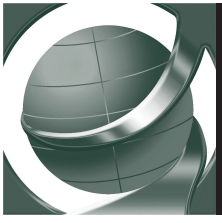
Data Conversion System and Viewer User's Guide

Release 7.3



SUNGARD iWORKS PROPHET

Actuarial Calculations

SunGard
Prophet
Sherwood House
Eastworth Road
Chertsey
Surrey
KT16 8SH
United Kingdom

Telephone:	+44 (0)1932 757575
Prophet Helpdesk:	+44 (0)1932 757555
Fax:	+44 (0)1932 757518
Email:	prophet.helpdesk@sungard.com
Web Site:	www.sungard.com/iWORKSprophet

Document No. PR730-3003-0206

Contents

Introduction

About this manual

This manual is the Data Conversion System and Viewer User's Guide. It provides Prophet and Glean users with:

- A description of the main features of the Data Conversion System (DCS).
- A description of the main features of Viewer.
- A tutorial that takes you through the steps involved in using DCS to create model point files for use in Prophet.

  If you will be using DCS to create data files for Glean you can still use this tutorial to gain a broad understanding of the facilities in DCS. The examples supplied with Glean also include example DCS programs which demonstrate how DCS can be used to create data files for Glean.
- Reference sections which provide additional information to help you use DCS.

How this manual is organised

This manual is organised in three parts:

- **Part 1** - Using DCS and Viewer
- **Part 2** - Data Conversion System Tutorial
- **Part 3** - Reference Sections

The first part describes how DCS works and the procedures involved in setting up and running a DCS program. It also explains how to use Viewer to look at DCS input and output files.

Part 2 incorporates the Data Conversion System Tutorial. This consists of a number of lessons, each of which can be regarded as being separate. You can take a break at the end of each lesson. However, the lessons need to be tackled in the specified order as each lesson uses and depends on the results from the previous lessons.

The final part contains reference sections that provide useful information for users of DCS.

Part 1 - Using DCS and Viewer

This part of the manual contains general details of how to use DCS and Viewer. It is recommended that you read this part of the manual to familiarise yourself with these applications, particularly if you are new to them.

Part 2 - Data Conversion System Tutorial

This part of the manual contains the Data Conversion System tutorial which consists of the following lessons:

1. **Setting up and running a simple DCS program**
   In this lesson you will set up a simple program to read a fixed format input file containing policies for two product types and produce output model point files for each of them. You will run it and look at the output files you have created.
2. **Validation and correction**
   In this lesson you will add validation and correction routines to the program you created in Lesson 1.

3. **Grouping in DCS**
   In this lesson you will modify your program so that it produces model point files containing grouped policy data rather than individual policy data.

4. **Further DCS topics**
   This lesson covers a number of further topics. You will enhance your program to use explicit declarations, produce summarised results, read values from a generic table and create ODBC compliant output files. You will also create a new program which produces a model point file for use in product design work.

5. **Multiple record formats**
   In the earlier lessons the input file that your program processed only contained a single record format. In this lesson you will create a new program to process an input file containing 3 different record formats.

6. **Units numbers and values**
   In this lesson you will enhance the program you created in the previous lesson to calculate the unit values for unit linked business using the numbers of units held in the input file and a set of prices held in a unit prices file.

7. **Model Point Files for Analysis of Movements**
   In this lesson you will create a model point file to be used in an Analysis of Movements run by comparing two input files.

8. **DCS program listings**
   This section contains listings of the DCS programs that you set up during this part of the tutorial. You can compare the programs that you create with these copies to help you diagnose any problems that may arise during the tutorial.

Pre-requisites

Before you tackle this tutorial you are expected to have:

- A basic knowledge of Windows
- Release 7.3 of DCS installed on your PC

For information on how to install DCS please refer to the "Installing Prophet" chapter of the Prophet User's Guide or the "Installing Glean" chapter of the Glean User's Guide.

Part 3 - Reference Sections

The reference sections contain information which will help you to use DCS. The sections are:

Section A - Input file types supported by DCS

Section B - Summary of DCS functions

Section C - Differences between the data grouping functionality in DCS and Prophet

Section D - Data grouping considerations and comparison checks

Section E - Guidelines for data grouping of particular life product types

Section F - Information on model point files

Section G - Procedures, functions and modules

Part 1 - Using DCS and Viewer

The first chapter in this part describes how DCS works and the procedures involved in setting up and running a DCS program. The second chapter explains how to use Viewer to look at DCS input and output files.

CHAPTERS IN THIS PART

CHAPTER 1

The Data Conversion System (DCS)

What is the Data Conversion System?

The Data Conversion System (DCS) enables you to convert external data files into files that can be used by Prophet, Glean or other applications. It has its own programming language and functions that allow you to specify the conversion process, including validation and correction routines.

You can program DCS to create one or more:

- Individual policy model point files which can be used in Prophet for individual policy calculation runs or as input for data grouping runs
- Grouped model point files that can be used in Prophet grouped data runs
- 2 Dimensional or Column format generic tables that can be used by Prophet
- Glean data files that can be used for experience analysis investigations
- Text, Microsoft Access and ODBC (Open Database Connectivity) format files

In addition, if you create model point files to be used in Prophet then DCS can use an associated Prophet Release 7.2 or later workspace to validate the values for the model point variables that have constraints applied to them in the workspace.

How is DCS used?

To convert an input file you need to create a DCS program that specifies the conversion process and the output files which are to be produced. You can then test the program and finally use it to convert and create the required output files in the specified format.

When you have developed a DCS program you can save and subsequently reload it and run it any number of times.

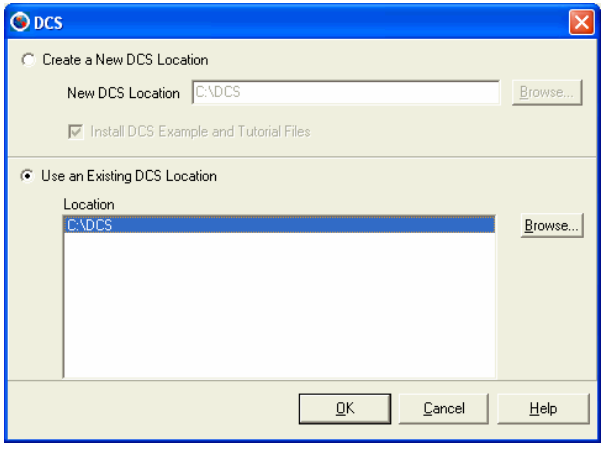
The input files can be in fixed format (ASCII or EBCDIC), delimited format, Model Point file format, Microsoft Excel format, Glean data format or in a database format such as Microsoft Access.

You can also use DCS to create model point files without the need for any input data. This option is useful for creating Prophet model points for a range of ages, policy terms, sex, smoker status, etc when goal seeking for a set of premium rates.

Managing DCS programs

A DCS program contains the information and instructions required to create model point files, Glean data files, text, delimited, Microsoft Access or ODBC files from your policy data. You create, save and manage all DCS programs using the DCS interface.

When you start DCS the following dialog appears. It allows you to select the location in which your DCS programs are held:

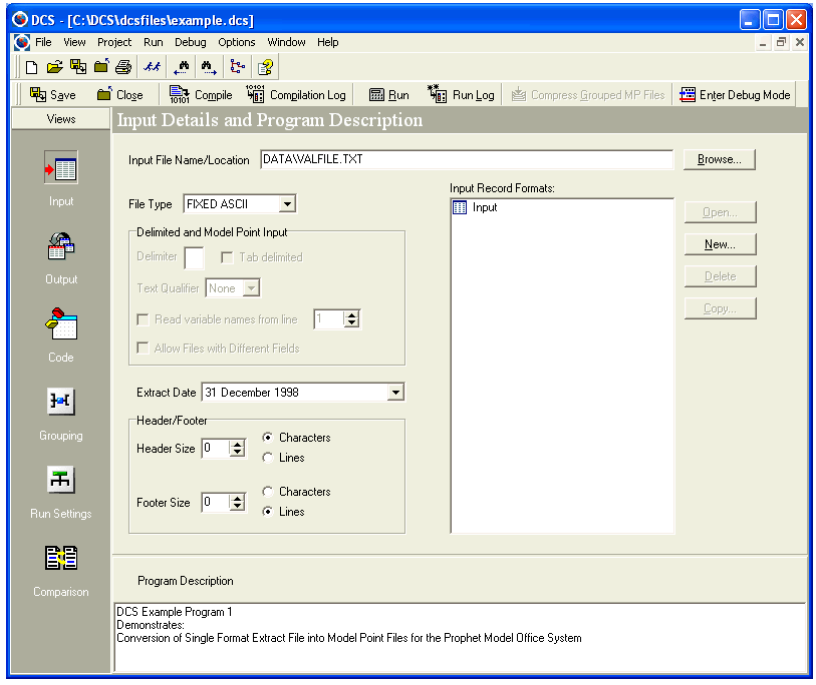


To open an existing location select **Use an Existing DCS Location** and then either select the location from the list or use the **Browse** button. To create a new location select **Create a New DCS Location**. If you want to install the example DCS programs so that you can look at how DCS programs should be set up or you want to run the DCS tutorial select "Install DCS Example and Tutorial Files".

Click **OK** and the DCS Program Selection dialog appears from which you can select an existing DCS program. Alternatively you can click **Cancel** and choose **New** in the **File** menu and then select **DCS Program File** to create a new program.

Note: If you launch DCS from within Glean this dialog is not displayed since the DCS locations are specified within Glean.

The Input Details view of the DCS program window then appears as follows:



From this view you specify the input files to be read by the program including details of their format. You also enter a description of the program. In the other views you can:

- Specify the output files to be created in the Output Details view
- Enter the DCS program code that specifies the conversion process in the Code view
- If you will be creating grouped output files, specify the rules for calculating the grouped values are specified in the Grouping view. For example, a policy record can create many exposure records and so it can be easier to analyse grouped policy data rather than individual policy data.
- Specify the DCS run settings in the Run Settings view.
- Carry out a comparison with another DCS program in the Comparison view so that you can see the differences between the two programs.

When you have specified and saved the DCS program you can then run it by clicking **Run**.

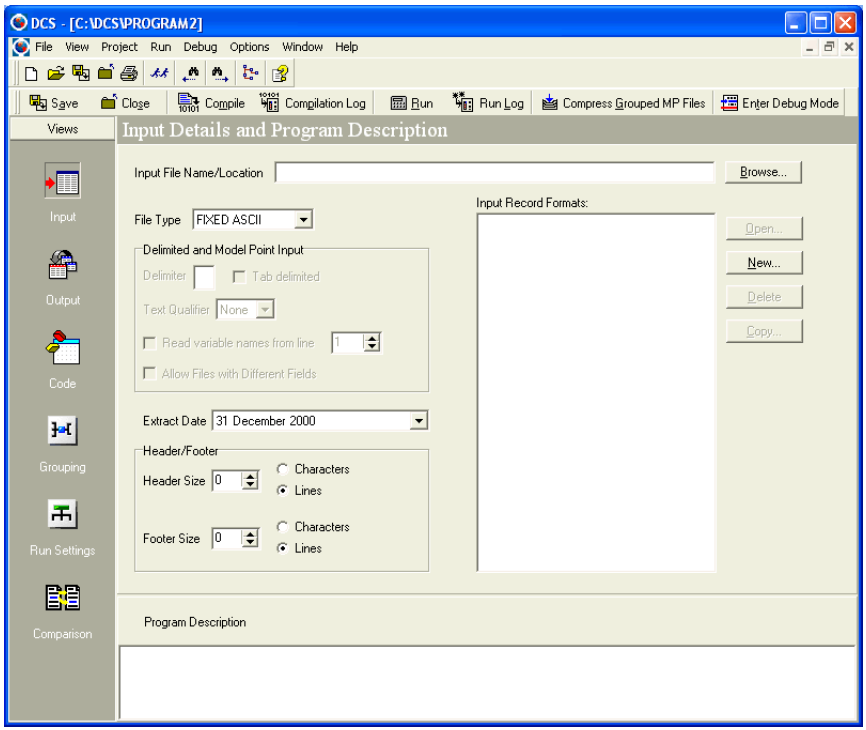
Creating a new DCS program

Always plan your data conversion needs before you start. When you have a clear idea of what is required, start DCS and create a new program by choosing **New** from the **File** menu and then selecting **DCS Program File**.

Next, enter a description of the program. Then enter the input and output file details as explained in the following sections.

Entering input files details

Input file details are entered in the Input Details view of the DCS program window. It appears similar to the following:



In this view you specify the following:

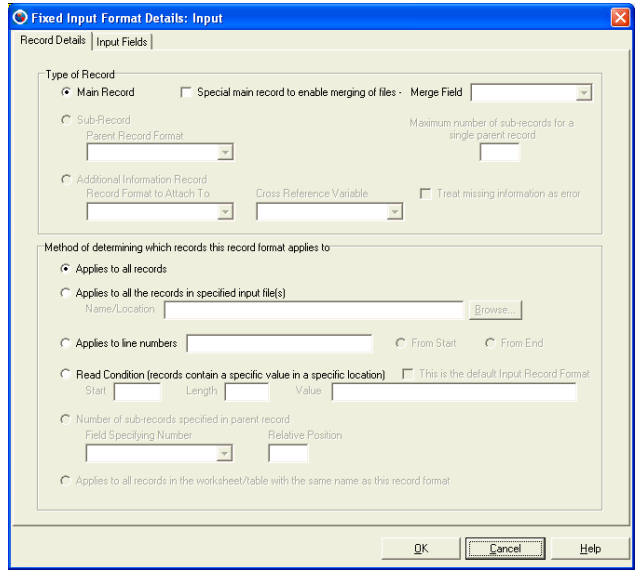
- The name and location of the input files. If the input file is in a database format then a data source is specified instead of a file name. If there are several data files with different formats then the names and locations are specified in the input record formats.

- The file format of the data input files being converted. The input files can be fixed format files, in ASCII or EBCDIC format, delimited format files such as a comma, space or tab delimited file, a Microsoft Excel file, a Model Point file, Glean data format or a database format file.
- The date for which the input files were extracted.
- Whether or not the input files have a header and/or footer. If so, the size of the header and/or footer can be specified in characters or lines.
- The format of the records in the input files. You must add a record format for each different type of record in the input files. The information you need to enter depends on the format of the input files.
- A description of the DCS program in the Program Description box.

Record format details

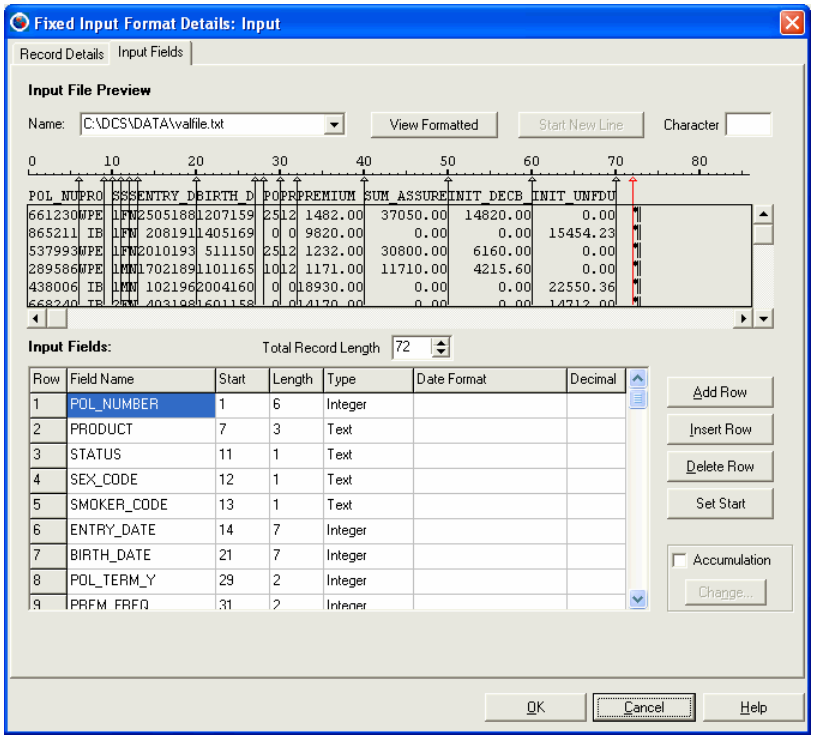
The record formats of the input files are listed and specified in the Input Record Formats list in the Input view of the DCS program window. When you click **New** to specify a new format or select a format from the Input Record Formats list and click **Open** the Format Details dialog is displayed.

Depending on the file format of the input files different options are available. For a fixed format file the Format Details dialog appears similar to the following:



From the Record Details tab you specify the type of record and the method of determining which records the record format applies to.

The fields in the input record format are displayed in the Input Fields tab which for a fixed format file appears similar to the following:



From the Input Fields tab of the Formats Details dialog you can specify or edit the fields for a record format.

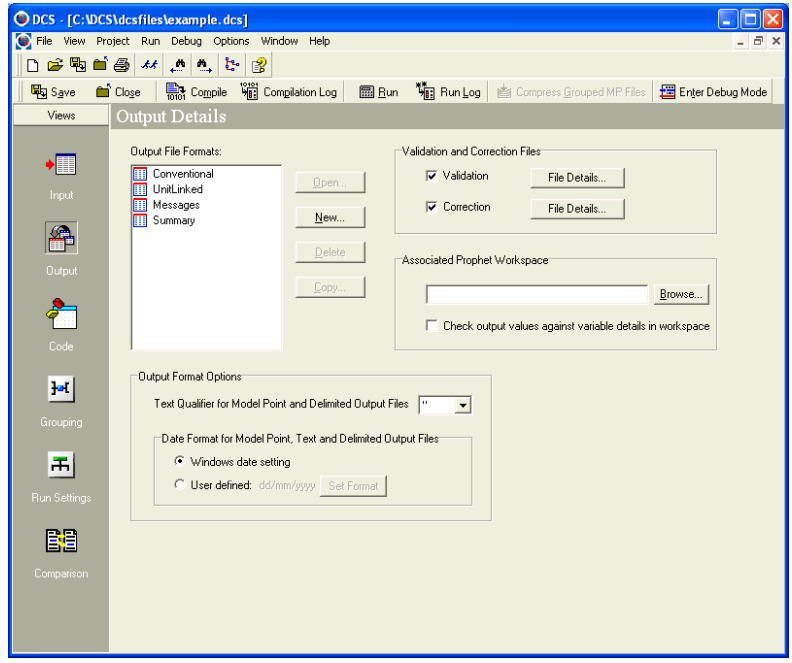
- **For fixed format input files**
  It specifies the total record length and the name, start, length, type, date format and number of decimals for each field to be read.
- **For delimited, model point, Excel, Glean and database format input files**
  It specifies the name, type, date format (date fields only), length (text fields only) and number of decimals (number fields only) for every field even if some fields will not be used (except for model point files for which you only need to include the fields you want to use).

You can also accumulate numeric amounts over successive records in an input file. For example, it can accumulate records that have the same policy number. If you want to accumulate records, you must specify in the record format which variable you want to accumulate on and which variables you want to accumulate.

The top part of the dialog displays a preview of the data in the input file. This is updated as you enter the field information so that you can check that this is consistent with the actual data in the file.

Entering output file details

Output file details contain the information that DCS needs to know about the output files you want to create. They are selected and entered in the Output Details view of the DCS program window. It appears similar to the following:



There are several tasks to perform when entering the output file details:

- Add an output file format for each type of output file the DCS program will create.

- Set the validation message file if your code uses the INVALID or TABLE_VALIDATE functions and you want DCS to write the validation messages to a separate file rather than to the run log.
- Set the correction message file if your code uses the CORRECT or TABLE_CORRECT functions and you want DCS to write the correction messages to a separate file rather than to the run log.

You can also optionally specify the name and location of a Prophet workspace. By specifying the workspace you can validate the values of the variables generated by the DCS program against the constrained variable values contained in the workspace.

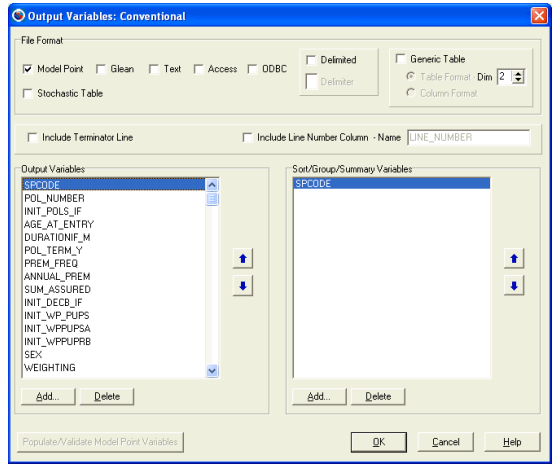
If you select the "Check output values against variable details in workspace" then DCS will check the values at runtime of every variable in the output format that has a constraint specified in the Prophet workspace.

If errors occur during the validation process a warning message is written to the runlog and, if specified to a .ERR file, for each value in every output record that does not conform to the constraints.

Output file formats

The Output file formats specify the format (Prophet model point, Glean data file, text, delimited, Microsoft Access. ODBC or Prophet Generic Table format) of the output files that you want your program to create, the variables that you want DCS to include in the output files and the variables, if any, by which the output records are to be sorted, grouped or summarised on.

When you create or edit an output file format by clicking **New** or **Open** in the Output Details view of the DCS program window, DCS displays the **Output Variables** dialog box from which you choose the variables and the output file formats. It appears similar to the following:



You must add an output file format for each different type of output file your program will produce. For example, if your input file contains records for both conventional and unit linked products, you could create an output file format containing certain variables from the conventional product records and another output file format containing certain variables from the unit linked product records. To do this, you would add one output file format for the conventional product records and another for the unit linked product records.

Also, if DCS writes validation messages to an output file (instead of the run log), then you need to add an output file format to specify which variables from the invalid records should be included in that output file. Similarly, if DCS writes correction messages to an output file, you need to add an output file format to specify which variables from the corrected records you want to be included in that output file. You can use the same output file format for both validation and correction message files.

If you are creating Glean data files you also need to select the Data Class for each variable from the following three options:

Data Class	Description
Analysis Factor	This option is selected if the variable is used to analyse the data. For example, it should be used for data items such as age, sex, vehicle group, product type and calendar year.
Value	This option is selected if the variable contains information that is to be analysed. For example, it should be used for claim numbers, claim amounts, premiums and exposure. This option cannot be selected if the data type is "Text".

Data Class	Description
Identifier	This option should be selected if the variable is used to identify each record. For example, it should be used for policy numbers.

Validation and correction messages

If a program uses functions that validate the data, then DCS generates a validation message each time one of these functions is called. Similarly, if a program uses functions that correct the data, then DCS generates a correction message each time one of these functions changes data when you run the program.

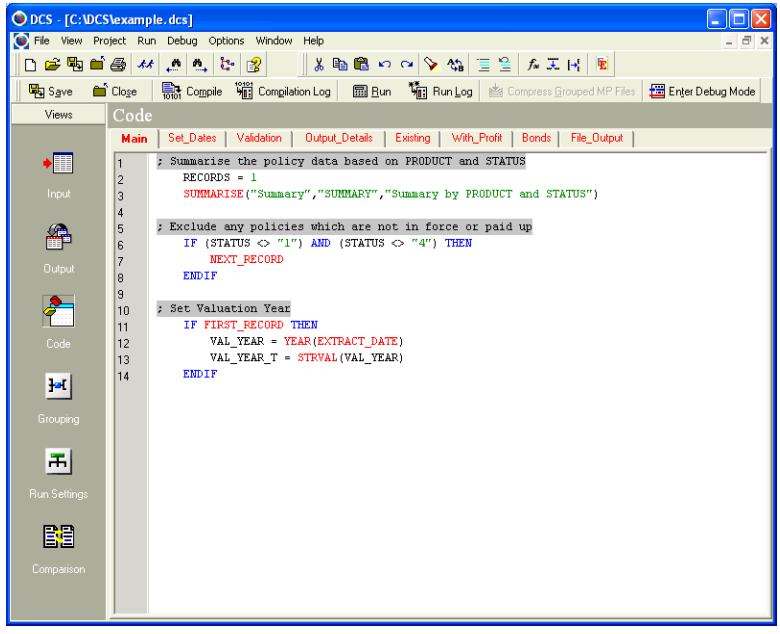
DCS can write validation and correction messages to the run log or to other files of your choice. If you don't specify a file for the validation or correction messages, DCS writes those messages to the run log.

Entering the program code

The code contains the instructions that you want DCS to follow to produce the output files. DCS has many built-in functions and statements that can perform:

- Specific calculations and data checks
- Read external tables
- Produce output results

The code is entered in the Code view of the DCS program window which appears similar to the following:



The code can be broken down into segments which appear on separate tabs. You can also create your own procedures and functions.

DCS functions and statements can be nested and can be used as arguments for other functions and statements. Note that a nested function or statement must return the appropriate type of data for the function or statement in which it is nested. You must also supply the necessary arguments for the nested function or statement.

Note: For information on how to write DCS procedures, functions and modules refer to Section G in the reference part of this manual.

Entering DCS functions, variables and statements

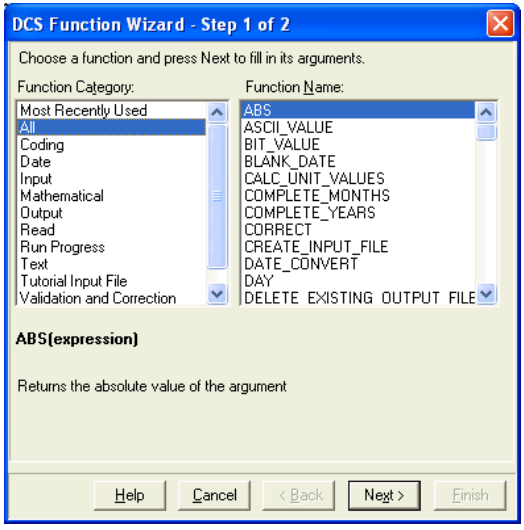
You can enter the DCS functions, variable names and statements directly into the code. Alternatively, you can use:

- The Function Wizard to select and insert DCS functions into your code
- The Insert Variable facility to select and enter variables into your code
- The Insert Statement facility to select and enter statements into your code.

All of these options are available from the code editor toolbar.

DCS Function Wizard

You can use the DCS Function Wizard to enter DCS functions into your code. To invoke it place the cursor in the code where the new function is to be inserted and either press Shift+F3 or click the Function Wizard toolbar button. The DCS Function Wizard appears similar to the following:



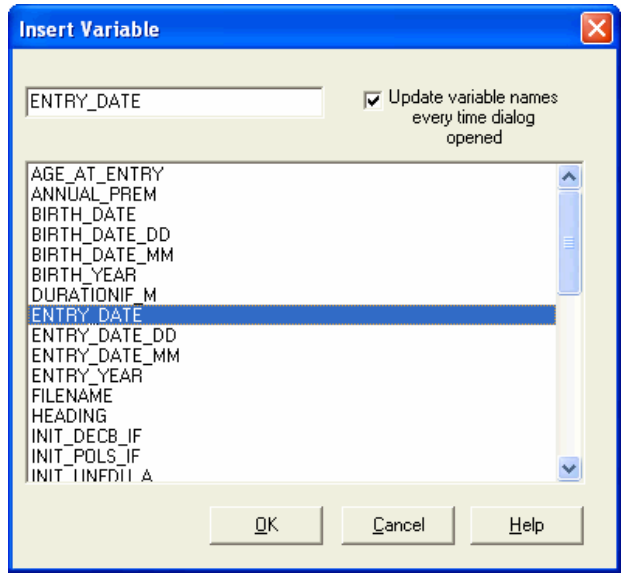
Select the function category and function name you require and then click **Next**. The Function Wizard then prompts you to enter any required arguments. For assistance and information on the selected function click **Help**.

Click **Finish** to enter the new function and its arguments into the code. If you do not enter any arguments and click **Finish** then you can subsequently enter them directly into the code.

Note: A summary of all the available DCS functions is provided in Section B of Part 3.

DCS Insert Variable facility

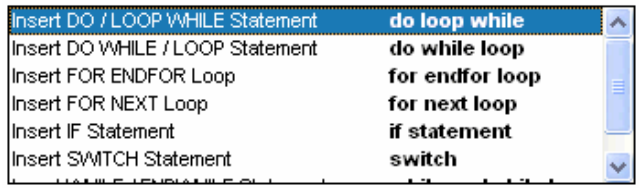
You can use the Insert Variable facility to enter variable names into your code. To do this, either place the cursor in the code where the new variable is to be inserted or type in the first few letters of the variable name. Then press F3 or click the Insert Variable toolbar button. A list of variables is then displayed similar to the following:



If you entered the first few characters of the variable name then the name of the first variable starting with those letters is highlighted. Select the required variable and click **OK**.

DCS Insert Statement facility

Instead of manually entering a statement such as a FOR NEXT loop in your DCS code you can use the Insert Statement facility. To do this you click the Insert Statement button from the Code Editor toolbar. You can then select and insert at the current cursor position the required outline statement from a list of available types. A list of statements appear similar to the following:



When you select and double click the required statement in the displayed list it is entered at the current cursor position. You can then update the outline statement as required.

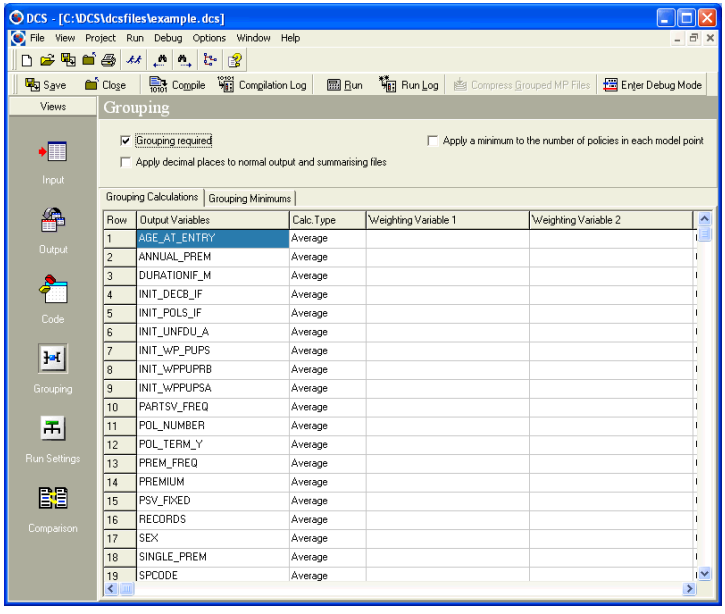
Grouping calculations

Grouping is the process by which sample policies are chosen to represent the individual polices. Policies with similar characteristics in terms of age, policy term, duration in-force, and so on, are grouped together to form these sample policies, which are known in Prophet as model points. If you are carrying out an experience analysis in Glean then you can group all the policies based on the Analysis Factor variables to create a smaller file which is quicker to analyse.

When grouping policies, you can calculate the weighted average age, policy term, duration in-force, and so on, in order to achieve a good model.

If you want to create a grouped output file, you must specify how the values should be calculated for the variables in that grouped output file.

This information is entered in the Grouping view of the DCS program window and appears similar to the following:



In the Grouping view you specify how the values are calculated for each output variable. You can:

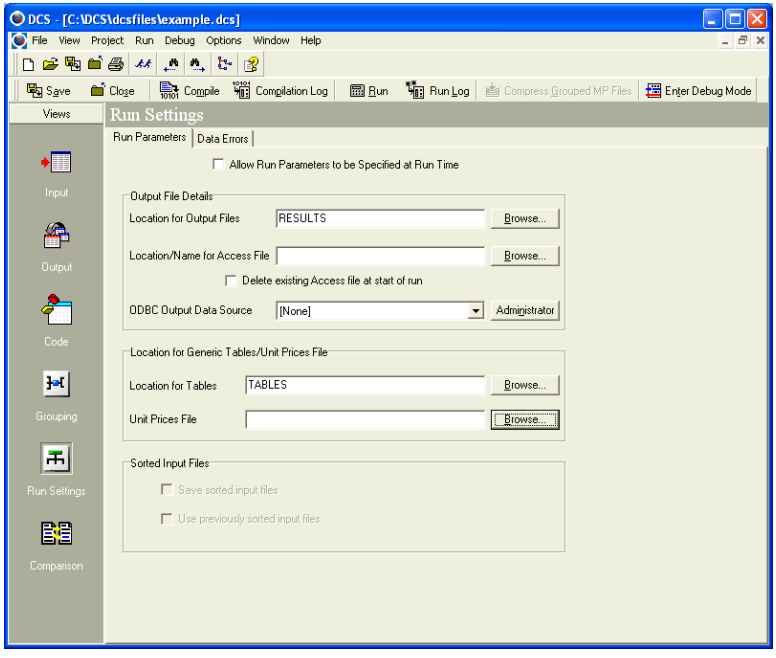
- **Select the Calculation Type to be applied**
  This can be **Average** to output the average value within the group, **Sum** to output the sum of the values within the group or **SumLog** to output the sum of log values within the group.

- **Set a Weighting Variable 1**
  The first weighting variable is selected from the list of variables in the program.
- **Set a Weighting Variable 2**
  The additional weighting variable is also selected from the list of variables in the program.
- **Set the number of Decimal places**
  This is selected from the dropdown list as required.

By using the grouping option to create a grouped output file you can specify the weighted average age, policy term, duration in force and so on which can help you to achieve a good model.

Setting up and running a DCS program

When the input data files, output data files and DCS program have been specified you then specify the run settings from the Run Settings view by clicking **Run Settings** in the **Views** list. The Run Settings view appears similar to the following:



You specify input and output folder locations and their associated options in the Run Parameters tab. Error processing is specified in the Data Errors tab.

Options in the Run Parameters tab enable you to specify:

- Whether a dialog is displayed at run time that allows the Run Parameters to be changed.
- The folder location in which the output files are to be created.
- The name and location of any Microsoft Access file which is to be created. Also whether any existing Access file should be deleted at the start of the run.
- Which ODBC data source to use (you only need to specify this information if one or more of the output formats specify that ODBC format outputs are required).

- Where to find the tables or unit prices file, if appropriate.
- Whether the sorted input files created during the run are retained. Also whether to use the sorted input files created in a previous run rather than the original unsorted files, thereby reducing the run time.

Options in the Data Errors tab enable you to:

- Specify the number of errors that can occur during a run before it should be stopped.
- Ignore any blank lines in the input file.
- Treat empty trailing fields as a zero or blank.
- Ignore records not matching any read condition.
- Use only the integer part of a number value in integer fields.
- Substitute a specified value for numeric fields containing specified text.
- Create additional Boolean variables that have the same names as the input variables but with the suffix "_missing_value" and which return TRUE if an input variable contains a missing or incorrect value.
- Write any input lines that cause an error to a .ERR file. This file is created in the same location as the DCS program file.

When you have specified the runs settings you save the DCS program and run it by clicking **Run** from the DCS program window toolbar.

Note: You can also run more than one DCS program by setting up options in the Multiple Run dialog which is accessed by choosing **Multiple Run** from the **Options** menu.

When the run has completed you can display the run log by clicking **Run Log** and the compilation log by clicking **Compilation Log**.

The generated output files can also be viewed using Viewer by clicking  on the toolbar.

What happens during a DCS run

When a DCS program is run for the first time it is automatically compiled by DCS. If any changes are made subsequently to the input or output details, the code, the grouping details or the run settings information then DCS recompiles the program when it is next run.

When DCS compiles a program it checks that the input details, output details and code are free from errors before running the program. If an error is detected DCS stops the run and displays the error. When the error is corrected the program can be run again. This process is repeated until all the errors are corrected. Note that DCS cannot complete a run until a program is free from such errors.

If you have selected the "Allow Run Parameters to be Specified at Run Time" option in the Run Settings view then the Run Parameters dialog will be displayed at the start of the run. This allows you to modify the settings for the run parameters from those specified in the program.

DCS then creates one or more output files by:

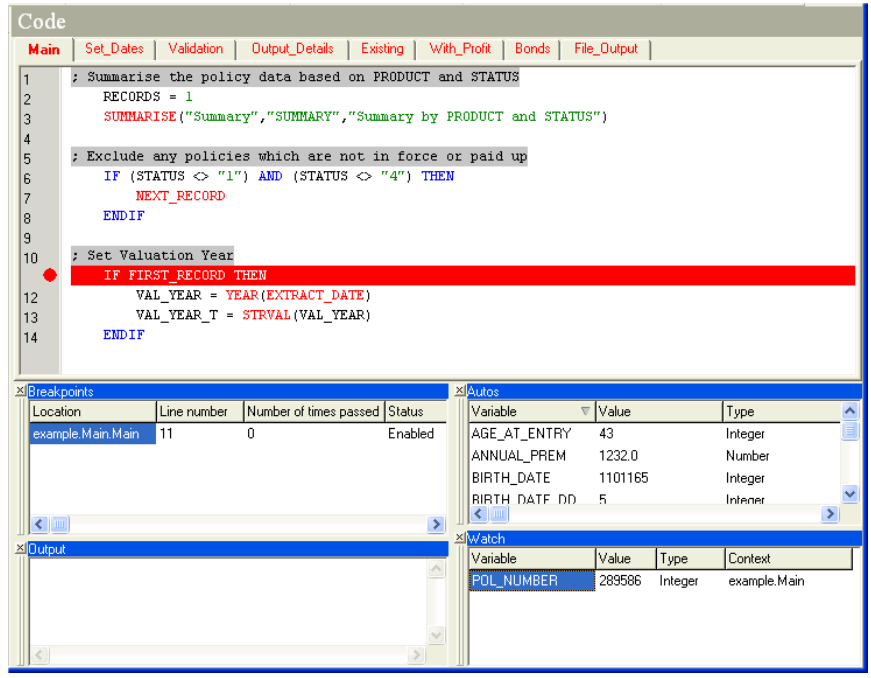
- Following the instructions in the code
- Using the run settings information
- Using the supplied input file(s) and tables

DCS is supplied with the Borland C++ compiler. However, faster run times can be achieved by licensing the optional Intel C++ compiler. You select which compiler to use by selecting **Compiler** in the **Options** menu.

Debugging DCS code

Using the DCS debugger you can run a DCS program and identify data input errors that can cause your program to fail or to spot failings in program logic.

The DCS debugger is displayed in the Code Window when you click **Enter Debug Mode**. The debugger appears similar to the following:



The debugger works in a similar manner to that of the VBA debugger in Microsoft Excel and has a toolbar from which you can control the main debug features.

From the debugger toolbar you can:

- Start, pause and stop a DCS debug session.
- Control the progress of a DCS debug session by stepping into, over or out of a section of code.
- Run and stop DCS code at the location of the cursor.

- Display any current active procedures or functions calls that have started but have not yet been completed.
- Manage the layout and visibility of the debug windows.
- Add a variable to the Watch window whose value you wish to monitor.
- Set or reset a breakpoint in the code. You cam also enable or disable all breakpoints in the program.

To support the debug session the following additional windows are available:

- **Breakpoints window**
  Allows you to manage the breakpoints that you set in the DCS code. Breakpoints are lines in the code that allow you to suspend code execution so that you can examine and change the values of variables or commence single stepping of the code.
- **Autos window**
  Lists all the variables used by the current function or procedure by name, value and type. The values in the window are updated only when program execution is suspended.
- **Watch window**
  Lists the variables that you have selected and whose values you wish to monitor. The variables are listed in the window by name, value, type and context. During code execution the values are continuously updated. This differs from the Autos window in which the values of the variables are updated when code execution is suspended.
- **Immediate window**
  Displays any messages that are sent to the run log file. In particular, you can use the MESSAGE function to display values of variables that you wish to examine.

You can hide or show any of the debug windows by choosing the appropriate option from the View menu or the Debug toolbar.

When you run DCS code in the debugger it is compiled to include debug information. You can then step through and make code changes as required in the debugger. Any changes made to the DCS code are saved when you exit the debugger. When the DCS code is subsequently recompiled outside of the debugger none of the debug information is included.

