawing Boxes

CSPro also allows the user to draw boxes, as both a means to help visually organize your data and to make the layout of your form look more professional. For example, if you wish to place fertility data on one portion of your form and then indicate to the viewer that these data are related, you could draw a box around the related items.

When you select multiple items with the mouse, you'll notice during the selection process a box that drags with you to show what you're including. To draw a box on a form, it seemed logical to have that same mechanism at work, so we've introduced the **Select Items/Boxes** button. Click on  to toggle between the two states. When you first click on this button it will appear depressed, and a floating toolbar will appear with the following buttons:

Click	To
	Allows you to toggle states between selecting items and drawing boxes without having to close down the toolbar
	Draw a box with an etched edge
	Draw a box with a raised edge
	Draw a box with a thin edge
	Draw a box with a thick edge

When you have finished drawing boxes and no longer need the box-draw toolbar, close it down by either toggling the  button, or close the box-draw toolbar.

Use Rosters

Create a Roster

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 subitem to a form. If you have a multiple item that has subitems, and you want to create a roster of the subitems, make sure you have the **Use subitems if present** box checked in the Drag Options dialog box.

Add Things to a Roster

Add Fields

Rosters can only include items from the same multiple record, or subitems 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.

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 gray space in the roster and select "Add Text". Note that you can choose 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. You can change this attribute for any box later if you want. Drawing boxes in a roster is essentially the same as drawing boxes on a form.

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 around the edges, the resize handles, at the corners and sides of the roster. Move the mouse pointer on top of a 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.

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.

Change Roster Occurrence Labels

Right click on an 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.

Horizontal alignment
This allows you to force the text to be left-aligned, right-aligned, or centered within the occurrence label area.

Vertical alignment
This allows you to force the text to be aligned at the top, middle, or bottom of the occurrence label area.

Font
To change the font of the occurrence label, choose the "Use custom font for text" radio button then click on the "Choose font" button.

Join and Split Roster Columns

By default, CSPro puts one field in each column. You can put two or more fields in a column by using the **Join** facility. To join two or more columns:

3 Select two or more adjacent columns. You can do this by holding down the Ctrl key and clicking on each column.
4 Right-click and choose **Join.**
5 Type in the text for the joined column.

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. To split a column:

1 Click on the column heading to select it
2 Right-click and choose Split

Rearrange Things

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 which text this is 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.

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. For more information about roster operations, see Resize and Position Things on 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 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 3 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 3 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.

Tips

- To select several items, hold down the left mouse button while dragging a selection box around the desired items.

- To select several items, you can also hold down the **Ctrl** key while individually clicking on each item 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 the both the **Shift** and **Ctrl** keys while individually clicking on either the edit box or its text for each different field.

Cut, Copy, or Paste Things

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 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 …

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

To copy things …

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

To paste things …

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

Tips
- You can paste cut or copied material to more than one location.
- Use undo if you paste to the wrong place.

See also: undo

Modify Things

Change Forms File Properties

Right click on the form file on the tree (top most entry) and choose Properties.

Label
This is descriptive text which 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.

Tip
- 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 which 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.

Tip
- 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 which helps you identify the current form. It may contain any characters (including blanks) and be up to 120 characters long.

Name
This is the name of the form 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.

Tip
- 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 which will be skipped to if the operator presses the plus (+) key on the numeric keypad. If the skip to field it is blank and the plus key is pressed, CSPro skips to the next field in sequence. Skip to is available only in operator controlled data entry mode.

Persistent
Check this box to make the field persistent.

Sequential
Check this box to make the field sequential.

Protected
Check this box to make the field protected.

Upper Case
Check this box to make the field upper case. This attribute only applies to fields whose corresponding dictionary item is alphanumeric.

Mirror
If this box is checked, the field is a mirror field. You cannot modify this property.

Use Enter Key
Check this box if you want to change the data entry option for this field 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 change the data entry option for this field 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.

Verify
Check this box if you want to change the data entry option for this field to verify the field when the operator is in verification mode. If left unchecked, verification is skipped. During verification mode, after each field is keyed, the value entered is compared with 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.

Label

This is descriptive text which 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.

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

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.

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.

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.

Note: You can change the font for all text fields on all forms by choosing Options/Global font from the main menu.

Delete Form Elements

To delete a field from a form:

1 Click on the field on the form, or on the forms tree.
2 Press **Delete**, or right-click and choose **Delete**.

To delete a form, do one of the following:

- Click on the form on the forms tree and press **Delete**
- Choose **Delete form** from the main menu at the top

Undo/Redo Changes

CSPro keeps track of the last dozen changes you have made to your forms on an "undo stack". Beware that not all changes can be undone.

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

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. You could probably make a nice song out of that.

Change 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.

Note: To change the order of fields in a roster you must drag and drop within the roster rather than on the tree.

Note: 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

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 at the end of each case entered to accept the case.

Allow partial save
If this box is checked, CSEntry will allow the operator to save a case from add, modify, or verify mode 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.

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.
Adds keystrokes for the operator, but controls flow.
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.
Fewer keystrokes for the operator.
More consistent with the heads down methodology.
Selecting this option will change the setting for **all** fields.

Some fields
You can not select this option, it is permanently deactivated. If this option is selected, it means there is a combination of enter options in affect—i.e., at least one field in the application has their "Use Enter Key" option checked, and at least one other field in the application whose "Use Enter Key" has **not** been checked. 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
All fields in the forms file can be forced.
Allows out-of-range values to be entered in the data file.
Allows editing of out-of-range values after keying.

No fields
All fields in the forms file cannot be forced.
Prevents any out-of-range values from being entered in the data file.
Forces the operator to edit out-of-range values.

Some fields require
Different fields in the forms file have different settings.

Select one of the other options to force all fields to the same setting.

Upper Case (alpha only)

At data entry time, alphanumeric fields can either allow upper and lower case 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

All alphanumeric fields in the forms file will be upper case only.
Allows case independent entry of alphabetic characters.
Useful for yes/non (Y/N) or letter character responses (A/B/C/D).

No fields

All alphanumeric fields in the forms file will be mixed upper and lower case.
Allows case sensitive entry of alphabetic characters.
Useful for names and addresses.

Some fields require

Different fields in the forms file have different settings.
Select one of the other options to force all fields to the same setting.

Verify

During verification, each item is either verified, that is keyed again and compared with 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

All fields are verified.
Allows maximum verification of data.

No fields

No fields are verified.
Useful when only a few fields are to be verified. Set all off, then set the verify property of each field to be verified.

Some fields require

Different fields in the forms file have different settings.
Select one of the other options to 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 10^{th} 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

From the **Options** menu, select **Default Text Font**.

Current default font is what CSPro will use whenever you add new text to any form.

- 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.

Change Field Font

From the **Options** menu, select **Field Font**.

- 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 make 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.

Tip
- If you change the field font you may want to also change the default text font and apply it to all items.

Change Error Sound

When an error occurs during data entry, an error box is shown of the screen and sound (beep, tone or other sound) is generated.

In order for the sound to be heard:

1 The computer must have sound card, with speaker connected and turned on.
2 The volume on the sound system must be turned on and sufficiently loud to be heard.
3 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.

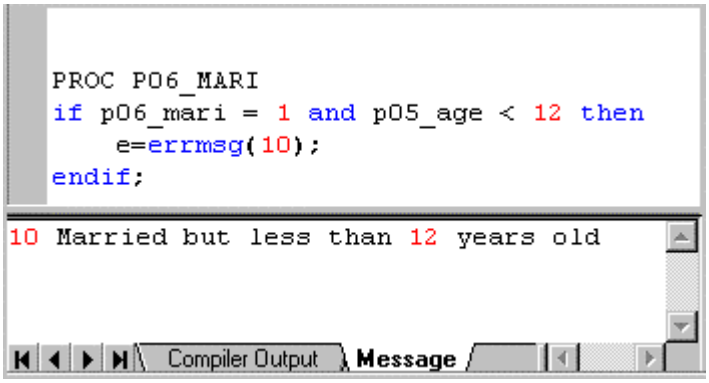
Add and Modify Procedures

View Logic

To toggle between the Logic Window and the Forms Window on the right-hand side of the screen

- Press  ; from the **View** menu check or uncheck **View Logic**; press **Ctrl+L**.

From the Logic Window you can create or modify procedures that add logic to your application. The view is divided into three general areas:



- Top: Text editor, where you write the logic statements
- Bottom: Compiler output tab, where CSPro error messages appear if you have errors in the logic
- Bottom: Message tab, where you create messages to show the keyer

Click on the forms 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.

Hint: Use the toolbar button  or Ctrl+L to switch back and forth between logic view and form view.

See also: Create and Edit Logic, CSPro language, 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 application—a field, form, roster, or level. First make sure the screen has:

- Logic View on the right.
- Forms tree on the left, so you can click on the item for which you want to write logic.

Note: You can see the logic for the whole application by clicking on the form file (usually the topmost node) on the forms tree.

Example: Programming a message for the keyer

Give a message 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
```

type

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

Example: Programming a skip

After the marital status question, 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, put:

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

Set Compiler Defaults

From within a data entry or batch application, an option is available that determines if variables must be declared. 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 encountered.

If you do not wish to declare your variables, you can change the behavior to implicit by going to the **Options** menu, and selecting the **Set Explicit** option. This will uncheck the option, allowing you to create variables "on the fly" simply by referring to them. However, this is not recommended by the CSPro team. It is much better programming practice to use keep the **set explicit** option ON. If variables are explicitly defined, the compiler can detect misspellings of variable names, which are difficult to find otherwise.

Alternatively, you may include a **set explicit** command in your program to override the system setting. See set statement for the command syntax. The following explains the impact of programmatically setting this switch, as opposed to using the system setting:

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

Note: 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.

(zFormF)

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 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 screen. When you are ready to compile the entire application, select the Batch Edit icon for the application from the 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.

Test and Run Applications

Run a Data Entry Application

Pressing  will launch CSEntry, and allow you to enter data for your application (you can also launch CSEntry by pressing **Ctrl+R**, or by selecting **Run** from the **Fil**e menu). 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.

Setup a Production System

Installing Data Entry Applications

To install data entry applications on each data entry computer you must:

Install the CSPro Data Entry Operator Software

Run the CSPro 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 CSPro components necessary to modify an application will NOT be installed.

Install the CSPro 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 CSPro 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 Data Entry Applications.

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 2.4\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: either [1] create it yourself using an ASCII editor (such as Notepad or Wordpad), or [2] 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 2.4
AppType=Entry

[DataEntryInit]
OperatorID=Emperor Armando
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=Emperor Armando
"Emperor Armando" 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: either [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 constricted due to options indicated in "Lock," the next feature). The other two permissible entries are "modify" or "verify".

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>
```

Summaries

Data Entry Designer Menu Summary

The following menu selections are available:

Files	
New	Create a new application.
Open	Open an existing application.
Close	Close an application.
Save	Save an application.
Insert File	Insert a file into an existing application.
Drop File	Drop a file from an existing application.
Compile	Compile the logic in the application.
Run	Run the application.
Edit	
Undo	Undo the most recent change.
Redo	Redo the last undo.
Cut	Copy selected element to clipboard and delete it.
Copy	Copy selected element to clipboard.
Paste	Paste element on clipboard to selected location.
Add Form	Add a form to the application.
Delete Form	Delete a form from the application.
Generate Forms	Generate forms using the Data Dictionary.
Delete	Delete selected objects.
Find	Find text in the procedures.
Find Next	Find the next occurrence of text in the procedures.
Replace	Replace text with new text in the procedures.
View	
Box Toolbar	Show or hide box drawing toolbar.
Names in Trees	Show names instead of labels in trees.
Full Screen	Hide the trees and show full screen view.
View Logic	Show or hide procedures in right-hand window.
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.
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.