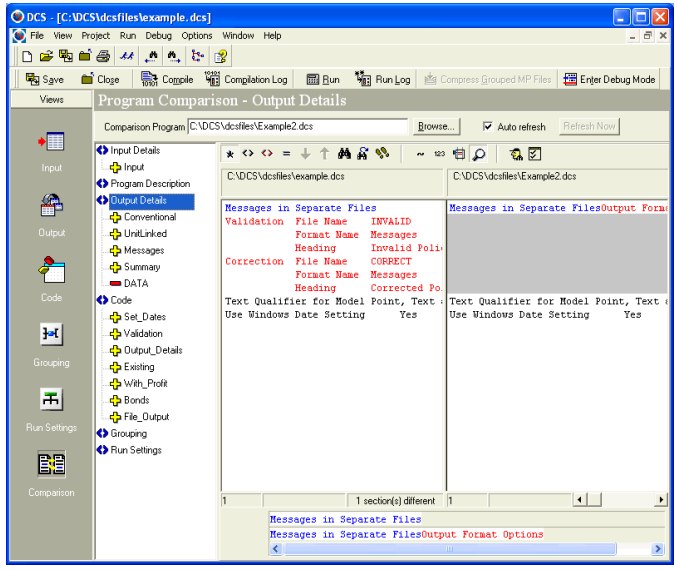
For more information on how to use the DCS debugger please refer to the online help.

Comparing DCS Programs

Comparisons between DCS programs can be produced that highlight any differences in:

- The input and output record format details.
- Program code and program description.
- Grouping details.
- Run Settings.

To compare these items you select the Comparison view and enter or browse to the DCS program you wish to compare with. The Comparison view appears:



Icons are used to indicate those aspects of the two DCS programs that differ. The differences are colour coded in separate panes. Comparison results can be printed or saved to a file with options to print only selected items, changed items, new items, deleted items or unchanged items.

If "Auto refresh" is selected then the comparison is updated each time this view is visited. Alternatively, you can manually refresh the comparison by not selecting the "Auto refresh" option and clicking **Refresh Now**.

Note: You can also use the compare feature to view and print textual listings of the current DCS program by not specifying a comparison program.

Viewing DCS files and logs

You can view the following files used by a DCS program:

- Input file(s)
- Output files
- Compilation log. The compilation log contains the messages that were generated the last time the program was compiled.
- Run log. The run log contains information about the most recent program run, including the run settings details, the files and tables used, the output files produced and the messages generated during the run.
- Error file. This is created only if you have selected the "Write input lines that cause an error to .ERR file" option in the Data Errors tab of the Run Settings view. It contains details of the errors encountered during the reading of the input files, including the actual records that caused the errors.

The input and output files are viewed using the Viewer application. The compilation, run and error logs are viewed from within DCS.

CHAPTER 2

Viewer

TOPICS IN THIS CHAPTER

What is Viewer?

Viewer is a separate application that can be used to view:

- Prophet model point files.
- Glean data files.
- Data Conversion System input and output files.
- Files created using the Prophet Table Editor, such as global, parameter, unit price files, generic, mortality, year dependent mortality and year index tables.
- Text files.
- Intermediate files created in Prophet dynamic runs

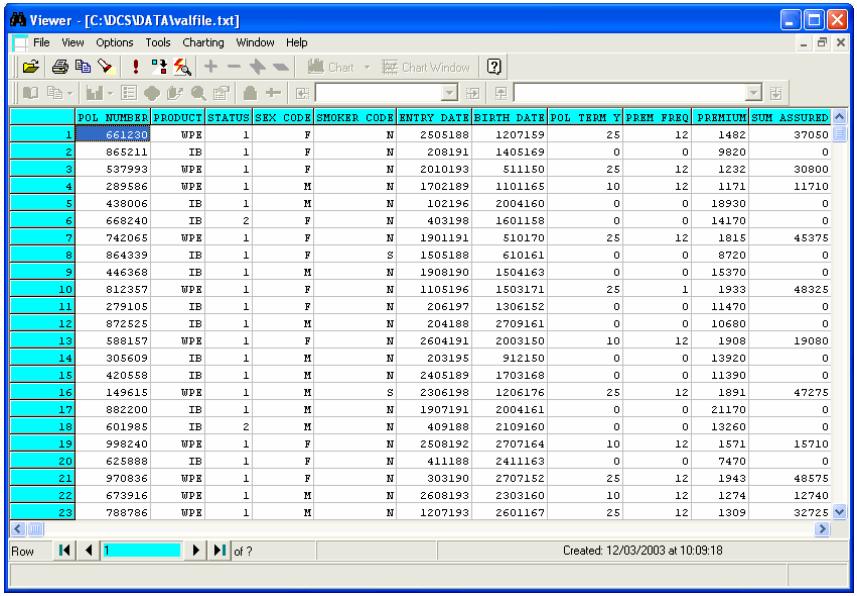
Viewer also has an analysis facility that enables you to investigate the data in an input file, a model point file or a Glean data file. In particular:

- Information about each variable can be displayed, such as the minimum, maximum, average and sum of its values.
- All the values which exist for particular variables can be displayed, including the frequency with which they exist in the data file.
- The average and sum of the values for a variable can be displayed for each of the values of another variable. For example, you can see how the average premium varies with the sex of the policyholder and term of the policy.
- The data in the file can be filtered so that the analysis only covers a sub-set of the data.
- The results of the analysis can be displayed in a chart as well as in a grid.

Viewer has been specifically developed to be used in conjunction with DCS to view what can potentially be very large files in a convenient way. You can open Viewer by selecting View Input File or View Output Files from inside DCS. You can also open it directly using the Viewer option in the "Prophet 7.3" or "Glean 7.3" program groups.

Using Viewer

A DCS input file displayed in Viewer looks similar to the following:



	POL_NUMBER	PRODUCT	STATUS	SEX_CODE	SMOKER_CODE	ENTRY_DATE	BIRTH_DATE	POL_TERM_Y	PREM_FREQ	PREMIUM	SUM_ASSURED
1	661230	WPE	1	F	N	2505188	1207159	25	12	1482	37050
2	865211	IB	1	F	N	208191	1405169	0	0	9820	0
3	537993	WPE	1	F	N	2010193	511150	25	12	1232	30800
4	289586	WPE	1	M	N	1702189	1101165	10	12	1171	11710
5	438006	IB	1	M	N	102196	2004160	0	0	18930	0
6	668240	IB	2	F	N	403198	1601158	0	0	14170	0
7	742065	WPE	1	F	N	1901191	510170	25	12	1815	45375
8	864339	IB	1	F	S	1505188	610161	0	0	8720	0
9	446368	IB	1	M	N	1908190	1504163	0	0	15370	0
10	812357	WPE	1	F	N	1105196	1503171	25	1	1933	48325
11	279105	IB	1	F	N	206197	1306152	0	0	11470	0
12	872525	IB	1	M	N	204188	2709161	0	0	10680	0
13	588157	WPE	1	F	N	2604191	2003150	10	12	1908	19080
14	305609	IB	1	M	N	203195	912150	0	0	13920	0
15	420558	IB	1	M	N	2405189	1703168	0	0	11390	0
16	149615	WPE	1	M	S	2306198	1206176	25	12	1891	47275
17	882200	IB	1	M	N	1907191	2004161	0	0	21170	0
18	601985	IB	2	M	N	409188	2109160	0	0	13260	0
19	998240	WPE	1	F	N	2508192	2707164	10	12	1571	15710
20	625888	IB	1	F	N	411188	2411163	0	0	7470	0
21	970836	WPE	1	F	N	303190	2707152	25	12	1943	48575
22	673916	WPE	1	M	N	2608193	2303160	10	12	1274	12740
23	788786	WPE	1	M	N	1207193	2601167	25	12	1309	32725

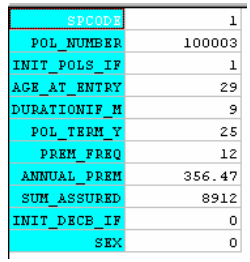
Viewer provides several different ways to view files:

Use this view	To display
Table	Model point files, Glean data files and DCS input files formatted in worksheet-like cells, rows and columns. For example, this is part of a model point file in table view:

	SPCODE	POL_NUMBER	INIT_POLS_IF	AGE_AT_ENTRY	DURATIONIF_M	POL_TERM_Y	PREM_FREQ	ANNUAL_PREM	SUM_ASSURED	IN
1	1	100001	1	19	57	23	1	439.1	10099	
2	1	100002	1	32	2	26	12	530.77	13800	
3	1	100003	1	29	9	25	12	356.47	8912	
4	1	100004	1	24	95	19	12	629.35	11958	
5	1	100006	1	28	33	20	12	682.25	13645	
6	1	100008	1	31	14	11	12	0	4823	
7	1	100010	1	23	29	10	12	528.65	4758	
8	1	100013	1	21	9	19	12	374.07	7107	
9	1	100014	1	25	9	10	12	477.58	4298	
10	1	100017	1	17	78	25	12	0	14766	

If this view is used with DCS input files with multiple record formats, the column headings automatically change to reflect the underlying format for the selected data row.

Row The values from a single row of a model point file, Glean data files or DCS input file. For example, this is a record in a model point file in row view:



SPCODE	1
POL_NUMBER	100003
INIT_POLS_IF	1
AGE_AT_ENTRY	29
DURATIONIF_M	9
POL_TERM_Y	25
PREM_FREQ	12
ANNUAL_PREM	356.47
SUM_ASSURED	8912
INIT_DECB_IF	0
SEX	0

Formatted Model point files, Glean data files and DCS input files in worksheet-like cells, rows and columns. This view is the same as the Table view.

Unformatted Model point files, Glean data files and DCS input files without any formatting.

ASCII DCS input files and text files without formatting. In this view, Viewer only displays characters in the ASCII character set.

Hex The binary value (in hexadecimal) of each byte in the file. Line numbers are shown in hex on the left and the actual characters are shown on the right. Viewer can display DCS input files and text files in this view.

Viewer includes a number of useful options that include:

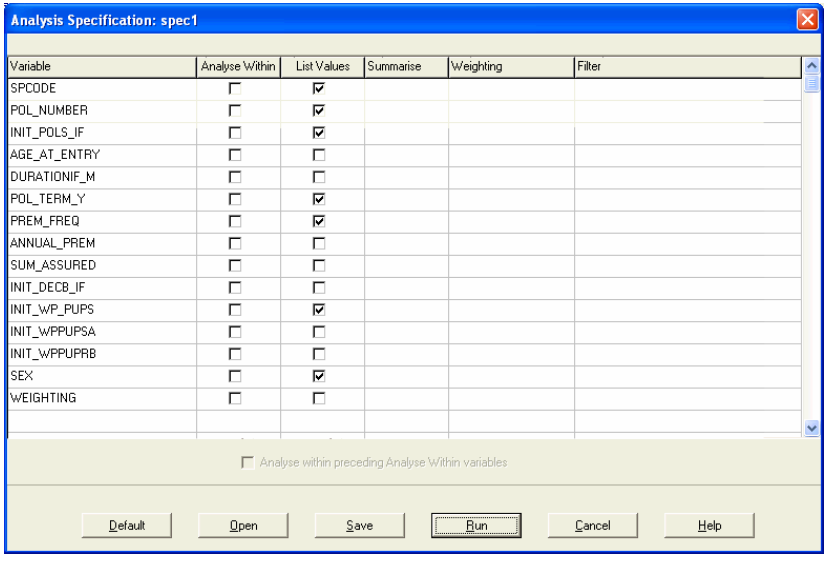
- Changing the order of the columns displayed by dragging and dropping the column headings
- Hiding selected columns using the **Del** key
- Displaying just a range of the rows using the **Range** option in the **View** menu
- Searching for particular text strings or values using the **Find** and **Go To** options in the **View** menu
- Specifying the format of numbers using the **Number Format** option in the **View** menu
- Copying the entire or part of the file to another application such as Microsoft Word or Excel using the **Copy** and **Copy Grid** options in the **View** menu
- Printing the whole or a part of the file

Using Viewer's analysis facility

In addition to using Viewer to view data files it can also be used to analyse and investigate that data using its analysis feature. This analysis feature enables you to investigate the values for each variable in a file by setting up an analysis specification which is then run. The results from the run can then be displayed and analysed in both grids and charts.

Setting up an analysis specification

To set up an analysis specification you select the **Analysis** option from the **Tools** menu. The Analysis Specification dialog is displayed from which you can select the variables in the file that you wish to investigate. A typical Analysis Specification dialog appears similar to the following:



The variables are displayed in the Variable column in the same order that they occur in the file. For each variable you can:

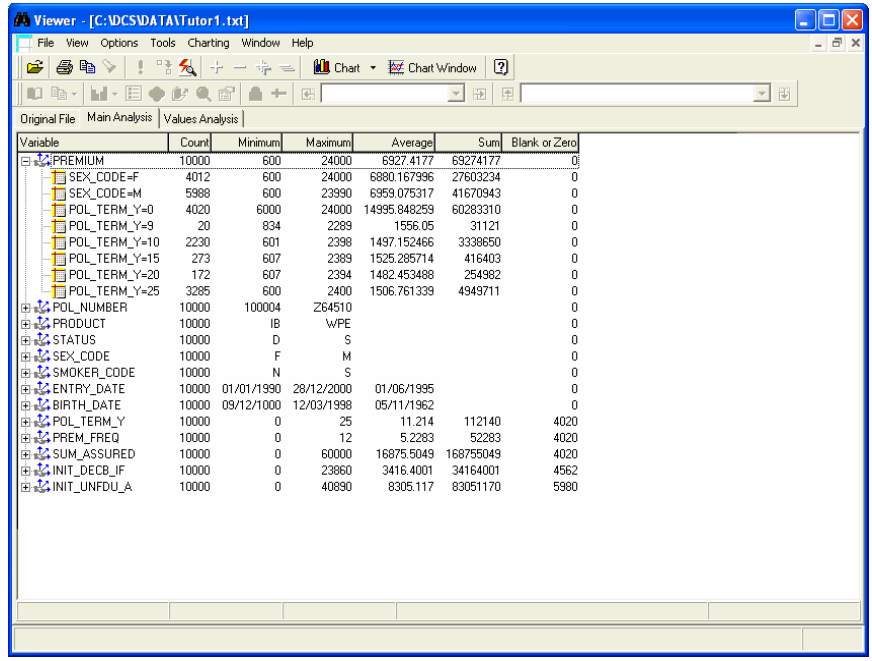
- Select the Analyse Within option, which means that the analysis will be carried out within each of the values for that variable.
- Select the List Values option to display the values that exist for that variable, including the frequency with which each of them exists.
- Summarise the values for the variable by selecting either Average and/or Sum in the Summarise column.

- Apply a weighting variable in the calculation of the average values by selecting a variable in the Weighting column.
- Filter the records that are included in the analysis by specifying one or more conditions in the Filter column.
- Rearrange the position of a variable by selecting and then dragging and dropping the variable in the order that you want it to be used in the analysis.

When an analysis specification has been specified it can be run. You can also save an analysis specification and then later retrieve it.

Running and displaying the analysis specification

When you click **Run** in the Analysis Specification dialog a run progress dialog is displayed. When the run finishes a new view is displayed with additional tabs at the top. It typically appears similar to the following:

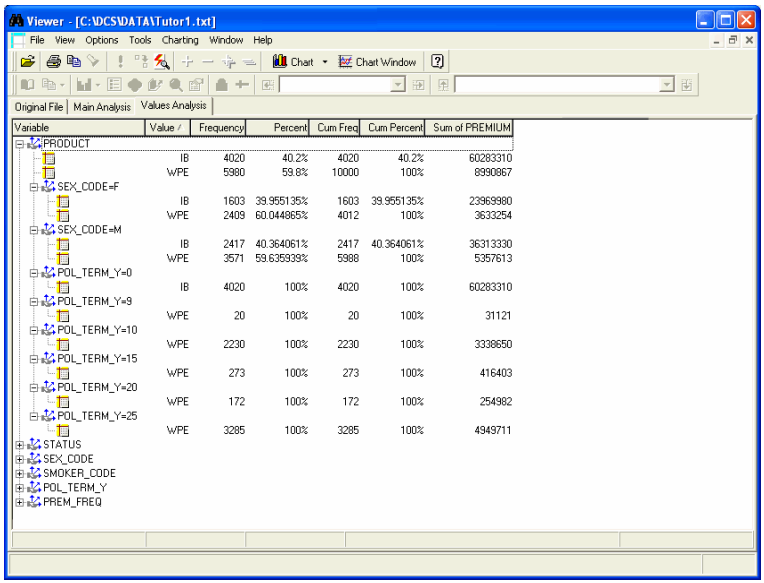


Variable	Count	Minimum	Maximum	Average	Sum	Blank or Zero
PREMIUM	10000	600	24000	6927.4177	69274177	0
SEX_CODE=F	4012	600	24000	6880.167996	27603234	0
SEX_CODE=M	5988	600	23990	6959.075317	41670943	0
POL_TERM_Y=0	4020	6000	24000	14995.848259	60283310	0
POL_TERM_Y=9	20	834	2289	1556.05	31121	0
POL_TERM_Y=10	2230	601	2398	1497.152466	3338650	0
POL_TERM_Y=15	273	607	2389	1525.285714	416403	0
POL_TERM_Y=20	172	607	2394	1482.453488	254982	0
POL_TERM_Y=25	3285	600	2400	1506.761339	4949711	0
POL_NUMBER	10000	100004	Z64510			0
PRODUCT	10000	IB	WPE			0
STATUS	10000	D	S			0
SEX_CODE	10000	F	M			0
SMOKER_CODE	10000	N	S			0
ENTRY_DATE	10000	01/01/1990	28/12/2000	01/06/1995		0
BIRTH_DATE	10000	09/12/1000	12/03/1998	05/11/1962		0
POL_TERM_Y	10000	0	25	11.214	112140	4020
PREM_FREQ	10000	0	12	5.2283	52283	4020
SUM_ASSURED	10000	0	60000	16875.5049	168755049	4020
INIT_DECB_IF	10000	0	23860	3416.4001	34164001	4562
INIT_UNFDU_A	10000	0	40890	8305.117	83051170	5980

The Original File tab displays the contents of the original data file. The Main Analysis contains the results of the analysis specification run with the variables listed in the order that they were specified.

In the screen above, you can see that the file contains 10000 policies, of which 4012 have SEX_CODE=F, and 5988 have SEX_CODE=M. You can also see how many values there are for POL_TERM_Y. Also shown are the minimum, maximum, average, sum and number of blank or zero values.

In the Values Analysis tab you can see the values for each variable where "List Values" was selected in the Analysis Specification dialog. Also the frequency with which each value appears and the percentage that it represents of the total number. You can expand or collapse the information for each variable. For example, in the following screen the variable PRODUCT was first selected and then **Expand Selected & Children** was selected from the **Expand/Collapse** option in the **View** menu:

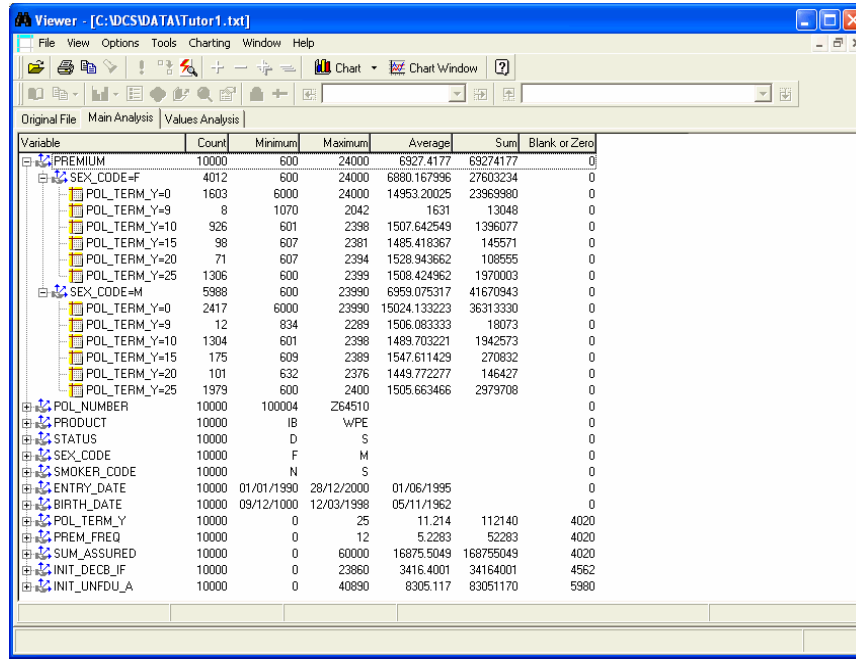


You can see the split of products according to SEX_CODE and POL_TERM_Y. It enables you to see that overall there is a 40.2%/58.8% split between investment bonds and endowments. When analysed according to sex code, it can be seen that there is no significant difference in the split between females and males.

You can sort the data according to either the values or the frequency by clicking on the appropriate column headers.

Using the Analyse within preceding Analyse Within variables option

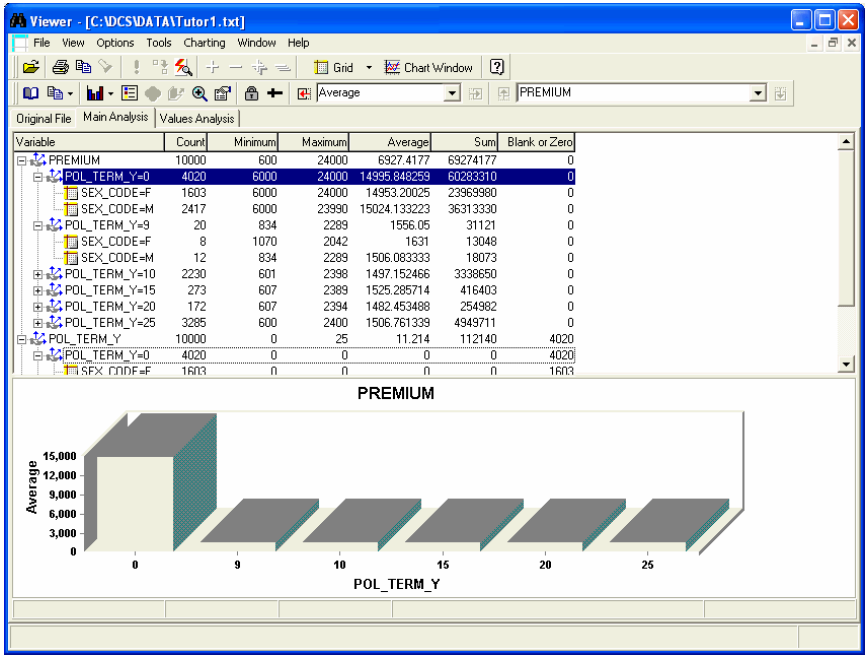
If two or more "Analyse Within" variables are selected in the Analysis Specification dialog you can specify the order in which the variables are analysed by rearranging the variables using drag and drop. You can also select the "Analyse within preceding Analyse Within variables" option if you wish each "Analyse Within" variable to be analysed within each of the "Analyse Within" variables higher up the list. For example, this enables the analysis to be done first according to SEX_CODE, and then according to POL_TERM_Y within SEX_CODE.



Variable	Count	Minimum	Maximum	Average	Sum	Blank or Zero
PREMIUM	10000	600	24000	6927.4177	69274177	0
SEX_CODE=F	4012	600	24000	6880.167996	27603234	0
POL_TERM_Y=0	1603	6000	24000	14953.20025	23969980	0
POL_TERM_Y=9	8	1070	2042	1631	13048	0
POL_TERM_Y=10	926	601	2398	1507.642549	1396077	0
POL_TERM_Y=15	98	607	2381	1485.418367	145571	0
POL_TERM_Y=20	71	607	2394	1528.943662	108555	0
POL_TERM_Y=25	1306	600	2399	1508.424962	1970003	0
SEX_CODE=M	5988	600	23990	6959.075317	41670943	0
POL_TERM_Y=0	2417	6000	23990	15024.133223	36313330	0
POL_TERM_Y=9	12	834	2289	1506.083333	18073	0
POL_TERM_Y=10	1304	601	2398	1489.703221	1942573	0
POL_TERM_Y=15	175	609	2389	1547.611429	270832	0
POL_TERM_Y=20	101	632	2376	1449.772277	146427	0
POL_TERM_Y=25	1979	600	2400	1505.663466	2979708	0
POL_NUMBER	10000	100004	Z64510			0
PRODUCT	10000	IB	WPE			0
STATUS	10000	D	S			0
SEX_CODE	10000	F	M			0
SMOKER_CODE	10000	N	S			0
ENTRY_DATE	10000	01/01/1990	28/12/2000	01/06/1995		0
BIRTH_DATE	10000	09/12/1000	12/03/1998	05/11/1962		0
POL_TERM_Y	10000	0	25	11.214	112140	4020
PREM_FREQ	10000	0	12	5.2283	52283	4020
SUM_ASSURED	10000	0	60000	16875.5049	168755049	4020
INIT_DECB_IF	10000	0	23860	3416.4001	34164001	4562
INIT_UNFDU_A	10000	0	40890	8305.117	83051170	5980

Displaying the results of the analysis in a chart

The results of the analysis can be displayed in a chart by selecting one of the options for the Chart button in the toolbar, for example "Display chart below grid". You can then specify what is displayed in the chart by choosing options from the two drop down lists that correspond to the columns and rows of the displayed data. Alternatively, you can click on the required row in the grid. For example:



You can also display the chart over the top of the grid by selecting the "Chart Window" toolbar button.

You can change the settings for the chart by either clicking on one of the toolbar buttons in the Chart toolbar or by double clicking on the part of the chart you want to change. Full details of the chart options are given in the Viewer Help.

Part 2 - Data Conversion System Tutorial

LESSONS IN THIS PART

LESSON 1

Setting up and running a simple DCS program

In this lesson you will set up a simple DCS program to read a fixed format input file and create individual policy model point files for the two products that it contains.

TOPICS COVERED IN THIS LESSON

Introduction

In this lesson you will set up a simple DCS program to read an input file, carry out a number of calculations and then produce some output files as follows:

Input file

The input file is in fixed ASCII format with no header lines or characters. It only has one record format. It contains details of 10,000 policies which are a mixture of with profit endowments and unit linked investment bonds. The date at which the file was produced was 31 December 2000.

The first four records of the input file are shown below:

```
533995 IB IMN0402199304051948  0 014850.00      0.00      0.00  22367.81
961375WPE IFS1501200012121972 1512 2327.00  34905.00      0.00      0.00
520588WPE IFN1708199613081967 25 1  941.00  23525.00   3764.00      0.00
534607WPE IFN0603199312071967 25 1 2301.00  57525.00  16107.00      0.00
```

Calculations required

The input file contains the date of birth of the policyholder and the date of commencement of the policy. You will calculate the age at entry from these dates.

You will also calculate the duration in force in months using the date at which the file was produced and the date of commencement of the policy.

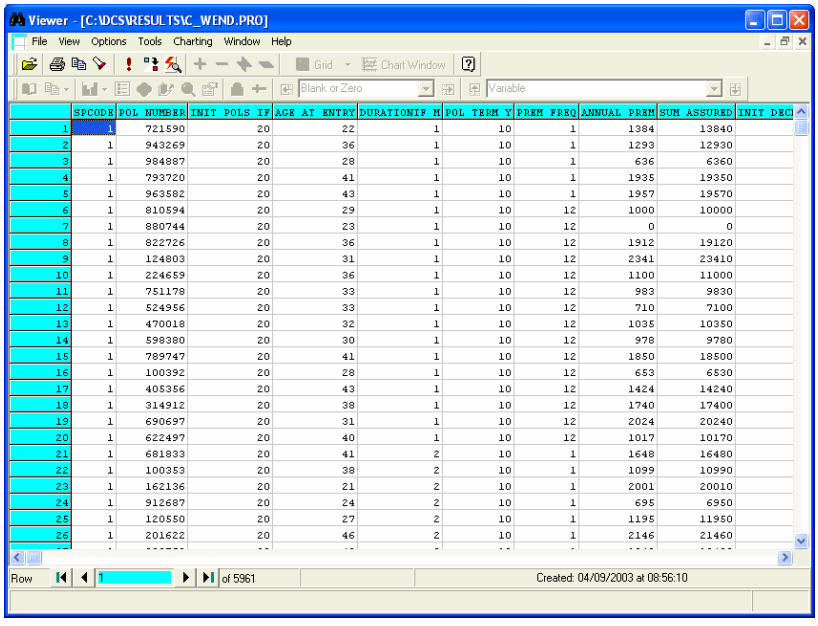
The sex and smoker status are held in the input file as text fields. You will convert them into the numbers required by Prophet.

As well as existing business model points, you will create a set of new business profile model points based on the profile of the new business actually sold in the previous year.

Output files to be produced

Two output files in model point file format are required, one for the with profit endowment and the other for the unit linked investment bond. Separate model points are required within each model point file for existing business and to form a profile for future new business.

These model point files will look similar to the following when viewed in Viewer:

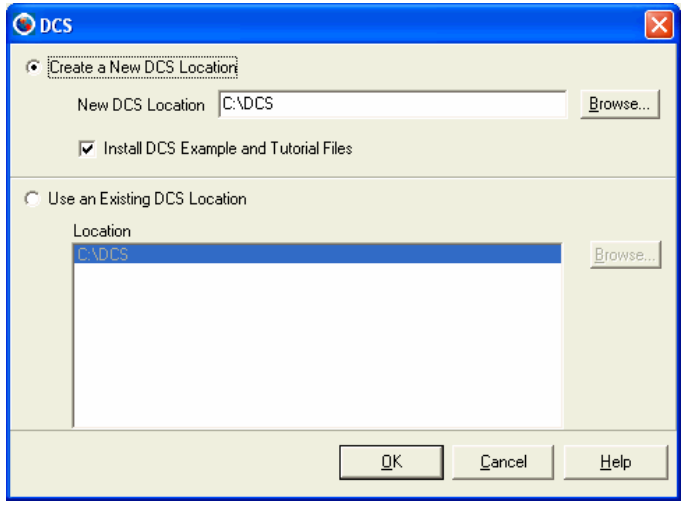


	SPCODE	POL_NUMBER	INIT_POLS_IF	AGE_AT_ENTRY	DURATIONIF_M	POL_TERM_Y	PREM_FREQ	ANNUAL_PREM	SUM_ASSURED	INIT_DEC
1	1	721590	20	22	1	10	1	1384	13840	
2	1	943269	20	36	1	10	1	1293	12930	
3	1	984887	20	28	1	10	1	636	6360	
4	1	793720	20	41	1	10	1	1935	19350	
5	1	963582	20	43	1	10	1	1957	19570	
6	1	810594	20	29	1	10	12	1000	10000	
7	1	880744	20	23	1	10	12	0	0	
8	1	822726	20	36	1	10	12	1912	19120	
9	1	124803	20	31	1	10	12	2341	23410	
10	1	224659	20	36	1	10	12	1100	11000	
11	1	751178	20	33	1	10	12	983	9830	
12	1	524956	20	33	1	10	12	710	7100	
13	1	470018	20	32	1	10	12	1035	10350	
14	1	598380	20	30	1	10	12	978	9780	
15	1	789747	20	41	1	10	12	1850	18500	
16	1	100392	20	28	1	10	12	653	6530	
17	1	405356	20	43	1	10	12	1424	14240	
18	1	314912	20	38	1	10	12	1740	17400	
19	1	690697	20	31	1	10	12	2024	20240	
20	1	622497	20	40	1	10	12	1017	10170	
21	1	681833	20	41	2	10	1	1648	16480	
22	1	100353	20	38	2	10	1	1099	10990	
23	1	162136	20	21	2	10	1	2001	20010	
24	1	912687	20	24	2	10	1	695	6950	
25	1	120550	20	27	2	10	1	1195	11950	
26	1	201622	20	46	2	10	1	2146	21460	

Starting DCS

In order to run DCS you need to have Prophet installed on your PC. If this is not the case you should refer to the Installing Prophet chapter of the Prophet User's Guide or the Installing Glean chapter of the Glean User's Guide.

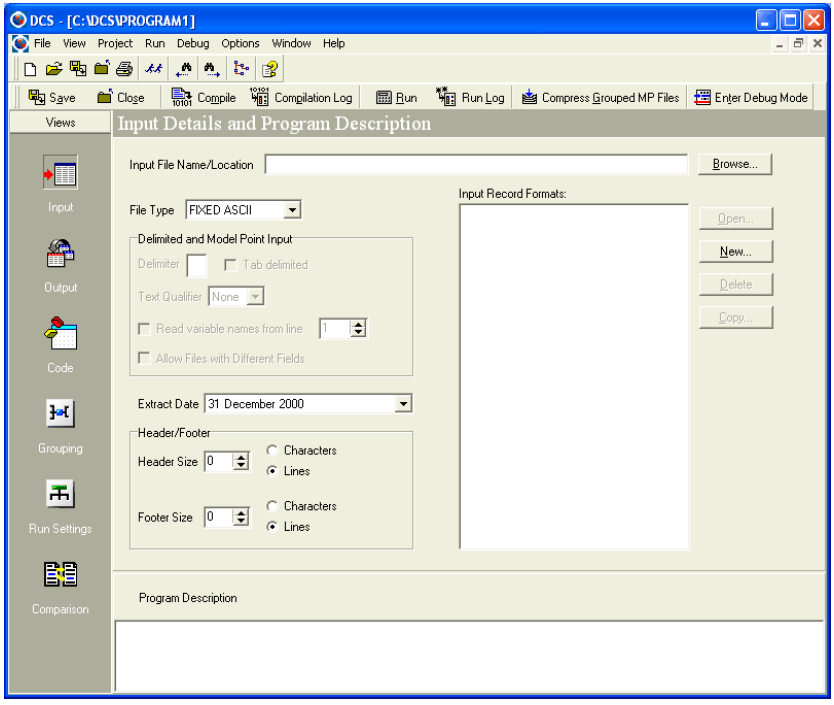
To start DCS you should click on the Start button and point to Programs. You should then select the Prophet 7.2 or Glean 7.2 program group and then click on the Data Conversion System option. The following dialog will then appear:



This dialog allows you to create a new DCS location in which to create DCS programs or to select an existing DCS location. You should select the "Create a New DCS Location" option. You should accept the default of C:\DCS, unless you already have a DCS location with this name in which case you should enter a different location (the remainder of this tutorial assumes that you use C:\DCS). You should ensure that the "Install DCS Example and Tutorial Files" option is selected because you will need these files for this tutorial.

You should now click **OK**.

When the DCS Program Selection dialog appears you should click on **Cancel** because you want to create a new program. You should then select **New** option in the **File** menu and choose **DCS Program File**. The following screen will then be displayed:



This is the **Input Details** view of the DCS program window. From this view you specify:

- The name and location of each input file. If the input file is in a database format then a data source is specified instead of a file name.
- The input file format of the input files. The input files can be fixed format files, in ASCII or EBCDIC format, delimited format files such as a comma, space or tab delimited file, a Microsoft Excel file, a database format file or a model point format file.
- The delimiter for delimited input files (for tab delimited files T should be specified for the delimiter). Also the text qualifier for delimited and model point input files. Also if a delimited file has a line containing the variable names, you can specify which line this is and also if different files contain different variables.

- The date for which the data was extracted.
- Whether or not the input files have a header or footer and, if so, the size of the header or footer.
- The format of the records in the input files. You must add a record format for each different type of record in the input files. The information you need to enter depends on the type of the input files.
- A description of the DCS program in the Program Description box.

You specify the output files to be generated by the DCS program in the **Output Details** view. The instructions and calculations to be carried out for each record are entered in the **Code** view and the settings for running the program are specified in the **Run Settings** view. If you are grouping the data from the input files then you specify the grouping settings in the **Grouping** view. If you want to compare two programs you use the **Comparison** view.

All these views are accessed from the Views list displayed at the left of the DCS program window.

Specifying the input file details

In this section you will enter details of the input file in the Input Details view of the DCS program window. Note that there is a deliberate error in one of the entries so that later in the lesson you can see how to identify and correct certain types of error.

The first thing you should do is enter the text **Tutorial Program 1** in the Program Description section.

Name/Location

You should enter the File/Name location as DATA\TUTOR1.TXT. This means that the DCS program will read the input file from the DATA sub-folder of the location where the DCS program file is held, that is C:\DCS\DATA in this tutorial. You can enter the full path to an input file if you wish to.

Type of input file

You need to select the type of input file. You have eight choices:

FIXED ASCII	EXCEL
FIXED EBCDIC	DATABASE
DELIMITED	MODEL POINT
ICL EBCDIC	GLEAN DATA

For FIXED format files the position of each field within each record is determined by its start position and length. For DELIMITED format files, a specific character such as a comma is used to separate each field within a record.

ASCII and EBCDIC are code formats used to map characters to the 256 possible combinations in a byte. For example, the letter A is represented by the number 65 in ASCII and 193 in EBCDIC. Generally, ASCII is used by PCs and EBCDIC by mainframes. If your input file was created on a mainframe and is still in EBCDIC format then by selecting FIXED EBCDIC or ICL EBCDIC it will be automatically translated by DCS as it is read.

For this tutorial you should select FIXED ASCII.

Delimiter

Since the input file is in FIXED ASCII format you don't need to enter a Delimiter. However, if the input file had been DELIMITED then you would have had to enter the appropriate delimiter, such as a comma or the letter T for tab delimited files. Text in delimited input files may be enclosed in single or double quotes and therefore the appropriate character, if any, should be chosen for the Text Qualifier.

Extract Date

You should leave the Extract Date unchanged as 31 December 2000.

Header

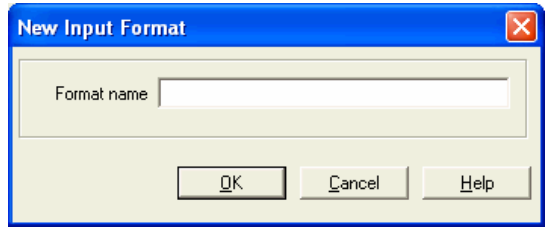
The input file has no header characters or lines so leave the Header Size field as 0.

Footer

The input file has no footer characters or lines so leave the Footer Size field as 0.

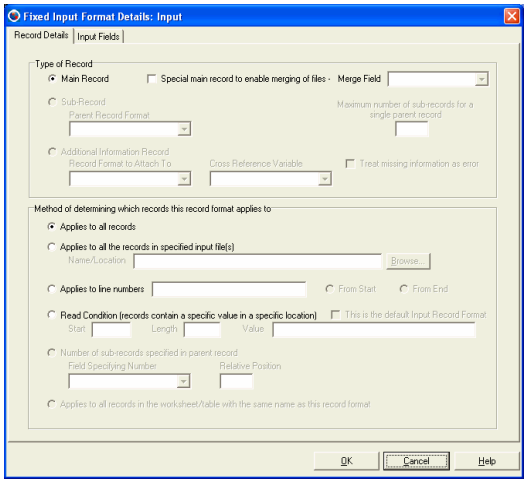
Input record formats

You now need to create an input record format in which you will enter details of all the data fields in the input file. So click on the **New** button and the following dialog will appear:



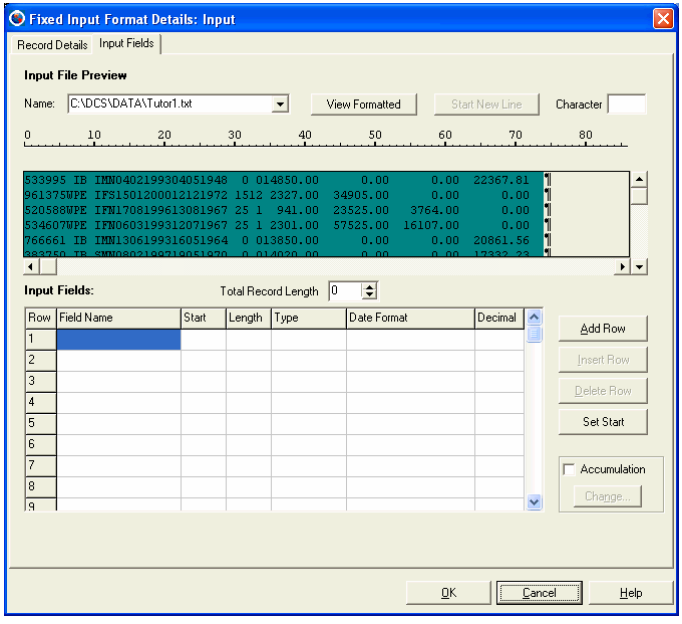
Enter "Input" as the name and then click **OK**.

The following dialog will then appear:

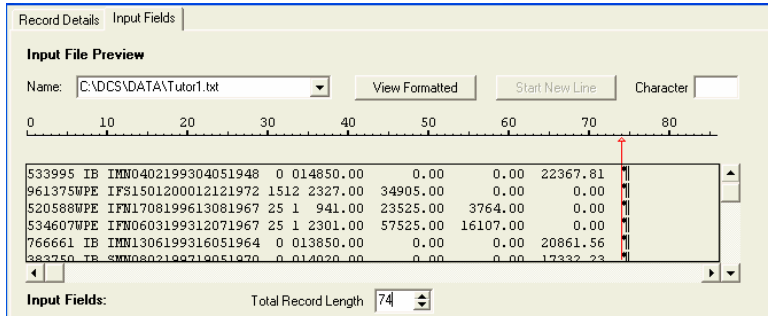


The options in the Record Details tab are only relevant for input files that have more than one record format, which is not the case for this input file. You will be covering such files in Lesson 5 but in the meantime you can ignore the options on this tab.