Data Entry Designer Toolbar Summary

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 .


Toggle between the logic view and form view.


Show last Dictionary window.


Show last Forms window.


Show last Batch Edit window.


Show last Cross Tabulation window.


Get Help.

Data Entry Designer Keys Summary

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 procedures.
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 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 + 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	Insert Clipboard contents clipboard.
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 Editing

Introduction to Batch Editing

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.

After keying or scanning your data, there will be errors in the data file. This is unavoidable, 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.

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.

For information on how to write, debug, and test your edit routines, continue reading.

To create these edit routines, you will create a batch edit application within CSPro, 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.

Editing Concepts

Editing Concepts

Screen Layout

The CSBatch screen is divided into three main work areas: the *Tree View*, the *Logic View,* and the *Message View*. They are found as follows:

Tree View: the window on the left half of the screen is the standard CSPro tree tab. If you only have a batch edit application open, at the bottom of this window you will notice three tabs: "Files", "Dicts", and "Edits". The "Edits" tab will be of primary importance to you.

Logic View: This is the window block on the right half of the screen, in the upper portion. It is devoted towards displaying the programming logic for the various edit levels, records, and items listed in the "Edits" tree tab.

Message View: This is the window block on the right half of the screen, in the lower portion. It is devoted to messaging. Similar to the Tree View, you have tabs available to you; 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; or, if the run was successful, it will state "Compile Successful." The "Message" tab is used to type in error messages you wish to use in debugging.

Within the Tree View, the "Edits" tab will be the most useful to you. The Logic View displays the programming logic (if any) for individual edit items. The Message View is where the results of your code compilation are viewed, and where user-defined messages are viewed and/or created.

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: Move Around an Application

Edit Tree

When you create a batch edit application, the edit tree will look identical to the dictionary tree; that is, edit items will be listed as found 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-level code, [2] user-defined functions, and the [3] global routine.

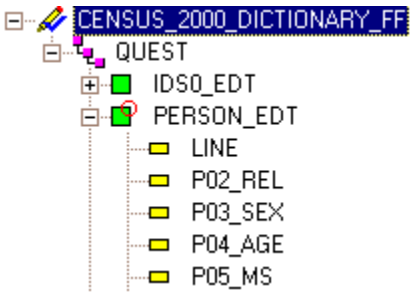
BatchEdit Level: This is the second-tier tree node, just below the root. It has a 1-to-1 correspondence with the same-named dictionary level.

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, 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 name 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 corrected.

The edit tree represents the order in which the logic associated with each edit item is executed. For example, review the following edit tree:



As indicated by the indentation, the preproc for the batch application will be executed before the preproc for the first level, and the preproc for the level will be executed before the preproc of the edit record. Finally, at the terminal leaf (the edit item), each items' preproc and postproc are executed before executing the postproc for the edit record.

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 BatchEdit File icon.

You can never delete edit levels, records, or items (i.e., the entries shown on the "Edits" tab). However, you can reorder edit items or records (though not levels) by dragging them within the "Edits" tree tab. When selecting a new edit item, the contents of the logic view will change to display the logic for the selected entity.

See also: Edit Logic, Edit Order

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 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 Edit Tree, you can reorder items and records (but not levels) on the 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. But you can change this by a simple drag and drop. Simply grab Relationship with the mouse, and drop it on top of Sex. The tree order will change to reflect the drag.

Having said all this, there are a few nuances to the editing process that you may wish to note, specifically with regard to the preprocs and postprocs execution:

1 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 interim of a preproc and postproc code block. Therefore it is suggested to simply maintain one code block for each item. If you do not write "preproc" or "postproc", the code by default will be in the postproc block. This is the suggested behavior, i.e., accept the default.

2 If a BatchEdit item is within a record that repeats, the logic will be repeated for each occurrence. For example, if you have a population record that allows 30 occurrences, the logic for each of its member items will repeat up to 30 times. 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.

3 If a record repeats, the associated logic for that record will NOT repeat; instead, it will be executed once and only once. 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.

4 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.

5 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.

Edit Logic

After creating a new batch edit application, you will see the Edits tree view on your left, with the root node selected, and the logic view on your right, with two lines of code provided: "PROC GLOBAL" and "PROC <BatchEdit_Filename>". 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.

There are two types of procedures for each edit item: a *preproc* and a *postproc*. By default, if not specified, logic will always be considered to be in the postproc portion of the edit entity.

When CSBatch begins to evaluate logic for a given item, it will execute the code contained in the preproc (if present) first. Once finished with that code, it will then proceed to the postproc of the edit item (again, if present). Usually, there is no need to use both the preproc and postproc sections, the postproc will suffice. However, there are times when it will make a difference, especially within an edit record or edit level. Read Edit Order, above, to see why.

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 invalid) or missing; then an appropriate response should be given.

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 am 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 is wrong (i.e., it should be blank). This type of error must be corrected, as your users will place very little faith in your data quality if this type of condition exists. Nor will they look too kindly on missing data either. Of course, nothing can correct for bad data, and if you find that a significant amount of your data is bad (poorly designed questionnaire, inadequate field procedures, inattentive coders and keyers, etc), you may want to consider whether the data should be used 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 age 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 they will be made male, and 50 percent of the time they will be made female. Note that no accommodation was made for other responses; for example, if there was fertility data present, you may not wish to make this person male. Or maybe this was 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 in to account other responses. We attempt do this 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):

```
                                (other  (non-
        (head) (spouse) (child)  rel)    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 is 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 rather than creating a table and assigning the values once and never changing them, the values are updated with original, correct values from your population.

Say for example you have a person and their age, relationship, and sex codes appear correct. You have done consistency checks among them, and everything checks out. You can use that person's values to update your cold-deck, making it a hot-deck. Take the table we had under the cold-deck example:

```
                                (other  (non-
        (head) (spouse) (child)  rel)    rel)
           1       2        3      4       5
(M) 1   35      50       10     41      65
(F) 2   32      48       10     37      68
```

If we found a male child whose age was 6, we would update this table yielding the following, revised table:

```
                                (other  (non-
        (head) (spouse) (child)  rel)    rel)
           1       2        3      4       5
(M) 1   35      50        6     41      65
(F) 2   32      48       10     37      68
```

We would proceed in this way for every person we encountered in the household who had the correct age, sex, and relationship values. Then, when we need to refer to the table due to a missing age, we have a much more representative value for the population. For a more detailed explanation of how to use hot-decks in your program, refer to the CSPro hot-deck example folder or to the United Nations Handbook on Population and Housing Census Edits.

Strategies

Finding Errors

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. However automated searches for your errors is quick and, if done properly, a reliable method to use. If done automatically with CSPro, you can write 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 subject-matter specialists.

Simple errors such as missing or out-of-range values should always be checked for each variable. For example, if a person can be no older than 110, then the first past check for errors might be as follows:

```
proc AGE
if $ in 0:110 then
    exit;  { the age range is OK, nothing else to do }
endif;
write ("Person %d, has incorrect age:  %d", curocc (PERSON_REC), $);
```

So what did we do? If a person's age is in the range 0 to 110 (0 is for infants!), then we're ok and the procedure is exited. If not, then the value is either outside this range or missing, in which case the subsequent write statement will be executed, telling us for for Person N, what their actual age is.

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 $ in 0:20 then { there are 0-20 children }
    if sex = 1 then
          write ("male has fertility info present");
          exit;
    else if sex = 2 then { possibly ok, check woman's age }
          if age < 15 then { 15 = minimum age for fertility }
                write ("woman is too young (%d) to have children",
age);
          endif;
    { else sex is incorrect, but that's another problem! don't worry
about here }
    endif;
else if $ = missing then { ok for men, just check women }
    if sex = 2 then { a problem if the woman is "of age" }
          if age >= 15 then
                write ("woman aged %d should have fertility info
present", age);
          endif;
    endif;
else { fertility value was out of range }
    write ("invalid fertility value (%d) found", $);
endif;
```

As you see, the complexity of your logic to find (and soon, correct!) errors is up to you. Just be sure to consider all situations/paths.

Correcting Errors

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 "improve" it, or 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' category. Thus, in case of an omission or an inconsistent, impossible, or unreasonable entry, a code for 'not reported' 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.

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 of a census could take years, and the possibility of human error is great. 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.

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.

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.

How to ...

Change Edit Order

By default a new data editing 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.

1 From the **Options** menu, select **Custom Order**.
2 Drag and drop items in the order 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 then 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.

Moving Around a Batch Application

To move around your batch edit application, select individual items from the "Edit" 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 batch edit application is displayed.

For example, suppose you select "age" from the Edits tree tab and there is no associated programming logic; you will see:

```
PROC AGE
```

in the LogicView. 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, and so it was unnecessary to explicitly state "postproc."

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:

[1] In the MessageView, 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 number it 10):

```
10 "Current age after imputation is %d"
```

[2] 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 simplying your work, after you run your program, a nice summary statistic will be generated for each user-defined error message, listing 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" command in the helps for a detailed explanation of all the options available to you.

Create a Specialized Report

We talked in Manipulating Automatic Reports about how to add messages to the automatic report which is generated after a run. This is great for debugging, but suppose you've got to submit a more user-friendly report to your coworkers or boss, how do you do this?

You can control it all via the "write" command—use it to put exactly what 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 was a population census, you'd probably have your levels of geography as your IDs; Province, District, Village, EA, etc. Using the "errmsg" command, the IDs would be displayed 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 }
```

Results of the run, written to the **.**wrt file, would be the following (actual numbers will vary depending on the questionnaire IDs of course!):