Click on the Input Fields tab and the following will be displayed:



You first need to enter the Total Record Length as 74. When you do this the Input File Preview area is refreshed and displays the contents of the input file named in the Name box. The Input File Preview area appears as follows:



If you do not enter a total record length DCS does not allow you to enter any input fields since it checks that the Start and Length values that you enter do not cause a field to extend beyond the end of the record.

The bottom part of the Input Fields tab enables you to enter the details of all the fields in the input file that you want to read. The following items need to be specified:

Field Name

This is the name of the variable that you will use to refer to this field in the code and in the output formats. If a field corresponds to a Prophet variable that you will want to use in the output model point files that will be created by the program then you should use the name of that variable here.

Start

This is the position of the first character of the field in the input file.

Length

This is the number of characters that the field takes up in the input file.

Type

This specifies the type of field. You can choose one of the following:

Text
Number
Integer
Date
ShortInt
Signed Binary
Binary
Packed
IBM Float

Date Format

For date fields you need to specify the format of the date. The default format is dd/mm/yyyy but this can be amended if a different format is used in the input file.

Decimal

For Number, Signed Binary, Binary, Packed and IBM Float fields you need to specify the number of decimal places. You can either select "File" or specify a value between 0 and 9 inclusive. If you select "File" then whatever number of decimal places is contained in the input file will be used. If a number is held in the input file as (say) an integer number of pence rather than as pounds and pence then by selecting 2 the number will be automatically converted into pounds and pence as it is read.

Note: For fixed format, database format and model point format input files you do not have to create fields in the input format for any data which you don't need to use in your program. For example, if the input file contained the name of the policyholder you could just ignore that information. The same is not true of delimited input and Excel files because DCS would have no way of knowing which particular fields had been left out.

You need to enter the following field information:

Row	Field Name	Start	Length	Type	Date Format	Decimal
1	POL_NUMBER	1	6	Integer		
2	PRODUCT	7	3	Text		
3	STATUS	11	1	Text		
4	SEX_CODE	12	1	Text		
5	SMOKER_CODE	13	1	Text		
6	ENTRY_DATE	14	8	Date	ddmmyyyy	
7	BIRTH_DATE	22	8	Date	ddmmyyyy	
8	POL_TERM_Y	31	2	Integer		
9	PREM_FREQ	33	2	Integer		
10	PREMIUM	35	8	Number		File
11	SUM_ASSURED	43	10	Number		File
12	INIT_DECB_IF	53	10	Number		File
13	INIT_UNFDU_A	63	10	Number		File

Notice that the start position is automatically calculated based on the values entered in the previous line. This should be overwritten when necessary so that blank spaces are ignored. You need to do this for STATUS which actually starts at position 11, not 10.

In the case of ENTRY_DATE you will need to modify the default Date Format of dd/mm/yyyy by clicking on it and then selecting the **None** option in the Separator section. DCS will then automatically use this as the default for BIRTH_DATE.

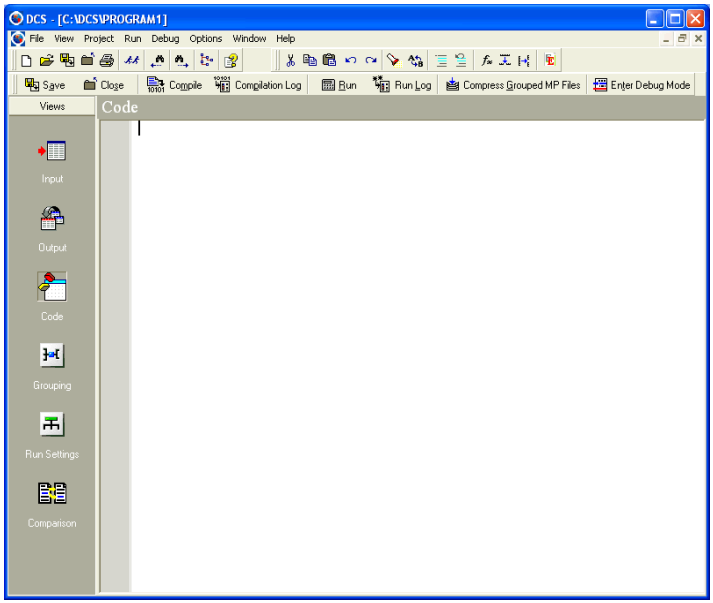
Note: You will be able to enter this information more quickly if you don't switch between the keyboard and the mouse. You will be able to just use the keyboard if you:

- Use the Tab key to move from one cell to the next. This also causes new rows to be added automatically at the end of the list when they are needed.
- Press the first letter to select the Type rather than selecting it from the drop down list box using the mouse. For example, type T to select Text.

When you have entered the field details click on **OK** to save the input record format.

Entering the code

In the Views list you should now click the **Code** icon so that you can enter the code using the DCS code editor. The Code view appears as follows:



You need to enter code in this view to carry out the required calculations and output the required files.

Function Wizard

To help you enter functions you can use the Function Wizard which operates in the same manner as the Function Wizard in Microsoft Excel. To use it click on the f_x toolbar button. In Section B of the Reference Section of this manual you will also find a list of all the functions.

Insert Variable option

To help you enter variable names you can use the Insert Variable option. It provides a list of all the variables in the input record formats and in the code. When you select a variable it is inserted into the code. To use it click on the toolbar button.

Insert Statement option

To help you create loops and branches you can use the Insert Statement option. It provides a list of all the statements that can be used in a DCS program. To insert a statement you place the cursor in the code where the statement is to be inserted and click on the  toolbar button. You can then select the appropriate statement from the displayed list and press Enter.

Excluding records not required

You first need to exclude records which have a status of other than I (that is in force). You exclude records using the NEXT_RECORD function. Therefore enter the following code:

```
; Exclude records that are not in force
  IF STATUS <> "I" THEN
    NEXT_RECORD
  ENDIF
```

Note: The first line starts with a semicolon which means that it is a comment line.

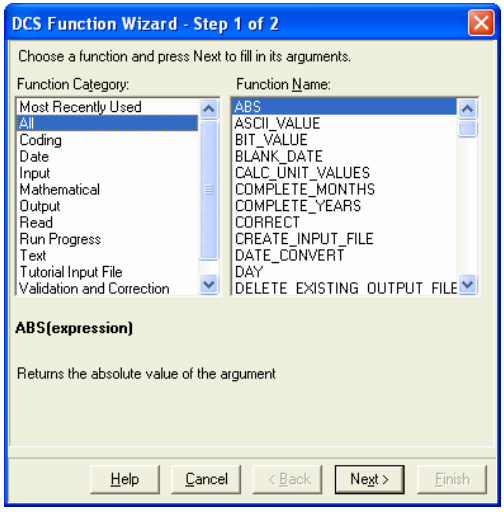
Valuation year

You now need to set the valuation year. This will be used in the calculations and in the header lines which will be written to the output files. It is therefore needed in both date and text forms. The EXTRACT_DATE function returns the extract date entered in the Input Details view and the YEAR function enables you to determine the year of a date. Therefore enter the following code:

```
; Set valuation year
  VAL_YEAR = YEAR(EXTRACT_DATE)
  VAL_YEAR_T = STRVAL(VAL_YEAR)
```

Using the Function Wizard

You can use the Function Wizard to help you enter the required code. To do this type in the code up to and including the "=" in the second line. Then click the  toolbar button. This dialog will appear.



In the Function Category list locate and select "'Date". Then in the Function Name list locate and select the "YEAR" function. Click **Next** and you will be given some information on the function, and prompted to supply the 'date' argument. Type in EXTRACT_DATE and click **Finish**. You can then enter STRVAL in the same way.

Entry year

The entry year needs to be set equal to the year part of the ENTRY_DATE. So enter the following:

```
; Calculate entry year
  ENTRY_YEAR = YEAR(ENTRY_DATE)
```

Age at entry

The age next birthday at entry needs to be calculated from the entry date and the birth date. We can use the COMPLETE_YEARS function to calculate the number of years between two dates, so enter the following:

```
; Calculate age at entry
  AGE_AT_ENTRY = COMPLETE_YEARS(ENTRY_DATE, BIRTH_DATE-1) + 1
```

Duration in force in months

The duration in force in months needs to be calculated from the entry date and the extract date, so enter the following:

```
; Calculate duration in force to higher month
  DURATIONIF_M = COMPLETE_MONTHS(EXTRACT_DATE, ENTRY_DATE) + 1
```

Premium

The input file contains the field PREMIUM. However, Prophet requires ANNUAL_PREM for the with profit endowment which is a regular premium product and SINGLE_PREM for the investment bond which is a single premium product. Therefore enter the following code:

```
; Set premium variables
  IF PRODUCT = "WPE" THEN
    ANNUAL_PREM = PREMIUM
    SINGLE_PREM = 0
  ELSE
    ANNUAL_PREM = 0
    SINGLE_PREM = PREMIUM
  ENDIF
```

Sex and smoker code

The sex and smoker code are stored as text values in the input file. They need to be converted to numeric format for use by Prophet, so enter the following:

```
; Convert sex and smoker codes to numeric format
  IF SEX_CODE = "F" THEN
    SEX = 1
  ELSE
    SEX = 0
  ENDIF

  IF SMOKER_CODE = "S" THEN
    SMOKER_STAT = 1
  ELSE
    SMOKER_STAT = 0
  ENDIF
```

Specify output files

You need to create two output model point files, one for the with profit endowment product and the other for the investment bond product. These files will require different lists of model point variables.

In DCS the OUTPUT function is used to produce output files which contain one line for each time the function is called. It has three arguments as follows:

Argument	Purpose
Format name	This is the name of the output format which contains the names of the variables to be included in the output file. It also includes the names of the variables that should be used to sort the output file. This will normally include SPCODE because Prophet requires that all the model points are in ascending order of their sub-product code within each model point file.
File name	This is the name of the file that is to be produced. In the case of model point files the name must be the same as the name of the corresponding Prophet product. If you enter the name without an extension, DCS will use the default extensions of .PRO for model point files, .GDF for Glean data files, .TXT for text files and .FAC for Generic Table files.
Heading	This is a heading that will be included at the start of the file.

If you are creating a large number of output files it is best to include a default name and format so that if any records are not allocated proper names and formats then the default is used. This problem can occur if you have a large number of IF / THEN / ENDIF statements to set the names and formats but you have not covered all circumstances.

Any variables that you create in the code retain their values from one record to the next. This can be useful in certain circumstances. For example, if each policy has several records in the input file you can create a variable called PREVIOUS_POL_NUMBER which is set equal to the current policy number near the end of the code and which then retains that value in the next record. You can then specify particular calculations to be carried out if the current policy number is the same as the previous policy number. If you don't want variables to retain their values then you must include code to reset them to the default values you require.

Enter the following code:

```
; Set output details (including defaults in case some
; policies are not given an appropriate filename)
  OUTPUT_FORMAT = "Conventional"
  FILENAME = "ERROR"
  HEADING = "Policies not allocated a filename as at 31.12."
            + VAL_YEAR_T

  IF PRODUCT = "WPE" THEN
    OUTPUT_FORMAT = "Conventional"
    FILENAME = "C_WEND"
    HEADING = "With Profits Endowments data as at 31.12."
              + VAL_YEAR_T
  ENDIF

  IF PRODUCT = "IB " THEN
    OUTPUT_FORMAT = "UnitLinked"
    FILENAME = "U_IB__"
    HEADING = "Investment Bonds data as at 31.12."
              + VAL_YEAR_T
  ENDIF
```

Sub-product code

The sub-product code needs to be set. For the purposes of this tutorial a sub-product code of 1 will be used for all the existing business in the input file. Therefore enter the following code:

```
; Set sub-product code to 1
  SPCODE = 1
```

Initial number of policies

The initial number of policies represented by each model point needs to be set to 1. Therefore enter the following code:

```
; Set initial number of policies to 1
  INIT_POLS_IF = 1
```

Output line for existing business

To output a line to the output files the OUTPUT function needs to be called. Therefore enter the following code:

```
; Output line for existing business
  OUTPUT(OUTPUT_FORMAT, FILENAME, HEADING)
```

Each time this function is called it creates a record in the output file which contains the current values for each of the variables that you specify in the output format.

New business profile

New business model points are also needed to provide the profile for future new business. The simplest way of doing this is to assume that future new business will have the same profile as the actual new business sold in the previous year, although this will not be an appropriate assumption to make under all circumstances. Enter the following code:

```
; Output line for new business profile
  IF ENTRY_YEAR = VAL_YEAR THEN
    SPCODE = SPCODE + 50
    DURATIONIF_M = 0
    INIT_DECB_IF = 0
    INIT_UNFDU_A = 0
    OUTPUT(OUTPUT_FORMAT, FILENAME, HEADING)
  ENDIF
```

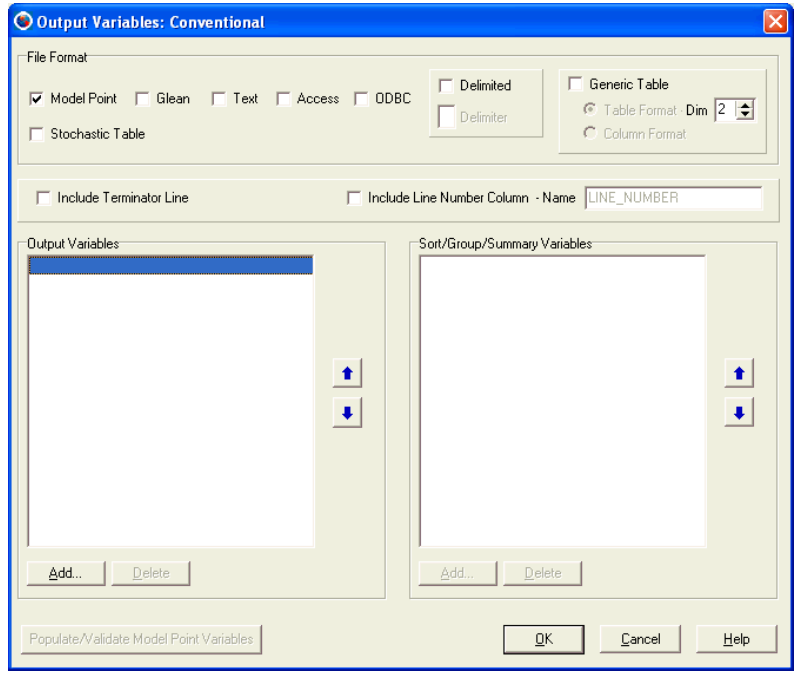
Note that the value of SPCODE has been increased by 50. Also that the starting position has been zeroised where appropriate.

You have now entered all the code that is required.

Specifying the output formats

You now need to create the output file formats **Conventional** and **UnitLinked** that you used in the code. These are specified in the Output Details view of the DCS program window.

To display the Output Details view click **Output** from the **Views** list. Then click **New** alongside the Output File Formats list. When prompted enter the name **Conventional** and then click **OK**. The following dialog will be displayed:



File Format

You need to select the required File Format. You have seven options:

File Format	Description
Model Point	This option produces output files in Prophet's model point file format (that is in a comma delimited format).
Glean	This option produces output files in Glean's data file format.

File Format	Description
Text	This option produces output files in a text format with each output variable in its own column with spaces to make all the columns properly aligned.
Access	This option produces output files in Microsoft Access 2000 format.
ODBC	This option produces output files in Access, dBase or FoxPro format (this aspect is covered in Lesson 4).
Delimited	This option produces output files in a delimited format where you can specify the delimiter.
Generic Table	This produces output files in Prophet Generic table format. You can choose the 2-Dim Table Format option to create 2-dimensional generic tables in table format which can be used for global and parameter files which are special cases of 2-dimensional generic tables. Alternatively, you can choose Column Format to create generic tables in column format which can be used for generic tables of any number of dimensions.

You should select the Model Point option.

The **Include Terminators Line** option allows you to specify that an additional line is created at the end of the output file that contains values to denote that it is the last line in the file.

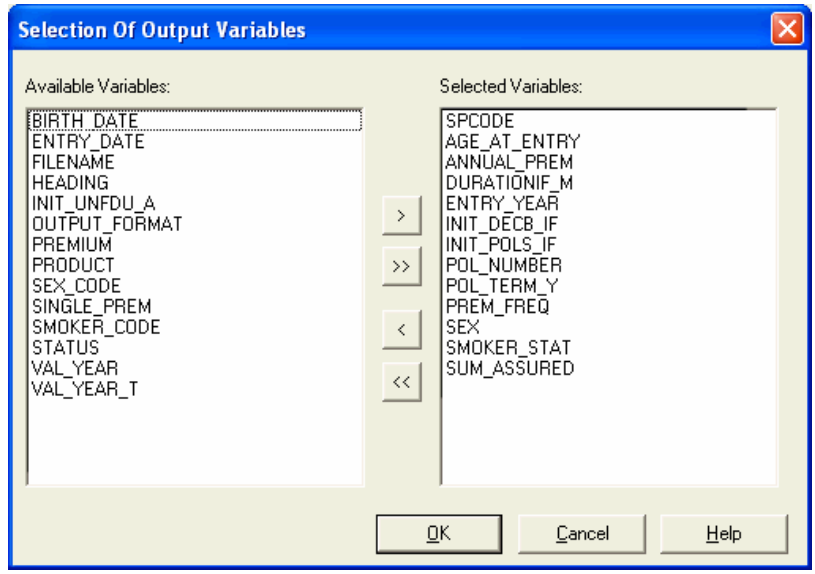
For the purposes of this tutorial do not select the **Include Terminators Line** option.

The **Include Line Number Column** option together with the name of a variable entered in the **Name** text box enables you to specify an additional variable in the output file that contains the number of the line within the output file. Generally, the additional variable is the first variable in the output file. However, if the first variable is SPCODE then the additional variable will be the second variable in the output line. The default name for the additional variable is LINE_NUMBER but you can specify a different name provided that it is not the same as a variable used elsewhere in the DCS program

For the purposes of this tutorial do not select the **Include Line Number Column** option.

Output Variables

You need to select the variables to be included in the output files that use this output format. Click **Add** in the Output Variables section to display the following dialog except that initially the right hand list will be empty:



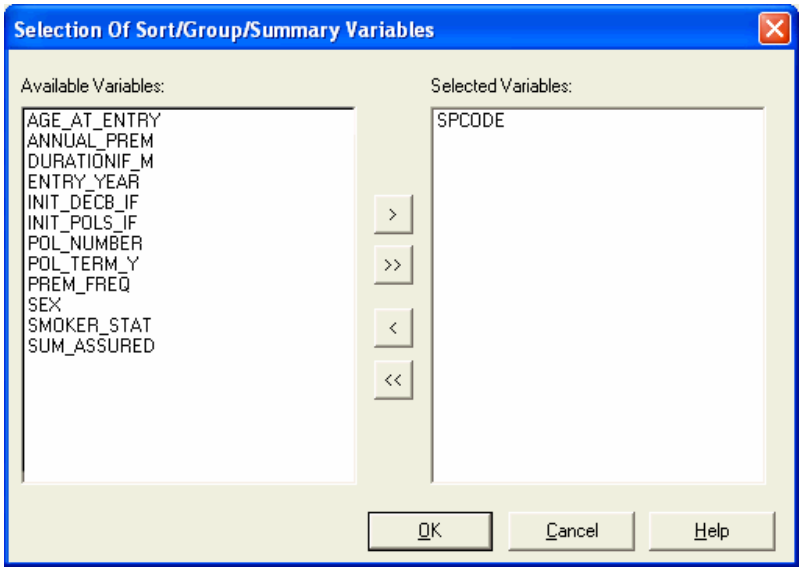
The left hand list contains all the variables which are either fields in the input file or which have been created by you as variables in the code.

You need to select variables from the left hand list into the right hand list so that it is the same as shown above.

The order of the variables doesn't matter except that SPCODE must be the first variable. If you need to, you can reorder them after you have clicked on **OK** to indicate that you have selected the variables that you require.

Sorting Variables

You need to select SPCODE as the variable on which the model point lines will be sorted. You need to do this so that the model points with SPCODEs of 1 and 51 are sorted into ascending order. Click on the **Add** button in the Sort/Group/Summary Variables section and the following dialog will appear except that initially the right hand list will be empty:

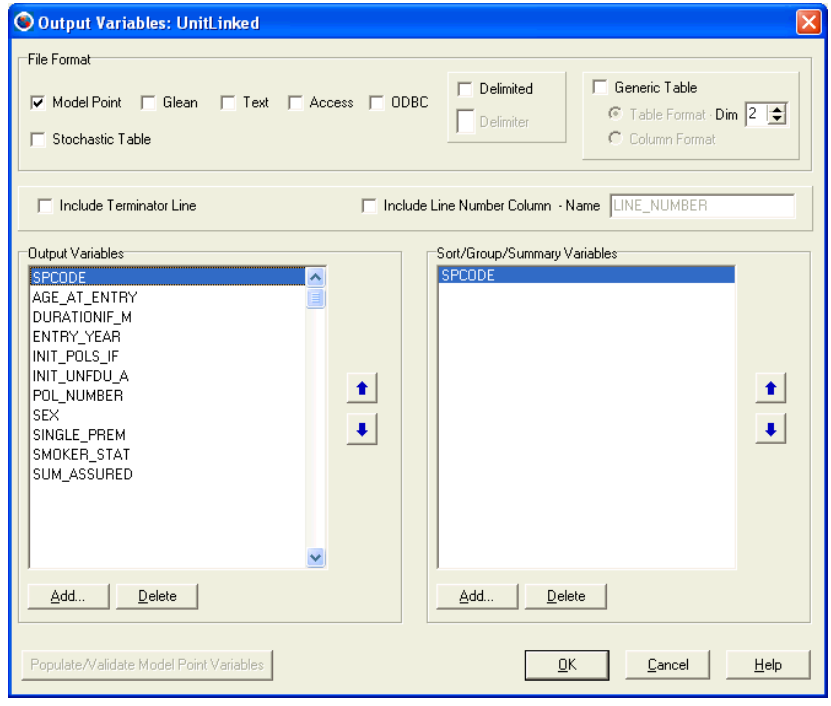


The left hand list contains all the variables which have been selected as output variables for this output format. You need to select SPCODE from the left hand list into the right hand list so that it is the same as shown above.

Now click on **OK** twice, first to confirm the sorting variables and then the output format itself. You will then be returned to the Output Details view of the DCS program window.

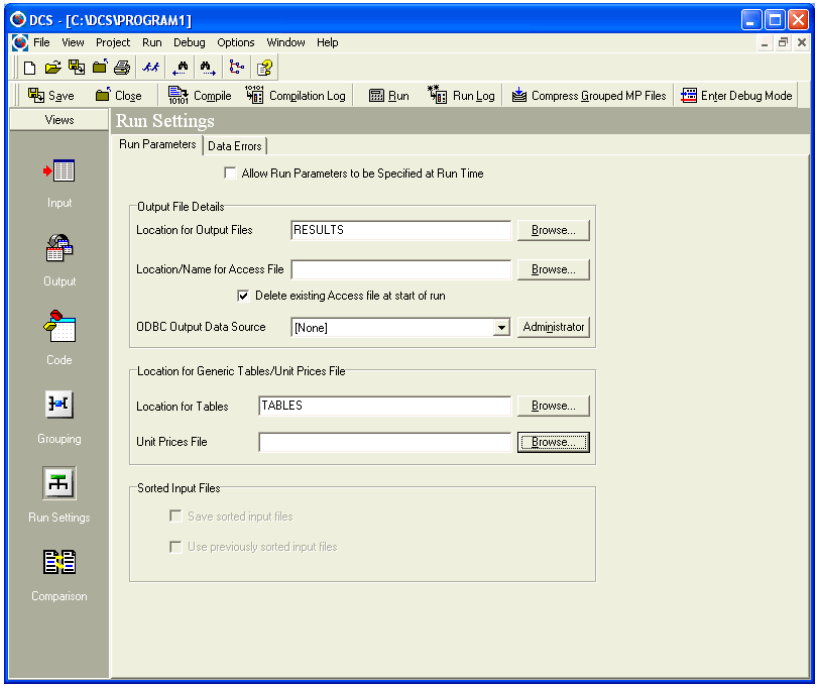
UnitLinked output format

You should now create the **UnitLinked** output format in exactly the same manner as the Conventional output format. You should set it up so that it looks as follows:



Setting up the run

You now need to specify various parameters which are required to run the program. Therefore from the **Views** list click **Run Settings**. The Run Settings view appears as follows:



You should change the Location for Output Files to RESULTS1 so that the output files are created in a RESULTS1 sub-folder of the DCS location.

On the Data Errors tab, you should leave the "Maximum Number of Runtime Errors" as zero. This means that the program will stop the first time that it detects an error.

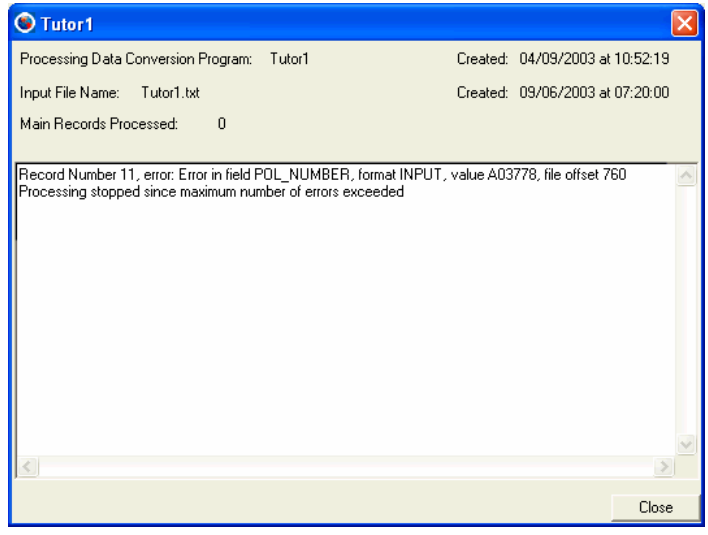
The other options will be covered in later lessons in this tutorial.

Running the program

You should now run the program by clicking on the **Run** button. You will first be prompted to save the program. Enter the name TUTOR1 and then click **Save**. The program will then be compiled and run.

When a DCS program runs, it automatically reads each input record in turn and then carries out the calculations and processes specified in the code for that record.

The following dialog will appear when the run commences at the end of compilation:



This type of message is produced if the value read for a field is inconsistent with its type. In this case the POL_NUMBER field is specified in the input record format to be an integer but when the input file was read non-numeric characters were found in the specified position (that is A03778). There are three reasons why this may have occurred:

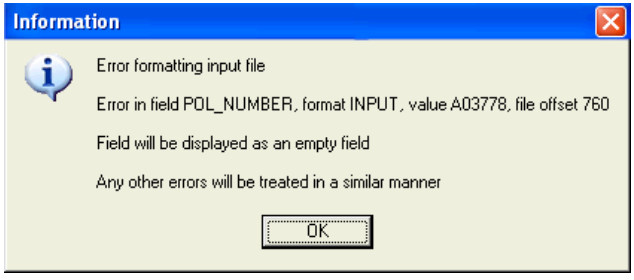
- POL_NUMBER has been incorrectly defined and should in fact be defined as a text field.
- The start and length details for POL_NUMBER have been incorrectly entered.
- The Total Record Length has been incorrectly specified.

Viewing the input file

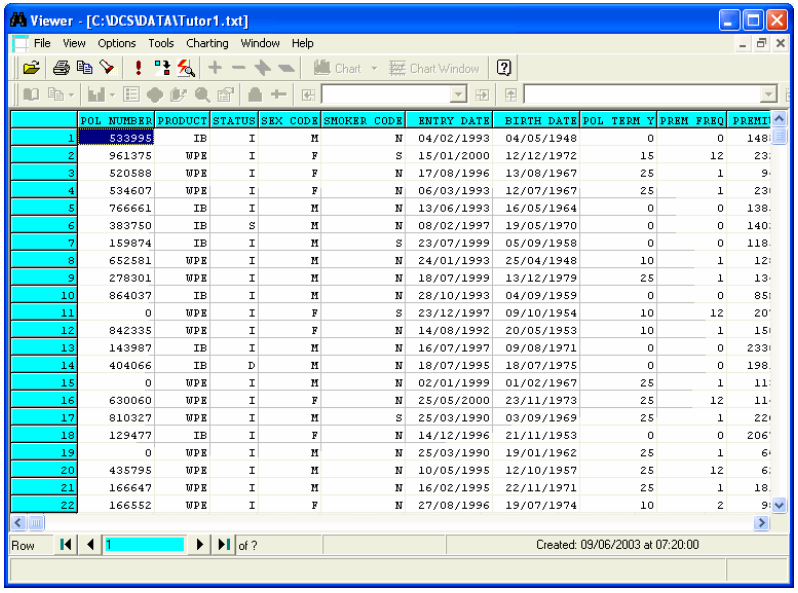
You can view the input file to help you identify the cause of such errors. To do this you should close the TUTOR1 dialog displayed above and then select the **Input File** option in the **View** menu.

Viewer main screen

Viewer will then be loaded and the following dialog will be displayed:



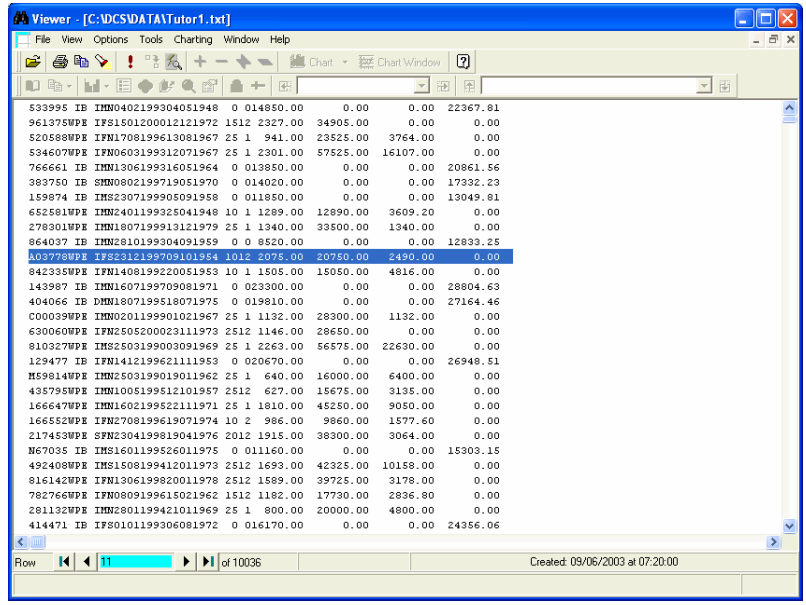
Click **OK** and the following screen is then displayed:



The Viewer is using the information in the record format in your DCS program to try and display the input file in neat columns. However, as you can see, it is displaying a policy number of zero for some of the rows.

Unformatted view

You should right click on your mouse and select the **Unformatted** option. The input file will now be displayed as follows:

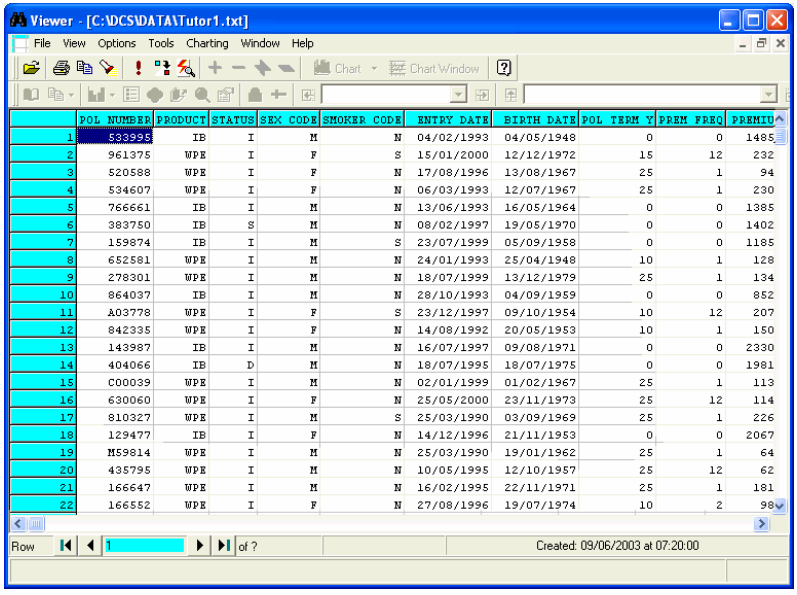


This is the way that the file is actually stored on the hard disk. As you can see, the data seems to be in the form that would be expected. In particular the policy number which is stored in the first 6 characters at the start of each line is normally an integer. However, if you look at row 11 you will see that the first character is A. Hence POL_NUMBER needs to be defined as a text variable.

Correcting the program

You should then return to DCS using either the Taskbar or by keying {Alt}{Tab}. You should open the Input record format, change the Field Type for POL_NUMBER to Text and click **OK**.

If you view the input file again in the same way as before the screen should show the following (you should click **OK** on the two warning dialogs which will appear during this process):



	POL_NUMBER	PRODUCT	STATUS	SEX_CODE	SMOKER_CODE	ENTRY_DATE	BIRTH_DATE	POL_TERM_Y	PREM_FREQ	PREMIU
1	533995	IB	I	M	N	04/02/1993	04/05/1948	0	0	1485
2	961375	WPE	I	F	S	15/01/2000	12/12/1972	15	12	232
3	520588	WPE	I	F	N	17/08/1996	13/08/1967	25	1	94
4	534607	WPE	I	F	N	06/03/1993	12/07/1967	25	1	230
5	766661	IB	I	M	N	13/06/1993	16/05/1964	0	0	1385
6	383750	IB	S	M	N	08/02/1997	19/05/1970	0	0	1402
7	159874	IB	I	M	S	23/07/1999	05/09/1958	0	0	1185
8	652581	WPE	I	M	N	24/01/1993	25/04/1948	10	1	128
9	278301	WPE	I	M	N	18/07/1999	13/12/1979	25	1	134
10	864037	IB	I	M	N	28/10/1993	04/09/1959	0	0	852
11	A03778	WPE	I	F	S	23/12/1997	09/10/1954	10	12	207
12	842335	WPE	I	F	N	14/08/1992	20/05/1953	10	1	150
13	143987	IB	I	M	N	16/07/1997	09/08/1971	0	0	2330
14	404066	IB	D	M	N	18/07/1995	18/07/1975	0	0	1981
15	C00039	WPE	I	M	N	02/01/1999	01/02/1967	25	1	113
16	630060	WPE	I	F	N	25/05/2000	23/11/1973	25	12	114
17	810327	WPE	I	M	S	25/03/1990	03/09/1969	25	1	226
18	129477	IB	I	F	N	14/12/1996	21/11/1953	0	0	2067
19	M59814	WPE	I	M	N	25/03/1990	19/01/1962	25	1	64
20	435795	WPE	I	M	N	10/05/1995	12/10/1957	25	12	62
21	166647	WPE	I	M	N	16/02/1995	22/11/1971	25	1	181
22	166552	WPE	I	F	N	27/08/1996	19/07/1974	10	2	98

You can now see the values displayed properly for all the records in the input file.

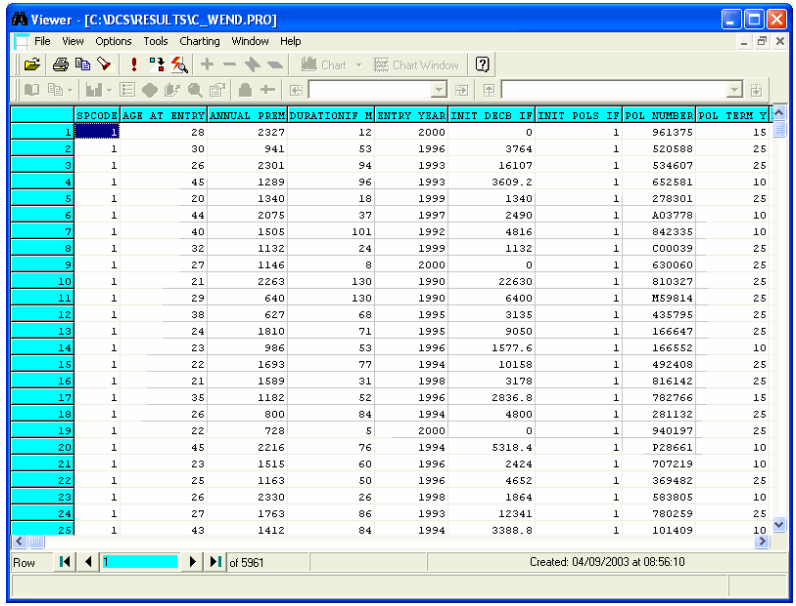
You should now close Viewer and return to DCS.

Re-running the program

You should click **Run** to re-run the program. This time it should run all the way through without any problems.

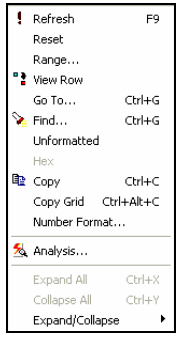
Viewing the output files

You should click the **View Output Files** toolbar button  . Viewer will be loaded again but with the Open DCS Output File dialog displayed so that you can select one of the two output files created in the run. Select C_WEND.PRO and click **Open**. The following screen will be displayed, although the order of the variables may be different depending on the order you selected them in the output format.



	SPCODE	AGE AT ENTRY	ANNUAL PREM	DURATIONIF M	ENTRY YEAR	INIT DECB IF	INIT POLS IF	POL NUMBER	POL TERM Y
1	1	28	2327	12	2000	0	1	961375	15
2	1	30	941	53	1996	3764	1	520588	25
3	1	26	2301	94	1993	16107	1	534607	25
4	1	45	1289	96	1993	3609.2	1	652581	10
5	1	20	1340	18	1999	1340	1	278301	25
6	1	44	2075	37	1997	2490	1	A03778	10
7	1	40	1505	101	1992	4816	1	842335	10
8	1	32	1132	24	1999	1132	1	C00039	25
9	1	27	1146	8	2000	0	1	630060	25
10	1	21	2263	130	1990	22630	1	810327	25
11	1	29	640	130	1990	6400	1	M59814	25
12	1	38	627	68	1995	3135	1	435795	25
13	1	24	1810	71	1995	9050	1	166647	25
14	1	23	986	53	1996	1577.6	1	166552	10
15	1	22	1693	77	1994	10158	1	492408	25
16	1	21	1589	31	1998	3178	1	816142	25
17	1	35	1182	52	1996	2836.8	1	782766	15
18	1	26	800	84	1994	4800	1	281132	25
19	1	22	728	5	2000	0	1	940197	25
20	1	45	2216	76	1994	5318.4	1	P28661	10
21	1	23	1515	60	1996	2424	1	707219	10
22	1	25	1163	50	1996	4652	1	369482	25
23	1	26	2330	26	1998	1864	1	583805	10
24	1	27	1763	86	1993	12341	1	780259	25
25	1	43	1412	84	1994	3388.8	1	101409	10

You should have a look at the options available in Viewer which enable you to view both input and output files. In particular you should look at the options on the right click menu:



LESSON 2

Validation and correction

In this lesson you will add validation and correction rules to the program you created in Lesson 1.

TOPICS COVERED IN THIS LESSON

Validation and correction functions

DCS provides 5 functions to carry out validation and correction:

Function	Description
INVALID	Validates a specific variable.
TABLE_VALIDATE	Validates a number of variables against a table of values.
CORRECT	Validates a specific variable and changes its value to a specified value if it fails the validation.
TABLE_CORRECT	Validates and corrects a number of variables against a table of values.
FAILED_VALIDATION	If a variable fails one of the 4 functions mentioned above for the current record then this function returns a value of TRUE. Otherwise it returns a value of FALSE.

If a variable fails the validation then a message is written to the run log or a validation file. Similarly if a variable is corrected a message is written to the run log or a correction file.

You can use IF / THEN /ENDIF and SWITCH statements to control the conditions under which these functions are used. This enables you to specify very precisely the validation and correction rules to be applied. This lesson demonstrates the use of both these statements.

Updating your program to carry out validation and correction

The following validation and correction rules should be added to your program:

- Any with profit endowment where the policy term is less than 10 years should be assumed to be invalid.
- Any with profit endowment policy where the premium frequency is not monthly, quarterly, half-yearly or annual should be assumed to be wrong and that it should be corrected to have a frequency of monthly.
- The minimum and maximum ages at entry should be assumed to be as follows, and if they are outside these ranges then they should be corrected as shown below:

Product	Minimum Age	Maximum Age	Corrected Age
With Profit Endowment	20	55	30
Investment Bond	20	75	50

This information needs to be set up in a 3-dimensional generic table as follows:

	A	B	C
1	PROD_NAME	VAR_NAME	AGE_AT_ENTRY
2	C_WEND	MIN_VAL	20
3	C_WEND	MAX_VAL	55
4	C_WEND	CHANGE_VAL	30
5	U_IB__	MIN_VAL	20
6	U_IB__	MAX_VAL	75
7	U_IB__	CHANGE_VAL	50

A file called CORRECT.FAC containing this information already exists in the TABLES sub-folder of the DCS location. If you wish to create this generic table yourself you should do so in Prophet, save it in a Table Location and then copy or move it to the TABLES sub-folder of the DCS location.

- A variable should be created to record whether the policy failed the validation.

To achieve this you should enter the following into your TUTOR1 program after the line which sets INIT_POLS_IF to 1. Use the Function Wizard and Insert Statement buttons to enter the functions and statements. As you do this click **Help** in the Function Wizard to understand how each function works.

```
; Validation of policy term
  IF PRODUCT = "WPE" AND POL_TERM_Y < 10 THEN
    INVALID(POL_TERM_Y)
  ENDIF

; Correction of premium frequency
  IF PRODUCT = "WPE" THEN
    SWITCH(PREM_FREQ)
    CASE 1,2,4,12:
  ; For these values nothing is done
    DEFAULT: CORRECT(PREM_FREQ,12)
    ENDSWITCH
  ENDIF

; Table correction of ages at entry
  PROD_NAME = FILENAME
  TABLE_CORRECT("CORRECT", PROD_NAME, AGE_AT_ENTRY)

; Variable to record failure of validation / correction
; tests
  IF FAILED_VALIDATION THEN
    IF_FAILED = 1
  ELSE
    IF_FAILED = 0
  ENDIF
```

Validation and correction message files

If you were to run the program now all the validation and correction messages would be written to the run log. However it is better to write them to specific validation and correction files instead because:

- The messages are not mixed up with other messages that are written to the run log.
- The files can be in model point, text, Access, Glean or ODBC formats.
- You can specify that additional variables are included in the files, such as the policy number.

In the run log only the following information is shown for each variable that fails the validation:

Record number
Variable failing the validation
Value that failed the validation
Corrected value (if applicable)

In the validation and correction message files this information is also always included. However, by setting up an output format you can specify the additional variables to be included.

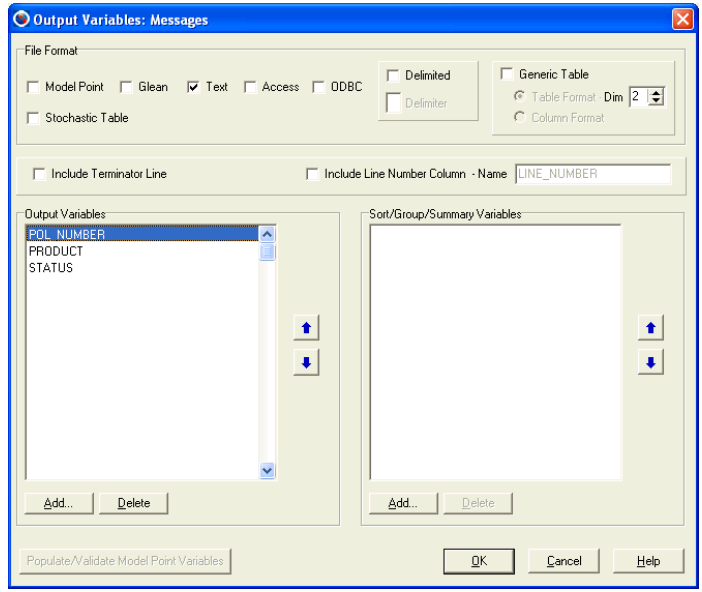
If you elect to write the validation and correction messages to separate files the messages do not appear in the run log.

Messages output format

Select the **Output Details** view and then create a new output format called Messages. Include the following variables in it:

POL_NUMBER
PRODUCT
STATUS

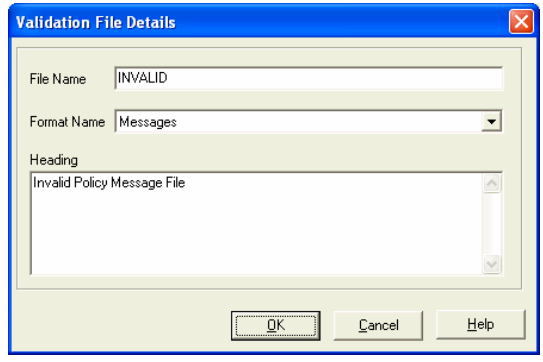
You should create the message files in Text format, so select the Text checkbox and uncheck the Model Point checkbox. It should therefore look as follows:



Note: If you want to you can also select Sorting variables so that the records in the validation and correction files appear in the order you want them to.

Validation messages

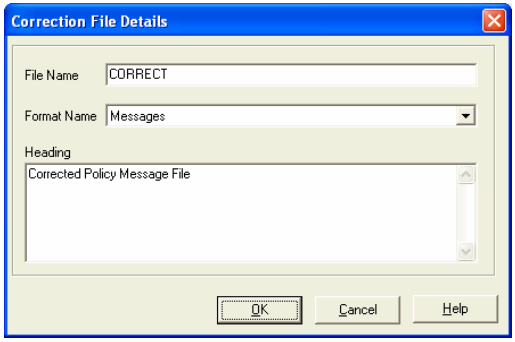
You should now select the **Validation** option and click the **File Details** button next to it. The following dialog will appear except that initially the entries will be blank:



Enter the details shown above and click **OK**.

Correction messages

Repeat this process for the **Correction** option and enter the information shown in the following dialog:

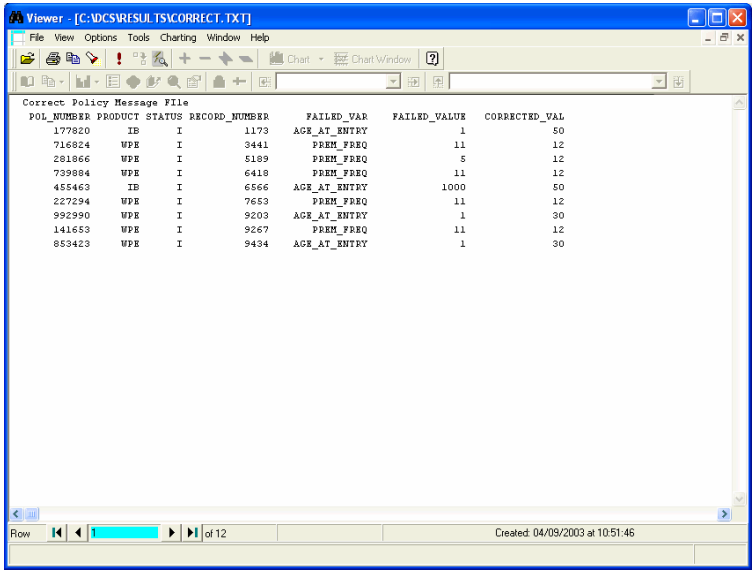


Run Settings

You need to specify the Location for Tables in the **Run Settings** view as the TABLES sub-folder because this is where the CORRECT.FAC file is held.

Running the program

Run your program and look at the output files INVALID.TXT and CORRECT.TXT in Viewer. For CORRECT.TXT the view will be as follows:



You can see from this how the values have been changed by the DCS program.

LESSON 3

Data grouping in DCS

In this lesson you will enhance your DCS program so that it produces grouped model point files instead of individual policy model point files. You will then further enhance it to produce both grouped and individual policy model point files from a single run.

TOPICS COVERED IN THIS LESSON

Details of the data grouping to be performed

In this lesson you will group the policies in the input file according to the following criteria:

Investment bond

The investment bond should just be grouped by age at entry:

Grouping Variable	Grouping Criteria
AGE_AT_ENTRY Age at entry	Policies should be grouped into the following bands: 19 and less 20 to 30 31 to 40 41 to 45 46 to 50 51 to 60 61 and over

With profit endowment

The with profit endowment should be grouped by three criteria:

Grouping Variable	Grouping Criteria
DURATIONIF_M Duration in force in months	The duration in force should be grouped monthly for business sold in the last 2 years. Business which is more than 2 years old but less than 25 years old should be grouped by year of issue. Business which is more than 25 years old should all be grouped together.
POL_TERM_Y Policy term in years	The policy term should be grouped into 5-yearly bands which are centred on the quinquennial policy terms. Policy terms of more than 28 years should all be grouped together.
PREM_FREQ Premium frequency	Monthly and quarterly policies should be grouped together and half yearly and annual policies should be grouped together.

The grouping criteria that you use need to reflect the features of the product that you know are important as regards profitability. Refer to Section A in the Reference Section of this manual for more information.

Enhancing the code for grouping

In order to group on a particular variable you normally need to create another variable which has a unique value for each group into which a policy can be placed. Therefore to group on AGE_AT_ENTRY for the Investment Bond a variable called AGE_GROUP should be created which is given 7 different values, one corresponding to each of the possible age bands. To achieve this you should enter the following in the code after the validation and correction lines:

```
; Calculation of grouping age
  IF AGE_AT_ENTRY <= 19 THEN
    AGE_GROUP = 18
  ELSEIF AGE_AT_ENTRY <= 30 THEN
    AGE_GROUP = 25
  ELSEIF AGE_AT_ENTRY <= 40 THEN
    AGE_GROUP = 35
  ELSEIF AGE_AT_ENTRY <= 45 THEN
    AGE_GROUP = 42
  ELSEIF AGE_AT_ENTRY <= 50 THEN
    AGE_GROUP = 47
  ELSEIF AGE_AT_ENTRY <= 60 THEN
    AGE_GROUP = 55
  ELSE
    AGE_GROUP = 65
  ENDIF
```

Note: In the formula above the grouping variable has been given values which correspond to approximately the mid points of the bands of AGE_AT_ENTRY. However, it would work equally as well if values of 1 to 7 were used instead. It is the fact that each band has a different value which is important.

Where the width of the band is the same for most or all of the bands it is easier to use the INT function rather than enter all the values using an IF statement. To achieve this for DURATIONIF_M enter the following code:

```
; Calculation of duration in months grouping period
  IF DURATIONIF_M <= 24 THEN
    DUR_GROUP = DURATIONIF_M
  ELSEIF DURATIONIF_M > 300 THEN
    DUR_GROUP = 301
  ELSE
    DUR_GROUP = INT((DURATIONIF_M - 25)/12) * 12 + 25
  ENDIF
```

Again, you can see how a unique value has been assigned for each band into which the policies are to be grouped.

Enter the following into the code to carry out the grouping of POL_TERM_Y and PREM_FREQ:

```
; Calculation of policy term grouping period
  IF POL_TERM_Y < 3 THEN
    POL_GROUP = 2
  ELSEIF POL_TERM_Y > 28 THEN
    POL_GROUP = 29
  ELSE
    POL_GROUP = INT((POL_TERM_Y - 3)/5) * 5 + 3
  ENDIF

; Calculation of premium frequency for grouping
  IF PREM_FREQ <= 2 THEN
    PREM_FREQ = 1
  ELSE
    PREM_FREQ = 12
  ENDIF
```

Note that in the case of PREM_FREQ a separate grouping variable has not been used. Instead the value of PREM_FREQ has been changed. Generally it is better to use a separate grouping variable for two reasons:

- You will get a better model if for variables such as the AGE_AT_ENTRY and DURATIONIF_M the weighted average of the policies going into each group is used rather than the mid-point of the range being used.
- If you will be creating several sets of grouped output files from within the same program you may want to use the exact values in a later calculation. You won't be able to do this if the exact value has been overwritten with the mid-point of the range.

In the case of PREM_FREQ the first point doesn't apply because it would obviously not be correct to calculate the average premium frequency since this might give a value of 11 which could not be properly processed by the formulas in your Prophet products.

The second point might apply to PREM_FREQ. If this were the case you would need to create a variable called (say) ACTUAL_FREQ which was set to be the same as the original premium frequency and which was then used in place of PREM_FREQ in the first line of the formula above.

The final change to the code is to replace the use of the OUTPUT function by the GROUP function in the two places where it occurs. The three arguments for the GROUP function (that is the output format name, the file name and the heading) are the same as for the OUTPUT function and you should therefore leave these unaltered.

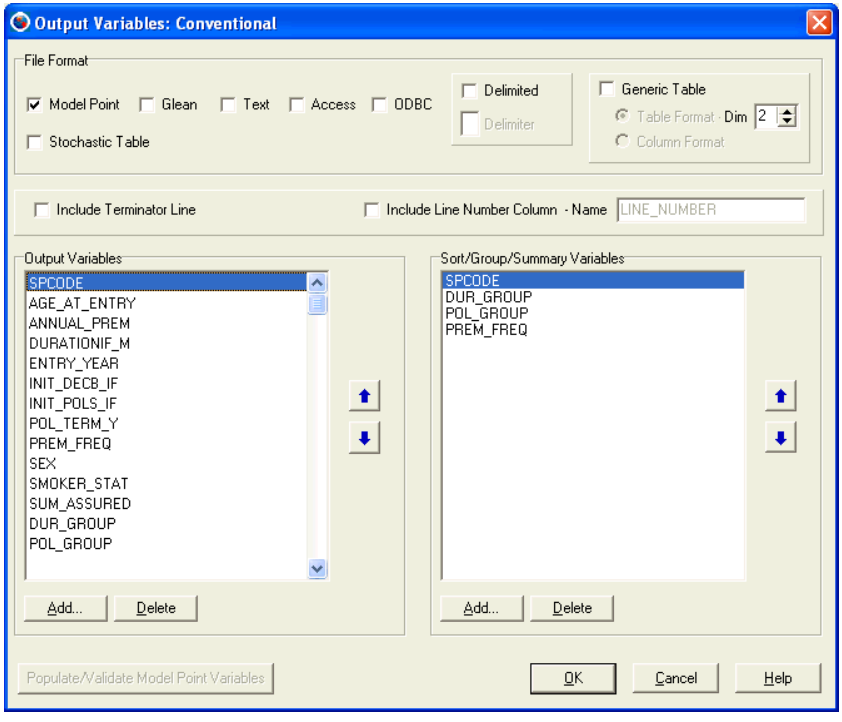
Changes to the output formats

You need to update the output formats to use the new grouping variables that you have created. Therefore open the Conventional output format. You need to add the DUR_GROUP and POL_GROUP variables to the list of output variables. You then need to add DUR_GROUP, POL_GROUP and PREM_FREQ to the list of Sorting, Grouping and Summarising variables.

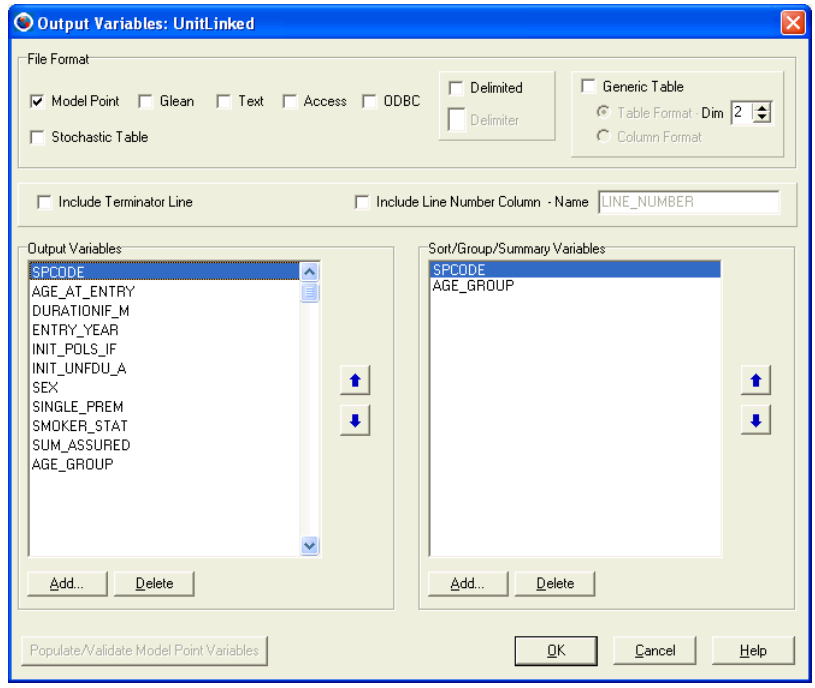
You also need to remove the variable POL_NUMBER from the output formats. The reason for this is that POL_NUMBER is a text variable and hence average or summed values cannot be calculated for it.

Note: SPCODE must still be the first variable in the list of Sorting, Grouping and Summarising variables because the existing business model points with an SPCODE value of 1 must be grouped separately from the new business model points with a value of 51.

Once you have made the changes the Conventional output format should be as follows:

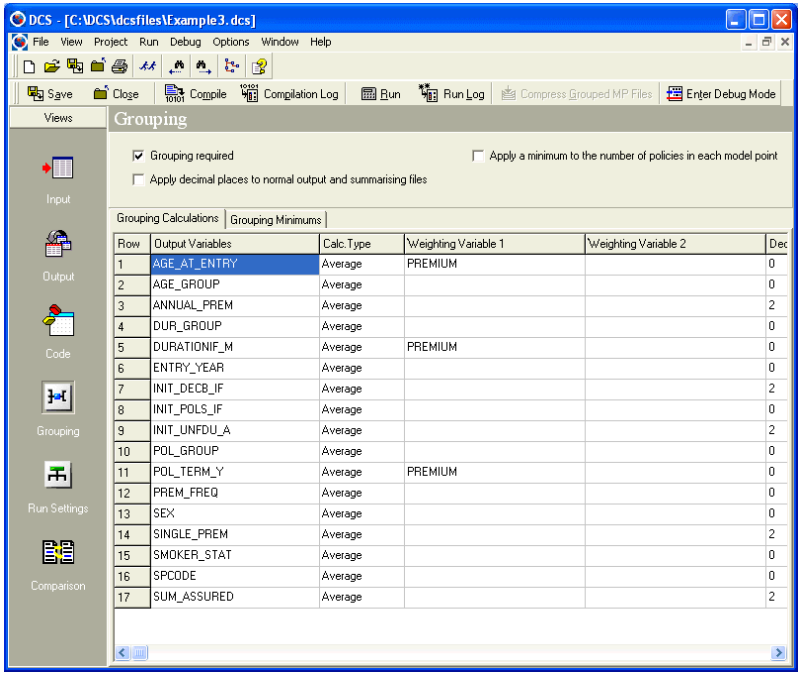


You also need to make changes to the UnitLinked format to add AGE_GROUP and delete POL_NUMBER so that it is as follows:



The Grouping view

You should now click on the **Grouping** icon in the Views list. The Grouping view appears as follows after you have selected the **Grouping required** checkbox, except that initially some of the entries will be blank:



The Grouping view lists all the numeric variables which are used in any of the output formats. For each one you need to specify how the values for grouped output are to be calculated. You have the following options:

Calc. Type
You can select either Average, Sum or SumLog. You should select Sum for variables such as INIT_POLS_IF where you want to sum up the values for all the policies going into a particular model point. However, for all the other variables you should select Average.

Weighting Variable 1 and Weighting Variable 2
You can select variables to be used to weight the calculation of the average value. For example, if you select PREMIUM for the variable AGE_AT_ENTRY then policies with a larger premium will be given a larger weighting in the calculation, which should result in a better data model. You can select either none, 1 or 2 weighting variables.

Decimal
You can select or enter the number of decimal places to be used for the values in the grouped output files. You need to ensure that you specify a value of 0 for variables such as POL_TERM_Y where the formulas in your products are set up to only expect integral values. Since you will normally be using the weighted average policy term the values will not be output as integral values if you specify anything other than 0.

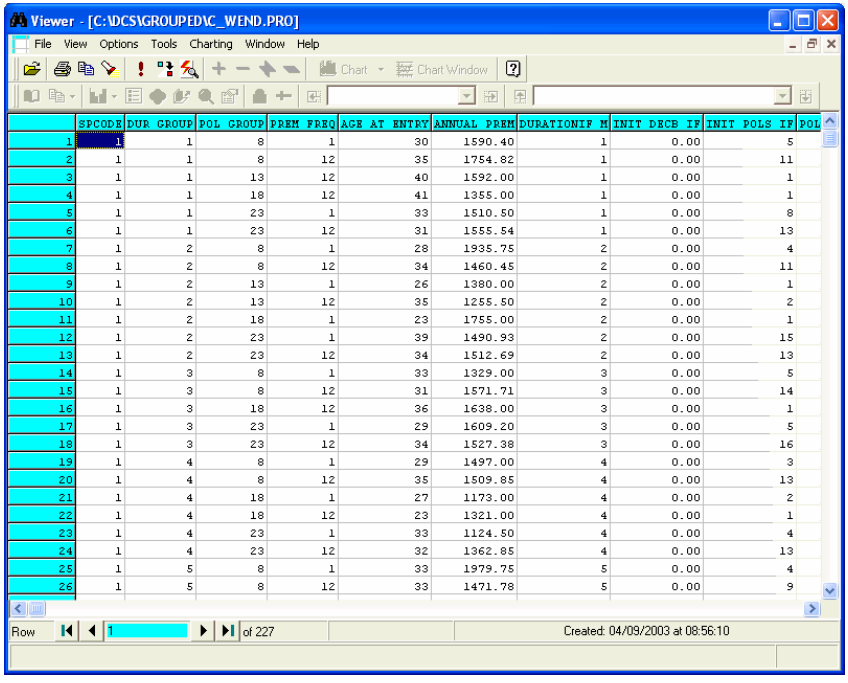
You should modify the Grouping view so that it contains the following information:

Output Variables	Calculation Type	Weighting Variable 1	Weighting Variable 2	Decimal Places
AGE_AT_ENTRY	Average	PREMIUM		0
AGE_GROUP	Average			0
ANNUAL_PREM	Average			2
DUR_GROUP	Average			0
DURATIONIF_M	Average	PREMIUM		0
ENTRY_YEAR	Average			0
INIT_DECB_IF	Average			2
INIT_POLS_IF	Sum			0
INIT_UNFDU_A	Average			2
POL_GROUP	Average			0
POL_TERM_Y	Average	PREMIUM		0
PREM_FREQ	Average			0
SEX	Average			0
SINGLE_PREM	Average			2
SMOKER_STAT	Average			0
SPCODE	Average			0
SUM_ASSURED	Average			2

Running your program

You should now change the Run Settings so that the Location for Output Files is set to GROUPED. This will ensure that you don't overwrite the output files that you created in earlier lessons.

You should now run your program in the normal manner. If you then look at the output files using Viewer you will see that they contain far fewer lines and that the variable INIT_POLS_IF has values of more than 1 to reflect the fact that each model point line now represents more than one policy. For example, for C_WEND.PRO the screen will appear as follows:



	SPCODE	DUR GROUP	POL GROUP	PREM FREQ	AGE AT ENTRY	ANNUAL PREM	DURATIONIF M	INIT DECB IF	INIT POLS IF	POL
1	1	1	8	1	30	1590.40	1	0.00	5	
2	1	1	8	12	35	1754.82	1	0.00	11	
3	1	1	13	12	40	1592.00	1	0.00	1	
4	1	1	18	12	41	1355.00	1	0.00	1	
5	1	1	23	1	33	1510.50	1	0.00	8	
6	1	1	23	12	31	1555.54	1	0.00	13	
7	1	2	8	1	28	1935.75	2	0.00	4	
8	1	2	8	12	34	1460.45	2	0.00	11	
9	1	2	13	1	26	1380.00	2	0.00	1	
10	1	2	13	12	35	1255.50	2	0.00	2	
11	1	2	18	1	23	1755.00	2	0.00	1	
12	1	2	23	1	39	1490.93	2	0.00	15	
13	1	2	23	12	34	1512.69	2	0.00	13	
14	1	3	8	1	33	1329.00	3	0.00	5	
15	1	3	8	12	31	1571.71	3	0.00	14	
16	1	3	18	12	36	1638.00	3	0.00	1	
17	1	3	23	1	29	1609.20	3	0.00	5	
18	1	3	23	12	34	1527.38	3	0.00	16	
19	1	4	8	1	29	1497.00	4	0.00	3	
20	1	4	8	12	35	1509.85	4	0.00	13	
21	1	4	18	1	27	1173.00	4	0.00	2	
22	1	4	18	12	23	1321.00	4	0.00	1	
23	1	4	23	1	33	1124.50	4	0.00	4	
24	1	4	23	12	32	1362.85	4	0.00	13	
25	1	5	8	1	33	1979.75	5	0.00	4	
26	1	5	8	12	33	1471.78	5	0.00	9	

Multiple grouping criteria

You can create more than one set of grouped output files using a single program. You can also create both grouped output files and individual policy output files using a single program. This requires either that the different sets of output files are written to different locations or that different file extensions are used for each set.

Using different output paths

The OUTPUT_FILE_PATH function allows you to change the output file path from that specified in the Run Settings view as the program is running. For example, to change the output file path to the GROUPED2 sub-folder of the DCS location you would enter the following in the code:

```
OUTPUT_FILE_PATH("GROUPED2")
```

To reset the path back to the path specified in the Run Settings view you would enter one of the following:

```
OUTPUT_FILE_PATH()
OUTPUT_FILE_PATH("")
OUTPUT_FILE_PATH
```

You should update your program to produce both individual policy and grouped output files by making the following changes immediately before each of the two lines that contain the GROUP function:

```
; Set output file path for individual policy output files
  OUTPUT_FILE_PATH("RESULTS2")
; Create individual policy output
  OUTPUT(OUTPUT_FORMAT, FILENAME, HEADING)
; Reset output file path for grouped output files
  OUTPUT_FILE_PATH
```

This will cause the individual policy results files to be written to the RESULTS2 sub-folder and the grouped output files to be written to the GROUPED sub-folder specified in the Run Settings view.

Using different file extensions

DCS uses the following default file extensions for output files if you do not specify an extension:

File Type	File Extension
Model point format	.PRO
Text format	.TXT
Glean data file format	.GDF
Delimited format	.DEL
Generic table format	.FAC

In the case of ODBC format files the default extension for the particular database format you are using is always used.

If you enter a file extension in the name of the output file in the OUTPUT or GROUP function calls then that extension will be used for the output files. If you have specified that both model point and text output files are to be produced for a particular output format then the specified extension will be used for the model point format output files, with .TXT being used for the text format output files.

You should change the output file names in your program by adding ".RPT" to the three lines where the variable FILENAME is set, for example:

```
FILENAME = "C_WEND.RPT"
```

You also need to change the line where PROD_NAME is set to exclude the extension as follows:

```
PROD_NAME = SUBSTR(FILENAME,1,6)
```

If you do not do this an error message will be produced during the run because the product names in the CORRECT generic table do not include the extension.

You should now rerun the program. Check that the model point format output files have been created with a .RPT extension rather than .PRO.

LESSON 4

Further DCS topics

In this lesson you will cover a number of miscellaneous further topics.

TOPICS COVERED IN THIS LESSON

Explicit declarations

In the program that you have set up in this tutorial any variables that you created in the code were automatically declared by DCS when it generated the C code from your program. There are two risks with this approach:

- If you slightly misspell the name of a variable somewhere in the code it will be treated as a different variable by DCS without you being aware of this.
- The first occurrence of each variable in the code determines the type that it is declared to be. For example, if the first occurrence of the variable A is "A=0" then it will be declared as an integer. If you subsequently use it as if it were a real number then the values will be truncated. To ensure that it is declared as a real number you need to ensure that the first occurrence is "A=0.0".

You can avoid both these problems occurring by including the following function at the top of your code:

```
EXPLICIT_DECLARATIONS
```

If this is present DCS will produce an error message during code generation for any variable which is not explicitly declared in the code. To declare a variable in the code you need to enter its name followed by its type at the top of the code.

You should first add EXPLICIT_DECLARATIONS as the first line of your TUTOR1 program and then try to compile it. You will then see all the variables that have not been declared listed in the compilation messages. You should then add the following declaration statements after the EXPLICIT_DECLARATIONS line and then recompile:

```
; Variable declarations
  VAL_YEAR INTEGER
  VAL_YEAR_T TEXT*4
  ENTRY_YEAR INTEGER
  AGE_AT_ENTRY INTEGER
  DURATIONIF_M INTEGER
  ANNUAL_PREM NUMBER
  SINGLE_PREM NUMBER
  SEX INTEGER
  SMOKER_STAT INTEGER
  OUTPUT_FORMAT TEXT
  FILENAME TEXT
  HEADING TEXT
  SPCODE INTEGER
  INIT_POLS_IF INTEGER
  PROD_NAME TEXT
```

```
IF_FAILED INTEGER
AGE_GROUP INTEGER
DUR_GROUP INTEGER
POL_GROUP INTEGER
RECORDS INTEGER
```

Note: For text variables you can specify the length of the variable as shown above for VAL_YEAR_T. If you do not specify a length DCS uses a default length of 65. Any trailing blanks are automatically suppressed when the output is produced.

Summarising

The SUMMARISE function allows you to summarise the information in the input file. For example, you could use this function to produce a summary for all the records in your input file of the number of policies, premium and sum assured by product and by status.

You are able to summarise the values of variables created in the code as well as those in the input file. The exact values that are summarised depends on where you place the SUMMARISE function in your code. If you place it right at the start then it will just be the values as held in the input file that are summarised. However, if you place it in the middle or near the end of your code then the values calculated in the code up to that point are reflected in the summary.

Changing the code

You first have to include the SUMMARISE function in the code. Therefore select the Code view and enter the following immediately after the variable declarations:

```
; Summarise the policy data based on PRODUCT and STATUS
  RECORDS = 1
  SUMMARISE("Summary","SUMMARY","Summary by PRODUCT and STATUS")
```

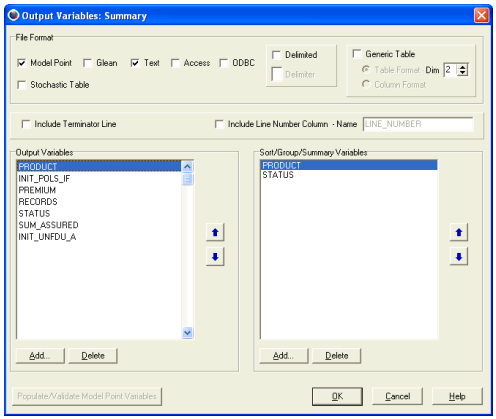
The SUMMARISE function has exactly the same arguments as the OUTPUT function, that is Output Format, File Name and Heading.

The "RECORDS = 1" line is included to provide a count of the number of records in the summary.

Summary output format

You now nned to create the Summary output format to specify the variables to be included in your summarised file and the variables that are used to summarise them.

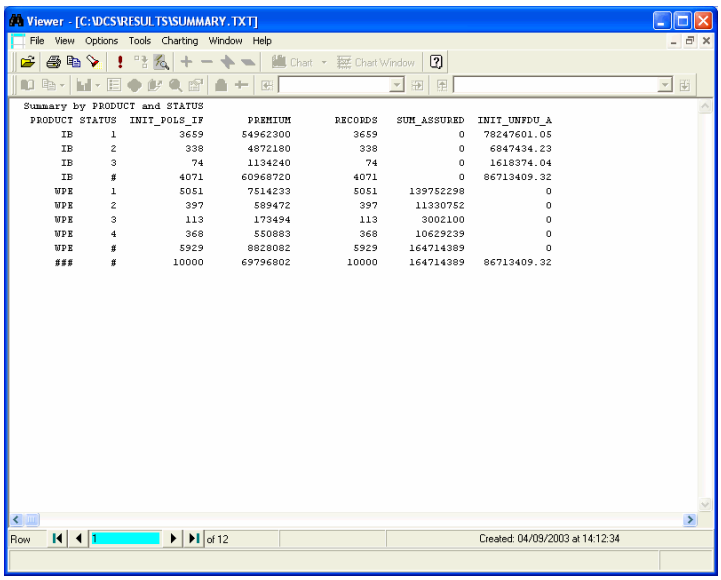
Therefore create a new output format with the name Summary as follows:



This will produce a summary by product and status.

Run the program

If you now run the program and look at the results file SUMMARY.TXT in Viewer you will see that the following results have been produced:



The # characters in the STATUS column indicate lines which total the values for the D, I and S statuses. Similarly for ### in the PRODUCT column.

For summaries of numeric variables - 9999 appears instead of the # characters.

Generic tables

The READ_GENERIC_TABLE function allows you to read values from generic tables in the same way as within Prophet. This allows you to use quite complex tabular data in your programs. However, for the tutorial you will just use this function to read the valuation year from a generic table rather than explicitly using the extract date.

Changing the code

To do this select the Code view and replace the code which sets the valuation year with the following:

```
; Set valuation year
  IF FIRST_RECORD THEN
    VAL_YEAR = READ_GENERIC_TABLE("EX1_CTRL","Y","VAL_YEAR")
    VAL_YEAR_T = STRVAL(VAL_YEAR)
  ENDIF
```

The FIRST_RECORD function is used so that the valuation year is only read from the generic table once. Reading generic tables can be quite slow and you should therefore only read them when you have to.

Setup Run

If you are using generic tables you must specify the location in which they are held in a similar manner to specifying the Location for Tables in a structure in Prophet. Select the Run Settings view and check that the Location for Tables is set to TABLES. This means that the generic tables will be read from the TABLES sub-folder of the location in which the DCS program is held.

Run the program

The EX1_CTRL.FAC file is supplied with the DCS tutorial files.

You can now run the program and it will read the valuation year of 2000 from the generic table.

ODBC format output files

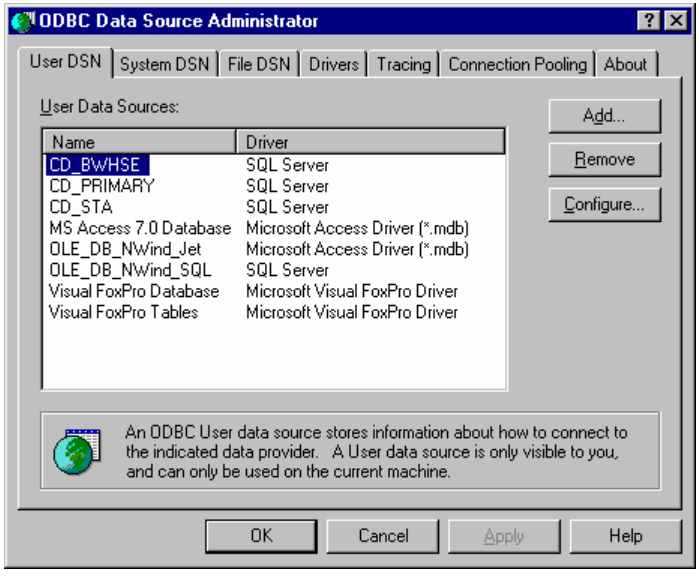
ODBC stands for Open Database Connectivity. It enables you to produce files in a variety of database formats such as Access. To do this you need to carry out the following steps:

Select ODBC option in output formats

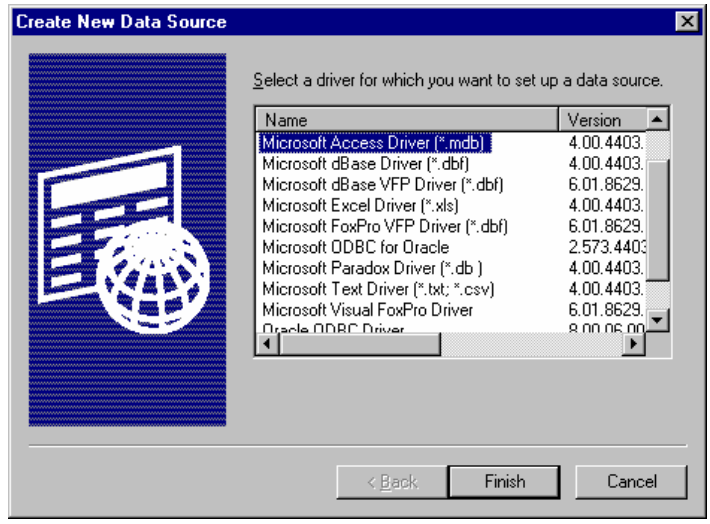
For all the output formats for which you want the results files to be in an ODBC format select the ODBC File Format option for that output format. You should open each of the Conventional and UnitLinked output formats and select ODBC format.

Create a data source

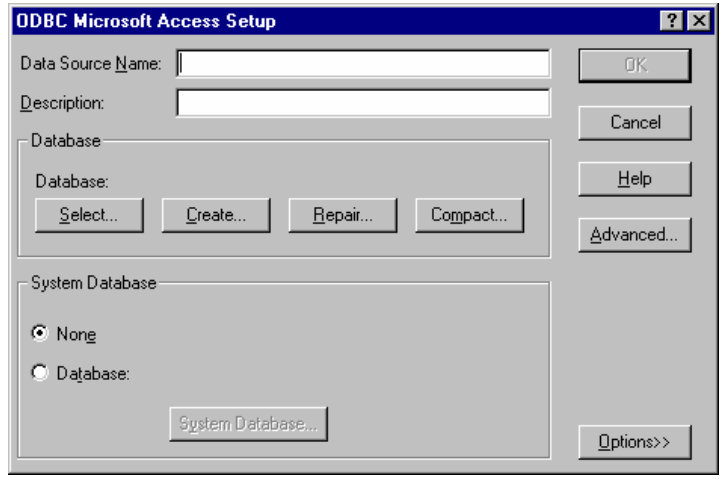
You need to create a data source which contains details for the files that you want to create in an ODBC format. To do this you use the "Administrator" button to the right of the ODBC Output Data Source option in the Run Settings view. When you click this button the following dialog appears, although the entries will vary according to the ODBC drivers installed on your PC:



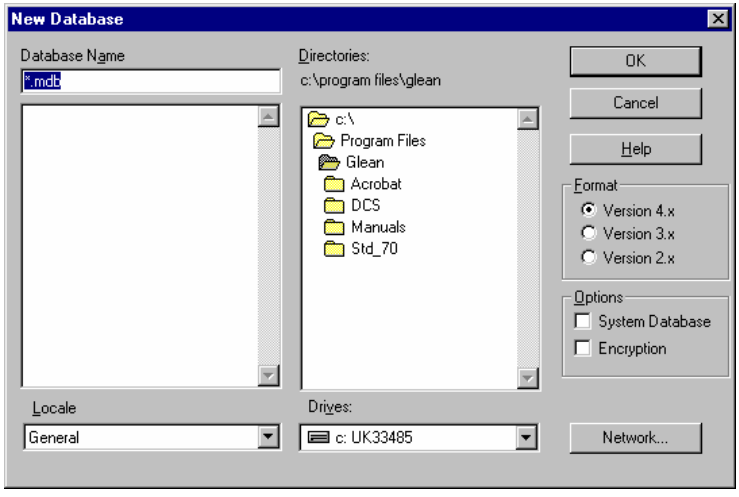
Click **Add** on the "User DSN" tab and the following dialog will appear, although the precise entries will vary depending on which ODBC drivers are installed on your PC:



Select the ODBC driver that you require depending on the database format that you require. For the purposes of the tutorial you should select the "Microsoft Access Driver (*.mdb)" option and click **Finish**. The following dialog will then appear (if you had chosen a different driver this dialog would be different):



You should enter both the Data Source Name and the Description as "Tutorial Data Source". You should then click **Create** so that the output files are created as tables in a new Access database. The following dialog will appear:



Enter the Database Name as TUTORIAL.MDB and select the directory as C:\DCS\GROUPED. Then click **OK**. A message should appear advising you that the database has been successfully created. Click **OK** on this message and again to save the data source. Then click **OK** or **Cancel** to close the ODBC Administrator.

Select Data Source in Run Settings view

In the Run Settings view select "Tutorial Data Source" in the ODBC Output Data Source drop down list box.