```
***************
 Province:   1
 District:   7
 Village :  30
 EA      :   4
***************
```

Add additional write statements throughout your program to get the customized report you want!

Use Hot Decks

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.

Set Compiler Defaults

From within a data entry or batch application, an option is available that determines if variables must be declared. 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 encountered.

If you do not wish to declare your variables, you can change the behavior to implicit by going to the **Options** menu, and selecting the **Set Explicit** option. This will uncheck the option, allowing you to create variables "on the fly" simply by referring to them. However, this is not

recommended by the CSPro team. It is much better programming practice to use keep the **set explicit** option ON. If variables are explicitly defined, the compiler can detect misspellings of variable names, which are difficult to find otherwise.

Alternatively, you may include a **set explicit** command in your program to override the system setting. See set statement for the command syntax. The following explains the impact of programmatically setting this switch, as opposed to using the system setting:

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

Note: 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.

(zOrderF)

Compile an Application

When CSPro compiles your application, 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 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 screen. When you are ready to compile the entire application, select the Batch Edit icon for the application from the 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**.

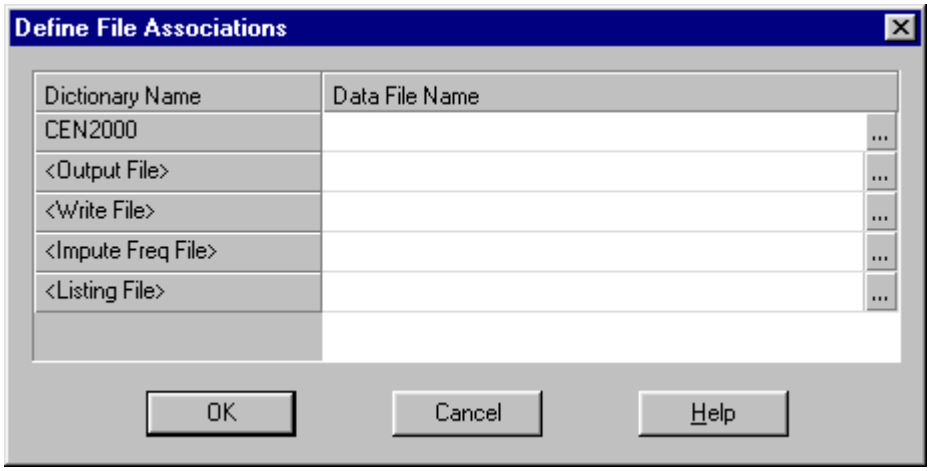
CSPro always compiles your application when you run the application. If there are errors, you cannot proceed until the error are corrected.

Tip

- During code development , you should only compile the logic for an individual Batch Edit entity. This saves time, because the system does not have to recompile the entire all the logic. Furthermore, your entire file may not be ready for compilation (i.e., there are unfinished parts awaiting someone's input), and hence you would not wish to compile the entire file's contents.

Run an Application

Press  on the toolbar; from the **Fil**e 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 ok, then you will receive the following dialog box:



BatchEdit Filename
This line is required, and asks what data file you wish to run your batch application against. This data file will not be modified in any way; it will only be opened, read, and closed.

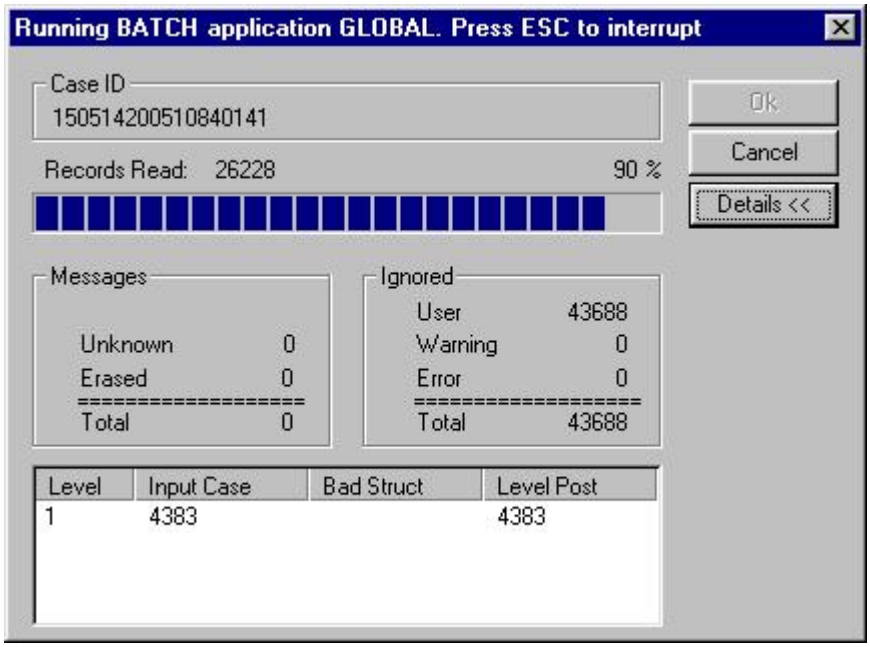
Output File
The output file is where the results of correcting your data will be written. If you are **not** making any corrections in your program, then the generated file will be an exact copy of the original data file. If you **are** making corrections to your data file, then this will be the corrected 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, no corrections will be made.

Write File
If you have any write functions in your program, they will write information to this file. The default file extension is .wrt, but you can use whatever you'd like. This field is optional; therefore, if your program contains write statements, but you forget to specify a write file, no file will be generated. Similarly, if you indicate a write file but your program does not contain any write statements, no file will be generated.

Impute Freq File
If your program contains any impute statements, the results of this command will be written to this file. The default file extension is .frq, but you can use whatever you'd like. This field is optional; therefore, if your program contains impute commands, but you forget to specify a frequency file, no file will be generated. Similarly, if you indicate a frequency file but your program does not contain any impute commands, no file will be generated.

Listing File
This line is required, and asks you what file you'd like to write the results of the run to. Results from errmsg functions will be written here. This file will always be generated, regardless of whether your program included errmsg commands.

After specifying your file(s), a progress dialog bar will be displayed as CSBatch works its way through your data file. If you choose to view the details of the run (just press the "Details" button), you'll get the following dialog:



After the run is completed, TextViewer will be launched to display the generated listing (.lst) file. 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.

Interpret Reports

See the explanation given under running your BatchEdit 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 2.4\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: either [1] create it yourself using an ASCII editor (such as Notepad or Wordpad), or [2] 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 2.4
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 that the batch edit program will run on; this file will not be modified during the run
- OutputData= the revised input data file will be saved as this file
- Listing= a report of the batch operation
- WriteData= if there is one or more write statements in your batch program, their text will be written here
- ImputeFreqs= if you have any impute statements in your batch program, their results will be written here

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.

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 an 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.

Summaries

Batch Edit Menu Summary

Files	
New	Create a new application.
Open	Open an existing application.
Close	Close an application.
Save	Save an application.
Insert File	Insert a file into an existing application (from Files tree).
Drop File	Drop a file from an existing application (from Files tree).
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.
Options	
Custom Order	Allow user defined order of editing.
Set Explicit	Require declaration of all variable names in logic.
View	
Names in Trees	Show names instead of labels in trees.
Full Screen	Hide the trees and show full screen view.
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.

Batch Edit Toolbar Summary

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.

Batch Edit Keys Summary

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 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 + 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	Insert Clipboard contents clipboard.
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.

Cross Tabulation

Introduction to Cross Tabulation

The Cross Tabulation module allows you to tabulate data quickly and easily.

Cross Tabulation Concepts
- Area Processing

Strategies
- Creating a frequency distribution
- Creating a cross tabulation

How to …
- Create a table
- Tabulate items with multiple occurrences
- Define a universe
- Change tabulation parameter
- Include percents
- Handle undefined values
- Tabulate values and/or weights
- Tabulate by geographic area
- Create an area names file
- Change the table title
- Add a table
- Insert a table
- Modify a table
- Delete a table
- Run a tabulation
- Create a thematic map of results
- Select table cells
- Copy all or part of a table
- Save tables
- Print tables

Cross Tabulation Concepts

Area Processing

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 IDs must be based on the Questionnaire IDs for the records you wish to tabulate. Area processing is in addition to any row/column items selected.

To use area processing you must have done the following:

1. Created an area names file (`.anm`)
2. 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.

Strategies

Creating a Frequency Distribution

A frequency tabulation shows the distribution of values in a data file for a particular item in the data dictionary.

- You can get a frequency distribution for any item by specifying it as the only row item with no column items.
- CrossTab automatically shows cumulative values for frequency distributions.
- Frequencies can have a Value and/or a Weight.
- Frequencies can have a Universe.
- Use **Total** or **Column** to get Percents.

Creating a Cross Tabulation

Getting the Basic Row/Column Variables Set

1. Select the Dictionaries [**Dicts**] tab to make the dictionary file structure visible.
2. Expand the tree until the item(s) you wish to use for a row or column variable appears on the tree.
3. 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.
4. You can repeat Step 3 for a total of two items (or subitems or value sets) per row and two items per column.

5. 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

1. Define the Universe for the table.

2. Define the Area for the table (required if you plan on creating a map from your table).

3. Select the Value Item you want to tabulate.

4. Select the Weight Item.

5. Modify the Title if desired, or use the one CrossTab automatically generates for you.

Tips
- Click here to discover how selecting an item as a row or column variable impacts the table layout.
- You can get a frequency distribution for any item by specifying it as the only row item with no column items.
- 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.)

How to ...

Create a 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 function is invoked for a table set, the area levels are included as additional row categories within which the other row variables are displayed.

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:

1. 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.

2. 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.

Tips

- 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.

Define a Universe

When you define a universe, CrossTab will only tabulate data records in the questionnaires that meet the conditions stipulated by you. A universe works like a filter, as the tables produced use

only a subset of the data file's records. Therefore, values in the table may be less 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.

To define a universe:

1. Select the Dictionaries [**Dicts**] tab to make the dictionary file structure visible.

2. Expand the tree until the item(s) you want to use are visible in the tree.

3. Click the  button to launch the **Universe** dialog box. (You can also press **Alt+U** or select the **Edit** menu item, then **Universe**.)

4. 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**.

5. Now select a relationship (=, <>, >, >=, <, <=) value.

6. 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`).

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
```

Tips
- You may enter several conditions using the **and / or**.
- You can 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.
- You can apply a universe to all the tables, by pressing the **Apply to All Tables** button.

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 either a data item or an actual numeric value to be used as a weighting factor during tabulation.

Include Percents

This option is located at the top of the **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 percents 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 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 rounding). If there are no column items selected, this setting has no effect.

- **Column:** 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 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.

Handle Undefined Values

To Include Undefined Row and Column Values

When this setting is used [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 tabulates 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 data dictionary. 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.

Tips

- 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

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 data dictionary as valid for a specific item. Hence, if the `.dcf` specified for tabulation includes the following definition for the field `Sex:`

```
Value Label     From
Male            1
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.

Tabulate Values and/or Weights

This section can be found at the bottom of the **Parameters** dialog box.

To Tabulate a Value

If you use a value item, the actual numeric value of that item for the current record in the data file will be added to the appropriate cell during tabulation. A value item usually represents a quantity of some sort. 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.

If you leave the value item blank, a value of `1` will be added into the appropriate cell during tabulation. 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.

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

Weighted Tabulations

If you choose a weight item and no value item, the actual numeric value of the weight item in the data file will be added into the appropriate cell during tabulation (thus the result is the same as if you used the item as a value with no weighting). If you choose a weight item and a value item, CrossTab will multiply together the actual values of these two items for the current record and add the result into the appropriate cell during tabulation.

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

Tabulate by Geographic Area

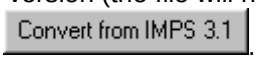
This dialog allows you to select the Questionnaire IDs to be used for area processing, as well as to specify the area names file.

3 Select one or more Questionnaire IDs from the left box (to select multiple items, hold down the **Ctrl** key when you make your selections).

4 Click 

to add the Questionnaire IDs to the Area IDs list.

5 Decide whether to show or not show areas where no data are tallied.

6 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 

.

Tips

- 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**.

Create an Area Names File

- The Area Names File is a text file that you can create using any text editor or word processor.
- It 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).

See also: Area Names File (.ANM)

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.

Tips
- 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

You can insert a table in one of the following three ways:

- Click on the  icon.
- Right-click anywhere in a table and select **Insert Table** from the pop-up menu.
- Select the **Edit** menu, then typing **I** to insert.

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.

Modify a Table

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

- altering the title heading
- changing the universe definition

- changing the parameters of an existing tabulation.
- shifting row/column variables (see **Tips** below) or
- removing row/column variables (see **Tips** below)

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.

Tips
- To exchange a row heading for a column heading, simply 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.

- To remove an item (i.e., one of the row or column headings) from a table, 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.

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.

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`).

1. Press the **Run Tabulation**  button.
2. You may be asked to save tables from a previous run.
3. Select the data file(s) you want to tabulate.
4. The results of the tabulation [the tables] will be shown.

Tips
You can also:

- type **Ctrl+R** to run the tabulation.

- tabulate multiple files in a single run.
- speed up identification of errors in your data file by using the Dump Undefined option.

Tabulating a Single Data File

When you're ready to produce your tables, do the following:

1. Press the **Run Tabulation**  button.
2. Navigate to the subdirectory that contains your data files.
3. Select the desired data file.
4. Press the **Open** key and the tables will be produced!

Tabulating Multiple Data Files

1. Press the **Run Tabulation**  button
2. Navigate to the subdirectory that contains your data files.
3 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.

4. Press **OK** when ready to tabulate.

Create a Thematic Map of Results

To Generate a Map from CrossTab

1. Produce a table using Area Processing from CrossTab.

2. When the finished table appears, click on the cell that represents the variable you wish to map.

3. Click on  [the MapViewer icon, located on the toolbar].

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

5. MapViewer 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:

1. Generate the first variable as described above using an `.mpc` file (rather than an `.mdf` or `.map` file).

2. Switch back to CrossTab (i.e., select it from the Windows 95 task bar or use **Alt+Tab**). Do **not** close down MapViewer.

3. 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).

4. 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.

5. Save the map data file (extension will be `.mdf`).

6. 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.

Select Table Cells

To select table cells …

1 Move the mouse pointer to the upper left-hand corner of the cells you wish to select.
2 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.
3 Then release the mouse button. The selected cells and their heads and stubs are highlighted.

Tips
- 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 block.

To select ALL cells …

From the **Edit** menu, select **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

1 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.
2 Click  on the toolbar; or from the **Edit** menu, select **Copy**; or press **Ctrl+C**.
3 Paste the cells you copied directly into a word processor or spreadsheet. They appear in tabular format.

See also: Save tables

Save Tables

To save entire tables …

7 Make sure that none of the table's cells are currently selected (press **Esc** if they are).
8 Click  on the toolbar; or from the **File** menu, select **Save As**; or press **Ctrl+S**.
9 If your tables file has multiple tables defined within it, a dialog box will appear which lets you pick which tables to save. Select all the tables that you would like to save in one file.
10 A **Save As** dialog box lets you enter the file directory (folder), file name and file type.

To save part of a table …

1 Select table cells you want to save. The corresponding boxheads and stubs of selected cells are also selected.
2 Click  on the toolbar; or from the **File** menu, select **Save As**; or press **Ctrl+S**.
3 A dialog box will appear asking if you want to save the whole table or just the selected parts. Choose **Selection** and press **OK**.
4 A **Save As** dialog box lets you enter the file directory (folder), file name and file type.

File types …

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.

Print Tables

To print entire tables …

1 Make sure that none of the table's cells are currently selected (press **Esc** if they are).
2 Click  on the toolbar; or from the **File** menu, select **Print**; or press **Ctrl+P**.
3 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.
4 A **Print** dialog box lets you select the printer, the page range, and the number of copies.

To save part of a table …

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

To preview the printing of tables …

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

Summaries

Cross Tabulation Menu Summary

The Cross Tabulation menu is displayed across the top of the window. It provides access to most features used in Cross Tabulation. The following menu options are available whenever the right-hand screen is displaying tables.

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 File	Insert a file into an existing application.
Drop File	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.
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.

Cross Tabulation Toolbar Summary

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


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.

Cross Tabulation Keys Summary

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 CSPro

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.

CSPro Language

Introduction to CSPro Language

The CSPro language lets you to 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 identify and correct errors after data capture is complete.

This section contains the following information:

Language Elements

- Declarations
- Events
- Order of Executing Data Entry Events
- Order of Executing Batch Edit Events

- Statements
- Functions
- Delimiters
- Comments

- Numeric Variables
- String Variables
- Numeric Arrays
- Reserved Words
- Data Items
- This Item ($)
- Subscripts
- Numbers
- Text Strings

- Expressions
- Substring Expressions
- Special Values

- Operators
- Operator Precedence
- And/Or Truth Table

- External Files
- Working Storage File
- Message File

Strategies

- Using Lookup files

Statements and Functions

- Alphabetical List

Language Elements

Declarations and Procedures

Declarations

Declarations are made in the `PROC GLOBAL` section. This is always the first procedure in an application. You can edit the PROC GLOBAL section by clicking on the topmost entry of the forms tree or order tree.

You must declare the following in the PROC GLOBAL section:

- Functions
- String variables
- Numeric variables
- Numeric arrays

Implicit vs. Explicit declaration:

The CSPro compiler operates in one of two modes:

> **implicit**: 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. 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 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.
>
> **explicit**: You must declare all variables not declared in your dictionary; otherwise, the variables will be flagged as errors by the compiler. The advantage is that you do not have to worry about misspelled names.

The default compiler mode is **explicit**. You can change the default mode on your computer by checking or unchecking the Options/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. If you include either a `set implicit` or `set explicit` statement in PROC GLOBAL, this will override the computer's default setting and your application will always give the same result on any machine.

Example:

```
PROC GLOBAL
  alpha FirstName; { this array will have a default length of 16 chars }
  numeric Count1, Count2;
  array AgeDeck (5, 2);

  function diff (x,y);
    diff = x - y;
  end;
```

Events

CSPro logic consists of a collection of events. Each event performs the operations you specify using CSPro statements and functions written in the CSPro Language.

Procedures can be any of the following kinds:

- global procedure
- forms file procedure
- level procedure
- form events
- roster events
- field events

The **global procedure** contains declarations and definitions. User-defined functions, tables, views, arrays, and case identifiers are declared in this section. 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.

The **forms file procedure** contains executable statements and assignments. The forms file procedure has two parts: a preproc and a postproc. In data entry applications, the form's file preproc is executed before any data are entered, and the postproc is executed after all data for the file are entered. Statements are assumed to be in the postproc unless it is explicitly stated that they are in the preproc. The global procedure is equivalent to the ISSA "level 0" procedure.

The **level**, **form**, **roster**, and **field events** contain executable statements and assignments. All can have a preproc and a postproc. Statements are assumed to be in the postproc unless it is explicitly stated that they are in the preproc. In addition, forms, rosters, and fields can have onfocus and killfocus events.

Events always fall under the "PROC" section, which is followed by the name of the field, roster, form, level, or forms file. For example,

```
PROC P05_AGE
preproc
{ logic to execute before cursor appears in the field...}
onfocus
{ logic to execute before cursor appears in the field, data entry only}
killfocus
{ logic to execute after the field is keyed, data entry only}
postproc
{ logic to execute after the field is keyed...}
```

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

Order of Executing Data Entry Events

CSPro executes application events one case at a time. During data entry, preproc, postproc, onfocus, and killfocus events are executed in the order in which they are encountered.

The following diagram illustrates the 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).

```
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: Three Fields, A, B, and C. From Field A, click on Field C. Assume Fields A-C all have preproc, onfocus, killfocus, and postproc events.
Result: Field 1's killfocus would fire, but it's postproc would not. Nothing would execute for Field B. Finally, Field C's preproc, global onfocus, and local onfocus would all execute.

Scenario 2: 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 fire, but it's postproc would not. Next, Form 1's onfocus would execute. Finally, Field 1's global onfocus and local onfocus would execute—but it's 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.

Order of Executing Batch Edit Events

CSPro executes the procedures in an application one case at a time. During data entry, preprocs and postprocs are executed in the order of the form structure unless otherwise specified.

The following diagram illustrates the order of operations for a two-level batch edit 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).

Form File preproc
    Level 1 preproc
        Form 1 preproc
            Field 11 preproc
                <entry of Field 11>
            Field 11 postproc
                :
            Field 14 preproc
                <entry of Field 14>
            Field 14 postproc
        Form 1 postproc
        Form 2 preproc
            Field 21 preproc
                <entry of Field 21>
            Field 21 postproc
                :
            Field 26 preproc
                <entry of Field 26>
            Field 26 postproc
        Form 2 postproc
        Level 2 preproc

Form 3 preproc
Field 31 preproc
<entry of Field 31>
Field 31 postproc
:
Field 34 preproc
<entry of Field 34>
Field 34 postproc
Form 3 postproc
Level 2 postproc
Level 1 postproc
Form File postproc

Statements and Functions

Statements

A procedure contains a series of statements. Each statement is a complete instruction to the computer. Every statement, except the assignment statement, begins with a command and ends with a semicolon (;). Statements are made up of a combination of commands, keywords, expressions, and functions. For example,

```
skip to Q103;
```

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

Assignment statements do not contain commands. For example,

```
SEX_RATIO = MALES / FEMALES;
```

calculates a sex ratio from the number of males and females.

Functions

Functions are of the form:

```
return-value = function-name(parameter-list)
```

where the **parameter-list** can contain zero or more parameters, depending on the function call's requirements.

Functions always return either a numeric or string value. Therefore, if you are going to assign a function's return value to a variable, the variable must be of the same type.

See also Function Declarations

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. Comments can be placed anywhere in an application and are not syntax checked. Comments can not be nested, that is comments within comments are not allowed.

The first line in the example below is a comment.

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

Variables and Constants

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.

Variable names must contain only letters, numbers, or the underscore ('_') character, and must begin with a letter. They can be up to 32 characters long. Variable names are case insensitive, meaning that uppercase and lowercase letters are considered the same. For example, "myvar" and "MyVar" are equivalent.

You may declare a numeric variable "on the fly", i.e., in any event. Numeric variables are global in scope, meaning you can assign or get the value of a variable from any other event.

Note: If you include `set explicit` in the global procedure, then you must declare all numeric variables in the `PROC GLOBAL` section.

Examples of the use of numeric variables:

```
x = 0;
NumOfKids = NumOfKids + 1;
```

See Also: Declarations, Numeric Arrays, Set, Events

String Variables

String variables in CSPro store alphanumeric data.

Variable names must contain only letters, numbers, or the underscore ('_') character, and must begin with a letter. They can be up to 32 characters long. Variable names are case insensitive, meaning that uppercase and lowercase letters are considered the same. For example, "myvar" and "MyVar" are equivalent.

You must declare a string variable in the GLOBAL procedure, using the alpha statement.

Example of the use string variables:

```
PROC GLOBAL
alpha reply;

PROC Q5

if q5 = 1 then
  reply = "Yes";
endif;
```

See Also: Declarations, Alpha, Set, Text Strings

Numeric Arrays

CSPro supports numeric arrays of up to three dimensions.

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.

You must declare arrays in the global procedure, using the array statement. The following would be an example of a numeric array declaration:

```
PROC GLOBAL
numeric MyArray (5,10);
```

The following are examples of using the MyArray array:

```
MYARRAY (1,3) = 0;
x = 2;
y = 1;
MYARRAY (X,Y) = 0;
Z = MYARRAY (X,Y);
```

See Also: Array, Declarations, Set

Alphanumeric Arrays

CSPro supports alphanumeric arrays of up to three dimensions.

Array names can contain only letters, numbers, or the underscore ('_') character, and must begin with a letter. Array names can be up to 32 characters long and 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.