Changing the Code

You cannot use the OUTPUT_FILE_PATH function in programs that produce ODBC format output files. The reason for this is that it is the ODBC data source that specifies the path to which the results are written. You therefore need to remove the following lines in the two places in which they appear in the code:

```
; Set output file path for individual policy output files
  OUTPUT_FILE_PATH("RESULTS2")
; Create individual policy output
  OUTPUT(OUTPUT_FORMAT, FILENAME, HEADING)
; Reset output file path for grouped output files
  OUTPUT_FILE_PATH
```

You also need to select ODBC in one or more output formats otherwise no ODBC output is created.

Run the program

You should now run the program in the normal manner.

At the end of the run you should be able to see a file called TUTORIAL.MDB in your C:\DCS\GROUPED folder in Windows Explorer. To open the file and view the results it contains you need to use Access or another package that can read Access format files. If you are able to do this then do so now.

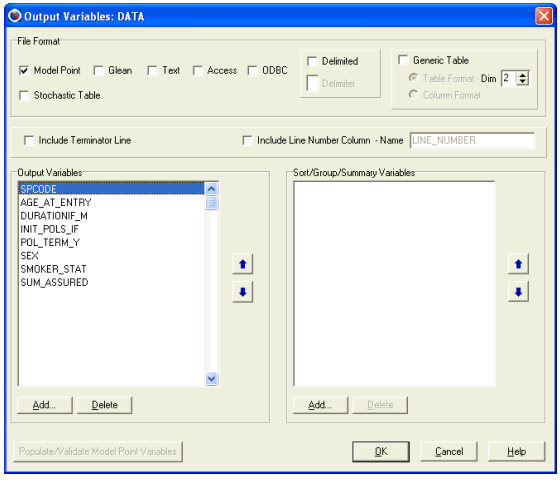
Creating model point files for product design work

You can use DCS to create model point files for product design work by creating programs that don't have an input file. For example, you might want to use Prophet to calculate a set of term assurance premium rates for every possible combination of age, policy term, sex and smoker status using Prophet's goal seeking facility. To do this you need to create a model point file that contains all the possible combinations.

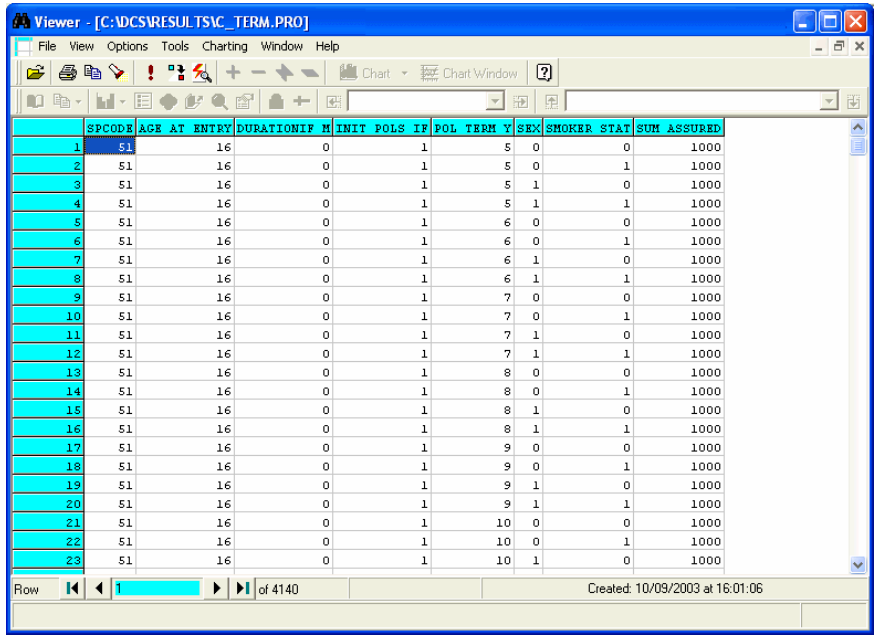
Select **New** option in the **File** menu and choose **DCS Program File** to create a new program. Enter the Program Description as "Tutorial Program 2" and then enter the following code:

```
SPCODE = 51
SUM_ASSURED = 1000
INIT_POLS_IF = 1
DURATIONIF_M = 0
FOR AGE_AT_ENTRY FROM 16 TO 60
  FOR POL_TERM_Y FROM 5 TO 65 - AGE_AT_ENTRY
    FOR SEX FROM 0 TO 1
      FOR SMOKER_STAT FROM 0 TO 1
        OUTPUT("DATA", "C_TERM",
        "Data for Determining Premium Rates")
      ENDFOR
    ENDFOR
  ENDFOR
ENDFOR
```

When there is no input file the code is only processed once. However, the use of the FOR loops in the code above means that many output lines will be written to the output file. You also need to create an output format with the name DATA as follows:



You should now set up the run to write the results to the RESULTS2 sub-folder. Run the program, saving it as TUTOR2 when prompted for a name. Then view the C_TERM.PRO output file that has been created:



	SPCODE	AGE AT ENTRY	DURATIONIF M	INIT POLS IF	POL TERM Y	SEX	SMOKER STAT	SUM ASSURED
1	51	16	0	1	5	0	0	1000
2	51	16	0	1	5	0	1	1000
3	51	16	0	1	5	1	0	1000
4	51	16	0	1	5	1	1	1000
5	51	16	0	1	6	0	0	1000
6	51	16	0	1	6	0	1	1000
7	51	16	0	1	6	1	0	1000
8	51	16	0	1	6	1	1	1000
9	51	16	0	1	7	0	0	1000
10	51	16	0	1	7	0	1	1000
11	51	16	0	1	7	1	0	1000
12	51	16	0	1	7	1	1	1000
13	51	16	0	1	8	0	0	1000
14	51	16	0	1	8	0	1	1000
15	51	16	0	1	8	1	0	1000
16	51	16	0	1	8	1	1	1000
17	51	16	0	1	9	0	0	1000
18	51	16	0	1	9	0	1	1000
19	51	16	0	1	9	1	0	1000
20	51	16	0	1	9	1	1	1000
21	51	16	0	1	10	0	0	1000
22	51	16	0	1	10	0	1	1000
23	51	16	0	1	10	1	0	1000

When you have done this close TUTOR2.

Creating new business model points which are not based on existing business

In Lesson 1 the new business model points were assumed to be based on the existing business that was sold in the previous year. There may be situations where this type of approach is not possible for the following reasons:

- The product that you wish to project is planned to be introduced in the future and there is therefore no existing business
- The mix of different ages, policy terms, premium sizes, etc for a product is expected to be significantly different in the future, perhaps due to tax changes or to a planned change to the marketing of the product

An approach similar to the one described in the previous section can be used in this situation. However, you need to specify that it is only run once during the processing of the input file because you do not want to create the new business model points every time a record in the input file is processed.

Suppose that you plan to introduce a second type of unit linked investment bond during the coming year and that you expect 25% of the policies to be sold to 40 year olds with an average premium size of 5000 and 75% to 50 year olds with an average premium size of 10000. To achieve this add the following code to the end of the code for program TUTOR1:

```
; Create new business model points for U_IB2_
  IF NEW_BUS_OUTPUT = 0 THEN
    SPCODE = 51
    DURATIONIF_M = 0
    INIT_UNFDU_A = 0
    FOR I FROM 1 TO 2
      SWITCH (I)
        CASE 1: SINGLE_PREM = 5000
                AGE_AT_ENTRY = 40
                INIT_POLS_IF = 1
        CASE 2: SINGLE_PREM = 10000
                AGE_AT_ENTRY = 50
                INIT_POLS_IF = 3
        DEFAULT:
      ENDSWITCH
      OUTPUT("UnitLinked", "U_IB2_", "")
      NEW_BUS_OUTPUT = 1
    ENDFOR
  ENDIF
```

Note: When the program is run NEW_BUS_OUTPUT will be given a default value of 0. Therefore the code above will be processed for the first record which does not have a terminated status (if a record has a terminated status this code will not be processed because of the use of the NEXT_RECORD function earlier in the code). When the code is processed NEW_BUS_OUTPUT will be set to 1. It will then retain this value for subsequent records and hence the code will not be processed again. An alternative would be to use the FIRST_RECORD or LAST_RECORD functions. However, these will not work if the first or last record has a terminated status.

You also need to declare NEW_BUS_OUTPUT as an integer at the start of the code.

If you now run the program and look at the output file U_IB2_.PRO you will see that it contains the 2 new business model points that are required.

Note: An alternative to entering the required values for the variables into the code is to create a generic table containing one line for each model point and then using the READ_GENERIC_TABLE function to read the values at run time.

LESSON 5

Multiple record formats

In this lesson you will create a new program that will read an input file that contains multiple record formats.

TOPICS COVERED IN THIS LESSON

Multiple record format input files

The policy extract file that you are converting may well contain several record formats rather than a single format as has been the case for the input file used in earlier lessons. In this lesson you will be creating a new program called TUTOR3 to convert such a file.

Second tutorial input file

A second tutorial input file has been provided which has a number of input record formats. It contains policy data for both a regular premium term assurance product and a regular premium unit linked investment plan. Each year the policyholders increase the premium that they pay under these plans. Details of each of these increments is held in a separate sub-record of the main record. This main record holds details which apply across all the increments such as the policy number and date of birth. The increment sub-records hold details of the date each increment took place and the increase in premium and sum assured.

In the case of the unit linked investment plan each increment can be invested in up to 10 unit funds. Details of the numbers of units currently held in each fund for each increment are held in sub-records of each increment sub-record. These fund sub-records hold details of the number of units held and the corresponding fund code.

Reading multiple record formats

If an input file only has one record format it is easy for DCS to read each record because the records just follow one after the other and each one has an identical format. If there is more than one record format then this is not the case. In particular DCS needs to know which record format follows the record that it has just read. There are two approaches that can be used in DCS to deal with this:

Read Condition
With this approach each record format needs to have a specific field which is in the same place in all the record formats and which specifies the type of that format. For example, there might be a field called "FormatType" which always starts at position 5 and is of length 2. Each record would then have a specific 2 character code in this position which identified the format of that record. This enables DCS to read these characters to determine the format of the record which follows the current record and then read the whole record accordingly.

Number of Sub-Records Specified in Parent Record
With this approach the current record contains a field which specifies the number of sub-records that follow it.

Input record formats

You should create a new program and enter a Program Description of "Tutorial Program 3". Select an Input File Type of FIXED ASCII, enter the file Name/Location as DATA\TUTOR3.TXT and leave the extract date unchanged. You should then create three record formats as follows:

Main Record

Name: Main
Type of Record: Main Record
Method of determining which records this record format applies to: Read Condition
This is the default Input Record Format: Not selected
Read Condition: Start = 1, Length = 4, Value = MAIN
Total Record Length: 38

Row	Field Name	Start	Length	Type	Date Format	Decimal
1	RECORD_TYPEM	1	4	Text		
2	POL_NUMBER	5	7	Integer		
3	PRODUCT	12	3	Text		
4	STATUS	15	1	Text		
5	SEX_CODE	16	1	Text		
6	SMOKER_CODE	17	1	Text		
7	ENTRY_DATE	18	8	Date	yyyymmdd	
8	BIRTH_DATE	26	8	Date	yyyymmdd	
9	POL_TERM_Y	34	2	Integer		
10	PREM_FREQ	36	2	Integer		

This record type is identified by the existence of the characters MAIN in positions 1 to 4 of the record.

Increment Record

Name: Increment
Type of Record: Sub-Record
Parent Record Format: Main
Max number of sub-records for a single parent record: 20
Method of determining which records this record format applies to: Read Condition
This is the default Input Record Format: Not selected
Read Condition: Start = 1, Length = 4, Value = INCR
Total Record Length: 30

Row	Field Name	Start	Length	Type	Date Format	Decimal
1	RECORD_TYPEI	1	4	Text		
2	INCRMT_DATE	5	8	Date	yyyymmdd	
3	ANNUAL_PREM	14	6	Number		2
4	SUM_ASSURED	23	7	Number		2

This record is a sub-record of the Main record and is identified by the existence of the characters INCR in positions 1 to 4 of the record.

Fund Record

Name: Fund
Type of Record: Sub-Record
Parent Record Format: Increment
Max number of sub-records for a single parent record: 10
Method of determining which records this record format applies to: Read Condition
This is the default Input Record Format: Not selected
Read Condition: Start = 1, Length = 4, Value = FUND
Total Record Length: 17

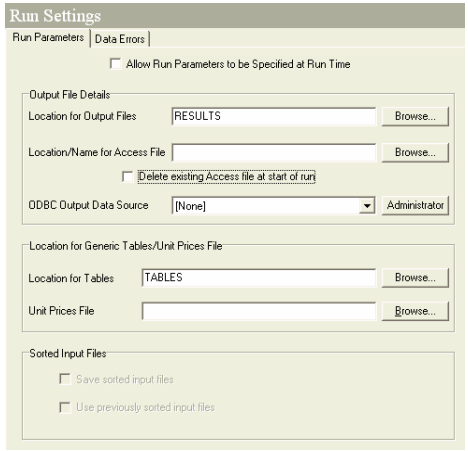
Row	Field Name	Start	Length	Type	Date Format	Decimal
1	RECORD_TYPEF	1	4	Text		
2	FUND_CODE	8	2	Integer		
3	NO_FUND_U	12	6	Number		2

This record is a sub-record of the Increment record and is identified by the existence of the characters FUND in positions 1 to 4 of the record.

Click **OK** to return to the main screen.

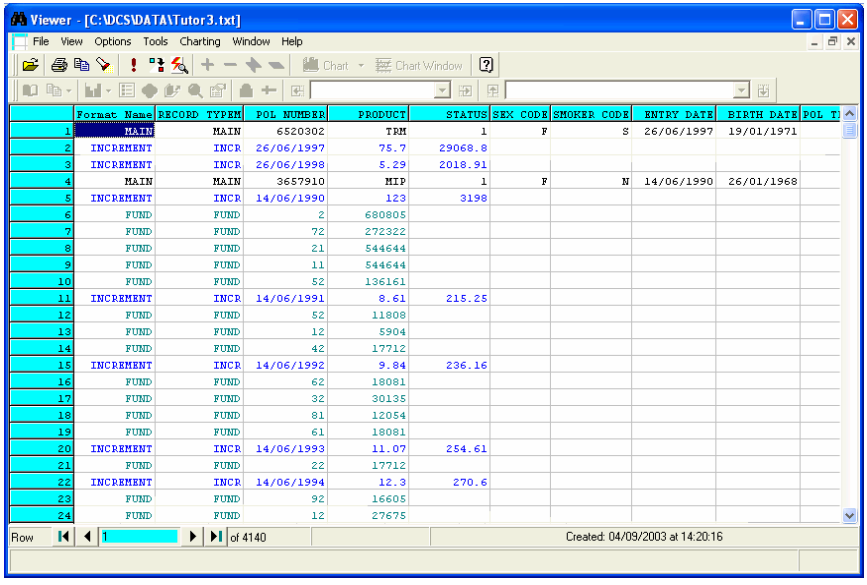
Setup Run

Enter the following information in the **Run Settings** view:



Viewing the input file

You should now view the input file. Since this is a new program you will be prompted to save it first. You should specify the name as TUTOR3. The following screen will then be displayed:



Note that as you cursor down the rows the column headers change to reflect the appropriate record format for the current row.

Close Viewer and return to DCS.

Entering the code

When DCS reads a record from an input file which contains multiple record formats it reads a main record along with all of its sub-records at the same time.

Therefore the variables in the Increment record format will be created as vectors, that is as 1 dimensional array variables. The length of the vector is equal to the value entered into the "Max No." field in the record format. Since you entered a value of 20 in the increment record format each variable will be a vector length of 20. For example, you need to refer to PREMIUM[2] or SUM_ASSURED[i].

Similarly the variables in the Fund record format will be created as arrays. For example, FUND_CODE will be created as an array of size 20 by 10 since you entered a value of 10 for the maximum number of entries for the fund record format. You need to specify the particular entries you want to refer to, for example:

```
FUND_CODE[3,4]
FUND_CODE[i,j]
```

The first argument specifies the number of the INCR record and the second specifies the number of the FUND record.

If you want to use any other vector or array variables in your code you have to explicitly declare them yourself because DCS cannot declare them automatically.

Note: You should normally declare the vector and array variables to have the same size as the maximum number of records for the relevant sub-record format. If a particular main record has more sub-records of a particular type than the maximum then DCS will produce an error message. However, it cannot identify situations where you have not declared vectors and arrays to be sufficiently large.

NO_OF_RECORDS function

For the Fund sub-records you need to also specify the particular Increment sub-record for which you want the number of records. For example, NO_OF_RECORDS("Fund", 3) returns the number of Fund sub-records for the 3rd Increment sub-record of the current main record.

Code

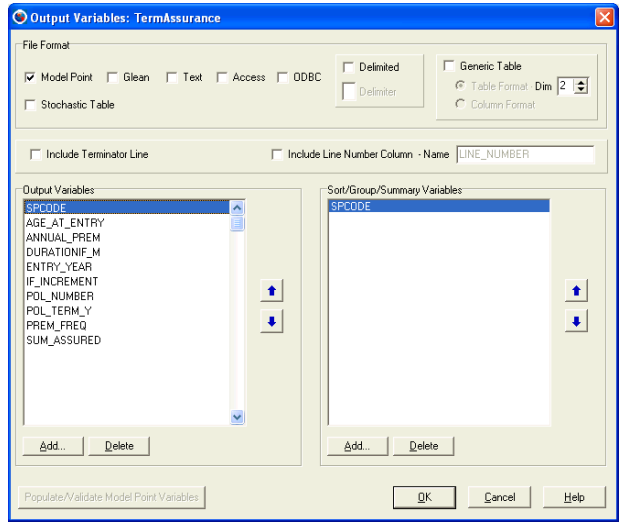
Enter the following code into the program:

```
; Declare variables
  VALN_YEAR DATE
  AGE_AT_ENTRY[20] INTEGER
  DURATIONIF_M[20] INTEGER
  IF_INCREMENT[20] INTEGER
  TERM_Y[20] INTEGER
  ENTRY_YEAR[20] INTEGER

; Set valuation year
  VALN_YEAR = EXTRACT_DATE

; Calculate values for each increment
  FOR i FROM 1 TO NO_OF_RECORDS("Increment")
    ENTRY_YEAR[i] = YEAR(INCRMT_DATE[i])
    AGE_AT_ENTRY[i] = COMPLETE_YEARS(INCRMT_DATE[i],
                                     BIRTH_DATE) + 1
    DURATIONIF_M[i] = COMPLETE_MONTHS(VALN_YEAR,
                                      INCRMT_DATE[i])
    TERM_Y[i] = POL_TERM_Y - (ENTRY_YEAR[i] -
                ENTRY_YEAR[1])

; Set variable if record is increment
  IF i = 1 THEN
      IF_INCREMENT[i] = 0
    ELSE
      IF_INCREMENT[i] = 1
  ENDIF
  ENDFOR

; Set sub-product code
  SPCODE = 1

; Create output
  IF PRODUCT = "TRM" THEN
    OUTPUT_FORMAT = "TermAssurance"
    FILENAME = "C_TERM"
    HEADING = "Term Assurance data as at 31.12.2000"
  ELSE
    OUTPUT_FORMAT = "UnitLinked"
    FILENAME = "U_MIP_"
    HEADING = "Maximum Investment Plan data as at
               31.12.2000"
  ENDIF
  OUTPUT(OUTPUT_FORMAT, FILENAME, HEADING)
```

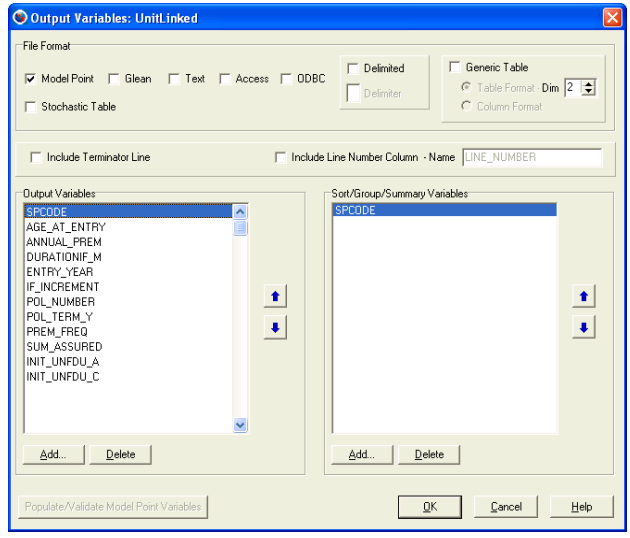
Output formats

Set the **Text Qualifier** to “None” then create the TermAssurance and UnitLinked output formats as shown in the following screens:

TermAssurance output format

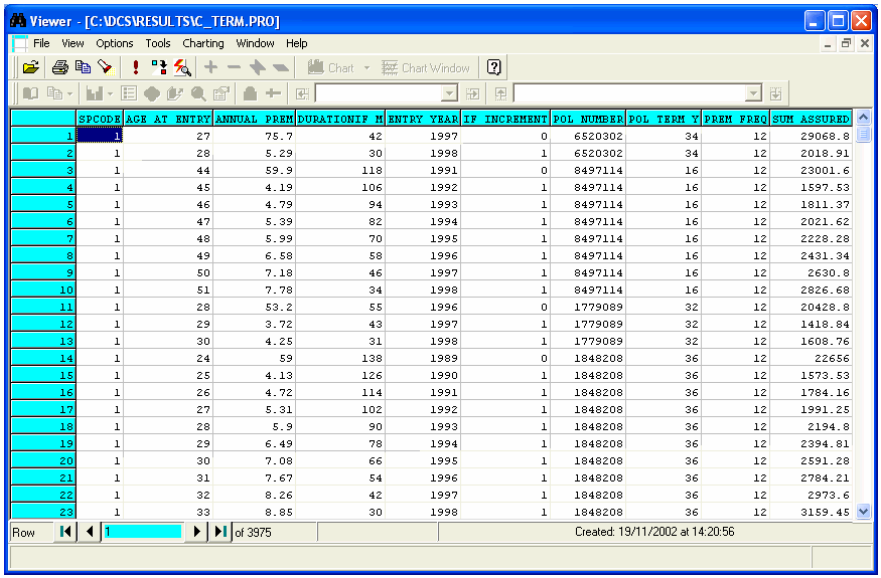


UnitLinked output format



Running the program

You should now run the program. If you then look at the C_TERM.PRO results file in Viewer the screen will be as follows:



	SPCODE	AGE AT ENTRY	ANNUAL PREM	DURATIONIF M	ENTRY YEAR	IF INCREMENT	POL NUMBER	POL TERM Y	PREM FREQ	SUM ASSURED
1	1	27	75.7	42	1997	0	6520302	34	12	29068.8
2	1	28	5.29	30	1998	1	6520302	34	12	2018.91
3	1	44	59.9	118	1991	0	8497114	16	12	23001.6
4	1	45	4.19	106	1992	1	8497114	16	12	1597.53
5	1	46	4.79	94	1993	1	8497114	16	12	1811.37
6	1	47	5.39	82	1994	1	8497114	16	12	2021.62
7	1	48	5.99	70	1995	1	8497114	16	12	2228.28
8	1	49	6.58	58	1996	1	8497114	16	12	2431.34
9	1	50	7.18	46	1997	1	8497114	16	12	2630.8
10	1	51	7.78	34	1998	1	8497114	16	12	2826.68
11	1	28	53.2	55	1996	0	1779089	32	12	20428.8
12	1	29	3.72	43	1997	1	1779089	32	12	1418.84
13	1	30	4.25	31	1998	1	1779089	32	12	1608.76
14	1	24	59	138	1989	0	1848208	36	12	22656
15	1	25	4.13	126	1990	1	1848208	36	12	1573.53
16	1	26	4.72	114	1991	1	1848208	36	12	1784.16
17	1	27	5.31	102	1992	1	1848208	36	12	1991.25
18	1	28	5.9	90	1993	1	1848208	36	12	2194.8
19	1	29	6.49	78	1994	1	1848208	36	12	2394.81
20	1	30	7.08	66	1995	1	1848208	36	12	2591.28
21	1	31	7.67	54	1996	1	1848208	36	12	2784.21
22	1	32	8.26	42	1997	1	1848208	36	12	2973.6
23	1	33	8.85	30	1998	1	1848208	36	12	3159.45

Row 1 of 3975 Created: 19/11/2002 at 14:20:56

You can see from this that there is a line for each increment. This has occurred because there were variables included in the TermAssurance output format from the Increment record format (or which were calculated at the increment level such as AGE_AT_ENTRY). DCS has recognised that this is the case and hence has included one for each such record. If the output format had just included variables from the Main record format (or which were calculated at the main record level) then there would have only been one line for each main record.

LESSON 6

Unit numbers and values

In this lesson you will enhance the program you created in the previous lesson so that it calculates the unit values for the unit linked business by multiplying the numbers of units as held in the input file by the corresponding unit prices held in a unit prices file. This facility is very similar to the unit numbers functionality which exists within Prophet.

TOPICS COVERED IN THIS LESSON

Input file details

To calculate unit values the input file must contain the unit numbers information as one or more pairs of variables:

- The number of units of a particular code
- The price code for that number of units

If it was possible for a policy to be linked to up to three different funds then there would need to be six fields in the input file as follows (unless the unit information was held in sub-records):

Number of units 1
Price code 1
Number of units 2
Price code 2
Number of units 3
Price code 3

If the unit information were held in sub-records of the main record then there would normally need to be three fields in the record format for those sub-records:

Record type
Number of units
Price code

The record type field is required so that DCS can recognise which are main records and which are the unit information records when it reads the input file (this aspect was covered in Lesson 5).

Note: The unit numbers calculation facility in DCS is only able to deal with the situation where policies are linked to both capital and accumulation units if different price codes are used to identify the two different types of units. If the same price code is used then this can be dealt with by adopting an alternative approach which involves the use of a generic table to hold the prices. This is dealt with later in this lesson.

You must use the following naming convention for input fields that hold unit number and price code information:

Fund code	FUND*CODE
Unit numbers	NO_FUND*U

The asterisk represents between 0 and 4 characters. Each pair of codes and numbers must have the same characters, for example FUND_1_CODE and NO_FUND_1_U. Therefore if it was possible for a particular policy to be linked to up to 5 funds then you could specify the following five sets:

FUND_1_CODE	NO_FUND_1_U
FUND_2_CODE	NO_FUND_2_U
FUND_3_CODE	NO_FUND_3_U
FUND_4_CODE	NO_FUND_4_U
FUND_5_CODE	NO_FUND_5_U

If the fund information is held in sub-records then you should use the names FUND_CODE and NO_FUND_U. This was the approach that was adopted in Lesson 5 when you specified the Fund format. You will be using the same program for this lesson.

The fund code variables can be specified as being text, number or integer. The fund codes can be up to 6 characters in length.

CALC_UNIT_VALUES function

To include the unit values functionality in your program you need to use the CALC_UNIT_VALUES function in your code. If you don't do this the unit number and fund code variables in the input file will just be treated in the same manner as any other variables.

You should include the following in the code for the TUTOR3 program that you created in Lesson 5. It should be inserted after the line where VALN_YEAR is calculated:

```
; Calculate unit values
  CALC_UNIT_VALUES("AC", "INIT_UNFDU_")
```

The first argument "AC" specifies that the input file contains unit funds of type A and C, that is A for accumulation units and C for capital units. The type for each fund code is held in the prices file. If there are any unit funds in the input file which have a type other than A or C then an error message will be produced during the run.

The second argument "INIT_UNFDU_" specifies the first part of the names of the variables that will return the unit values. DCS adds the type to this to give the complete name. Therefore in this case two variables will be created called INIT_UNFDU_A and INIT_UNFDU_C which return the values for type A and type C respectively.

You can use the INIT_UNFDU_ variables in the same way as any other variables. In particular you can include them in the output files so that they can be used by your unit linked Prophet products.

SUMMARISE_UNITS function

The SUMMARISE_UNITS function enables you to summarise the unit fund information by fund code.

Enter the following into your code below the CALC_UNIT_VALUES function:

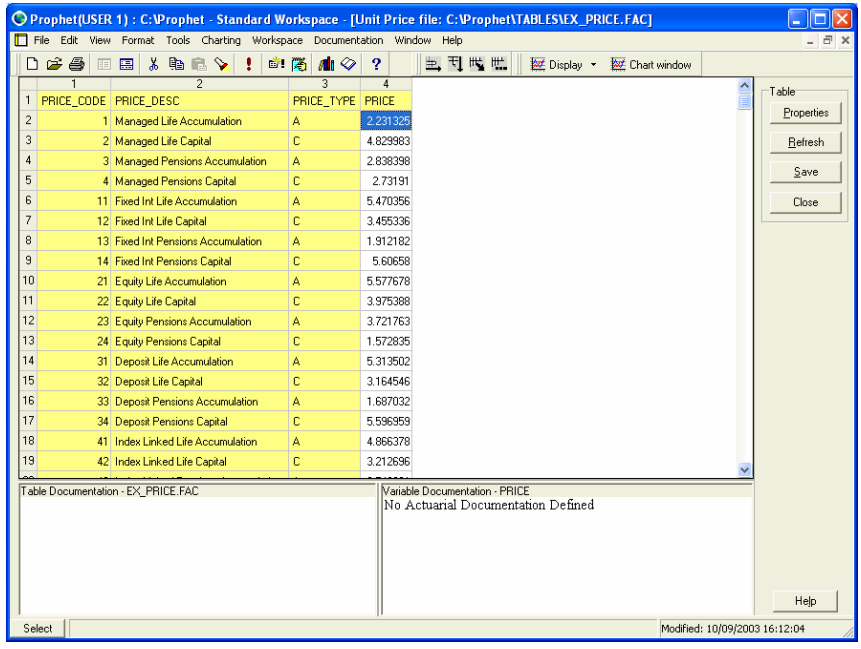
```
; Summarise unit values
  NO_OF_LINKS = 1
  SUMMARISE_UNITS("SumUnits", "SUMUNITS",
                  "Summary of units")
```

The NO_OF_LINKS variable is included so that the summary can include the number of policies linked to each fund type.

The arguments for the SUMMARISE_UNITS function are the same as for the OUTPUT, GROUP and SUMMARISE functions.

Unit prices file

The unit prices files used by DCS have exactly the same format as those used by Prophet. To create them you should use the Table Editor in Prophet. A unit prices file called EX_PRICE.FAC is supplied with DCS in the TABLES sub-folder. If you copy the file to a Table Location for a Prophet workspace and load it into the Table Editor it looks as follows:

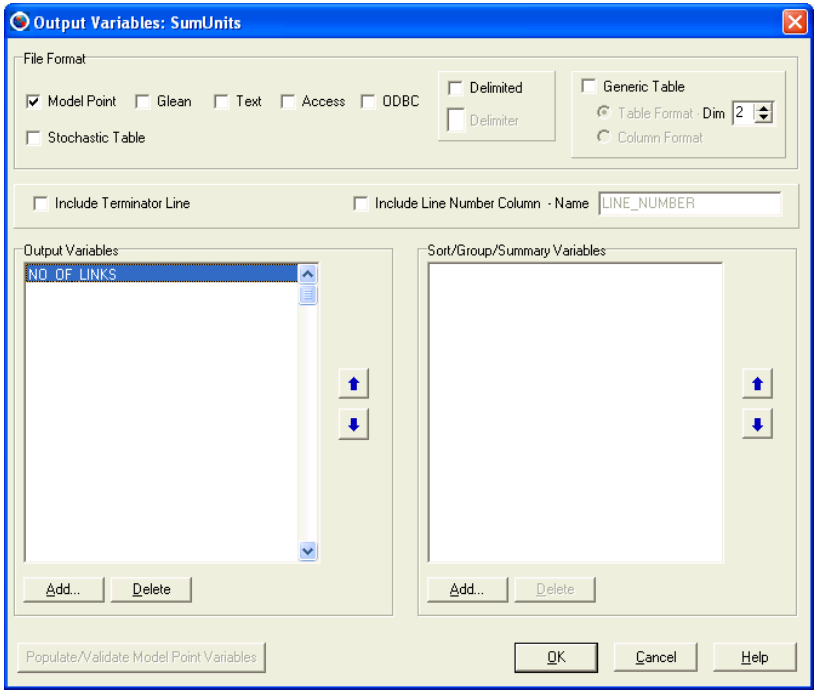


	1	2	3	4
1	PRICE_CODE	PRICE_DESC	PRICE_TYPE	PRICE
2	1	Managed Life Accumulation	A	2.231325
3	2	Managed Life Capital	C	4.829983
4	3	Managed Pensions Accumulation	A	2.838398
5	4	Managed Pensions Capital	C	2.73191
6	11	Fixed Int Life Accumulation	A	5.470356
7	12	Fixed Int Life Capital	C	3.455336
8	13	Fixed Int Pensions Accumulation	A	1.912182
9	14	Fixed Int Pensions Capital	C	5.60658
10	21	Equity Life Accumulation	A	5.577678
11	22	Equity Life Capital	C	3.975388
12	23	Equity Pensions Accumulation	A	3.721763
13	24	Equity Pensions Capital	C	1.572835
14	31	Deposit Life Accumulation	A	5.313502
15	32	Deposit Life Capital	C	3.164546
16	33	Deposit Pensions Accumulation	A	1.687032
17	34	Deposit Pensions Capital	C	5.596959
18	41	Index Linked Life Accumulation	A	4.866378
19	42	Index Linked Life Capital	C	3.212696

You should note from this that the price types allowed for in the unit prices file are A and C, which is consistent with the way you have specified the CALC_UNIT_VALUES function call in your program.

Output formats

You need to set up an output format called SumUnits to specify the output variables and summarising variables for the SUMMARISE_UNITS function. You should set it up as follows:



You should also add the variables INIT_UNFDU_A and INIT_UNFDU_C as output variables in the UnitLinked output format.

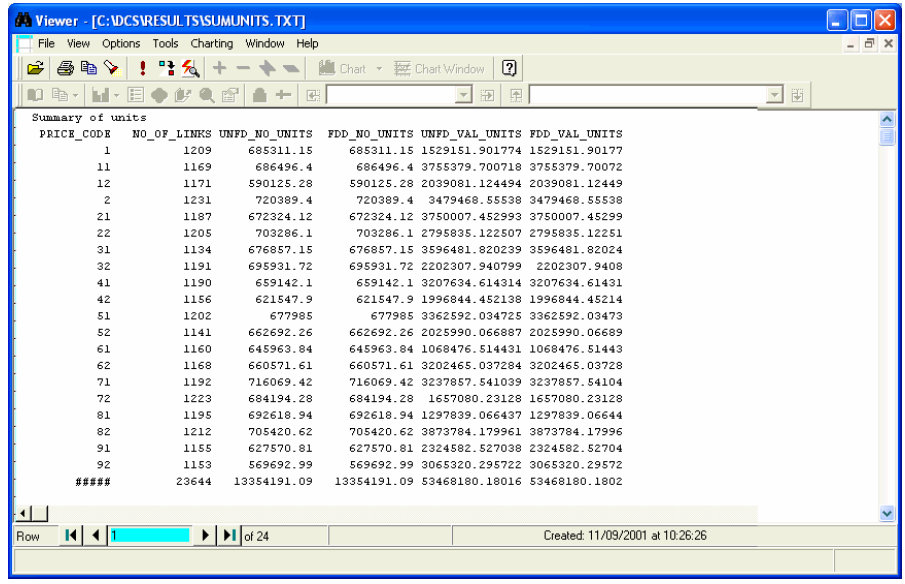
Setting up and running the program

You need to change the Run Settings view so that the Prices File is specified as TABLES\EX_PRICE.FAC.

Run

You should now run the program and view the U_MIP_.PRO output file. You will be able to see that the unit values are included for the variables INIT_UNFDU_A and INIT_UNFDU_C.

You should also view the SUMUNITS.TXT file. It will appear as follows:



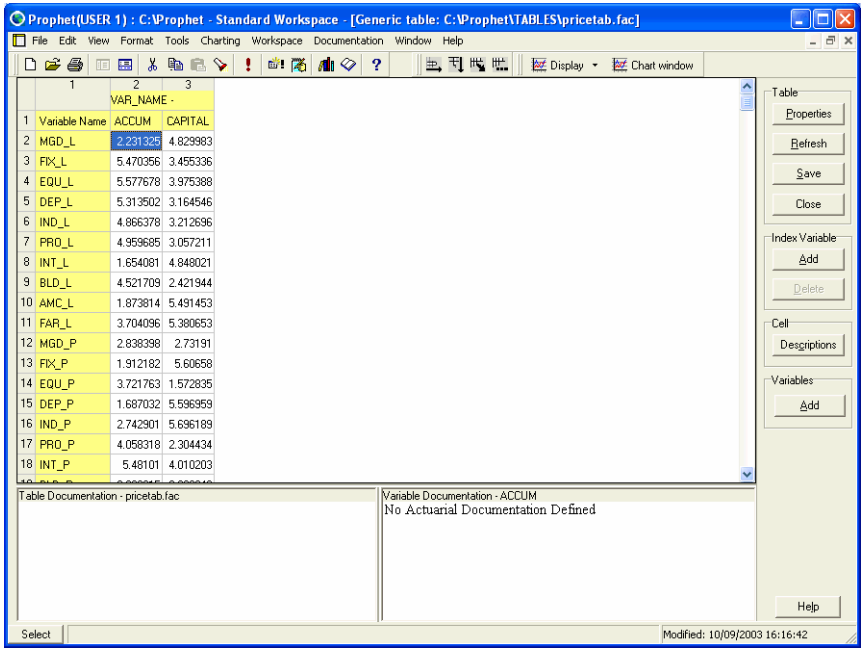
You can see from this how the results are summarised by price code.

Alternative approach using a generic table

As was mentioned earlier in this lesson, the CALC_UNIT_VALUES function cannot be used in situations where both capital and accumulation units have been given the same fund code. This section explains how you can cater for this situation by using a generic table to hold the unit prices. However, there is no program for you to set up and run.

Generic table

You would need to create a generic table which has FUND_CODE and FUND_TYPE as the two index variables. The screen below shows the same prices information as the unit prices file used earlier in this lesson but in the form of a generic table:



Note how the different values for FUND_TYPE of ACCUM and CAPITAL appear as column headers.

Input file

If each policy can be linked to up to 3 funds and the fund information is held in the main record then you could set up the input format with the following variables:

Fund Codes	FUND_CODE_1	FUND_CODE_2	FUND_CODE_3
Number of Accumulation Units	NO_ACC_U_1	NO_ACC_U_2	NO_ACC_U_3
Number of Capital Units	NO_CAP_U_1	NO_CAP_U_2	NO_CAP_U_3

Code

You would enter the following into the code to calculate the values of the units in the variables INIT_UNFDU_A and INIT_UNFDU_C:

```
; Calculate unit values
  INIT_UNFDU_A = 0.0
  INIT_UNFDU_C = 0.0
  FOR FUND FROM 1 TO 3
  SWITCH(FUND)
    CASE 1: FUND_CODE = FUND_CODE_1
            NO_ACC_UNITS = NO_ACC_U_1
            NO_CAP_UNITS = NO_CAP_U_1

    CASE 2: FUND_CODE = FUND_CODE_2
            NO_ACC_UNITS = NO_ACC_U_2
            NO_CAP_UNITS = NO_CAP_U_2

    CASE 3: FUND_CODE = FUND_CODE_3
            NO_ACC_UNITS = NO_ACC_U_3
            NO_CAP_UNITS = NO_CAP_U_3
    DEFAULT:
  ENDSWITCH
  PRICE_A = READ_GENERIC_TABLE("PRICETAB", "Y",
            FUND_CODE, "ACCUM")
  PRICE_C = READ_GENERIC_TABLE("PRICETAB", "Y",
            FUND_CODE, "CAPITAL")
  INIT_UNFDU_A = INIT_UNFDU_A + NO_ACC_UNITS * PRICE_A
  INIT_UNFDU_C = INIT_UNFDU_C + NO_CAP_UNITS * PRICE_C

  ENDFOR
```

Note: The example above assumes that the life funds have different fund codes from the pension funds. However, it is easy to extend this method to cover the situation where the same codes are used for both by having separate columns in the generic table. Within the code you would then need to determine whether a particular record represented a life product or a pensions product, perhaps based on the product code.

LESSON 7

Model point files for Analysis of Movements

In this lesson you will see how to use a simple approach to produce model point files for Analysis of Movements calculations. The approach illustrated uses extract files at two different dates and, using broad assumptions, constructs movement records for the period between those two dates.

TOPICS COVERED IN THIS LESSON

Introduction

For Analysis of Movements calculations your model point files need to contain model point lines for every policy that has been in force at any time during the Inter-Valuation Period (IVP) which is being investigated. For example, if the IVP is from 31 December 1999 to 31 December 2000 the model point file needs to include details of policies which were in force on 31 December 1999 but which were surrendered or became death claims at some point during 2000. If you were just carrying out normal projections your model points files would normally just contain details of the policies that were in force or paid up at 31 December 2000.

Different approaches are required depending on how the movement information is available in your extract files.

Single input file

If your year end policy extract file contains:

- Details of all the policies that were in force at any time during the IVP, and
- Sufficient information to enable the status of each policy at the start and end of the IVP to be determined, and also the month(s) in which any status changes occurred

then there are no special issues involved in creating the required model point files using DCS. A possible exception to this is where a policy has been altered during the year. This information may not be held in the file in which case you will probably have to assume that no alterations were made to any policies.

Note: The EXAMPAOM.DCS program which is one of the example programs supplied with DCS illustrates how model point files can be created when there is a single input file.

Two input files

If your extract files only contain details of the policies that are in force at the date of the extract then you can create the required model point files by comparing the data from two extract files. One of these needs to contain the policy data at the start of the IVP and the other needs to contain the policy data at the end of the IVP.

The main difficulties with this approach are as follows:

- For policies that terminate during the year you won't know the reason why. You can probably make the assumption that any policy that terminated during the year in which the policy was due to mature was a maturity. However, for other terminations you will not be able to distinguish between deaths and surrenders. From other data you may be able to make an assumption based on the relative numbers of deaths and surrenders and allow for that in the code. For example, you might assume that 1 in every 100 terminations was a death, with the other 99 being surrenders. Another possibility would be to create a generic table containing the policy numbers of all the death claims. The DCS program could then read that table to determine whether a termination was a death or not.
- You will not know in which month during the year a change of status occurred. You will therefore normally need to assume the middle of the year, except for maturities where you will be able to assume that the maturity month is the same as the entry month for policies with integral policy terms. (If you had separate input files for each month during the IVP rather than just at the start and end of the IVP it would be possible to determine the month of movement accurately by in effect comparing all 13 input files.)
- New policies sold during the year which terminated before the end of the year will not be included in the analysis.

These difficulties mean that it is better to use a single input file if this is possible. If you are using two input files you can use the merge facility to combine the records relating to the same policy from the two files.

Separate movements file

It is possible that details of the movements are kept in a separate file from the one holding the in force policy data. Depending on the structure of the data in the two files it should be possible to combine the data using the merge facility.

The remainder of this lesson explains how to set up a DCS program which creates a model point file by comparing two input files.

Analysis of Movements manuals

For additional information on Analysis of Movements you should refer to the Analysis of Movements manuals.

Model Point File requirements for Analysis of Movements

As was mentioned earlier, the model point files used in an Analysis of Movements run need to contain data for all the policies that were in force at any time during the IVP. In addition to the normal model point variables, there are four which need to be included to specify the details of the movements.

Additional model point variables

The four additional model point variables required to specify the details of the movements are as follows:

Variable	Specifies
STATUS_1	Status of the policy at the start of the IVP
STATUS_2	Status of the policy at the end of the IVP
MVT_MONTH_1	Month in which the policy achieved STATUS_1 (0 for the start of the IVP and month of issue for new business)
MVT_MONTH_2	Month in which the policy achieved STATUS_2 (IVP in months + 1 for policies which are in force or paid up at the end of the IVP, for example 13 for a full year's analysis)

Status field values

The following values are used for the two status fields:

Status Value	Status
0	Not yet In Force (that is, new business)
1	In Force
2	Surrender
3	Death
4	Maturity
5	Other off
6	Part Surrender
7	Alteration
8	Transition from In Force to Paid Up

Status Value	Status
9	Transition from Paid Up to In Force
10	Expected New Business
20	Paid Up

DURATIONIF_M

The variable DURATIONIF_M needs to hold the duration in force in months at the end of the IVP.

Model Point Lines

There should normally be only one model point line in the model point files for each policy. The exceptions to this are policies that are altered, made paid up or are partially surrendered during the IVP. For these policies there should be two model point lines, one to represent the position before the movement and the second to represent the position after the movement.

The DCS code needs to allow for the following possibilities:

1. The policy is in the start of year file but not the end of year file.
2. The policy is in the end of year file but not the start of year file.
3. The policy is in both the start of year file and the end of year file.

In the last case the code needs to allow for both altered and non altered policies.

For the first case the policy details to be output are those in the start of year input file rather than the end of year input file. However, the program cannot know whether to output a line until it has read the next record to see if it relates to the same policy or not (the next record could be either in the same input file or the other input file as compared to the current record). It is therefore necessary to output the model point line for this case after the next record has been read. The program therefore needs to retain the values to be output from one record to the next. For this reason all the variable names in the input record format must be different from those names required in the output model point files.

DCS features used

To create Analysis of Movements model point files by comparing two extract files you use the following features:

Feature	Purpose
Multiple input files	You are able to specify multiple input files by entering a name that uses a wildcard, for example DATA\VAL*.TXT, or by entering a number of names separated by commas, for example DATA\VAL97.TXT, DATA\VAL98.TXT. For Analysis of Movements you should use the second approach because you always want to be sure which file is treated as the first file and which as the second.
FILE_NUMBER function	This function returns the order number for the input file from which the current record was read. Using the previous example, the function would return a value of 1 for records read from DATA\VAL97.TXT and 2 for records read from DATA\VAL98.TXT.
SORT_INPUT function	This function enables the records in the input file(s) to be sorted before the calculations specified in the code are carried out. For Analysis of Movements you should sort on the policy number and FILE_NUMBER so that when the code is processed the records from the two files related to the same policy number are next to each other, with the record relating to the start of the IVP always being before the record relating to the end of the IVP.
PREVIOUS function	This function allows you to read the value for a variable from an earlier record (normally the one immediately before the current record). For example, PREVIOUS("POL_NUMBER") will return the policy number from the previous record.

Notes:

1. If multiple input files are used they must all be of the same type and contain the same fields. The only exception to this is if you can use the Read Condition facility to identify that different record formats are to be read from different files.
2. The SORT_INPUT function can only be used with input files that contain a single input record format.
3. The PREVIOUS function can only be used for variables that are in an input record format, or which have been declared in the code or which have been used in the code prior to point where the PREVIOUS function is used.

Input record format

In this lesson you will be creating a program to read the policy details from two input files called VAL97.TXT and VAL98.TXT which contain data as at 31 December 1997 and 31 December 1998 respectively. The input files only contain data for a with profit endowment product with a name of C_END_.

You should do the following:

- Create a new program by selecting **New** in the **File** menu and choose **DCS Program File**.
- In the Input Details view enter the following as the input file Name/Location:
  C:\DCS\DATA\VAL97.TXT, C:\DCS\DATA\VAL98.TXT
- Select **DELIMITED** as the Type.
- Select Tab delimited since the input files that the program will be reading are Tab delimited.
- Change the Extract Date to 31 December 1998.
- Enter **0** for the Header and footer sizes.
- Enter the Program Description as **Tutorial Program 4**.
- Create a new input record format by clicking **New.** Enter the name of the new record format as **Input** and then enter the following field information in the Input Fields tab:

Row	Field Name	Type	Length	Decimal
1	POLICY_NUMBER	Integer		
2	PREMIUM	Number		File
3	SUM_ASSD	Number		File
4	DECLARED_BON	Number		File
5	FREQ	Integer		
6	AGE	Integer		
7	ISSUE_YEAR	Integer		
8	ISSUE_MONTH	Integer		
9	POLICY_TERM	Integer		

- Click **OK** to save the record format.
- Click **Save** to save the DCS program. Enter the name as **TUTOR4**.

Entering the code

Select the Code view to enter the code for the program:

Sorting the input files

You need to sort the two input files so that all the records are in policy number order, with the record from the second input file for a particular policy always following after the record for that policy from the first input file. This is very important because the processing carried out by the program is completely dependent upon this. Therefore enter the following into the code:

```
; Sort input files based on policy number and file number
  SORT_INPUT("POLICY_NUMBER", "FILE_NUMBER")
```

Creating variable for FILE_NUMBER

There is a need to be able to refer to the value of FILE_NUMBER from the previous record. However, you cannot use FILE_NUMBER as an argument in a PREVIOUS function call since it is a function and not a variable. You therefore need to create a variable which is set equal to FILE_NUMBER. Hence enter the following into the code:

```
; Create a variable set equal to FILE_NUMBER because
; FILE_NUMBER cannot be used as an argument to the
; PREVIOUS function because it is a function, not a variable
  FILE_NO = FILE_NUMBER
```

First OUTPUT call

A model point line needs to be written out for those policies that are only in the start of year file and not the end of year file. These are policies that have terminated during the year.

If the maturity year is 1998 then we will assume that the policy terminated because it matured. In other cases we will assume that it was surrendered. For maturities we will assume that the movement month was the same as the issue month. For surrenders we will assume that the movement month was 6 (that is half way through the year).

Enter the following into the code:

```
; Model point output for policies that terminated
; during the year
  IF POLICY_NUMBER <> PREVIOUS("POLICY_NUMBER")
  AND PREVIOUS("FILE_NO") = 1 THEN
    STATUS_1 = 1
    MVT_MONTH_1 = 0
    IF PREVIOUS("ISSUE_YEAR") + PREVIOUS("POLICY_TERM")
    = 1998 THEN
      STATUS_2 = 4
      MVT_MONTH_2 = PREVIOUS("ISSUE_MONTH")
    ELSE
      STATUS_2 = 2
      MVT_MONTH_2 = 6
    ENDIF
    OUTPUT("Standard", "C_END_",
    "Model Point File for C_END_ at 31 Dec 1998")
  ENDIF
```

Note the use of the PREVIOUS function because we are outputting a model point line in respect of the previous record. For the first record, this function returns either 0 or blank depending on the type of the variable.

Second OUTPUT call

A model point line needs to be written out for those policies that are in both the start of year file and the end of year file and which were altered during the IVP. Two lines need to be output, one relating to the policy details at the start of the year and the other relating to the policy details at the end of the year. The second OUTPUT call deals with the start of year details. (The fourth OUTPUT call which is covered later deals with the end of year details.)

The details required for this call are therefore those at the start of the year, that is from the previous record. The alterations are assumed to occur in month 6.

Enter the following code:

```
; If policy in both start of year and end of year file
  IF POLICY_NUMBER = PREVIOUS("POLICY_NUMBER") AND
  FILE_NUMBER = 2 THEN

; Check if altered policy
    IF PREMIUM <> PREVIOUS("PREMIUM")
    OR SUM_ASSD <> PREVIOUS("SUM_ASSD")
    OR FREQ <> PREVIOUS("FREQ")
    OR AGE <> PREVIOUS("AGE")
    OR ISSUE_YEAR <> PREVIOUS("ISSUE_YEAR")
    OR ISSUE_MONTH <> PREVIOUS("ISSUE_MONTH")
    OR POLICY_TERM <> PREVIOUS("POLICY_TERM") THEN
      ALTERED_POLICY = 1
    ELSE
      ALTERED_POLICY = 0
    ENDIF
```

```
; If altered policy, output model point line for policy
; information at start of year
    IF ALTERED_POLICY = 1 THEN
      STATUS_1 = 1
      STATUS_2 = 7
      MVT_MONTH_1 = 0
      MVT_MONTH_2 = 6
      OUTPUT("Standard", "C_END_",
      "Model Point File for C_END_ at 31 Dec 1998")
    ENDIF
  ENDIF
```

Set output details

At this point in the code you need to set the output details. This means that for the OUTPUT calls from this point onwards in the code the values for the current record are output. For the earlier OUTPUT calls the values from the previous record are output because the values for these variables will have been retained as the program moved forward from the previous record to this one.

Enter the following code:

```
; Set output details so that details for the current
; policy are output by any future use of the OUTPUT
; function for this record
  SPCODE = 1
  POL_NUMBER = POLICY_NUMBER
  AGE_AT_ENTRY = AGE
  ANNUAL_PREM = PREMIUM
  SUM_ASSURED = SUM_ASSD
  INIT_DECB_IF = DECLARED_BON
  PREM_FREQ = FREQ
  POL_TERM_Y = POLICY_TERM
  ENTRY_YEAR = ISSUE_YEAR
  ENTRY_MONTH = ISSUE_MONTH
  DURATIONIF_M = (1998 - ENTRY_YEAR) * 12 + 13 - ENTRY_MONTH
```

Third OUTPUT call

A model point line needs to be written out for those policies that are in the end of year file but not in the start of year file. In most cases these will be new policies issued during the IVP in the month specified by the value of the variable ISSUE_MONTH. Any policies which were issued prior to the IVP are given a status of 2 at the start of the IVP (that is surrendered) and hence will be treated as reinstatements during the IVP.

Enter the following code:

```
; If current policy only in end of year file
  IF POLICY_NUMBER <> PREVIOUS("POLICY_NUMBER") AND
  FILE_NUMBER = 2 THEN
    STATUS_2 = 1
    MVT_MONTH_2 = 13
    IF ENTRY_YEAR = 1998 THEN
      STATUS_1 = 0
      MVT_MONTH_1 = ISSUE_MONTH
    ELSE
      STATUS_1 = 2
      MVT_MONTH_1 = 6
    ENDIF
    OUTPUT("Standard", "C_END_",
    "Model Point File for C_END_ at 31 Dec 1998")
  ENDIF
```

Fourth OUTPUT call

A model point line needs to be written out for those policies that are in both the start of year file and the end of year file. Different statuses and movement months are required depending on whether or not the policy has been altered during the IVP.

Enter the following code:

```
; If current policy in both start of year file and end
; of year file
  IF POLICY_NUMBER = PREVIOUS("POLICY_NUMBER") AND
  FILE_NUMBER = 2 THEN
; If not an altered policy
    IF ALTERED_POLICY = 0 THEN
      STATUS_1 = 1
      STATUS_2 = 1
      MVT_MONTH_1 = 0
      MVT_MONTH_2 = 13

; If an altered policy
    ELSE
      STATUS_1 = 7
      STATUS_2 = 1
      MVT_MONTH_1 = 6
      MVT_MONTH_2 = 13
    ENDIF
    OUTPUT("Standard", "C_END_",
    "Model Point File for C_END_ at 31 Dec 1998")
  ENDIF
```

Fifth OUTPUT call

A model point line needs to be written out if the last policy is only in the start of year file. Normally the model point line for this policy would have been written out in the first OUTPUT call when the next record was processed. However,

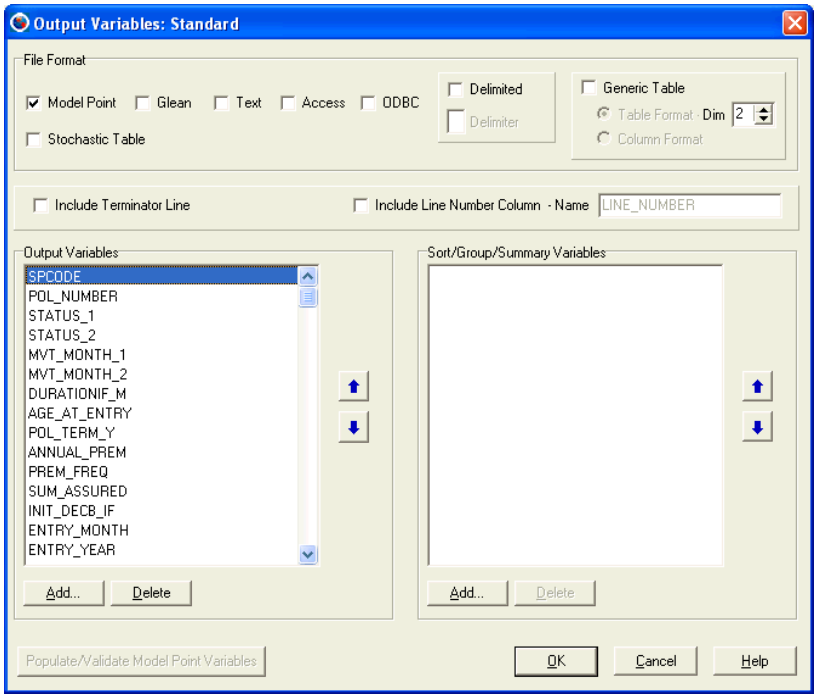
for the last record the program will terminate without this happening because there isn't another record to be processed.

Enter the following code:

```
; If last record and policy only in start of year
; file then output model point line
  IF LAST_RECORD AND FILE_NUMBER = 1 THEN
    STATUS_1 = 1
    MVT_MONTH_1 = 0
    IF ISSUE_YEAR + POLICY_TERM = 1998 THEN
      STATUS_2 = 4
      MVT_MONTH_2 = ISSUE_MONTH
    ELSE
      STATUS_2 = 2
      MVT_MONTH_2 = 6
    ENDIF
    OUTPUT("Standard", "C_END_",
    "Model Point File for C_END_ at 31 Dec 1998")
  ENDIF
```

Output format

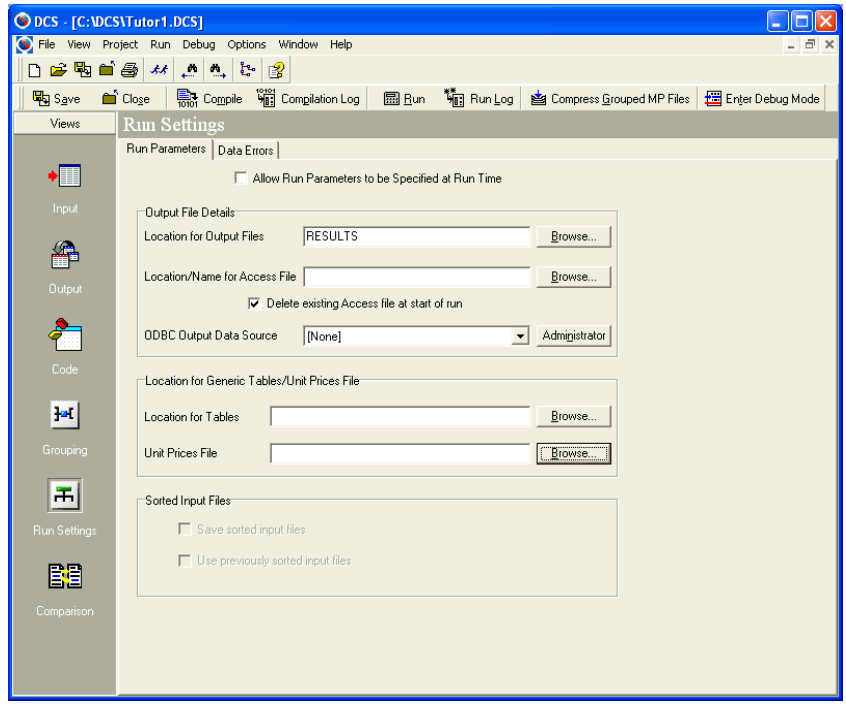
Create a new output file format called **Standard** and select variables for it as follows:



Then click **OK** to save it.

Running the program

In the **Run Settings** view enter the information shown below:



Run

Run the program and then look at the input files used and the output file created. Parts of these files are shown below.

Start of Year Input File (VAL97.TXT)

	POLICY NUMBER	PREMIUM	SUM ASSD	DECLARED BON	FREQ	AGE	ISSUE YEAR	ISSUE MONTH	POLICY TERM
1	1000005	484	10000	892	12	43	1996	7	20
2	1000003	410	6000	2320	1	39	1983	9	15
3	1000002	420	6000	523	12	39	1995	3	15
4	1000001	711	7100	1098	12	35	1992	1	10

End of Year Input File (VAL98.TXT)

	POLICY NUMBER	PREMIUM	SUM ASSD	DECLARED BON	FREQ	AGE	ISSUE YEAR	ISSUE MONTH	POLICY TERM
1	1000001	711	7100	1311	12	35	1992	1	10
2	1000004	850	11000	566	12	47	1996	12	13
3	1000005	669	10000	1222	12	43	1996	7	15
4	1000006	350	6000	0	12	38	1998	4	18

Note: If there is more than one input file for a DCS program Viewer lists them all so that you can select the required one. To switch to a different input file in Viewer select **Alternative Input File** from the **View** menu or in the right click menu.

Output Model Point File (C_END_.PRO)

	SPCODE	POL NUMBER	STATUS 1	STATUS 2	MVT MONTH 1	MVT MONTH 2	DURATIONIF M	AGE AT ENTRY	POL TERM Y	ANNUAL PREM
1	1	1000001	1	1	0	13	84	35	10	711
2	1	1000002	1	2	0	6	46	39	15	420
3	1	1000003	1	4	0	9	184	39	15	410
4	1	1000004	2	1	6	13	25	47	13	850
5	1	1000005	1	7	0	6	30	43	20	484
6	1	1000005	7	1	6	13	30	43	15	669
7	1	1000006	0	1	4	13	9	38	18	350

Each of the policies in the run is an example of the types of movement that can occur:

Policy Number	Example of
1000001	In force policy throughout
1000002	Surrender during the IVP
1000003	Maturity during the IVP
1000004	Reinstatement during the IVP
1000005	Alteration of policy term and premium during the IVP
1000006	New business during the IVP

DCS program listings

This section contains listings for the four DCS programs which you have set up as you went through the tutorial. These will help you to identify any mistakes that you made.

Note: Where the same program is used in more than one lesson the changes for the current lesson are enclosed in boxes.

Program listing for TUTOR1 in Lesson 1

DCS Program Specification SunGard 06/01/2006
Program Name: TUTOR1

Program Details

Volume: Directory: C:\DCS Created: 06/01/2006 at 12:10:24

Description: Tutorial Program 1

Run Settings

Run Parameters

Allow run parameters to be specified at runtime	No
Location for Output Files	RESULTS1
Location/Name for Access File	
Delete existing Access file at start of run	Yes
ODBC Data Source	
Location for Tables	TABLES
Unit Prices File	

Data Errors

Maximum Number of Runtime Errors Before Termination	0
Ignore Blank Lines (except in header and footer)	No
Treat Missing Trailing Fields as Zero or Blank	No
Ignore any Records not Matching any Read Condition	No
Read Integer Part of Number Values in Integer Fields	No
Treat Following Text Values in Numeric Fields as having a value of	No
Create " _missing_value" Variables for Every Input Variable	No
Write input lines that cause an Error to .ERR file	Yes
Check output values against variable details in associated Prophet workspace	No

Input File(s)

Input File Name/Location	DATA\TUTOR1.TXT
Extract Date	31 December 2000
Type	FIXED ASCII
Header Size	0 Lines
Footer Size	0 Lines

Input Record Format Name = Input, Record Length = 74

Type of Record: Main Record

Method of determining which records this record format applies to: Applies to all records

	Field Name	Start	Length	Type	Date Format	Decimal
1	POL_NUMBER	1	6	Text		
2	PRODUCT	7	3	Text		
3	STATUS	11	1	Text		
4	SEX_CODE	12	1	Text		
5	SMOKER_CODE	13	1	Text		
6	ENTRY_DATE	14	8	Date	ddmmyyyy	
7	BIRTH_DATE	22	8	Date	ddmmyyyy	
8	POL_TERM_Y	31	2	Integer		
9	PREM_FREQ	33	2	Integer		
10	PREMIUM	35	8	Number		File
11	SUM_ASSURED	43	10	Number		File
12	INIT_DECB_IF	53	10	Number		File
13	INIT_UNFDU_A	63	10	Number		File

Program listing for TUTOR1 in Lesson 1

DCS Program Specification SunGard 06/01/2006
Program Name: TUTOR1

Output Format Options

Text Qualifier for Model Point, Text and Delimited Output Files None
Use Windows Date Setting Yes

Output File Format Name = Conventional
Format = Model Point
Include Terminator Line: No
Include Line Number Column: No

Output Variables	Sort/Group/Summary Variables
SPCODE	SPCODE
AGE_AT_ENTRY	
ANNUAL_PREM	
DURATIONIF_M	
ENTRY_YEAR	
INIT_DECB_IF	
POL_NUMBER	
POL_TERM_Y	
PREM_FREQ	
SEX	
SMOKER_STAT	
SUM_ASSURED	
INIT_POLS_IF	

Output File Format Name = UnitLinked
Format = Model Point
Include Terminator Line: No
Include Line Number Column: No

Output Variables	Sort/Group/Summary Variables
SPCODE	SPCODE
AGE_AT_ENTRY	
DURATIONIF_M	
ENTRY_YEAR	
INIT_UNFDU_A	
POL_NUMBER	
SEX	
SINGLE_PREM	
SMOKER_STAT	
SUM_ASSURED	
INIT_POLS_IF	

Program Code

```
; Exclude records that are not in force
 IF STATUS <> "I" THEN
   NEXT_RECORD
 ENDIF

; Set valuation year
 VAL_YEAR = YEAR(EXTRACT_DATE)
 VAL_YEAR_T = STRVAL(VAL_YEAR)

; Calculate entry year
 ENTRY_YEAR = YEAR(ENTRY_DATE)

; Calculate age at entry
 AGE_AT_ENTRY = COMPLETE_YEARS(ENTRY_DATE, BIRTH_DATE-1) + 1
```

Prophet Data Conversion System 2 Release 7.3

Program listing for TUTOR1 in Lesson 1


```
; Calculate duration in force to higher months
 DURATIONIF_M = COMPLETE_MONTHS(EXTRACT_DATE, ENTRY_DATE) + 1

; Set premium variables
 IF PRODUCT = "WPE" THEN
   ANNUAL_PREM = PREMIUM
   SINGLE_PREM = 0
 ELSE
   ANNUAL_PREM = 0
   SINGLE_PREM = PREMIUM
 ENDIF

; Convert sex and smoker codes to numeric format
  IF SEX_CODE = "F" THEN
    SEX = 1
  ELSE
    SEX = 0
  ENDIF

  IF SMOKER_CODE = "S" THEN
    SMOKER_STAT = 1
  ELSE
    SMOKER_STAT = 0
  ENDIF

; Set output details (including defaults in case some
; policies are not given an appropriate filename)
  OUTPUT_FORMAT = "Conventional"
  FILENAME = "Error"
  HEADING = "Policies are not allocated a filename as at 31.12."
            + VAL_YEAR_T

  IF PRODUCT = "WPE" THEN
    OUTPUT_FORMAT = "Conventional"
    FILENAME = "C_WEND"
    HEADING = "With Profits Endowments data as at 31.12."
              + VAL_YEAR_T
  ENDIF

  IF PRODUCT = "IB" THEN
    OUTPUT_FORMAT = "UnitLinked"
    FILENAME = "U_IB__"
    HEADING = "Investment Bonds data as at 31.12."
              + VAL_YEAR_T
  ENDIF

; Set sub-product code to 1
  SPCODE = 1

; Set initial number of policies to 1
  INIT_POLS_IF = 1

; Output line for existing business
  OUTPUT(OUTPUT_FORMAT, FILENAME, HEADING)

; Output line for new business profile
  IF ENTRY_YEAR = VAL_YEAR THEN
    SPCODE = SPCODE + 50
    DURATIONIF_M = 0
    INIT_DECB_IF = 0
    INIT_UNFDU_A = 0
    OUTPUT(OUTPUT_FORMAT, FILENAME, HEADING)
  ENDIF
```

Program listing for TUTOR1 in Lesson 2

DCS Program Specification SunGard 06/01/2006
Program Name: TUTOR1

Program Details

Volume: Directory: C:\DCS Created: 06/01/2006 at 12:34:56

Description: Tutorial Program 1

Run Settings

Run Parameters

Allow run parameters to be specified at runtime	No
Location for Output Files	RESULTS1
Location/Name for Access File	
Delete existing Access file at start of run	Yes
ODBC Data Source	
Location for Tables	TABLES
Unit Prices File	

Data Errors

Maximum Number of Runtime Errors Before Termination	0
Ignore Blank Lines (except in header and footer)	No
Treat Missing Trailing Fields as Zero or Blank	No
Ignore any Records not Matching any Read Condition	No
Read Integer Part of Number Values in Integer Fields	No
Treat Following Text Values in Numeric Fields as having a value of	No
Create "_missing_value" Variables for Every Input Variable	No
Write input lines that cause an Error to .ERR file	Yes
Check output values against variable details in associated Prophet workspace	No

Input File(s)

Input File Name/Location	DATA\TUTOR1.TXT
Extract Date	31 December 2000
Type	FIXED ASCII
Header Size	0 Lines
Footer Size	0 Lines

Input Record Format Name = Input, Record Length = 74

Type of Record: Main Record

Method of determining which records this record format applies to: Applies to all records

	Field Name	Start	Length	Type	Date Format	Decimal
1	POL_NUMBER	1	6	Text		
2	PRODUCT	7	3	Text		
3	STATUS	11	1	Text		
4	SEX_CODE	12	1	Text		
5	SMOKER_CODE	13	1	Text		
6	ENTRY_DATE	14	8	Date	ddmmyyyy	
7	BIRTH_DATE	22	8	Date	ddmmyyyy	
8	POL_TERM_Y	31	2	Integer		
9	PREM_FREQ	33	2	Integer		
10	PREMIUM	35	8	Number		File
11	SUM_ASSURED	43	10	Number		File
12	INIT_DECB_IF	53	10	Number		File
13	INIT_UNFDU_A	63	10	Number		File

Prophet Data Conversion System 1 Release 7.3

Program listing for TUTOR1 in Lesson 2

DCS Program Specification SunGard 06/01/2006
Program Name: TUTOR1

Messages in Separate Files

Validation File Name INVALID
Format Name Messages
Heading Invalid Policy Message File

Correction File Name CORRECT
Format Name Messages
Heading Corrected Policy Message File

Output Format Options

Text Qualifier for Model Point, Text and Delimited Output Files None
Use Windows Date Setting Yes

Output File Format Name = Conventional
Format = Model Point
Include Terminator Line: No
Include Line Number Column: No

Output Variables Sort/Group/Summary Variables

SPCODE SPCODE
AGE_AT_ENTRY
ANNUAL_PREM
DURATIONIF_M
ENTRY_YEAR
INIT_DECB_IF
POL_NUMBER
POL_TERM_Y
PREM_FREQ
SEX
SMOKER_STAT
SUM_ASSURED
INIT_POLS_IF

Output File Format Name = UnitLinked
Format = Model Point
Include Terminator Line: No
Include Line Number Column: No

Output Variables Sort/Group/Summary Variables

SPCODE SPCODE
AGE_AT_ENTRY
DURATIONIF_M
ENTRY_YEAR
INIT_UNFDU_A
POL_NUMBER
SEX
SINGLE_PREM
SMOKER_STAT
SUM_ASSURED
INIT_POLS_IF

Prophet Data Conversion System 2 Release 7.3

Program listing for TUTOR1 in Lesson 2

DCS Program Specification SunGard 06/01/2006
Program Name: TUTOR1

Output File Format Name = Messages
Format = Model Point, Text
Include Terminator Line: No
Include Line Number Column: No

Output Variables Sort/Group/Summary Variables

POL_NUMBER
PRODUCT
STATUS

Program Code

```
; Exclude records that are not in force
 IF STATUS <> "I" THEN
   NEXT_RECORD
 ENDIF

; Set valuation year
 VAL_YEAR = YEAR(EXTRACT_DATE)
 VAL_YEAR_T = STRVAL(VAL_YEAR)

; Calculate entry year
 ENTRY_YEAR = YEAR(ENTRY_DATE)

; Calculate age at entry
 AGE_AT_ENTRY = COMPLETE_YEARS(ENTRY_DATE, BIRTH_DATE-1) + 1

; Calculate duration in force to higher months
 DURATIONIF_M = COMPLETE_MONTHS(EXTRACT_DATE, ENTRY_DATE) + 1

; Set premium variables
 IF PRODUCT = "WPE" THEN
   ANNUAL_PREM = PREMIUM
   SINGLE_PREM = 0
 ELSE
   ANNUAL_PREM = 0
   SINGLE_PREM = PREMIUM
 ENDIF

; Convert sex and smoker codes to numeric format
  IF SEX_CODE = "F" THEN
    SEX = 1
  ELSE
    SEX = 0
  ENDIF

  IF SMOKER_CODE = "S" THEN
    SMOKER_STAT = 1
  ELSE
    SMOKER_STAT = 0
  ENDIF

; Set output details (including defaults in case some
; policies are not given an appropriate filename)
  OUTPUT_FORMAT = "Conventional"
  FILENAME = "Error"
  HEADING = "Policies are not allocated a filename as at 31.12."
            + VAL_YEAR_T

  IF PRODUCT = "WPE" THEN
    OUTPUT_FORMAT = "Conventional"
    FILENAME = "C_WEND"
    HEADING = "With Profits Endowments data as at 31.12."
              + VAL_YEAR_T
  ENDIF
```

Prophet Data Conversion System 3 Release 7.3

Program listing for TUTOR1 in Lesson 2


```
  IF PRODUCT = "IB" THEN
    OUTPUT_FORMAT = "UnitLinked"
    FILENAME = "U_IB__"
    HEADING = "Investment Bonds data as at 31.12."
              + VAL_YEAR_T
  ENDIF

; Set sub-product code to 1
  SPCODE = 1

; Set initial number of policies to 1
  INIT_POLS_IF = 1

; Validation of policy term
  IF PRODUCT = "WPE" AND POL_TERM_Y < 10 THEN
    INVALID(POL_TERM_Y)
  ENDIF

; Correction of premium frequency
  IF PRODUCT = "WPE" THEN
    SWITCH(PREM_FREQ)
    CASE 1,2,4,12:
  ; For these values nothing is done
    DEFAULT: CORRECT(PREM_FREQ,12)
    ENDSWITCH
  ENDIF

; Table Correction of ages at entry
  PROD_NAME = FILENAME
  TABLE_CORRECT("CORRECT", PROD_NAME, AGE_AT_ENTRY)

; Variable to record failure of validation / correction
; tests
  IF FAILED_VALIDATION THEN
    IF_FAILED = 1
  ELSE
    IF_FAILED = 0
  ENDIF

; Output line for existing business
  OUTPUT(OUTPUT_FORMAT, FILENAME, HEADING)

; Output line for new business profile
  IF ENTRY_YEAR = VAL_YEAR THEN
    SPCODE = SPCODE + 50
    DURATIONIF_M = 0
    INIT_DECB_IF = 0
    INIT_UNFDU_A = 0
    OUTPUT(OUTPUT_FORMAT, FILENAME, HEADING)
  ENDIF
```

Program listing for TUTOR1 in Lesson 3

DCS Program Specification SunGard 06/01/2006
Program Name: TUTOR1

Program Details

Volume: Directory: C:\DCS Created: 06/01/2006 at 15:06:52

Description: Tutorial Program 1

Run Settings

Run Parameters

Allow run parameters to be specified at runtime	No
Location for Output Files	GROUPED
Location/Name for Access File	
Delete existing Access file at start of run	Yes
ODBC Data Source	
Location for Tables	TABLES
Unit Prices File	

Data Errors

Maximum Number of Runtime Errors Before Termination	0
Ignore Blank Lines (except in header and footer)	No
Treat Missing Trailing Fields as Zero or Blank	No
Ignore any Records not Matching any Read Condition	No
Read Integer Part of Number Values in Integer Fields	No
Treat Following Text Values in Numeric Fields as having a value of	No
Create "_missing_value" Variables for Every Input Variable	No
Write input lines that cause an Error to .ERR file	Yes
Check output values against variable details in associated Prophet workspace	No

Input File(s)

Input File Name/Location	DATA\TUTOR1.TXT
Extract Date	31 December 2000
Type	FIXED ASCII
Header Size	0 Lines
Footer Size	0 Lines

Input Record Format Name = Input, Record Length = 74

Type of Record: Main Record

Method of determining which records this record format applies to: Applies to all records

	Field Name	Start	Length	Type	Date Format	Decimal
1	POL_NUMBER	1	6	Text		
2	PRODUCT	7	3	Text		
3	STATUS	11	1	Text		
4	SEX_CODE	12	1	Text		
5	SMOKER_CODE	13	1	Text		
6	ENTRY_DATE	14	8	Date	ddmmyyyy	
7	BIRTH_DATE	22	8	Date	ddmmyyyy	
8	POL_TERM_Y	31	2	Integer		
9	PREM_FREQ	33	2	Integer		
10	PREMIUM	35	8	Number		File
11	SUM_ASSURED	43	10	Number		File
12	INIT_DECB_IF	53	10	Number		File
13	INIT_UNFDU_A	63	10	Number		File

Program listing for TUTOR1 in Lesson 3

DCS Program Specification SunGard 06/01/2006
Program Name: TUTOR1

Messages in Separate Files

Validation File Name INVALID
Format Name Messages
Heading Invalid Policy Message File

Correction File Name correct
Format Name Messages
Heading Corrected Policy Message File

Output Format Options

Text Qualifier for Model Point, Text and Delimited Output Files None
Use Windows Date Setting Yes

Output File Format Name = Conventional
Format = Model Point
Include Terminator Line: No
Include Line Number Column: No

Output Variables	Sort/Group/Summary Variables
SPCODE	SPCODE
AGE_AT_ENTRY	DUR_GROUP
ANNUAL_PREM	POL_GROUP
DURATIONIF_M	PREM_FREQ
ENTRY_YEAR	
INIT_DECB_IF	
POL_TERM_Y	
PREM_FREQ	
SEX	
SMOKER_STAT	
SUM_ASSURED	
INIT_POLS_IF	
DUR_GROUP	
POL_GROUP	

Output File Format Name = UnitLinked
Format = Model Point
Include Terminator Line: No
Include Line Number Column: No

Output Variables	Sort/Group/Summary Variables
SPCODE	SPCODE
AGE_AT_ENTRY	AGE_GROUP
DURATIONIF_M	
ENTRY_YEAR	
INIT_UNFDU_A	
SEX	
SINGLE_PREM	
SMOKER_STAT	
SUM_ASSURED	
INIT_POLS_IF	
AGE_GROUP	

Program listing for TUTOR1 in Lesson 3

DCS Program Specification SunGard 06/01/2006
Program Name: TUTOR1

Output File Format Name = Messages
Format = Model Point, Text
Include Terminator Line: No
Include Line Number Column: No

Output Variables Sort/Group/Summary Variables

POL_NUMBER
PRODUCT
STATUS

Grouping Calculations/Decimal Places

Decimal places not to be applied to normal output and summarising files

Output Variables	Calculation Type	Weighting Variable 1	Weighting Variable 2	Decimal Places
AGE_AT_ENTRY	Average	PREMIUM		0
AGE_GROUP	Average			0
ANNUAL_PREM	Average			2
DUR_GROUP	Average			0
DURATIONIF_M	Average	PREMIUM		0
ENTRY_YEAR	Average			0
INIT_DECB_IF	Average			2
INIT_POLS_IF	Sum			0
INIT_UNFDU_A	Average			2
POL_GROUP	Average			0
POL_TERM_Y	Average	PREMIUM		0
PREM_FREQ	Average			0
SEX	Average			0
SINGLE_PREM	Average			2
SMOKER_STAT	Average			0
SPCODE	Average			0
SUM_ASSURED	Average			2

Program Code

```
; Exclude records that are not in force
 IF STATUS <> "I" THEN
   NEXT_RECORD
 ENDIF

; Set valuation year
 VAL_YEAR = YEAR(EXTRACT_DATE)
 VAL_YEAR_T = STRVAL(VAL_YEAR)

; Calculate entry year
 ENTRY_YEAR = YEAR(ENTRY_DATE)

; Calculate age at entry
 AGE_AT_ENTRY = COMPLETE_YEARS(ENTRY_DATE, BIRTH_DATE-1) + 1

; Calculate duration in force to higher months
 DURATIONIF_M = COMPLETE_MONTHS(EXTRACT_DATE, ENTRY_DATE) + 1

; Set premium variables
 IF PRODUCT = "WPE" THEN
   ANNUAL_PREM = PREMIUM
   SINGLE_PREM = 0
 ELSE
   ANNUAL_PREM = 0
   SINGLE_PREM = PREMIUM
```