You must declare arrays in the global procedure; see the array statement for syntax and other details.

An example of an alphanumeric array declaration and its usage is as follows:

```
PROC GLOBAL
  array alpha(10) crop (20); { 20 crop names, each up to 10 characters
long }

PROC MY_PROGRAM
preproc
  crop(1)= "maize";
  crop(2)= "wheat";
  crop(3)= "rice";
  crop(4)= "potatoes";
```

If you attempt to assign a string to the variable that is longer than the space allocated, 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 that 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 default to a length of 16. See the alpha declaration for more information on this.

See Also: Array, Alpha, Declarations, Set

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	editnote	level	setlb
add	else	loadcase	setub
advance	elseif	locate	skip
all	end	log	soccurs
alpha	endbox	maketext	sort
and	enddo	max	special
array	endif	min	specific
average	endgroup	missing	sqrt
box	endlevel	next	stat
break	enter	noccurs	stop
by	errmsg	noinput	strip
case	exec	not	sum
clear	exit	notappl	summary
close	exp	numeric	sysdate
cmcode	filename	onfocus	sysparm
compare	find	open	systime
concat	float	or	then
count	for	pos	title
crosstab	function	poschar	to
curocc	getbuffer	postproc	tonumber
default	getnote	preproc	totocc
delcase	if	proc	until
delete	impute	putnote	update
demenu	in	random	visualvalue
demode	insert	recode	vset
denom	int	reenter	where
disjoint	ioerror	retrieve	while
display	key	seed	write
do	killfocus	selcase	writecase
edit	length	set	

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.

Data Items

Data items are defined in a data dictionary. You can assign or get the value of a data item in any procedure.

Examples of the use of data items:

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

See Also: Dictionaries, Record Items, $

This Item ($)

The dollar sign ($) is a short-hand way of referring to a data item if used within it'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 specify 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 expression

```
SEX(1)
```

refers to the sex of the first child. The expression

```
SEX(3)
```

refers to the sex of the third child. The expression

```
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 repeating, 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. For example:

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 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.

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 region 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. For example,

MyString=‘That’s great!’; would set MyString to ⇨that⇦, and the trailing ⇨s great!⇦ would be considered outside the string, and therefore a compiler error

So 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';

Note: Strings that are surrounded by quotation marks will appear in pink. Strings that are surrounded by apostrophes will appear in black. We recommend using quotations, as it will be quickly apparent whether you have terminated your string properly or not.

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 expressions evaluate to numbers. The following are numeric expressions:

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

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

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

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

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

Substring Expressions

A substring expression lets you extract a part (substring) of a string. It takes the form:

```
string[start:length]
```

where **start** gives the starting character position of the substring in 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 string.

For example, 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"
```

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 your 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.

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 can be used to combine multiple if statements. For example, consider the following more traditional program:

```
if fertility = notappl then { fertility is blank }
  if sex = 2 and age >= 10 then { but this is a woman of child-bearing
age }
    fertility = 0; { then she should have a non-blank response }
  endif;
endif;
```

This could more succinctly be written as follows:

```
if fertility = notappl then { fertility is blank }
  fertility = 0 <=> sex = 2 and age >= 10;
endif;
```

Note:

This is not a standard programming operator, and if your code will be developed, edited, and/or reviewed by multiple individuals, it may be more understandable to use an if-then-else statement series.

See also Operators and 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.

or

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

and

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

Files

External Files

An external file is an ASCII text file that you can use in a data entry or batch application, other than the primary data file. 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. External files which are only read from, can always be shared. External files which are written to can only be used by one user at a time.

See also: Insert a File in an Application, Drop a File from an Application, Using Lookup Files, External File Functions

Sharing External Files

External files can be used by several different users 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:

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

2. 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:
    a. loadcase without a var-list
    b. retrieve
    c. key

    In this case the file is opened just before the first file function is executed, but 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.

Working Storage File

Working storage contains alphanumeric variables and data items used in a procedure 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 a File in an Application, Drop a File from an Application

Message File

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

Basic Messages

Each line in the message file contains one message. A message consists of a message number followed by text. The message text can be up to 240 characters long. It is displayed when an errmsg function with the message number is executed in a data entry application.

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

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

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

Messages with Parameters

Parameters can be specified in the errmsg function. These parameters can be numeric expressions or string expressions. String parameters in the error message text are indicated by **%s**. Integer numeric parameters are indicated with **%d**. Decimal numeric parameters are indicated with **%f**.

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

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

The application could use this as follows:

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

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

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

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

Any number of different messages can be included in the Message File. The errmsg function can be used in any dictionary, form, group, or field procedure, or in a user-defined function. The maximum number of parameters in an errmsg function is 20.

Strategies

Using Lookup Files

A lookup file (external file) is an ASCII text file that you can use in a Data Entry or Batch application from which you retrieve data to display on a form or to use in a calculation. 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 make sure 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.

A lookup file requires a CSPro data dictionary.

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

1 Create the lookup file and its data dictionary
2 Close the lookup file's data dictionary
3 Create a Data Entry or Batch Edit Application with a standard forms file and data dictionary.
4 Insert the lookup file's data dictionary into the application.
5 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 2.4\Examples".

Statements and Functions

Alphabetical List of Statements and Functions

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.
clear	Initializes the memory values of data items defined in external files and working storage to zero or blank.
close	Closes a previously opened external file.
cmcode	Returns the number of months since the beginning of the century 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.
exit	Ends a procedure before normal processing is expected to end.
exp	Returns the value of **e** raised to a given power.
filename	Returns the data file name currently associated with a data dictionary.
find	Determines the existence of a case in an external file that matches a condition.
for	Loops through multiple records or items.
function	Declares a user-defined function.
getbuffer	Returns a string containing the contents of a data item.
getnote	Gets a data entry field note and assigns it to a string.
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.
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.
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.
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.
retrieve	Reads a case from the current position of an external file into memory.
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.
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 batch edit run before the last case is processed.
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.
systime	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	During data entry, the **visualvalue** function 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.
writecase	Writes a case from memory to an external file.

Declaration Statements

Alpha Statement

The **alpha** statement declares alphanumeric variables used in the application.

Format:

```
alpha [(len)] var-1[, var-2[..., var-n]]);
```

[] indicates that this part is optional.

Examples:

```
PROC GLOBAL
  alpha a,b,c;
  alpha(10) x,y;
```

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 an alpha var is assigned a string which is shorter than the space that has been allocated for it, the trailing character positions will be blank-filled. Conversely, if the string to be assigned is longer than the space allocated for the alpha var, then the string will be truncated. The following two examples, using the declaration of x above, should clarify this point:

```
x = "hi mom";
```

x will now equal `"hi mom    "`
`1234567890`

```
x = "good night, mom";
```

x will now equal `"good night"`
`1234567890`

See also: Numeric, Set, Alphanumeric Arrays

Array Statement

The **array** statement allows the declaration of a 1, 2, or 3 dimension array. Arrays can contain either numeric or alphanumeric values.

Format:

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

`[]` indicates that this part is optional.

Example:

```
PROC GLOBAL
  array age_HD (2,8);  { sex by relationship }
  array alpha(15) crop (10); { 10 crop names, each up to 15 characters
long }
  numeric 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 }

  crop(1)= "maize";
```

```
crop(2)= "wheat";
crop(3)= "rice";
crop(4)= "potatoes";
{ continue with crop initializations }
```

Description:

The **array** statement defines a variable which contains an array of either numeric or alphanumeric values. Only one variable can be defined in each array statement (note in the example above there is a separate declaration for each array cell). The array can have 1, 2, or 3 dimensions. The array declaration(s) must appear in the global section, much like the numeric statement. The initial array contents are zero (if numeric) and blank (if alphanumeric) until a value for each dimension is assigned.

Whenever the array variable is used in the application, a value or numeric expression for each dimension must be given. For example, in the example above the variables "male" and "female" are used as the first subscript to the array, while a number is used for the second subscript.

See Also: numeric arrays, alphanumeric arrays

Function Statement

The **function** keyword allows the creation a user-defined function. Function declarations must be coded in the PROC GLOBAL section.

Format:

```
function function-name([p-1[,p-2[...,p-n]]]);
  statements;
  function-name = expression;
end;
```

[] indicates that this part is optional.

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}
```

Description:

User-defined functions are defined 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.

Each parameter specifies a numeric or alphanumeric 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.

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.

A user-defined function returns a single numeric value. To specify the return value, assign a numeric value to the name of the function (see examples above).

User-defined functions can contain CSPro statements and functions, and other user-defined functions. If no return value is assigned to the function, a DEFAULT value is returned. User-defined functions cannot be recursive (i.e., they can not call themselves), though they can call other functions (either user-defined or system-supplied). Function arguments are optional.

Example Application:

An example of a user-defined function can be found in the *DateCheck* folder, under the *Examples* folder.

See also: if

Numeric Statement

The **numeric** keyword allows the creation of "on-the-fly" variables (i.e., variables not associated with any dictionary). Numeric declarations must be coded in the PROC GLOBAL section.

Format:

```
numeric var1[, var2, ...];
```

[] indicates that this part is optional.

Example:

```
PROC GLOBAL
  numeric x, y, z, temp;
```

Description:

User-defined numeric variables must be declared with the numeric declaration if the `set explicit` option is active (Options/Set Explicit on the toolbar) or if a set explicit statement is included in the PROC GLOBAL section of your program. A numeric variable is significant to 15 digits. It is equivalent to a float or double variable.

See also: Alpha, Set, Array

Preproc Event

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

Format:
```
preproc
```

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

Description:
A preproc procedure can be coded in a proc for any run, case, level, record, form, roster, or field.

In data entry applications, the statements in preproc procedure are executed when you move **forward** onto an object, that is flow onto it, advance to it, step to it, go to it, click on it, tab to it, or manually skip to it. Preproc statements are NOT executed when you move **backward** onto an object, that is reenter it, go backwards to it, or backtab 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, preprocs are 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 statements are considered postproc statements by default.

See Also: proc, postproc, onfocus, killfocus, Events, Order of Executing Data Entry Events, Order of Executing Batch Edit Events

Proc Event

The **proc** statement declares the beginning of the procedures for a data entry or batch processing element.

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

Example:
```
PROC AGE
```

Description:
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.

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, postproc, onfocus, killfocus, Events, Order of Executing Data Entry Events, Order of Executing Batch Edit Events

Postproc Event

The **postproc** statement declares that the following statements are executed at the end of a run, case, level, record, form, roster, or field.

Format:

```
postproc
```

Example:

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

Description:
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, backtab 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, postprocs are 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 statements are considered postproc statements by default.

See Also: proc, preproc, onfocus, killfocus, Events, Order of Executing Data Entry Events, Order of Executing Batch Edit Events

Set Statement

The **set** statement switches the values of various system parameters.

Format:

```
set explicit | implicit;
```

| indicates that either keyword may be used.

Example:

```
PROC GLOBAL
      set explicit;
      numeric x,y,z;
```

Description:

Set explicit/implicit overrides the compiler's default setting of this switch. If used, it must be the first line coded in the `PROC GLOBAL` section.

set explicit means that any numeric or string variables used in your program must be declared using numeric or alpha statements in the `PROC GLOBAL` section. If they are not, a compiler error message is generated when an undeclared variable is used.

set implicit means that numeric or string variables do not have to be declared in the `PROC GLOBAL` section.

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 be hard to find otherwise.

See also: Set Attributes, Numeric, Alpha

Set Attributes Statement

The **set** statement switches the values of various system parameters.

Format:

```
set attributes (field-1[, field-2, ..., field-N]) display | visible
| autoskip | return | protect | hidden | native;
```

`[]` indicates that this part is optional
`|` indicates that one of the attributes may be selected

Example:

```
PROC QUEST
preproc
      set attributes (total_HH_income) protect;
```

Description:

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 attribute will override any statically-set attribute(s). Field properties set dynamically can be placed anywhere in the program **except** in the `PROC GLOBAL` section.

One or more dictionary items can be named in the field list. However, only one attribute setting can be used in a 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.

`autoskip` This is equivalent to leaving the statically-set field property "Use Enter Key" unchecked. If this option is used, the cursor automatically advances to the next field,

after the maximum number of characters have been entered. This option will override any statically-set field property settings.

`return` This is equivalent to checking the statically-set field property "Use Enter Key." If this option is used, the operator must press the <Enter> key to advance from the listed field(s). This option will override any statically-set field property settings.

`protect` This is identical to the statically-set field property "protected." If a field is set to 'protect', the operator will not be able to enter it. If the field was already statically set to "protected," the setting has no effect.

`hidden` If a field is visible, its value will now be hidden from view; if it was already hidden, the setting has no effect.

`native` Regardless of what settings have been made dynamically in the program, if a field is set to native, all field settings will revert to their initial, statically-set properties.

See also: Set Explicit | Implicit, Numeric, Alpha

Assignment and Recode Statements

Assignment Statement

The **assignment** statement sets a variable equal to the value of an expression.

Format:

```
numeric-variable = numeric-expression;
string-variable = string-expression;
```

Examples:

```
AGE = 10;
Q102 = PREV_AGE;
Y = sqrt(X);
NAME = "John Doe";
```

Description:

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.

Recode (Box) Statement

(Note to ISSA users: The **Recode** and **Box** statements are identical.)

The **recode** statement assigns a value to a variable based on the value of one or more other variables.

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.

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;
```

Description:

The recode statement is used to recode 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 (e.g., 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 and a string if var-out is an alphanumeric variable.

See also: If

Impute Function

The **impute** statement assigns a value to a data item and logs the frequency of assignments.

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.

Example:

```
      impute(P04_AGE, TEMPAGE) title("Age updated via TempAge")
vset(2);
```

Description:

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 number or logical expression; for example, '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 IMPUTES 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 divvied up in to 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

Code Example:
A code example of this statement can be found in the *Examples/Impute* folder.

Program Control Statements

Do Statement

The **do** statement executes one or more statements repeatedly while a logical condition is true, or until a logical condition is no longer true.

Format:

```
do [[varying] var = expression] while/until condition [by expression]
  statements;
enddo;
```

[] indicates that this part is optional.

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.

Description:

The do statement executes one or more statements repeatedly, in a loop, 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 **do** 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 affect on the command, and so may be omitted.

See Also:

for, while , if

Exit Statement

The **exi**t statement terminates a procedure or function before normal processing is expected to end.

Format:

```
exit;
```

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 }
```

Description:

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.

See also: skip case, stop statement

For Statement

The **for** statement loops through multiple records or items.

Format:

```
  for num-var in group do
      statements;
  enddo;
```

Example:

```
  PROC QUEST
  spouse = 0;
  for i in PERSON_EDT do
    if relationship = 2 and spouse = 0 then
      spouse = i;
    endif;
  enddo;

  if not spouse in 0,2 then
    for i in PERSON_EDT do
      errmsg ("Person %d's relationship=%d",i, relationship);
    enddo;
  endif;
```

Description:

The **for** statement executes one or more statements repeatedly within the loop for each occurrence of a multiply-occurring record or item named by **group**. In our example above, 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 current occurrence being examined. It cannot be changed inside the loop, but it can be referenced. Its starting value will be always be 1, and its ending value will be determined by the number of occurrences of the item or record named. So, for example, in our example above, if there were five people in the household, the loop would execute five times.

The **for** statement should be used outside of the controlling item named by **group**. Note in the example above that the code is executed in the QUEST procedure. It should not be located in the PROC PERSON_EDT, or any of the data items in that record.

See Also: do , while

If Statement

The **if** statement executes statements conditionally.

Format:

```
if condition then
    statements;
[elseif condition then
    statements;]
[else
    statements;]
endif;
```

[] indicates that this part is optional.

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;
```

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, elseif keywords do not require extra endif keywords. 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.

While Statement

The **while** statement executes one or more statements repeatedly while a logical condition is true.

Format:

```
while condition do
  statements;
enddo;
```

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;
```

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—therefore, be sure to remember to increment (or decrement) your variables 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.

See Also:

do, for , if

Data Entry Statements and Functions

Accept Function

The **accept** function returns the number of a choice from a list made by the data entry operator.

Format:

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

Example:

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

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.

See Also:
accept, do, noinput

Advance Statement

The **advance** statement moves forward field-by-field to a specified field during data entry.

Format:

```
advance [to] field-name;
```

[] indicates that this part is optional.

Example:

```
FIRST_WOMAN = 0;
do i = 1 while i <= totocc(PERSON)
    if SEX(I) = 2 then
        FIRST_WOMAN = i;
        advance to CHILD;
    endif;
enddo;
```

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** were pressed repeatedly until either the specified field appears or one of the procedures executed during the advance goes to a different field.

Note that the advance statement behaves differently from the skip statement.

See Also:
skip, do, if, totocc

Demode Function

The **demode** function returns the current data entry mode.

Format:

```
i = demode();
```

Example:

```
if demode() = add then
    V103 = 3;
endif;
```

Description:

There are three data entry operator modes: add, to input new cases, and modify, to change cases that have already been entered, and verify, to reenter the cases checking for inconsistencies between the first and second entry. The demode function returns a value of **add, modify**, or **verify** depending on the current data entry mode. The demode function is often used to limit the execution of certain statements to a mode; for example, initialization of variables might be performed in add and verify mode, but left unaltered for modify mode.

Return value:
The function returns an integer value of 1, 2 or 3:

1 corresponds to add mode
2 corresponds to modify mode
3 corresponds to verify mode

Editnote Function

The **editnote** function displays the data entry field note box for adding or changing.

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

[] indicates that this part is optional.

Example:
```
PROC COOKING
if $ = 9 then
    OTHER = editnote();
endif;
```

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.

See also: Getnote, Putnote

Endlevel Statement

The **endlevel** statement ends data entry for the current level of the current questionnaire.

Format:
```
endlevel;
```

Example:
```
if MORE_WOMEN = 0 then
    endlevel;
endif;
```

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.

Note: 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.

Endgroup Statement

(**Note**: this function has superceded the endsect statement. endsect will continue to work, but is being dropped in favor of endgroup.)

The **endgroup** statement ends data entry for the current record.

Format:

```
endgroup;
```

Example:

```
if KIDSBORN = 0 then
    endgroup;
endif;
```

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.

Code Example:

A more thorough example of this statement can be found in the *Examples\ItemDrivenDE* folder.

Enter Statement

The **enter** statement enters data from a secondary form file.

Format:

```
    enter form-file-name
```

Example:

```
if V108 = 6 then
    enter OTHERS;
endif;
```

Description:

The enter statement allows the use of a secondary form file to enter data to 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**.

Getnote Function

The **getnote** function returns the field note string for the current field.

Format:

```
s = getnote()
```

Example:

```
PROC BIRTH_PLACE
if length(getnote()) > 0 then
      FLD_NOTE = editnote();
endif;
```

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.

See also: Putnote, Editnote , if, length

Killfocus Event

The **killfocus** statement indicates that the following statements are executed when a form, roster, or field stops being active.

Format:

```
killfocus
```

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 }
```

Description:
A killfocus procedure can be coded in a proc for any form, roster, or field data entry applications. Killfocus procedures are not executed in batch applications.

Statements in a killfocus procedure are executed whenever you move off of 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, flow off of it either by logic or by operator intervention, such as clicking off it with the mouse, manually

skipping from it, or backtabbing from it. They are also executed when you perform noinput of a field or when the field is protected.

See Also: onfocus, proc, preproc, postproc, Events, Order of Executing Data Entry Events, Order of Executing Batch Edit Events, reenter , if

Noinput Statement

The **noinput** statement prevents input for the current field during data entry.

Format:

```
noinput;
```

Example:

```
PROC Q102
preproc
if Q101 <> 1 then
    noinput;
endif;
```

Description:

The noinput statement prevents input of a field during data entry. The noinput statement can only be coded 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 requesting 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 requesting 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 backtab to the field. It is not possible to backtab to a field that is protected.

Onfocus Event

The **onfocus** statement indicates that the following statements are executed when a form, roster, or field becomes active.

Format:

```
onfocus
```

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 }
```

Description:

An onfocus procedure can be coded in a proc for any form, roster, or field data entry applications. Onfocus procedures are not executed in batch applications.

Statements in an onfocus procedure are executed whenever 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 backtabbing to it. They are also executed when you perform noinput of a field or when the field is protected.

See Also: killfocus, proc, preproc, postproc, Events, Order of Executing Data Entry Events, Order of Executing Batch Edit Events

Putnote Function

The **putnote** function set the data entry field note to a given value.

Format:

```
b = putnote(string);
```

Example:

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

Description:
The putnote function places a string in the note for the current field. If the string is empty, there fill be no note for the field.

Notes are stored in a file called <data file name>.NOT.

Return value:
The function returns a logcial value of 1 (true) if a case is found and 0 (false) otherwise.

See also: Getnote, Editnote

Reenter Statement

The **reenter** statement forces the data entry operator to re-enter a field.

Format:

```
reenter [field-name];
```

[] indicates that this part is optional.

Example:

```
if KIDS = 1 & BOYS = 0 & GIRLS = 0 then
    reenter KIDS;
endif;
```

Description:
The reenter statement is used to force the entry operator to reenter the datum for the current variable, or for a variable that was entered earlier.

Field-name specifies the field to be reentered. If no field-name is specified, the current field must be reentered. Field-name must be earlier on the data path than the current variable. If it isn't, an error message will be displayed during data entry and data entry will be aborted.

When a reenter statement is executed, the preproc for field-name will not be executed. If field-name is on a different form from the current variable, that form will be displayed automatically. The postproc of field-name will be executed normally after field-name has been reentered.

[When the reenter statement executes, all fields between the current field and field-name are removed from the data path and are displayed in yellow. Pressing the ENTER or TAB key will confirm previously entered information and change the color to green. If reentering a field changes the data path, any previously entered fields that are not on the new data path will remain displayed in yellow and will not be saved when data entry ends for that questionnaire.]

Selcase Function

The **selcase** function allows a data entry operator to select and load a case from an external file.

Format:

```
b = selcase(ext-dict-name,alphanumeric-expression[,offset]);
```

[] indicates that this part is optional.

Example:

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

Description:

The selcase 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** if a key is selected and **false** otherwise.

Skip Statement

The **skip** statement jumps to a specified field during data entry.

Format:

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

[] indicates that this part is optional.

Example 1:

```
if Q305 <> 2 then
    skip to Q307;
endif;
```

Example 2:

```
PROC Q203
preproc
if Q202 <> 1 then
    skip to next Q201;
endif;
```

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.

Note in Example 2 above, that if the skip occurs the resultant value of Q203 will be notappl.

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.

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.

Visualvalue Function

During data entry, the **visualvalue** function returns the value of a data item prior to it's input.

Format:

```
i = visualvalue(numeric-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;
```

Description:

The **visualvalue** function is used to access the contents of a data item before the data item has been keyed. In the example above, 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.

See Also:
accept, if, do, noinput

Batch Edit Statements

Skip Case Statement

The **skip case** statement ends processing of the current case in a batch edit run.

Format:

```
skip case;
```

Example:

```
if totocc(HOUSING) > 1 then
      errmsg("Too many housing records");
      skip case;
endif;
```

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.

See also: Stop, Exit , errmsg, if, totocc

Stop Statement

The **stop** statement ends a batch edit run before the last case is processed.

Format:

```
stop;
```

Example:

```
if TOTAL_ERRORS > 100 then
      stop;
endif;
```

Description:

The stop statements ends batch edit processing. If an output file is specified, neither the current case nor subsequent cases are placed in the output file.

See also: Skip case, exit statement

Numeric Functions

Cmcode Function

The **cmcode** function returns the number of months since the beginning of the centry given a month and year.

Format:

```
i = cmcode(month,year);
```

Example 1:

```
XMONTH = 06;
XYEAR = 81;
DATE = cmcode(XMONTH,XYEAR);
```

The value of DATE for the given parameters June 1981, would be (81 x 12) + 6 = 978.

Example 2:

```
XMONTH = 2;
XYEAR = 2000;
DATE = cmcode(XMONTH,XYEAR);
```

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 twelve, 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.

Exp Function

The **exp** function returns the value of **e** raised to a given power.

Format:

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

Example:

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

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.

Int Function

The **int** function returns the integer portion of a numeric expression.

Format:

```
i = int(numeric-expression);
```

Example:

```
x = int(5 / 3);
```

The value of x would be 1.

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.

Log Function

The **log** function returns the base-10 logarithm of a numeric expression.

Format:

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

Example:

```
X = log(Y);
```

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.

Random Function

The **random** function returns a pseudo-random integer in a given range.

Format:

```
i = random(min-value,max-value);
```

Example:

```
NUM = random(1,100);
```

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 the 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.

Seed Function

The **seed** function initializes the random number generator to a particular starting place.

Format:

```
b = seed(numeric-expression);
```

Example:

```
OK = seed(1009);
```

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.

Sqrt Function

The **sqrt** function returns the square root of a numeric expression.

Format:

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

Example:

```
X = sqrt(Y);
X = sqrt(12);
```

Description:

The sqrt function returns the square root of numeric-expression. The result of the 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 the numeric-expression is negative, the function returns the special value DEFAULT.

String Functions

Compare Function

The **compare** function returns alphabetical order (i.e., collating sequence) of the two strings.

Format:

```
i = compare(string-1,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.

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 for 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

Note:

Direct string comparisons can also be made. For example, the following code is permissible (and in fact preferable to usage of the compare function):

```
if string1 < string2 then
  <statements>;
endif;
```

Concat Function

The **concat** function joins two or more strings into one string.

Format:

```
s = concat(string-2,string-2[,...,string-n]);
```

[] indicates that this part is optional.

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                    "
```

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.

Edit Function

The **edit** function converts a number to a string.

Format:

```
s = edit(edit-pattern,numeric-expression);
```

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);
```

Description:

The edit function converts a number to a character string defined by the given edit pattern. The edit pattern is a string containing Zs or 9s (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 made from the number.

See also: Tonumber , sysdate

Filename Function

The **filename** function returns the data file name currently associated with a data dictionary.

Format:
```
s = filename(dict-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.

Description:
The filename function returns the fully qualified name of the data file referenced by the data dictionary with the name **dict-name**.

Return value:
The function returns a string containing the file name.

Getbuffer Function

The **getbuffer** function returns a string containing the contents of a data item.

Format:
```
s = getbuffer(item-name);
```

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)));
```

Description:
The data item may be numeric or alphanumeric. The getbuffer function always returns a string containing the data item's contents.

Therefore in both examples above, it does not matter whether getbuffer is used on 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 a data item contains you would use the getbuffer to return the alphanumeric characters it contains.

Return value:
The function returns a string containing the data item's contents.

Length Function

The **length** function returns the length of a dictionary item or a string.

Format:
```
i = length(string-exp);
```

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
```

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 an integer length of the string.

See also:
alpha, strip

Maketext Function

The **maketext** function formats a text string with inserted values.

Format:
```
s = maketext(string-exp[,p1[,p2[,...,pn]]]);

          [] indicates that this part is optional.
```

Example:

```
TEXT = maketext("Sex = %d", SEX);
```

Description:

The maketext function formats a text string with inserted values. Each parameter (e.g., p**1**) is sequentially inserted into the text string. Parameters can be numeric or alphanumeric expressions, but the type of parameter must match the type of the receiving field in the string expression.

In the string expression

%[*n*]d = Insert a number and display it as an integer
%[*n.d*]f = Insert a number and display it as a decimal value
%[*n.d*]s = Insert a text string

where ***n*** is the size of the field and ***d*** is the number of decimal places to show for a number.

Numbers are never truncated. Text strings are truncated only if ***.d*** is used.

If ***n*** is positive, the insert is right justified in the size of the field. If ***n*** is negative, the insert is left justified in the size of the field. If ***n*** is a positive number with a leading zero, the insert is right justified in the size of the field and zero filled to the left.

When inserting a number, if ***n*** is preceded by a **+**, the sign of the number is always displayed.

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 the formatted string.

Pos Function

The **pos** function returns the start position of a substring within the source string.

Format:

```
i = pos (substring, source);
```

Example 1:

```
X = pos("L","FOR THE CHILDREN");
```

The value of X will be 4.

Example 2:

```
X = pos("DRE","CHILDREN");
```

The value of X will be 5, as this is where the substring "dre" begins in the source string.

Example 3:

```
X = pos("DCN","CHILDREN");
```

The value of X will be 0. The substring "lcn" does not exist in the source string.

Description:

The pos function searches for a pattern string within a source string. It returns the beginning position of the first occurrence of the pattern string. The substring and source strings are case-sensitive, therefore "children" is completely different than "CHILDREN."

Note:

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 }
```

Return value:

The function returns an integer position. If the pattern string is not found, 0 is returned.

See also:

poschar, alpha statement, strip

Poschar Function

The **poschar** function returns the location of the first character found from the pattern string within the source string.

Format:

```
i = poschar (pattern, source);
```

Example 1:

```
X = poschar("L","CHILDREN");
```

The value of X will be 4.

Example 2:

```
X = poschar("LCN","CHILDREN");
```

The value of X will be 1, as ‘c’ is the first character encountered in the source string.

Description:

The poschar function searches for a collection of characters within the source string. It returns the beginning position of the first occurrence of the pattern string. The substring and source strings are case-sensitive, therefore "children" is completely different than "CHILDREN."

Note:

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";
poschar (" ", myStr);
    { will return 5, as it finds a blank after the ‘s’ in Kids }
poschar (" ", 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 }
```

Return value:

The function returns an integer position. If no characters from the pattern string are found, 0 is returned.

See also:

pos, alpha statement, strip

Strip Function

The **strip** function removes trailing blanks from a string.

Format:

```
s = strip(string-exp);
```

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
```

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.

See also:

alpha, concat, length

Tonumber Function

The **tonumber** function converts a string to a number.

Format:

```
d = tonumber(string-exp);
```

Example:

```
n = tonumber(CASEID);
```

Description:

The tonumber function converts a string into a number. Leading blanks in the string are ignored. The conversion stops at the first non-numeric character.

Return value:

The function returns a decimal number. If the string begins with a non-numeric character, the function returns DEFAULT.

See also: Edit

Multiple Occurrence Functions

Average Function

The **average** function returns the average of an item that occurs multiple times.

Format:

```
      d = average(multiple-item [where condition]);
```

[] indicates that this part is optional.

Examples:

```
AVG_INCOME = average(INCOME);
AVG_FEMALE_INCOME = average(INCOME where SEX = 2);
```

Description:

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.

Count Function

The **count** function returns the number of occurrences for a repeating form or roster.

Format:

```
i = count(multiple-item [where condition]);
```

[] indicates that this part is optional.

Examples:

```
TOTAL_PERSONS = count(PERSONS);
NUM_CHILDREN = count(PERSONS where RELATIONSHIP = 3);
```

Description:

During data entry, the occurrence value is updated after the postproc of the first field within a repeating form or roster is executed. If the count function is executed prior to the form or roster, it returns 0. If it is executed from a field within the form or roster, it returns the current occurrence number. If it is executed after the form or roster, it returns the total number of occurrences in the form or roster.

During batch editing, count always returns the total number of occurrences in the multiply-repeating item/record.

If a **where** condition is included, the function returns the number of occurrences for which the condition is true.

If the **where** condition is not included, the count function and the noccurs function return the same result.

Return value:
The function returns an integer count value.

See also: Noccurs, Soccurs, Totocc, Curocc

Curocc Function

The **curocc** function returns the current occurrence number for a roster, form, or record.

Format:
```
i = curocc([group]);
```

[] indicates that this part is optional.

Examples 1:
```
PROC RELATION
if curocc(PERSON_REC) = 1 then
  if (RELATION <> 1) then
    errmsg("First person must be head of household.");
  endif;
endif;
```

Description:
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 it's sequence number within the group.

Return value:
The function returns an integer occurrence number.

See also: Totocc, Noccurs, Soccurs, Count , if, errmsg

Delete Function

The **delete** function removes a record or item occurrence from the current case.

Format:
```
b = delete(group[occ]);
```

[occ] is required for multiply-occurring records or items, and
is not required for singly-occurring records or items

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;
```

Description:

The **delete** function removes incomplete or otherwise unneeded records or item occurrences. It can be used to remove singly- or multiply-occurring records, though if the record is non-repeating (such as the housing record), 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.

See also:

if, totocc, while

Insert Function

The **insert** function creates a record or item occurrence in the current case.

Format:

```
b = insert(group[occ]);
```

[occ] is required for multiply-occurring records or items, and
is not required for singly-occurring records or items

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 property setting of required=yes.

Description:
The insert function inserts missing or otherwise needed data records or item occurrences. 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.

See also:
do, ifIf_Statement, totoccTotocc_Function, count

Max Function

The **max** function returns the maximum value of an item that occurs multiple times.

Format:

```
d = max(multiple-item [where condition]);
```

[] indicates that this part is optional.

Examples:

```
MAX_INCOME = max(INCOME);
MAX_FEMALE_INCOME = max(INCOME where SEX = 2);
```

Description:

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 maximum value.

See Also: Min

Min Function

The **min** function returns the minimum value of an item that occurs multiple times.

Format:
```
d = min(multiple-item [where condition]);
```

[] indicates that this part is optional.

Examples:
```
MIN_INCOME = min(INCOME);
MIN_MALE_INCOME = min(INCOME where SEX = 1);
```

Description:
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 minimum value.

See also: Max

Noccurs Function

The **noccurs** function returns the number of occurrences of a roster, form, or record.

Format:
```
i = noccurs(group);
```

Example:

```
TOTAL_PERSONS = noccurs(PERSON);
```

Description:
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.

noccurs is equivalent to the count function without the where phrase.

Return value:
The function returns an integer occurrence number.

See also: Totocc, Curocc, Soccurs, Count

Soccurs Function

The **soccurs** function returns the total number of occurrences of a record.

Format:
```
i = soccurs(record-name);
```

Example:
```
NUM_HH_MEMBERS = soccurs(PERSON_REC);
```

Description:
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.

During data entry 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 an integer value of the number of occurrences.

See also: Totocc, Curocc, Noccurs, Count

Sort Function

The **sort** function sort occurrences of records or items based on the value of an item.

Format:
```
b = sort(group using item);
```

Example:

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

Description:
The sort function sorts the multiple records or items in the specified group in ascending order 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.

Sum Function

The **sum** function returns the sum of an item that occurs multiple times.

Format:
```
d = sum(multiple-item [where condition]);
```

[] indicates that this part is optional.

Example:
```
TOTAL_INCOME = sum(INCOME);
TOTAL_FEMALE_INCOME = sum(INCOME where SEX = 2);
```

Description:
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 a **where** condition is included, the function returns the sum 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 of the sum.

Totocc Function

The **totocc** function returns the number of multiply occurring records or the number of multiply-occurring items in a group.

Format:
```
i = totocc([group]);
```

[] indicates that this part is optional.

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;
```

Description:

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.