Prophet Data Conversion System 3 Release 7.3

Program listing for TUTOR1 in Lesson 3


```
 ENDIF

; Convert sex and smoker codes to numeric format
  IF SEX_CODE = "F" THEN
    SEX = 1
  ELSE
    SEX = 0
  ENDIF

  IF SMOKER_CODE = "S" THEN
    SMOKER_STAT = 1
  ELSE
    SMOKER_STAT = 0
  ENDIF

; Set output details (including defaults in case some
; policies are not given an appropriate filename)
  OUTPUT_FORMAT = "Conventional"
  FILENAME = "Error.RPT"
  HEADING = "Policies are not allocated a filename as at 31.12."
            + VAL_YEAR_T

  IF PRODUCT = "WPE" THEN
    OUTPUT_FORMAT = "Conventional"
    FILENAME = "C WEND.RPT"
    HEADING = "With Profits Endowments data as at 31.12."
              + VAL_YEAR_T
  ENDIF

  IF PRODUCT = "IB" THEN
    OUTPUT_FORMAT = "UnitLinked"
    FILENAME = "U IB  .RPT"
    HEADING = "Investment Bonds data as at 31.12."
              + VAL_YEAR_T
  ENDIF

; Set sub-product code to 1
  SPCODE = 1

; Set initial number of policies to 1
  INIT_POLS_IF = 1

; Validation of policy term
  IF PRODUCT = "WPE" AND POL_TERM_Y < 10 THEN
    INVALID(POL_TERM_Y)
  ENDIF

; Correction of premium frequency
  IF PRODUCT = "WPE" THEN
    SWITCH(PREM_FREQ)
    CASE 1,2,4,12:
  ; For these values nothing is done
    DEFAULT: CORRECT(PREM_FREQ,12)
    ENDSWITCH
  ENDIF

; Table Correction of ages at entry
  PROD_NAME = substr(FILENAME,1,6)
  TABLE_CORRECT("CORRECT", PROD_NAME, AGE_AT_ENTRY)

; Variable to record failure of validation / correction
; tests
  IF FAILED_VALIDATION THEN
    IF_FAILED = 1
  ELSE
    IF_FAILED = 0
  ENDIF

; Calculation of grouping age
```

Program listing for TUTOR1 in Lesson 3

DCS Program Specification SunGard 06/01/2006
Program Name: TUTOR1

```
  IF AGE_AT_ENTRY <= 19 THEN
    AGE_GROUP = 18
  ELSEIF AGE_AT_ENTRY <= 30 THEN
    AGE_GROUP = 25
  ELSEIF AGE_AT_ENTRY <= 40 THEN
    AGE_GROUP = 35
  ELSEIF AGE_AT_ENTRY <= 45 THEN
    AGE_GROUP = 42
  ELSEIF AGE_AT_ENTRY <= 50 THEN
    AGE_GROUP = 47
  ELSEIF AGE_AT_ENTRY <= 60 THEN
    AGE_GROUP = 55
  ELSE
    AGE_GROUP = 65
  ENDIF

; Calculation of duration in months grouping period
  IF DURATIONIF_M <= 24 THEN
    DUR_GROUP = DURATIONIF_M
  ELSEIF DURATIONIF_M >300 THEN
    DUR_GROUP = 301
  ELSE DUR_GROUP = INT((DURATIONIF_M - 25)/12) * 12 + 25
  ENDIF

; Calculation of policy term grouping period
  IF POL_TERM_Y < 3 THEN
    POL_GROUP = 2
  ELSEIF POL_TERM_Y > 28 THEN
    POL_GROUP = 29
  ELSE
    POL_GROUP = INT((POL_TERM_Y - 3)/5) * 5 + 3
  ENDIF

; Calculation of premium frequency for grouping
  IF PREM_FREQ <= 2 THEN
    PREM_FREQ = 1
  ELSE
    PREM_FREQ = 12
  ENDIF

; Set output file path for individual policy output files
  OUTPUT_FILE_PATH("RESULT2")

; Create individual policy output
  OUTPUT(OUTPUT_FORMAT, FILENAME, HEADING)

; Reset output file path for grouped output files
  OUTPUT_FILE_PATH

; Output line for existing business
  GROUP(OUTPUT_FORMAT, FILENAME, HEADING)

; Output line for new business profile
  IF ENTRY_YEAR = VAL_YEAR THEN
    SPCODE = SPCODE + 50
    DURATIONIF_M = 0
    INIT_DECB_IF = 0
    INIT_UNFDU_A = 0

; Set output file path for individual policy output files
  OUTPUT_FILE_PATH("RESULT2")

; Create individual policy output
  OUTPUT(OUTPUT_FORMAT, FILENAME, HEADING)

; Reset output file path for grouped output files
  OUTPUT_FILE_PATH

GROUP(OUTPUT_FORMAT, FILENAME, HEADING)
```

Prophet Data Conversion System 5 Release 7.3

Program listing for TUTOR1 in Lesson 3


```
ENDIF
```

Program listing for TUTOR1 in Lesson 4

DCS Program Specification SunGard 06/01/2006
Program Name: TUTOR1

Program Details

Volume: Directory: C:\DCS Created: 06/01/2006 at 16:31:12

Description: Tutorial Program 1

Run Settings

Run Parameters

Allow run parameters to be specified at runtime	No
Location for Output Files	RESULTS
Location/Name for Access File	
Delete existing Access file at start of run	Yes
ODBC Data Source	Tutorial Data Source
Location for Tables	TABLES
Unit Prices File	

Data Errors

Maximum Number of Runtime Errors Before Termination	0
Ignore Blank Lines (except in header and footer)	No
Treat Missing Trailing Fields as Zero or Blank	No
Ignore any Records not Matching any Read Condition	No
Read Integer Part of Number Values in Integer Fields	No
Treat Following Text Values in Numeric Fields as having a value of	No
Create "_missing_value" Variables for Every Input Variable	No
Write input lines that cause an Error to .ERR file	Yes
Check output values against variable details in associated Prophet workspace	No

Input File(s)

Input File Name/Location	DATA\TUTOR1.TXT
Extract Date	31 December 2000
Type	FIXED ASCII
Header Size	0 Lines
Footer Size	0 Lines

Input Record Format Name = Input, Record Length = 74

Type of Record: Main Record

Method of determining which records this record format applies to: Applies to all records

	Field Name	Start	Length	Type	Date Format	Decimal
1	POL_NUMBER	1	6	Text		
2	PRODUCT	7	3	Text		
3	STATUS	11	1	Text		
4	SEX_CODE	12	1	Text		
5	SMOKER_CODE	13	1	Text		
6	ENTRY_DATE	14	8	Date	ddmmyyyy	
7	BIRTH_DATE	22	8	Date	ddmmyyyy	
8	POL_TERM_Y	31	2	Integer		
9	PREM_FREQ	33	2	Integer		
10	PREMIUM	35	8	Number		File
11	SUM_ASSURED	43	10	Number		File
12	INIT_DECB_IF	53	10	Number		File
13	INIT_UNFDU_A	63	10	Number		File

Program listing for TUTOR1 in Lesson 4

DCS Program Specification SunGard 06/01/2006
Program Name: TUTOR1

Messages in Separate Files

Validation File Name INVALID
Format Name Messages
Heading Invalid Policy Message File

Correction File Name CORRECT
Format Name Messages
Heading Corrected Policy Message File

Output Format Options

Text Qualifier for Model Point, Text and Delimited Output Files None
Use Windows Date Setting Yes

Output File Format Name = Conventional
Format = Model Point, ODBC
Include Terminator Line: No
Include Line Number Column: No

Output Variables Sort/Group/Summary Variables

SPCODE SPCODE
AGE_AT_ENTRY DUR_GROUP
ANNUAL_PREM POL_GROUP
DURATIONIF_M PREM_FREQ
ENTRY_YEAR
INIT_DECB_IF
POL_TERM_Y
PREM_FREQ
SEX
SMOKER_STAT
SUM_ASSURED
INIT_POLS_IF
DUR_GROUP
POL_GROUP

Output File Format Name = UnitLinked
Format = Model Point, ODBC
Include Terminator Line: No
Include Line Number Column: No

Output Variables Sort/Group/Summary Variables

SPCODE SPCODE
AGE_AT_ENTRY AGE_GROUP
DURATIONIF_M
ENTRY_YEAR
INIT_UNFDU_A
SEX
SINGLE_PREM
SMOKER_STAT
SUM_ASSURED
INIT_POLS_IF
AGE_GROUP

Prophet Data Conversion System 2 Release 7.3

Program listing for TUTOR1 in Lesson 4

DCS Program Specification SunGard 06/01/2006
Program Name: TUTOR1

Output File Format Name = Messages
Format = Model Point, Text
Include Terminator Line: No
Include Line Number Column: No

Output Variables Sort/Group/Summary Variables

POL_NUMBER
PRODUCT
STATUS

Output File Format Name = Summary
Format = Model Point, Text
Include Terminator Line: No
Include Line Number Column: No

Output Variables	Sort/Group/Summary Variables
PRODUCT	PRODUCT
INIT_POLS_IF	STATUS
PREMIUM	
RECORDS	
STATUS	
SUM_ASSURED	
INIT_UNFDU_A	

Grouping Calculations/Decimal Places

Decimal places not to be applied to normal output and summarising files

Output Variables	Calculation Type	Weighting Variable 1	Weighting Variable 2	Decimal Places
AGE_AT_ENTRY	Average	PREMIUM		0
AGE_GROUP	Average			0
ANNUAL_PREM	Average			2
DUR_GROUP	Average			0
DURATIONIF_M	Average	PREMIUM		0
ENTRY_YEAR	Average			0
INIT_DECB_IF	Average			2
INIT_POLS_IF	Sum			0
INIT_UNFDU_A	Average			2
POL_GROUP	Average			0
POL_TERM_Y	Average	PREMIUM		0
PREM_FREQ	Average			0
PREMIUM	Average			0
RECORDS	Average			0
SEX	Average			0
SINGLE_PREM	Average			2
SMOKER_STAT	Average			0
SPCODE	Average			0
SUM_ASSURED	Average			2

Program Code

```
EXPLICIT_DECLARATIONS
; Variable Declarations
  VAL_YEAR INTEGER
  VAL_YEAR_T TEXT*4
  ENTRY_YEAR INTEGER
  AGE_AT_ENTRY INTEGER
```

Program listing for TUTOR1 in Lesson 4


```
  DURATIONIF_M INTEGER
  ANNUAL_PREM NUMBER
  SINGLE_PREM NUMBER
  SEX INTEGER
  SMOKER_STAT INTEGER
  OUTPUT_FORMAT TEXT
  FILENAME TEXT
  HEADING TEXT
  SPCODE INTEGER
  INIT_POLS_IF INTEGER
  PROD_NAME TEXT
  IF_FAILED INTEGER
  AGE_GROUP INTEGER
  DUR_GROUP INTEGER
  POL_GROUP INTEGER
  RECORDS INTEGER

; Summarise the policy data based on PRODUCT and STATUS
  RECORDS = 1
  SUMMARISE("Summary", "SUMMARY", "Summary by PRODUCT and STATUS")
```

```
; Exclude records that are not in force
 IF STATUS <> "I" THEN
   NEXT_RECORD
 ENDIF
```

```
; Set valuation year
 IF FIRST_RECORD THEN
   VAL_YEAR = READ_GENERIC_TABLE("EX1_CTRL","Y","VAL_YEAR")
   VAL_YEAR_T = STRVAL(VAL_YEAR)
 ENDIF
```

```
; Calculate entry year
 ENTRY_YEAR = YEAR(ENTRY_DATE)

; Calculate age at entry
 AGE_AT_ENTRY = COMPLETE_YEARS(ENTRY_DATE, BIRTH_DATE-1) + 1

; Calculate duration in force to higher months
 DURATIONIF_M = COMPLETE_MONTHS(EXTRACT_DATE, ENTRY_DATE) + 1

; Set premium variables
 IF PRODUCT = "WPE" THEN
   ANNUAL_PREM = PREMIUM
   SINGLE_PREM = 0
 ELSE
   ANNUAL_PREM = 0
   SINGLE_PREM = PREMIUM
 ENDIF

; Convert sex and smoker codes to numeric format
  IF SEX_CODE = "F" THEN
    SEX = 1
  ELSE
    SEX = 0
  ENDIF

  IF SMOKER_CODE = "S" THEN
    SMOKER_STAT = 1
  ELSE
    SMOKER_STAT = 0
  ENDIF

; Set output details (including defaults in case some
; policies are not given an appropriate filename)
  OUTPUT_FORMAT = "Conventional"
  FILENAME = "Error.RPT"
  HEADING = "Policies are not allocated a filename as at 31.12."
```

Program listing for TUTOR1 in Lesson 4


```
            + VAL_YEAR_T

  IF PRODUCT = "WPE" THEN
    OUTPUT_FORMAT = "Conventional"
    FILENAME = "C_WEND.RPT"
    HEADING = "With Profits Endowments data as at 31.12."
              + VAL_YEAR_T
  ENDIF

  IF PRODUCT = "IB" THEN
    OUTPUT_FORMAT = "UnitLinked"
    FILENAME = "U_IB__.RPT"
    HEADING = "Investment Bonds data as at 31.12."
              + VAL_YEAR_T
  ENDIF

; Set sub-product code to 1
  SPCODE = 1

; Set initial number of policies to 1
  INIT_POLS_IF = 1

; Validation of policy term
  IF PRODUCT = "WPE" AND POL_TERM_Y < 10 THEN
    INVALID(POL_TERM_Y)
  ENDIF

; Correction of premium frequency
  IF PRODUCT = "WPE" THEN
    SWITCH(PREM_FREQ)
    CASE 1,2,4,12:
  ; For these values nothing is done
    DEFAULT: CORRECT(PREM_FREQ,12)
    ENDSWITCH
  ENDIF

; Table correction of ages at entry
  PROD_NAME = substr(FILENAME,1,6)
  TABLE_CORRECT("CORRECT", PROD_NAME, AGE_AT_ENTRY)

; Variable to record failure of validation / correction
; tests
  IF FAILED_VALIDATION THEN
    IF_FAILED = 1
  ELSE
    IF_FAILED = 0
  ENDIF

; Calculation of grouping age
  IF AGE_AT_ENTRY <= 19 THEN
    AGE_GROUP = 18
  ELSEIF AGE_AT_ENTRY <= 30 THEN
    AGE_GROUP = 25
  ELSEIF AGE_AT_ENTRY <= 40 THEN
    AGE_GROUP = 35
  ELSEIF AGE_AT_ENTRY <= 45 THEN
    AGE_GROUP = 42
  ELSEIF AGE_AT_ENTRY <= 50 THEN
    AGE_GROUP = 47
  ELSEIF AGE_AT_ENTRY <= 60 THEN
    AGE_GROUP = 55
  ELSE
    AGE_GROUP = 65
  ENDIF

; Calculation of duration in months grouping period
  IF DURATIONIF_M <= 24 THEN
    DUR_GROUP = DURATIONIF_M
  ELSEIF DURATIONIF_M >300 THEN
```

Program listing for TUTOR1 in Lesson 4


```
    DUR_GROUP = 301
  ELSE
    DUR_GROUP = INT((DURATIONIF_M - 25)/12) * 12 + 25
  ENDIF

; Calculation of policy term grouping period
  IF POL_TERM_Y < 3 THEN
    POL_GROUP = 2
  ELSEIF POL_TERM_Y > 28 THEN
    POL_GROUP = 29
  ELSE
    POL_GROUP = INT((POL_TERM_Y - 3)/5) * 5 + 3
  ENDIF

; Calculation of premium frequency for grouping
  IF PREM_FREQ <= 2 THEN
    PREM_FREQ = 1
  ELSE
    PREM_FREQ = 12
  ENDIF

; Output line for existing business
  GROUP(OUTPUT_FORMAT, FILENAME, HEADING)

; Output line for new business profile
  IF ENTRY_YEAR = VAL_YEAR THEN
    SPCODE = SPCODE + 50
    DURATIONIF_M = 0
    INIT_DECB_IF = 0
    INIT_UNPDU_A = 0

GROUP(OUTPUT_FORMAT, FILENAME, HEADING)
 ENDIF
```

Program listing for TUTOR2 in Lesson 4

DCS Program Specification SunGard 06/01/2006
Program Name: TUTOR2

Program Details

Volume: Directory: C:\DCS Created: 06/01/2006 at 16:43:16

Description: Tutorial Program 2

Run Settings

Run Parameters

Allow run parameters to be specified at runtime	No
Location for Output Files	RESULTS2
Location/Name for Access File	
Delete existing Access file at start of run	Yes
ODBC Data Source	
Location for Tables	
Unit Prices File	

Data Errors

Maximum Number of Runtime Errors Before Termination	0
Ignore Blank Lines (except in header and footer)	No
Treat Missing Trailing Fields as Zero or Blank	No
Ignore any Records not Matching any Read Condition	No
Read Integer Part of Number Values in Integer Fields	No
Treat Following Text Values in Numeric Fields as having a value of	No
Create "_missing_value" Variables for Every Input Variable	No
Write input lines that cause an Error to .ERR file	Yes
Check output values against variable details in associated Prophet workspace	No

Output Format Options

Text Qualifier for Model Point, Text and Delimited Output Files "
Use Windows Date Setting Yes

Output File Format Name = DATA
Format = Model Point
Include Terminator Line: No
Include Line Number Column: No

Output Variables Sort/Group/Summary Variables

SPCODE
AGE_AT_ENTRY
DURATIONIF_M
INIT_POLS_IF
POL_TERM_Y
SEX
SMOKER_STAT
SUM_ASSURED

Program Code

```
SPCODE = 51
SUM_ASSURED = 1000
INIT_POLS_IF = 1
DURATIONIF_M = 0
FOR AGE_AT_ENTRY FROM 16 TO 60
  FOR POL_TERM_Y FROM 5 TO 65
    FOR SEX FROM 0 TO 1
      FOR SMOKER_STAT FROM 0 TO 1
        OUTPUT("DATA", "C_TERM", "Data for Determing Premium Rates")
```

Prophet Data Conversion System 1 Release 7.3

Program listing for TUTOR2 in Lesson 4


```
        ENDFOR
      ENDFOR
    ENDFOR
ENDFOR
```

Program listing for TUTOR3 in Lesson 5

DCS Program Specification SunGard 07/01/2006
Program Name: TUTOR3

Program Details

Volume: Directory: C:\DCS Created: 07/01/2006 at 10:08:28

Description: Tutorial Program 3

Run Settings

Run Parameters

Allow run parameters to be specified at runtime	No
Location for Output Files	RESULTS
Location/Name for Access File	
Delete existing Access file at start of run	No
ODBC Data Source	
Location for Tables	TABLES
Unit Prices File	

Data Errors

Maximum Number of Runtime Errors Before Termination	0
Ignore Blank Lines (except in header and footer)	No
Treat Missing Trailing Fields as Zero or Blank	No
Ignore any Records not Matching any Read Condition	No
Read Integer Part of Number Values in Integer Fields	No
Treat Following Text Values in Numeric Fields as having a value of	No
Create " missing_value" Variables for Every Input Variable	No
Write input lines that cause an Error to .ERR file	Yes
Check output values against variable details in associated Prophet workspace	No

Input File(s)

Input File Name/Location	DATA\Tutor3.txt
Extract Date	31 December 2000
Type	FIXED ASCII
Header Size	0 Lines
Footer Size	0 Lines

Input Record Format Name = Main, Record Length = 38

Type of Record: Main Record

Method of determining which records this record format applies to: Read condition
Read Condition
Start 1
Length 4
Value MAIN

	Field Name	Start	Length	Type	Date Format	Decimal
1	RECORD_TYPEM	1	4	Text		
2	POL_NUMBER	5	7	Integer		
3	PRODUCT	12	3	Text		
4	STATUS	15	1	Text		
5	SEX_CODE	16	1	Text		
6	SMOKER_CODE	17	1	Text		
7	ENTRY_DATE	18	8	Date	yyyymmdd	
8	BIRTH_DATE	26	8	Date	yyyymmdd	
9	POL_TERM_Y	34	2	Integer		
10	PREM_FREQ	36	2	Integer		

Prophet Data Conversion System 1 Release 7.3

Program listing for TUTOR3 in Lesson 5

DCS Program Specification SunGard 07/01/2006
Program Name: TUTOR3

Input Record Format Name = Increment, Record Length = 30

Type of Record: Sub-Record
Main Record Name Main
Maximum Number 20

Method of determining which records this record format applies to: Read condition
Read Condition
Start 1
Length 4
Value INCR

	Field Name	Start	Length	Type	Date Format	Decimal
1	RECORD_TYPEI	1	4	Text		
2	INCRMT_DATE	5	8	Date	yyyymmdd	
3	ANNUAL_PREM	14	6	Number		2
4	SUM_ASSURED	23	7	Number		2

Input Record Format Name = Fund, Record Length = 17

Type of Record: Sub-Record
Main Record Name Increment
Maximum Number 10

Method of determining which records this record format applies to: Read condition
Read Condition
Start 1
Length 4
Value FUND

	Field Name	Start	Length	Type	Date Format	Decimal
1	RECORD_TYPEF	1	4	Text		
2	FUND_CODE	8	2	Integer		
3	NO_FUND_U	12	6	Number		2

Output Format Options

Text Qualifier for Model Point, Text and Delimited Output Files None
Use Windows Date Setting Yes

Output File Format Name = TermAssurance
Format = Model Point
Include Terminator Line: No
Include Line Number Column: No

Output Variables	Sort/Group/Summary Variables
SPCODE	SPCODE
AGE_AT_ENTRY	
ANNUAL_PREM	
DURATIONIF_M	
ENTRY_YEAR	
IF_INCREMENT	
POL_NUMBER	
POL_TERM_Y	
PREM_FREQ	
SUM_ASSURED	

Program listing for TUTOR3 in Lesson 5

DCS Program Specification SunGard 07/01/2006
Program Name: TUTOR3

Output File Format Name = UnitLinked
Format = Model Point
Include Terminator Line: No
Include Line Number Column: No

Output Variables	Sort/Group/Summary Variables
SPCODE	SPCODE
AGE_AT_ENTRY	
ANNUAL_PREM	
DURATIONIF_M	
IF_INCREMENT	
POL_NUMBER	
POL_TERM_Y	
PREM_FREQ	
SUM_ASSURED	
ENTRY_YEAR	

Program Code

```
; Declare variables
 VALN_YEAR DATE
 AGE_AT_ENTRY[20] INTEGER
 DURATIONIF_M[20] INTEGER
 IF_INCREMENT[20] INTEGER
 TERM_Y[20] INTEGER
 ENTRY_YEAR[20] INTEGER

; Set valuation year
 VALN_YEAR = EXTRACT_DATE

; Calculate values for each increment
 FOR i FROM 1 TO NO_OF_RECORDS("Increment")
   ENTRY_YEAR[i] = YEAR(INCRMT_DATE[i])
   AGE_AT_ENTRY[i] = COMPLETE_YEARS(INCRMT_DATE[i],BIRTH_DATE) + 1
   DURATIONIF_M[i] = COMPLETE_MONTHS(VALN_YEAR, INCRMT_DATE[i])
   TERM_Y[i] = POL_TERM_Y - (ENTRY_YEAR[i] - ENTRY_YEAR[1])

; Set variable if record is increment
  IF i = 1 THEN
    IF_INCREMENT[i] = 0
  ELSE
    IF_INCREMENT[i] = 1
  ENDIF
 ENDFOR

 ; Set sub-product code
   SPCODE = 1

 ; Create output
   IF PRODUCT = "TRM" THEN
     OUTPUT_FORMAT = "TermAssurance"
     FILENAME = "C_TERM"
     HEADING = "Term Assurance data as at 31.12.2000"
   ELSE OUTPUT_FORMAT = "UnitLinked"
     FILENAME = "U_MIP_"
     HEADING = "Maximum Investment Plan data as at 31.12.2000"
   ENDIF
   OUTPUT(OUTPUT_FORMAT, FILENAME, HEADING)
```

Program listing for TUTOR3 in Lesson 6

DCS Program Specification SunGard 07/01/2006
Program Name: TUTOR3

Program Details

Volume: Directory: C:\DCS Created: 07/01/2006 at 10:56:30

Description: Tutorial Program 3

Run Settings

Run Parameters

Allow run parameters to be specified at runtime	No
Location for Output Files	RESULTS
Location/Name for Access File	
Delete existing Access file at start of run	No
ODBC Data Source	
Location for Tables	TABLES
Unit Prices File	TABLES\EX_PRICE.FAC

Data Errors

Maximum Number of Runtime Errors Before Termination	0
Ignore Blank Lines (except in header and footer)	No
Treat Missing Trailing Fields as Zero or Blank	No
Ignore any Records not Matching any Read Condition	No
Read Integer Part of Number Values in Integer Fields	No
Treat Following Text Values in Numeric Fields as having a value of	No
Create " missing value" Variables for Every Input Variable	No
Write input lines that cause an Error to .ERR file	Yes
Check output values against variable details in associated Prophet workspace	No

Input File(s)

Input File Name/Location	DATA\Tutor3.txt
Extract Date	31 December 2000
Type	FIXED ASCII
Header Size	0 Lines
Footer Size	0 Lines

Input Record Format Name = Main, Record Length = 38

Type of Record: Main Record

Method of determining which records this record format applies to: Read condition
Read Condition
Start 1
Length 4
Value MAIN

	Field Name	Start	Length	Type	Date Format	Decimal
1	RECORD_TYPEM	1	4	Text		
2	POL_NUMBER	5	7	Integer		
3	PRODUCT	12	3	Text		
4	STATUS	15	1	Text		
5	SEX_CODE	16	1	Text		
6	SMOKER_CODE	17	1	Text		
7	ENTRY_DATE	18	8	Date	yyyymmdd	
8	BIRTH_DATE	26	8	Date	yyyymmdd	
9	POL_TERM_Y	34	2	Integer		
10	PREM_FREQ	36	2	Integer		

Program listing for TUTOR3 in Lesson 6

DCS Program Specification SunGard 07/01/2006
Program Name: TUTOR3

Input Record Format Name = Increment, Record Length = 30

Type of Record: Sub-Record
Main Record Name Main
Maximum Number 20

Method of determining which records this record format applies to: Read condition
Read Condition
Start 1
Length 4
Value INCR

	Field Name	Start	Length	Type	Date Format	Decimal
1	RECORD_TYPEI	1	4	Text		
2	INCRMT_DATE	5	8	Date	yyyymmdd	
3	ANNUAL_PREM	14	6	Number		2
4	SUM_ASSURED	23	7	Number		2

Input Record Format Name = Fund, Record Length = 17

Type of Record: Sub-Record
Main Record Name Increment
Maximum Number 10

Method of determining which records this record format applies to: Read condition
Read Condition
Start 1
Length 4
Value FUND

	Field Name	Start	Length	Type	Date Format	Decimal
1	RECORD_TYPEF	1	4	Text		
2	FUND_CODE	8	2	Integer		
3	NO_FUND_U	12	6	Number		2

Output Format Options

Text Qualifier for Model Point, Text and Delimited Output Files None
Use Windows Date Setting Yes

Output File Format Name = TermAssurance
Format = Model Point
Include Terminator Line: No
Include Line Number Column: No

Output Variables	Sort/Group/Summary Variables
SPCODE	SPCODE
AGE_AT_ENTRY	
ANNUAL_PREM	
DURATIONIF_M	
ENTRY_YEAR	
IF_INCREMENT	
POL_NUMBER	
POL_TERM_Y	
PREM_FREQ	
SUM_ASSURED	

Program listing for TUTOR3 in Lesson 6

DCS Program Specification SunGard 07/01/2006
Program Name: TUTOR3

Output File Format Name = UnitLinked
Format = Model Point
Include Terminator Line: No
Include Line Number Column: No

Output Variables Sort/Group/Summary Variables

SPCODE SPCODE
AGE_AT_ENTRY
ANNUAL_PREM
DURATIONIF_M
IF_INCREMENT
POL_NUMBER
POL_TERM_Y
PREM_FREQ
SUM_ASSURED
ENTRY_YEAR
INIT_UNFDU_A
INIT_UNFDU_C

Output File Format Name = SumUnits
Format = Model Point, Text
Include Terminator Line: No
Include Line Number Column: No

Output Variables Sort/Group/Summary Variables

NO_OF_LINKS

Program Code

```
; Declare variables
 VALN_YEAR DATE
 AGE_AT_ENTRY[20] INTEGER
 DURATIONIF_M[20] INTEGER
 IF_INCREMENT[20] INTEGER
 TERM_Y[20] INTEGER
 ENTRY_YEAR[20] INTEGER

; Set valuation year
 VALN_YEAR = EXTRACT_DATE

; Calculate unit values
 CALC_UNIT_VALUES("AC", "INIT_UNFDU_")

; Summarise unit values
 NO_OF_LINKS = 1
 SUMMARISE_UNITS("SumUnits", "SUMUNITS", "Summary of units")

; Calculate values for each increment
 FOR i FROM 1 TO NO_OF_RECORDS("Increment")
   ENTRY_YEAR[i] = YEAR(INCRMT_DATE[i])
   AGE_AT_ENTRY[i] = COMPLETE_YEARS(INCRMT_DATE[i],BIRTH_DATE) + 1
   DURATIONIF_M[i] = COMPLETE_MONTHS(VALN_YEAR, INCRMT_DATE[i])
   TERM_Y[i] = POL_TERM_Y - (ENTRY_YEAR[i] - ENTRY_YEAR[1])

; Set variable if record is increment
  IF i = 1 THEN
    IF_INCREMENT[i] = 0
  ELSE
    IF_INCREMENT[i] = 1
  ENDIF
 ENDFOR

 ; Set sub-product code
   SPCODE = 1
```

Prophet Data Conversion System 3 Release 7.3

Program listing for TUTOR3 in Lesson 6

DCS Program Specification SunGard 07/01/2006
Program Name: TUTOR3

```
IF PRODUCT = "TRM" THEN
  OUTPUT_FORMAT = "TermAssurance"
  FILENAME = "C_TERM"
  HEADING = "Term Assurance data as at 31.12.2000"
ELSE OUTPUT_FORMAT = "UnitLinked"
  FILENAME = "U_MIP_"
  HEADING = "Maximum Investment Plan data as at 31.12.2000"
ENDIF
OUTPUT(OUTPUT_FORMAT, FILENAME, HEADING)
```

Prophet Data Conversion System 4 Release 7.3

Program listing for TUTOR4 in Lesson 7

DCS Program Specification SunGard 07/01/2006
Program Name: TUTOR4

Program Details

Volume: Directory: C:\DCS Created: 07/01/2006 at 13:43:54

Description: Tutorial Program 4

Run Settings

Run Parameters

Allow run parameters to be specified at runtime	No
Location for Output Files	RESULTS
Location/Name for Access File	
Delete existing Access file at start of run	Yes
ODBC Data Source	
Location for Tables	
Unit Prices File	

Data Errors

Maximum Number of Runtime Errors Before Termination	0
Ignore Blank Lines (except in header and footer)	No
Treat Missing Trailing Fields as Zero or Blank	No
Ignore any Records not Matching any Read Condition	No
Read Integer Part of Number Values in Integer Fields	No
Treat Following Text Values in Numeric Fields as having a value of	No
Create " missing_value" Variables for Every Input Variable	No
Write input lines that cause an Error to .ERR file	Yes
Check output values against variable details in associated Prophet workspace	No

Input File(s)

Input File Name/Location	C:\DCS\DATA\VAL97.TXT, C:\DCS\DATA\VAL98.TXT		
Extract Date	31 December 1998		
Type	DELIMITED	Delimiter	T
		Text Qualifier	None
		Read variable names from line	No
Header Size	0 Lines		
Footer Size	0 Lines		

Input Record Format Name = Input

Type of Record: Main Record

Method of determining which records this record format applies to: Applies to all records

	Field Name	Type	Date Format	Length	Decimal
1	POLICY_NUMBER	Integer			
2	PREMIUM	Number			File
3	SUM_ASSD	Number			File
4	DECLARED_BON	Number			File
5	FREQ	Integer			
6	AGE	Integer			
7	ISSUE_YEAR	Integer			
8	ISSUE_MONTH	Integer			
9	POLICY_TERM	Integer			

Output Format Options

Text Qualifier for Model Point, Text and Delimited Output Files	None
Use Windows Date Setting	Yes

Program listing for TUTOR4 in Lesson 7

DCS Program Specification SunGard 07/01/2006
Program Name: TUTOR4

Output File Format Name = Standard
Format = Model Point
Include Terminator Line: No
Include Line Number Column: No

Output Variables	Sort/Group/Summary Variables
SPCODE	SPCODE
POL_NUMBER	
STATUS_1	
STATUS_2	
MVT_MONTH_1	
MVT_MONTH_2	
DURATIONIF_M	
AGE_AT_ENTRY	
POL_TERM_Y	
ANNUAL_PREM	
PREM_FREQ	
SUM_ASSURED	
INIT_DECB_IF	
ENTRY_MONTH	
ENTRY_YEAR	

Program Code

```
; Sort input files based on policy number and file number
 SORT_INPUT("POLICY_NUMBER", "FILE_NUMBER")

; Create a variable set equal to FILE_NUMBER because
; FILE_NUMBER cannot be used as an argument to the
; PREVIOUS function because it is a function not a variable
 FILE_NO = FILE_NUMBER

; Model point output for policies that terminated during the year
 IF POLICY_NUMBER <> PREVIOUS("POLICY_NUMBER")
 AND PREVIOUS("FILE_NO") = 1 THEN
   STATUS_1 = 1
   MVT_MONTH_1 = 0
   IF PREVIOUS("ISSUE_YEAR") + PREVIOUS("POLICY_TERM") = 1998 THEN
     STATUS_2 = 4
     MVT_MONTH_2 = PREVIOUS("ISSUE_MONTH")
   ELSE
     STATUS_2 = 2
     MVT_MONTH_2 = 6
   ENDIF
   OUTPUT("Standard", "C_END_", "Model Point File for C_END_ at 31 Dec 1998")
 ENDIF

; If policy in both start of year and end of year file
 IF POLICY_NUMBER = PREVIOUS("POLICY_NUMBER")
 AND FILE_NUMBER = 2 THEN

; Check if altered policy
  IF PREMIUM <> PREVIOUS("PREMIUM")
  OR SUM_ASSD <> PREVIOUS("SUM_ASSD")
  OR FREQ <> PREVIOUS("FREQ")
  OR AGE <> PREVIOUS("AGE")
  OR ISSUE_YEAR <> PREVIOUS("ISSUE_YEAR")
  OR ISSUE_MONTH <> PREVIOUS("ISSUE_MONTH")
  OR POLICY_TERM <> PREVIOUS("POLICY_TERM") THEN
    ALTERED_POLICY = 1
  ELSE
    ALTERED_POLICY = 0
  ENDIF

; If altered policy, output model point line for policy information at start of year
  IF ALTERED_POLICY = 1 THEN
    STATUS_1 = 1
    STATUS_2 = 7
    MVT_MONTH_1 = 0
```

Program listing for TUTOR4 in Lesson 7


```
    OUTPUT("Standard", "C_END_", "Model Point File for C_END_ at 31 Dec 1998")
  ENDIF
 ENDIF

; Set output details so that details for the current policy are output by any
; future use of the OUTPUT function for this record
 SPCODE = 1
 POL_NUMBER = POLICY_NUMBER
 AGE_AT_ENTRY = AGE
 ANNUAL_PREM = PREMIUM
 SUM_ASSURED = SUM_ASSD
 INIT_DECB_IF = DECLARED_BON
 PREM_FREQ = FREQ
 POL_TERM_Y = POLICY_TERM
 ENTRY_YEAR = ISSUE_YEAR
 ENTRY_MONTH = ISSUE_MONTH
 DURATIONIF_M = (1998-ENTRY_YEAR) * 12 + 13 - ENTRY_MONTH

; If current policy only in end of year file
 IF POLICY_NUMBER <> PREVIOUS("POLICY_NUMBER")
 AND FILE_NUMBER = 2 THEN
   STATUS_2 = 1
   MVT_MONTH_2 = 13
   IF ENTRY_YEAR = 1998 THEN
     STATUS_1 = 0
     MVT_MONTH_1 = ISSUE_MONTH
   ELSE
     STATUS_1 = 2
     MVT_MONTH_1 = 6
   ENDIF
   OUTPUT("Standard", "C_END_", "Model Point File for C_END_ at 31 Dec 1998")
 ENDIF

; If current policy in both start of year file and end of year file
 IF POLICY_NUMBER = PREVIOUS("POLICY_NUMBER")
 AND FILE_NUMBER = 2 THEN

; If not an altered policy
   IF ALTERED_POLICY = 0 THEN
     STATUS_1 = 1
     STATUS_2 = 1
     MVT_MONTH_1 = 0
     MVT_MONTH_2 = 13

; If an altered policy
   ELSE
     STATUS_1 = 7
     STATUS_2 = 1
     MVT_MONTH_1 = 6
     MVT_MONTH_2 = 13
   ENDIF
   OUTPUT("Standard", "C_END_", "Model Point File for C_END_ at 31 Dec 1998")
 ENDIF

; If last record and policy only in start of year file then output model point line
 IF LAST_RECORD AND FILE_NUMBER = 1 THEN
   STATUS_1 = 1
   MVT_MONTH_1 = 0
   IF ISSUE_YEAR + POLICY_TERM = 1998 THEN
     STATUS_2 = 4
     MVT_MONTH_2 = ISSUE_MONTH
   ELSE
     STATUS_2 = 2
     MVT_MONTH_2 = 6
   ENDIF
   OUTPUT("Standard", "C_END_", "Model Point File at 31 Dec 1998")
 ENDIF
```

Part 3 - Reference Sections

SECTIONS IN THIS PART

SECTION A

Input file types supported by DCS

This section sets out details of the types of files that the Data Conversion System is able to read and convert into the formats required for Prophet and Glean calculations.

TOPICS IN THIS SECTION

Input file formats

DCS is able to read five main types of file:

- Fixed format
- Delimited format
- Excel format
- Database format
- Model Point format
- Glean data format

Fixed format

For these types of file the position of each field within each record in the file is determined by where it starts and how long it is. For example, a particular field might start at position 23 and be of length 8.

This type of file can be in either ASCII, EBCDIC or ICL EBCDIC format.

Delimited format

For these types of file there is a delimiter character between each field. This delimiter will normally be a space or a comma but can be other characters including a Tab. Text fields may be contained within text qualifiers of either single quotes or double quotes, or there may be no text qualifier.

This type of file must be in ASCII format.

Excel format

The name of each record format that you set up must be the same as the name of the worksheet within the workbook which contains the data that you want to read for that format.

Each worksheet that you read should have a row containing the column names followed by the data rows (the column names can be different from the field names specified for the record format in DCS). There should be no other data in that worksheet.

A "Populate" option is provided which enables you to set up all the field information for an input record format by reading that information from the appropriate worksheet of the input file.

You enter the name and location of the Excel workbook file in the same way as for any other type of input file and then select the File Type as EXCEL.

Database format

The name of each record format that you set up must be the same as the name of the table within the database file which contains the data that you want to read for that format.

A "Populate" option is provided which enables you to set up all the field information for an input record format by reading that information from the appropriate table of the input file.

You need to set up a data source which connects to the database file you want to read. You do this by first selecting the File Type as DATABASE and then using the Administrator button.

Model Point File format

Model Point files are ASCII format files that have a delimiter character between each field or data element. The delimiter character can be a space or a comma. Text fields may be contained within text qualifiers of either single quotes or double quotes, or there may be no text qualifier.

Lines containing the names of fields in a Model Point file must start with a ! (exclamation) or & (ampersand) character with each field name separated by a comma or a space.

Lines containing data corresponding to the fields must start with a * (star character) with each data element separated by a comma or space. Note that the decimal point character used in numbers must be in UK format, i.e. a full stop or period character and not a comma.

You will normally use this option when you are reading model point format files created by either DCS or Prophet.

Glean Data format

A Glean data format files is generated by Glean when it exports data from a model. DCS can also generate Glean data format files whose contents are specified in a DCS output file format.

Field types

The input file can contain the following field types, although the last 4 are only available for fixed format files:

Text

Number

Integer

Date

Short Integer

Signed Binary

Binary

Packed

IBM Float

Multiple record formats

DCS is able to read files which contain either a single record format or those that have multiple record formats. A file might have multiple record formats to cater for policies where there are a number of different benefits attached to each policy or where each increment is recorded as a separate sub-record.

In order to read a file that contains multiple record formats DCS needs to know where each record starts and ends. There are two ways in which this can be done:

1. Each record contains a field which is at the same position in each record type and which holds a different number or text string depending on the record type. For example, the main record type might contain the character M at a particular position while the benefit record type contains the character B. If the value is in a binary field you should enter \x followed by the value you require in hex notation. For example, enter \x0F if the value you require has a byte value of 15.
2. Each record contains a field which specifies the number of sub-records that follow that particular record. For example, the main record might contain a field that specified that 3 benefit records followed it for the current policy.

Multiple input files

A DCS program is able to read several input files at the same time using the following capabilities:

Wildcards in the input file name

If there are several files which are identical in format but contain different data they can be processed one after the other by using wildcards in the specification of the input file name. This facility might be used if a company had a number of administration systems covering different products but which produced output files with identical formats.