See also: Curocc, Count, Soccurs, Noccurs

General Functions

Clear Function

The **clear** function initializes the memory values of data items defined in external files and working storage to zero or blank.

Format:

```
b = clear(item-1 [...,item-n]);
```

[] indicates that this part is optional.

Example:

```
OK = clear(WORKDICT);
```

Description:

The clear function assigns zeros to all numeric items, assigns blanks to all alphanumeric items, and sets the number of occurrences of records or items to 0 in items item-1 to item-n. Items item-1 to item-n can be external dictionaries, working storage dictionaries, or any component in these dictionaries.

Return value:

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

Errmsg (Display) Function

The **errmsg** function displays a message on the data entry screen or writes a message to the batch edit report.

Format 1:

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

[] indicates that this part is optional.
| indicates that one or the other keyword may be used.

Format 2:

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

[] indicates that this part is optional.
| indicates that one or the other keyword may be used.
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.

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);
```

Description:

The errmsg function displays a message to the user during program execution in a Data Entry application or writes a message to a listing (.lst) file 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.

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

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.

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 3 above, the output listing will show the number of times there was more than one head of household divided by the number of people 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 hit (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.

Special Function

The **special** function determines whether a variable's value is MISSING, NOTAPPL, or DEFAULT.

Format:

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

Example:

```
if special(XVAR) then
  YVAR = 99;
else
  YVAR = XVAR;
endif;
```

Description:

The special function checks if the value of variable-name is a special value (i.e., MISSING, NOTAPPL, or DEFAULT).

Return value:

The function returns a logical value of 1 (true) if the variable is a special value and 0 (false) otherwise.

See also:

Special Values, if

Sysdate Function

The **sysdate** function returns the current system date as an integer.

Format:

```
i = sysdate([date-format]);
```

[] indicates that this part is optional.

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
```

Description:

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 YYYYY 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.

Sysparm Function

The **sysparm** function returns the value of the 'parameter' variable stipulated in the data entry or batch edit pff file.

Format:

```
alphanumeric-var = sysparm();
```

Example:

```
PROC GLOBAL
alpha(30) MyParam;

PROC MyFile
preproc
MyParam = sysparm();
```

Description:

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.

Systime Function

The **systime** function returns the current system time as an integer.

Format:

```
i = systime();
```

Example:

```
TIME = systime();
```

```
HOUR = int(TIME / 10000);
MIN = int(TIME / 100) % 100;
SEC = TIME % 10000;
```

Description:

The systime function returns the 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.

Write Function

The **write** function write text to an write file.

Format:

```
[b =] write(string-exp[,p1[,p2[,...,pn]]]);
```

[] indicates that this part is optional.

Example:

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

Description:

The write function writes text to file that can be used as a report. Each parameter (e.g., p**1**) 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

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.

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.

External File Functions

Close Function

The **close** function closes a previously opened external file.

Format:
```
b = close(ext-dict-name);
```

Example:
```
OK = close(LOOKUP);
```

Description:
Under most circumstances neither an **open** or a **close** function is necessary to manipulate an external file. In data entry applications, by default, an external file is opened when it is operated on with an external file function, such as loadcase or writecase, and closed immediately afterward. In batch applications, by default, an external file is opened at the beginning of the run and closed at the end.

If you want to control the opening and closing of an external file, you can use the **open** and **close** functions to do this. If you code an **open** function anywhere is the application logic, then you must control ALL the opening and closing of the file

The open function opens the specified external file and leaves it open. The **close** function closes an open external file.

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 successful and 0 (false) otherwise.

See also:
open

Delcase Function

The **delcase** function marks a case for deletion in an external file based on a key.

Format:
```
b = delcase(ext-dict-name[,var-list]);
```

[] indicates that this part is optional.

Example:
```
OK = delcase(GNMR31,MCLUST,MHHNUM,MLINE);
```

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.

Find Function

The **find** function determines the existence of a case in an external file that matches a specified condition.

Format:
```
b = find(ext-dict-name,rel-op,alpha-ex);
```

Example:
```
OK = find(CODE,>=,"10100201");
```

Description:

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 dictionary name defined in the data dictionary 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.

See also:
locate

Key Function

The **key** function returns the key of the case at the current position in an external file.

Format:
```
s = key(ext-dict-name);
```

Example:
```
THE_KEY = key(LOOKUP);
```

Description:
The key function returns a string containing the key of the case in the current position in an external file.

The **ext-dict-name** must be supplied. It is the dictionary name defined in the data dictionary 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 functions returns a string containing the key. If no key is present, a null string is returned.

Loadcase Function

The **loadcase** function reads a specified case from an external file into memory.

Format:
```
b = loadcase(ext-dict-name[,var-list]);
```

`[]` indicates that this part is optional.

Example:

```
OK = loadcase(SAMPDICT,CLUSTER,HH);
```

Description:

The loadcase function reads a case from an external data 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 dictionary name defined in the data dictionary 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 the case in the external file 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 physical case in the file.

Return value:

The function returns a value 1 (true) if the case was loaded successfully, 0 (false) otherwise.

See also:

retrieve, writecase

Locate Function

The **locate** function finds but does not load a case in an external file that matches a specified condition.

Format:

```
b = locate(ext-dict-name,rel-op,alpha-ex);
```

Example:

```
OK = locate(CODE,>=,"10100201");
```

Description:

The locate function searches the index of an external file and finds the first case that matches the specified condition. The file is positioned to the case's location in the file, 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 dictionary name defined in the data dictionary 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 logcial value of 1 (true) if a case is found and 0 (false) otherwise.

See also:
find

Open Function

The **open** function opens and keeps open an external file.

Format:
```
b = open(ext-dict-name);
```

Example:
```
OK = open(LOOKUP);
```

Description:
Under most circumstances neither an **open** or a **close** function is necessary to manipulate an external file. In data entry applications, by default, an external file is opened when it is operated on with an external file function, such as loadcase or writecase, and closed immediately afterward. In batch applications, by default, an external file is opened at the beginning of the run and closed at the end.

If you want to control the opening and closing of an external file, you can use the **open** and **close** functions to do this. If you code an **open** function anywhere is the application logic, then you must control ALL the opening and closing of the file

The **open** function opens the specified external file and leaves it open. The close function closes an open external file.

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 file is opened and 0 (false) otherwise.

Retrieve Function

The **retrieve** function reads into memory a case from the current position of an external file.

Format:
```
b = retrieve(ext-dict-name);
```

Example:
```
OK = retrieve(LOOKUP);
```

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.

See also:
loadcase

Writecase Function

The **writecase** function writes a case from memory to an external file.

Format:
```
b = writecase(ext-dict-name[,var-list]);
```

[] indicates that this part is optional.

Example:
```
OK = writecase(KIDS,CLUSNUM,HHNUM,LINE);
```

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 dictionary name defined in the data dictionary 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.

See also:
loadcase

Files

File Types

Data Entry Applications consist of the following files:

- Data Entry Application file (.ENT) specifies all other files contained in the data entry application and includes other application information.
- Forms file (.FMF) specifies the data entry forms. There is usually one form 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) contains text for messages displayed during data entry.
- Help file (.HPF) is reserved for future use.
- Other Data Dictionary files (.DCF) is optional, representing secondary data files (such as lookup files) which are read and/or written to during data entry.

Batch Edit Applications consist of the following files:

- Batch Edit Application file (.BCH) specifies all other files contained in the batch edit application and includes other batch edit application information as well.
- Edit Order file (.ORD) specifies the order in which the logic in the application is executed. An edit order file (.ORD) contains one data dictionary file (.DCF) which represents the primary data file that is being edited modified.
- Logic file (.APP) contains CSPro language statements.
- Message file (.MGF) contains text for messages displayed during data entry or batch editing.
- Other Data Dictionary files (.DCF) is optional, representing secondary data files (such as look up files) which are read and/or written to during batch edit.

Cross Tabulation applications consist of the following files:

- Cross Tabulation Application file (.XTB) specifies all other files contained in the cross tabulation application.
- Cross Tabulation Specification file (.XTS) contains variable names and other parameters which define the tables. The file also names the associated data dictionary file.
- Logic file (.APP) is reserved for future use.
- Message file (.MGF) is reserved for future use.
- Help file (.HPF) is reserved for future use.

- Data Dictionary file (.DCF) contains the physical format of the data file(s) to tabulate.

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.

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.

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.

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.

Form File (.FMF)

- The forms file contains information about forms, their fields, text and rosters. The forms file also contains the name of the associated data dictionary file. Fields and rosters have links into the data dictionary.

- The flow during data entry, that is, the order in which forms and fields are entered, is defined in the forms file, not in the data dictionary.

- CSPro allows you to explicitly open, close and save forms files independently of the application file. When you do so, the associated data dictionary file is also opened, closed or saved.

- Note that if you open a forms file you will not have access to its application's logic. Generally, only advanced users open forms files explicitly.

- The forms file is 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 name of the associated data dictionary file.

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.

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.

Logic File (.APP)

- The logic file contains all the CSPro language statements which control the application. There is one logic file associated with each application.

- CSPro does not allow you to explicitly open the logic file. It is opened only when you open its associated application.

- By default, the logic file has the same name as the application file, just a different extension. This is not a requirement, however. Advanced users who change the name of this file must also remember to change the corresponding name in the application file.

- The logic file is an ASCII text file which may be viewed with any text editor, such as CSPro's Text Viewer or the Windows Notepad. While you may make changes to this file outside the CSPro environment, CSPro provides a powerful text editor which is integrated with the CSPro compiler.

Messages File (.MGF)

The message file is a text file where you can store message text and an associated message number. For more information see Message File.

Helps File (.HPF)

- The help file contains information related to **CAPI** (**C**omputer **A**ssisted **P**ersonal **I**nterviewing) data entry applications. Such information includes question text to appear on the screen with each field and help screens to appear when the operator presses the help (**F1**) key.

- The CSPro interface does not fully support CAPI applications in this release.

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

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 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.)

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 2.4

[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`

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.

Index

o

!

$

%

&

*

.

/

^

|

+

<

=

>

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z