Input record format specifies the input file it applies to

If there are several files which have different formats then the names can be specified in each of the input record formats which are set up. Each of these files is then processed one after the other. This facility might be used for group pensions business where the details for active, pensioner, spouse and deferred members are held in different files, each of which contains different data fields.

Additional information record

If there is a separate file that contains additional data then it can be associated with the data in the main input file. For example, for group pensions business the current salary of each member might be in a different file to that containing details of each member's age, retirement date, etc.

Merging of input files

If there are several files that contain data related to the same policies then the data in those files can be merged together, based on a field such as the policy number. For example, motor insurance data will often consist of separate policy and claims files. This facility can be used to merge the files so that the claims data can be associated with the corresponding policy data. (The Glean General Insurance example demonstrates how this can be done in DCS.)

Generic tables

It is possible to read data from generic tables which is then used in conjunction with the data from the main input files. This is useful in situations where there is additional data for each record which is too extensive to be sensibly entered directly into the program code. For example, if the premiums were not held in the main input file they could be calculated using premium rates held in a generic table. Generic tables can hold both numeric and text data which is indexed by up 10 dimensions. If the premium rates varied by age, policy term, sex and smoker status they could be read from a 4-dimensional generic table.

Transferring the file to the PC or network

In order for DCS to be able to read the input file it needs to be either on the PC which is running DCS or on a network server which can be accessed from that PC. Therefore if the file was originally created as an extract from a mainframe administration system it needs to be transferred to the PC or network server.

If the file was created in EBCDIC format it may be possible to convert it to ASCII format as part of the process of transferring it. However, if the file contains packed, binary or IBM Float fields then it must be left in EBCDIC format and the conversion to ASCII format carried out by DCS. The reason for this is that for these types of fields it is not possible to carry out the conversion on a character by character basis without corrupting the values.

File sizes / hard disk requirements

Typically each 10,000 policies will require between 1 MB and 2 MB of disk space. Therefore if the input file contains 1 million policies it will normally require between 100 MB and 200 MB of disk space. A similar amount of disk space will be required for the Prophet model point and Glean data files that are created by DCS. In addition a similar amount of disk space should be allowed for the temporary storage which is required by the sorting and grouping processes that DCS can carry out.

There is a limit to the size of the input files that DCS can process of 2,048 GB if you are using Windows NT, Windows 2000 or Windows XP and 2 GB if you are using Windows 98 or Windows Millennium.

For large files you should set the amount of memory to be used for sorting and grouping to be at least 1% of the size of the input file. For example, for a 2,000 MB input file you should use at least 20 MB of memory for sorting and grouping.

SECTION B

Summary of DCS functions

TOPICS IN THIS SECTION

DCS functions summary

The functions provided by the DCS programming language are summarised in the following table:

Function	Description
ABS	Returns the absolute value of the argument
ASCII_VALUE	Returns the ASCII value of a character in a string
BIT_VALUE	Returns the value (1 or 0) of the nth bit of a binary field
BLANK_DATE	Returns the blank date of 1 January 100
CALC_UNIT_VALUES	Returns the value of the units for each main record based on the number of units in each fund and the unit price
COMPLETE_MONTHS	Returns the complete number of months between two dates
COMPLETE_YEARS	Returns the complete number of years between two dates
CORRECT	Returns a corrected value to be used in the calculations where there is incorrect data
CREATE_INPUT_FILE	Creates a data file containing a specified number of main records based on a specified input file format.
DATE_CONVERT	Converts a year, month and day into a date
DAY	Returns the day of a date
DELETE_EXISTING_OUTPUT_FILES	Deletes existing output files
DIV	Returns the integer part of the division of two integers
DO LOOP/WHILE	Controls looping
DO WHILE/LOOP	Controls looping

Function	Description
EXACT_YEARS	Returns the exact difference between two dates expressed in years and fractions of years
EXP	Returns the exponential of the argument
EXPLICIT_DECLARATIONS	Requires that all variables created within the code are declared before use
EXTRACT_DATE	Returns the extract date of the input file(s), as specified in the main window of DCS
FAILED_VALIDATION	Enables you to test whether data has failed any validation checks or whether data correction has taken place
FILE_NUMBER	Returns the order number of the input file from which the current record was read.
FIRST_RECORD	Returns TRUE if the current record is the first record in the file, and FALSE otherwise
FOR/FROM/TO/STEP /ENDFOR	Controls looping
FRACT	Returns the fractional part of the argument
GROUP	Produces grouped output files
GROUP_RECORD	Outputs a single record for inclusion in a grouped output file.
IF/THEN/ELSEIF/ELSE/ENDIF	Enables commands to be performed according to certain conditions
INPUT_FILE_DATE_TIME	Returns a text string containing the date and time of the input file from which the current main record was read
INPUT_FILE_EXTENSION	Returns the extension of the input file from which the current main record was read
INPUT_FILE_LOCATION	Returns the location of the input file from which the current main record was read
INPUT_FILE_NAME	Returns the name of the input file from which the current main record was read

Function	Description
INSERT_C_CODE /END_C_CODE	Enables raw C code to be entered in the DCS program. This allows more complex functionality, which is not supported by the DCS language, to be incorporated
INT	Returns the integer part of the argument
INVALID	Allows data validation
LAST_RECORD	Returns TRUE if the current record is the last record in the file, otherwise returns FALSE
LEAP_YEAR	Returns TRUE if the year is a leap year, otherwise returns FALSE
LEN	Returns the length of a string
LN	Returns the natural log of the argument
LOG	Returns the log to base 10 of the argument
LOOP	Passes control to the top of the nearest enclosing FOR or WHILE loop. LOOP skips the commands between it and the next ENDFOR or ENDWHILE and returns to the beginning of the structure
MAIN_FORMAT_NAME	Returns the name of the main record format for the current main record
MAX	Returns the maximum of a series of numbers or dates
MAX_NUM_SUBRECORDS	Returns the number that was entered in the DCS User Interface for MaxNo for the sub-record format with the given name.
MESSAGE	Writes a message to the run log file
MIN	Returns the minimum of a series of numbers or dates
MOD	Returns the remainder of the division of two integers
MONTH	Returns the month of a date
MONTHS_ADD	Adds a specified number of months to a date

Function	Description
MONTHS_SUB	Subtracts a specified number of months from a date
NEAREST_MONTHS	Returns the difference between two dates to the nearest number of months
NEAREST_YEARS	Returns the difference between two dates to the nearest integral number of years
NEXT_RECORD	Causes the program to move to the next record and hence skip any remaining processing for the current record
NEXT_VALUE	Returns the value of an input variable from the next record
NO_OF_CURRENT_RECORDS	Returns the number of records of the specified input format that were read for the current main record and which have not been suppressed using the SUPPRESS_RECORD function
NO_OF_RECORDS	Returns the number of records of the specified input format name that were read for the current main record
NUMVAL	Converts the value of a string variable to a number
OUTPUT	Outputs records to a file
OUTPUT_ARRAY_AS_CONSTANTS	Causes an array variable to be output as a series of additional fields rather than as separate records
OUTPUT_FILE_PATH	Enables you to send output files to a location other than the one specified in the Setup Run information
OUTPUT_RECORD	Outputs a single record to a file
PREVIOUS	Returns the value of an input variable or a variable in the code from a previous record.
PREVIOUS_VALUE	Returns the value of an input variable or a variable in the code from a previous record

Function	Description
QUITLOOP	Terminates the innermost FOR or WHILE loop currently being executed. The program continues with the command following the most immediate ENDFOR or ENDWHILE
RAND	Returns a random number which is greater than or equal to zero and less than one
RE_START	Restarts the DCS program to read an input file again from the beginning
READ_GENERIC_TABLE	Reads a numeric value from a generic table
READ_GENERIC_TABLE_TEXT	Reads a text value from a generic table
READ_INPUT_FOOTER	Returns a value from a specified number of characters from the footer record of an input file.
READ_INPUT_FOOTER_TEXT	Returns a text value from a specified number of characters from the footer record of an input file.
READ_INPUT_HEADER	Returns a value from a specified number of characters from the header record of an input file.
READ_INPUT_HEADER_TEXT	Returns a text value from a specified number of characters from the header record of an input file.
RECORD_COUNT	Returns the number of the record currently being processed
RECORD_SUPPRESSED	Returns True or False depending on whether a record has been suppressed or not
ROUND	Returns the integer nearest to the argument
ROUND_DOWN	Returns the argument rounded down to the specified number of decimal places
ROUND_NEAR	Returns the argument rounded to the specified number of decimal places, rounded to the nearest

Function	Description
ROUND_UP	Returns the argument rounded up to the specified number of decimal places
RUN_DCS_PROGRAM	Enables another DCS program to be run from within the current program
RUN_PARAMETER	Enables numeric parameters to be specified at run time
RUN_PARAMETER_TEXT	Enables text parameters to be specified at run time
SET_MAIN_RECORD	Sets the main format to be ouput.
SET_NUM_SUBRECORDS	Sets the number of sub-records to write for the given instances of the higher level records.
SORT_ARRAY	Sorts the values in an array variable
SORT_CURRENT_RECORDS	Sorts all the records of a specified format
SORT_INPUT	Causes the input file(s) to be sorted before the remainder of the code is processed
STRVAL	Converts the value of a numeric variable to a string
SUBSTR	Returns a substring of a string
SUMMARISE	Produces output files summarised by the summary variables specified in the output file format
SUMMARISE_RECORD	Outputs a single record for inclusion in an output file containing summarised information
SUMMARISE_UNITS	Produces output files summarised by the summary variables specified in the output file format plus PRICE_CODE
SUMMARISE_UNITS_RECORD	Outputs a single record for inclusion in an output file containing summarised unit number and value information
SUPPRESS_RECORD	Suppresses a record so that it is not included in the output

Function	Description
SWITCH/ENDSWITCH	Enables commands to be performed according to certain conditions. Easier to use in some circumstances than a series of nested IF statements
TABLE_CORRECT	Returns a corrected value to be used in the calculations where there is incorrect data, reading generic tables for the data bounds
TABLE_VALIDATE	Allows data validation, reading generic tables for the data bounds
TERMINATE	Allows you to terminate the program
TIME	Returns the number of seconds elapsed since midnight as an integer
TODAY	Returns the current date
TRIM	Removes leading spaces from character variables
UBOUND	Returns an integer that represents the largest available subscript for the indicated dimension of an array.
WHILE/ENDWHILE	Controls looping
YEAR	Returns the year of a date
YEARS_ADD	Adds a specified number of years to a date
YEARS_SUB	Subtracts a specified number of years from a date

Full details of all the DCS programming functions and statements are contained in Help in DCS.

SECTION C

Differences between the data grouping functionality in DCS and Prophet

This section details the differences between the grouping functionality in DCS and Prophet and the factors that should affect your choice as to whether to use DCS or Prophet for grouping your policy data.

TOPICS IN THIS SECTION

Differences in grouping functionality

The main differences between the grouping functionality in DCS and Prophet are as follows:

- DCS provides Average and Sum as the calculation types within a group whereas Prophet also provides Fixed, Range and Count. You can achieve the same effect as Fixed, Range and Count within DCS by setting the relevant variables equal to the required values. However, care is needed if individual policy output is subsequently produced for the same policy, as may be the case for new business model points. Alternatively, you can create extra variables within the code.
- DCS applies the same grouping criteria for a particular variable in all the output files produced from a particular DCS program. For example, you can define DURATIONIF_M as a weighted average using WEIGHT_PREM that is to be output to 0 decimal places. DCS then applies this definition to all grouped output files containing DURATIONIF_M. However, within Prophet, you specify separate grouping calculations for each product, enabling you to have different calculations for the same variable across products.
- DCS provides the facility for you to specify that each grouped model point must represent at least a minimum number of individual policies. This enables you to avoid the situation where some model points represent tens of thousands of policies whereas others only represent one or two. This facility does not yet exist in Prophet.
- In Prophet you cannot create new variables or calculate values to help with the grouping. In DCS you can, which means that it is more flexible. However, setting up the grouping is more complex.

Choosing between DCS and Prophet

Both DCS and Prophet use the same basic approach to carrying out the grouping. Therefore you can achieve the same results with either system, although one of them may be more appropriate depending upon your exact requirements. Therefore when deciding whether to carry out the grouping of your data within DCS or Prophet, you should consider the following factors:

- If you use a system other than DCS for carrying out the conversion process then you should use Prophet to produce grouped model point files. With DCS you would need a separate program for each product, which would be time consuming to set up, check and maintain.

- If you only require grouped output files, then producing them using DCS saves the additional step of producing the individual policy files in DCS and then setting up and running Prophet to produce the grouped output files.
- Using Prophet you can easily set up and run 5 different model point groupings. In DCS you can use the OUTPUT_FILE_PATH function to produce several sets of grouped output files (that is, with different grouping criteria) in different output directories from within the same program. However, this is more complex to program.
- Using Prophet, you can run the grouping for a single product. With DCS the entire extract file needs to be processed each time.
- It is simpler to set up grouping ranges in Prophet than in DCS because dialogs are provided to help you do this. In DCS you have to specify the grouping ranges using formulas.
- Using Prophet you can more easily batch grouping runs with calculation runs.
- Using DCS you can set a minimum for the number of policies that a model point must represent.

SECTION D

Data grouping considerations and comparison checks

The topics in this section discuss the factors that you consider when setting the grouping rules and the types of comparison checks that you can apply to help check that the grouped model points adequately model the individual policy data.

TOPICS IN THIS SECTION

Grouping considerations

The factors you should consider when setting the grouping rules to produce grouped model points include:

The relative importance of a particular product

If only a small amount of business has been sold then only a relatively small number of model points would be justified. Indeed, it may be better to combine the product with another larger product which has similar profitability characteristics. However, if the run times are not significant for this business you may decide that the work involved in producing the grouped data is not justified and that you will carry out individual policy calculations.

The purpose for which the model will be used

Models can be developed for a number of purposes, for example:

- Bonus prospects
- Capital requirements
- Embedded values
- Effect of AIDS mortality
- Short term financial monitoring
- Asset/liability modelling
- Stochastic modelling

A different set of model points could, in theory, be justified for each of these purposes.

Run times

If an excessive number of model points are produced the time taken for a complete run of the model office could become a problem.

Comparison checks

Once the business has been grouped and a Prophet calculations run made, the results must be checked against results from other systems. These could include:

- Published valuation
- Bonus reserve valuation
- Expense analysis
- Accounting information

You should also apply reasonability checks and, if possible, compare the results produced using grouped data with those produced using individual policy data.

These comparisons will provide checks on the underlying data, the profit test assumptions and the accumulation of results, as well as on the choice of grouping rules.

SECTION E

Guidelines for data grouping of particular life product types

The topics in this section set out guidelines on grouping particular life insurance product types. In most cases the guidelines are obvious and are based on an understanding of the important profitability criteria for each product. However, slavish adherence to the guidelines may not result in an adequate model of a particular product or company. You should carefully consider the features of the particular products and company that is being modelled.

These products do not form a comprehensive list of life product types. However, they do cover the main categories and it should be possible to extend the ideas to other types.

TOPICS IN THIS SECTION

Annuities in payment

The most important factors are:

- Current age
- Period to expiry of guarantee period
- Sex
- Single/joint life

The mathematical reserves for this business are very sensitive to age as well as the other factors which affect mortality. For this reason, small age bands are required. Depending upon the number of policies involved, 3-year age bands between ages 60 and 96 should be sufficient with wider bands outside this range.

Prophet needs to be supplied with the age at entry and the duration in-force in months. You can calculate these in DCS by setting the:

- Age at entry equal to the age last birthday at the start of the projection.
- Duration in-force equal to 6 months.

Within the Prophet product, the policy term can then be set equal to:

```
100 - current age, subject to a minimum of 15.
```

For guaranteed annuities, the period to the expiry of the guarantee is also important. You must look at the policy data to see what the spread of periods is. If most of the annuities were issued with an original guarantee period of 5 years, then a grouping by curtate term in years with curtate terms of 5 or more grouped together should be sufficient. The guarantee period must be set equal to the curtate term to run plus 1 if the assumed duration in-force is 6 months.

It should not be necessary to have different sets of model points for different escalation rates since using the weighted average of the escalation rate for each group should produce a good approximation.

Endowments

The most important factors are:

- Policy term
- Duration in-force

It should be sufficient to model the policy term in 5-year bands centred around the quinquennial policy terms with all terms of 28 or more grouped together.

The duration in-force should be modelled monthly for a number of years (up to 4) and annually thereafter. After 15 or 20 years it may be possible to use 5-yearly bands or even to group all durations together. The following factors will affect the choice of the number of years for which the duration needs to be modelled monthly:

- The number of months for which zillmerisation results in zero reserves.
- The initial commission earnings period.

Age at entry is unlikely to have a significant impact for endowments. However, endowments are likely to be a major class of business for with profit offices and, therefore, some banding by age at entry could be justified.

It is likely that joint life first death policies will form a significant proportion of the total and, therefore, separate groupings may be justified.

Investment bonds

The most important factors are:

- Current age
- Duration in-force

In practice, both these factors may not be particularly important unless they affect the assumed lapse rates or there is some clawback of commission on early surrender.

Maximum investment plans

The most important factors are:

- Policy term
- Duration in-force

The same considerations regarding policy term and duration in-force apply here as apply to conventional endowments.

In addition, premium bands may also be required if the allocation percentages vary with premium size. If premium bands were not used in this situation the allocation percentage would be determined by the average premium size of a particular group, which may well be different from the actual average allocation percentage which takes into account the varying premium sizes.

Regular premium pension plans

The most important factors are:

- Policy term
- Duration in-force
- Retirement age

In many ways this type of product is similar to endowments. However, age may be more important, particularly if there is no return on death or a return of premiums with no interest. In practice most pension policies are written to ages 60, 65, 70 or 75 and it is possible to group on retirement age.

Single premium pension plans

The most important factors are:

- Duration to run
- Retirement age

For most single premium policies duration in-force is not an important factor, except for with profits business where the terminal bonus scale may depend on year of issue.

Temporary annuities in payment and guaranteed income bonds

The most important factor is the period to expiry/maturity. This should be treated in a similar way to the period to expiry of the guarantee period in annuities in payment. However, in the case of guaranteed income bonds, it may well be appropriate to group by months to maturity rather than years to maturity if it is a significant proportion of the business of your office.

Age is unlikely to be an important factor but this should be investigated.

Term assurances

The most important factors will be:

- Age at entry
- Policy term
- Duration in-force

You need to take special care here otherwise a very large number of model points will be produced. This number may be out of all proportion to the financial significance of this class of business.

Most policies will have an age at entry in the range 25 to 50. Quinquennial age bands should be sufficient for this range, with separate bands for all ages under 25 and all ages over 50.

It should be sufficient to model the policy term in 5-year bands centred around the quinquennial terms with all terms of 28 or more grouped together.

For most offices, the relatively small volumes of this business should enable the duration in-force to be modelled annually from issue, with 5-year bands used for business in-force for more than 15 or 20 years.

It may be necessary to treat males and females separately and also smokers and non-smokers.

Variable whole life plans

The most important factors will be:

- Age at entry
- Duration in-force
- Sum assured level

If there is a nil allocation period that varies with age at entry, you will probably achieve the best results by using an age band grouping that corresponds with the nil allocation periods. Having a different age band group for each nil allocation period may result in too many model points, in which case a number of nil allocation periods will need to be grouped together.

The number of years which are modelled monthly will largely be determined by the maximum nil allocation period and also by the initial commission earnings period.

The decision as to whether to group by sum assured level will depend upon the relative amounts of business at the different levels and also the effect of the different levels on profitability.

For this type of contract the cost of mortality will almost certainly be met by unit cancellation. It is, therefore, likely that a fairly relaxed attitude can be taken to factors such as sex and joint life since any approximations in the modelling are unlikely to have a significant impact on profitability.

Whole life

The most important factors are:

- Age at entry
- Duration in-force
- Premium paying term

Five-yearly age bands should be sufficient, with wider bands perhaps being appropriate for ages under 30 and over 50. In the latter case, the number of policies is likely to be small unless special policies designed for this age group are offered.

For duration in-force the same considerations apply as for endowments.

It is likely that premiums will cease at certain specified ages, for example 65 or 90. It should be possible to limit the number of such ages to 2 or 3 for modelling purposes and for policies to be allocated to the nearest premium cease age. These ages can then be used as an additional grouping factor.

SECTION F

Information on model point files

This section sets out details of the most significant Prophet model point variables that need to be included in the model point files that will be used for Prophet calculations. It also covers the issues related to including new business model points in the model point file.

TOPICS IN THIS SECTION

Significant model point variables

The Prophet libraries have been set up to expect certain policy information to be provided in a particular form, for example age at entry. The extract data may be available in the form of, say, date of issue and date of birth. Although the Prophet calculations could be modified to accept data in this form, the best approach is normally to use the facilities within DCS to create new fields in the required format.

If the data is to be grouped, DCS can also be used to create certain additional variables that may be needed in the grouping process. An example of this is a weighting variable.

The following sections consider each of the most significant model point variables that may need to be created. Obviously, the list is not exhaustive since the data and product features will vary considerably between offices.

Sub-product code

The sub-product code is specified using the variable SPCODE. It is used to control the accumulation of the results during the calculation process. If the value for SPCODE for a model point is the same as the value for the previous model point then the results will be summed together. If the value for a model point is different from the value for the previous model point then the previous summation will be written to a results file on the hard disk and a new summation will be started.

Often it will only be necessary to use one value of SPCODE for existing business and another for new business. However, if there is any reason for retaining separate results for different sub-sets of the model points, such as there being a number of variants for which separate details are required, then more than one value of SPCODE can be used for either existing business or new business.

The rules for sub-product codes are:

- Every value for SPCODE must be an integer in the range 1 to 99 inclusive.
- The values for SPCODE must always be in ascending order.
- SPCODE must be the first variable in the model point file.

Age at entry

The standard Prophet library formulas require that the age at entry be specified rather than the date of birth. The age should normally be age nearest at entry but, in practice, the use of age next birthday will often be adequate, particularly if the policy data will be grouped. It should be noted that Prophet allows non-integral ages to be used (and also non-integral years of entry for year dependent mortality tables and AIDS tables).

For a joint life policy the age of the second life will also be required. It is necessary to consider at this stage how joint life policies will be treated. The Prophet library formulas allow different ages and different mortality tables to be used for each life. An alternative is to treat the policy as a single life policy for an equivalent age. Another possibility is to treat it as a joint life policy with equal average ages. In all cases the necessary ages need to be created in the data conversion process.

It is also necessary to consider how mortality differences between males and females and smokers and non-smokers will be treated. One possibility is to treat females as males who are (say) 4 years younger than their actual age. A similar approach could be adopted for smokers and non-smokers. Alternatively, the sex (SEX) and smoker status (SMOKER_STAT) variables can used in the Prophet product to specify that different mortality tables are used.

Policy term

Prophet requires that the original policy term in years be specified using the variable POL_TERM_Y for the term in years and the variable ADD_TERM_M for any additional term in months if the policy term is not an integral number of years. Both of these variables must have integral values.

For whole life policies and annuities in payment the policy term will be closely linked to the age at entry. Therefore, provided that you do not want to group on policy term, the policy term for a whole life product may be set up as a formula within the product equal to (say) 112 less the age at entry.

Duration in-force

Prophet requires that the duration in-force be specified to the higher month using the variable DURATIONIF_M. This is the same as the number of months' premiums that would have been paid on a monthly premium policy.

If you will be grouping the data you will often want to group by year of issue for business which is more than a few years old. This will result in maturities mainly occurring in the middle of each future year. If the projection is being used for short term monitoring or for modelling guaranteed income bonds this may not be adequate. This problem can be overcome by creating a new variable which contains the curtate number of months to maturity. Policies within (say) 2 years of maturity could then be grouped by month to maturity, although care would be needed to ensure that the number of model points does not become excessive.

For some products, for example investment bonds and annuities in payment, the duration in-force will not normally have a significant impact on future cashflows. For these products the duration in-force could be arbitrarily set equal to 12 months and the age at entry set equal to one year less than the current age. The ability to do this does depend upon there being no commission clawback, etc on surrender. Also, this approach will only be appropriate for immediate annuities with a guarantee period if the policies are also grouped by outstanding guarantee period.

In-force/PUP number of policy variables

In-force policies and PUPs have to be projected separately if you want to allow for future PUPs, although it is possible to project both of them forwards within the same Prophet product.

Two variables to calculate the number of polices should normally be set up within DCS:

- **In-force number variable - INIT_POLS_IF**
  This should have a value of:
  1 for an in-force policy and
  0 for a paid-up policy.
- **PUP number variable - INIT_PUPS_IF, INIT_WP_PUPS or INIT_NP_PUPS**
  This should have a value of:
  0 for an in-force policy and
  1 for a paid-up policy.

These variables are required for the following purposes:

- They specify to the Prophet product the number of in force policies and paid up policies that the model point represents. In the case of grouped model points the values for them can be summed to calculate the number of policies and PUP'd policies that each model point represents.
- They can be used as weighting variables in the grouping process. For example, for the sum assured for in force policies the variable SUM_ASSURED needs to be the average for just the in force policies. If a model point contains both in force and PUP'd policies then an unweighted average would be calculated across both the in force and PUP'd policies. This problem can be overcome by making INIT_POLS_IF a weighting variable. Similarly for variables such as INIT_WPPUPSA the variable INIT_WP_PUPS should be used as a weighting variable.

Weighting variable

If you are going to group the data you will normally wish to calculate the weighted average age at entry, policy term, duration in-force, etc, in order to achieve a good model. The premium size will often be the best choice for determining the weights to be given to each policy.

If you have no PUP'd policies then ANNUAL_PREM can be used. However, if you do have PUP'd policies this would give no weight to the PUP'd policies because ANNUAL_PREM only relates to premium paying policies. It is therefore better to calculate a weighting variable called WEIGHTING which is set equal to the current annual premium for premium paying policies and (say) 10% of the original premium for PUP'd policies.

If your data does not provide the original premium for PUP'd policies, resulting in 0 for the weighting variable, both DCS and the grouping in Prophet will use a value of 0.000001 for WEIGHTING in the weighting calculation. It will do this for any weighting variable which has a value of 0.

Unit values for unit linked business

The Prophet calculations normally require that the unfunded value of the units attaching to each policy is provided in the model point file. If a particular product has two unit types (such as Capital and Accumulation) then the values are required separately for each unit type.

If the extract file only contains the numbers of units rather than the values then DCS can be used to multiply these by the appropriate unit prices and then accumulate the value for each policy. However, if a product can be linked to a large number of funds it may be better to perform this process on the mainframe to reduce the amount of data that needs to be transferred.

Alternatively the UNIT_NUMBERS indicator could be selected for the product and the numbers of units included in the model point file. The prices information would then also need to be entered in a separate unit prices file and the calculation of the unit values would then take place in the Prophet calculation run. However, it is normally difficult to use this approach if grouping of the policies is required because the number of units for each unit fund would need to held in a different variable and the fund codes would have to have number values. If policies can be linked to a wide range of different unit funds then this would not be practical. The best approach would be to create two sets of model points files using DCS, one of which only contained the unit values and the other which

contained the unit numbers and the fund codes. The first of these sets would be used to create grouped model point files.

For regular premium products which can have PUPs, or to which single premiums can be added, it will normally be necessary to have separate variables for unit values in respect of PUPs or single premiums.

New business model points

The previous sections have dealt mainly with the modelling of existing business. However, the new business normally also has to be included in the model office projections.

It should be noted that normally when creating the model point files only the mix of new business for each product by age, term, etc. needs to be considered. The actual levels of future new business are normally specified in the new business sales file.

The same Prophet product can be used to project the new business model points as is used for the existing business model points. This is achieved by including the new business model points at the end of the existing business model points. The only differences for the new business model points are:

- New business model points must have a sub-product code which is above the value specified by you for "Last Sub-Product Code for Existing Business" in the run setting (the default for this value is 50).
- New business model points which have a sub-product code which is above the value specified by you for "Last Sub-Product Code for New Business based on Sales Volumes" in the run setting are used to model new members into existing group pension schemes (the default for this value is 99).
- Certain model point variables such as duration in-force, value of existing units, attaching bonus, etc. must be set to zero.

Often it will be possible to use the same sub-product code for all the new business model points for a given product. However, if, for example, for with profit endowments the ability to change the relative levels of new business for 10 and 25 year terms was required in the new business file, then these terms would need to be given different sub-product codes.

The best approach to adding the new business model points to the end of the existing business model points is to use DCS to base the new business model points on the existing business. It is necessary to decide which business will form the basis for the new business model points. An obvious choice is the business sold during the previous year (this is the approach used in Lesson 1). However, there may have been certain events during the previous year, such as the introduction of new legislation, which will result in a change in the mix of business within a product. In this situation it may be possible to choose a different period which will reflect the likely mix of future new business. If this is

not possible then you can include code in your DCS program to create the required model points. This approach can also be used for a new product which is planned to be introduced in the future. The "Creating new business model points which are not based on existing business" section in Lesson 4 explains how this can be done.

Having decided which existing business will be used for determining the new business model points, DCS is used to create one output file for each product containing both existing business model points and new business model points. Alternatively, if you are using Prophet to group your data you may decide to use DCS to output one or more files containing a mixture of products and then split them into separate model point files for each product as part of your data grouping run in Prophet.

For the new business model points it is necessary to set to zero the variables that specify the starting position for the model point, for example the duration in-force, value of units, etc.

This approach of creating the new business model points in DCS has the benefit of enabling changes in the grouping criteria to be easily and quickly made for both existing and new business policies and of removing the need to edit model point files to add the new business model points.

An alternative approach is to use the Table Editor or a word processing package to type in the required new business model points. This is quite simple to do since the format of the existing business model points can be followed. However, these would need to be retyped if for any reason the existing business had to be remodelled. Also, the Table Editor is limited to 16,384 model points.

SECTION G

Procedures, Functions and Modules

This section sets out details of how procedures, functions and modules can be used within your DCS code.

TOPICS IN THIS SECTION

Procedures and Functions

You can simplify DCS programming tasks by breaking DCS programs into smaller logical components. These components within DCS are called procedures and functions.

Procedures and functions are useful for condensing repeated or shared tasks, such as frequently used calculations.

There are three major benefits of programming with procedures and functions:

- Procedures and functions allow you to break your DCS programs into discrete logical units. It then becomes much easier to find or modify the DCS code compared to an entire DCS program without procedures or functions.
- Procedures and functions used in one DCS program can act as building blocks for other DCS programs, usually with little or no modification.
- Procedures and functions allow you to reuse code.

With DCS you can create:

- Procedures that do not return a value.
- Functions that return a value.

What is a procedure?

A procedure is a unit of code enclosed between the Proc and EndProc statements that performs a task but doesn't return a value. The following is an example of a procedure that checks a policy term.:

```
Proc CheckPolicyTerm()
   ;Validation of policy term
   IF PRODUCT = "WPE" AND POL_TERM_Y < 10 THEN
         INVALID(POL_TERM_Y)
   ENDIF
EndProc
```

What is a function

A function is a unit of code enclosed between the Function and EndFunction statements. Like a procedure, a function performs a specific task. Unlike a procedure, however, a function also returns a value.

The following example is a Function that checks a policy term as the previous procedure example but also returns a boolean indicating if the policy term was valid:

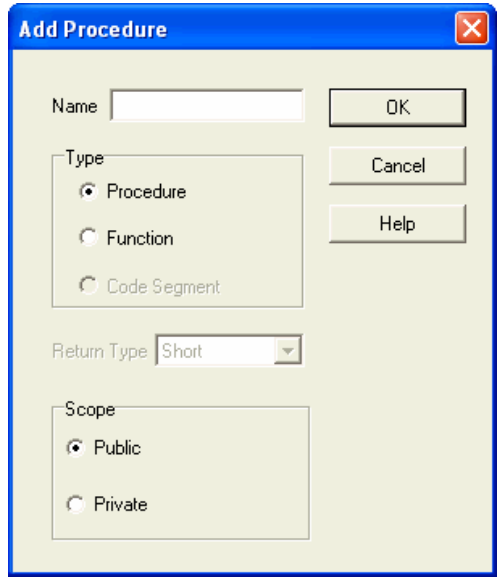
```
Function PolicyTermIsValid() As Boolean
   ;Validation of policy term
   IF PRODUCT = "WPE" AND POL_TERM_Y < 10 THEN
      INVALID(POL_TERM_Y)
      Return False
   ELSE
      Return True
   ENDIF
EndFunction
```

Adding procedures and functions

Procedures and functions can be added to code in two different ways. You can:

- Add procedures and functions by writing the code directly. This is covered in detail later in this section.
- Add procedures and functions by using the 'Add Procedure' dialog.

To use the 'Add Procedure' dialog choose **Add Procedure…** in the **Project** menu to create a new procedure or function. The Add Procedure dialog will then appear and looks similar to the following:



In the dialog:

- Set the name of the procedure or function to be created.
- Select whether a procedure or function is required.
- If a function is required you also need to set the return type for the function.
- Click **OK** and the procedure or function will be added to the code editor at the current insertion point.

If the procedure or function requires parameters the declaration of the procedure or function will need editing. This is covered in detail later in this section.

Declaring procedures

The syntax for a procedure is (where [ ] denote optional elements):

```
[Public | Private] Proc ProcName (arguments)
      statements
EndProc
```

Each time the procedure is called, the statements between Proc and EndProc are executed. A procedure can be placed anywhere in the DCS code as long as they are placed before any code that calls the procedure.

Public versus Private procedures

Procedures in the main DCS code should be declared without the private or public prefix as they can only be private. If you do add a public or private prefix a compile error will occur.

In DCS code modules procedures can be defined as public or private. If a procedure is defined as private it can only be used within the DCS code module where it is defined. If a procedure is defined as public it can called from the main DCS code or other DCS code modules if it is made available using the USES clause. This is covered in more detail later in the chapter.

For procedures declared in a DCS code module the default is Public if it is not set in the procedure declaration.

Procedure name

The length of a procedure name cannot exceed 33 characters. If the name of a procedure exceeds 33 characters then during compilation the name will get truncated. A warning will be displayed in the compilation log.

Procedure arguments

The arguments for a procedure are a list of variables representing the arguments that are passed to the procedure when it is called. It is like a variable declaration. Multiple arguments are separated by commas. Note that it is optional to specify arguments.

The argument list has the following syntax and parts (where [ ] denote optional elements):

```
[ByVal | ByRef] varname [As] type
```

The declaration of an argument consists of 3 parts:

- The scope of the argument
- The variable name of the argument
- The variable type for the argument

Scope of argument

Variables can be declared by value (ByVal) or by reference (ByRef). It is optional to specify this. If ByVal or ByRef is not declared then ByVal is assumed by default.

When an argument is declared by value using ByVal then a copy of the variable is passed into the procedure. If the procedure changes the value, the change affects only the copy and not the variable itself. Passing by value is the default in DCS.

When a variable argument is declared by reference it gives the procedure access to the actual variable. As a result, the variable's actual value can be changed by the procedure to which it is passed.

Declaring an argument

The variable name is the name of the variable representing the argument and follows the standard naming convention of variables within DCS.

If the argument is an array then the variable name must also define the number of dimensions in the array. This can be done by having a array dimension declaration after the variable name. For an array argument the syntax is:

```
[ByVal | ByRef] VarName[0,0,…,0] [As] type
```

The 0's in the declaration are just placeholders to indicate a dimension. For example:

- Short array has 1 dimension then declare argument as:

  VarName[0] As Short

- Number array with 3 dimensions then declare argument as:

  VarName[0,0,0] As Number

Note that DCS has a function called UBOUND that can be used to determine the number of elements within an array if this is not fixed. For example the following statements in a procedure would total all the elements in the array variable passed in and set it to the variable TotalValue:

```
Proc ExampleProc(Byval viVar[0] As Short)

   TotalValue, liIdx As Short
   FOR liIdx:=1 To UBOUND(viVar,1)
       TotalValue = TotalValue + viVar[liIdx]
   Next

EndProc
```

The type for an argument must also be declared. This can be any of the variable types supported by DCS (Integer, Number, Short (Short Integer), Text, Date or Boolean). If the variable type is Text then no length should be specified as text variables are variable length within a procedure.

Examples

The following examples illustrate the declaration of a procedure:

- Proc NoArg
  statements
  EndProc
- Proc NumberArg(Byval viVar As Short, Byval vrVar As Number)
  statements
  EndProc
- Proc TextArg(Byref vsVar Text)
  statements
  EndProc
- Proc ArrayArg(Byref rrVar[0,0] As Number)
  statements
  EndProc
- Proc MixedArg(Byval viVar As Short, Byref rsVar As Text)
  statements
  EndProc

Calling procedures

Once a procedure has been defined it can be called from within the main DCS code. When a procedure is called the execution of the program will transfer to the procedure. Upon completion of the procedure control will be returned to the line following the procedure call.

The syntax for calling a procedure is as follows (where [ ] denote optional elements):

```
[Call] ProcName [argumentlist]
```

The 'Call' part of the statement is optional. The 'ProcName' is the name of the procedure that is being called and must be declared prior to the call statement. If the procedure has an argument list then the call statement must have the same number of arguments as that defined by the procedure declaration and the variables must have the same types.

For example if the code is as follows:

```
Proc ExampleProc(Byval viVar As Short)

      Procedure Statements

EndProc
X = X + 1
Call ExampleProc(X)
Y = Y + 1
```

then:

- First the line 'X = X + 1' would be executed.
- Then the procedure ExampleProc is called with X as an argument. The statements within that procedure would get executed.
- When the statements in the procedure ExampleProc have been executed control is returned to where the procedure was called from. Execution will therefore continue at the line 'Y = Y + 1'.

Function Declaration

The syntax for a function is as follows(where [ ] denote optional elements):

```
[Public | Private] Function FunctionName (arguments) As Type
      statements
      Return Value
EndFunction
```

Each time the function is called, the statements between Function and EndFunction are executed. A function can be placed anywhere in the DCS code as long as they are placed before any code that calls the function.

Like a procedure, a function can take arguments, perform a series of statements, and change the value of its arguments. Unlike a procedure, a function returns a value. There are three differences between procedure and functions:

- You call a function by including the function name and arguments on the right side of a larger statement or expression (returnvalue = function()).
- Functions have data types, just as variables do. This determines the type of the return value.
- You return a value by assigning it to the Return statement. When the function returns a value, this value can then become part of a larger expression. For example if within a function there is a variable called CalculatedValue and this value is to be returned by the function then you would write the statement:

  ```
  Return CalculatedValue
  ```

The rules for declaring a function follow that of declaring a procedure. In addition a Type for the function needs to be defined that specifies the data type of the return value. The Types allowed are the same as those allowed for the arguments. It should be noted that functions cannot return an array.

Examples

The following examples illustrate the declaration of a function:

- Function NoArg As Short
  statements
  EndFunction

- Function NumberArg(Byval vrVar As Number) As Short
  statements
  EndFunction
- Function TextArg(Byref vsVar Text) As Text
  statements
  EndFunction
- Function ArrayArg(Byref rrVar[0,0] As Number) As Short
  statements
  EndFunction

Calling functions

Once a function has been defined it can be called from within the main DCS code. When a function is called the execution of the program will transfer to the function. Upon completion of the function control will be returned to the line following the function call.

The syntax for calling a function is as follows (where [ ] denote optional elements):

```
[ReturnValue=] FunctionName [argumentlist]
```

The 'FunctionName' is the name of the function that is being called and must be declared prior to the statement from where the function is called. If the function has an argument list then the statement must have the same number of arguments as that defined by the function declaration and the variables must have the same types.

The type of the return value variable ReturnValue must also be the same as the declaration return type specified in the function declaration. Note that if a function is used within a statement then the ReturnValue= part of the statement is not required.

For example if the code is as follows:

```
Function ExampleFunction (Byval viVar As Short) As Short

      Function Statements

EndFunction
X = X + 1
Y = ExampleFunction (X)
Z = X + Y
```

then:

- First the line 'X = X + 1' would be executed.

- Then the function ExampleFunction is called with X as an argument. The statements within that function would get executed.
- When the statements in the function ExampleFunction have been executed control is returned to where the function was called from and sets the return value of the function to the variable Y.
- Execution will then continue at the line 'Z = X + Y'.

Note that as well as setting a function to a variable it can also be used within statements. For example, the previous example could also be written as:

```
Function ExampleFunction (Byval viVar As Short) As Short

      Function Statements

EndFunction
X = X + 1
Z = X+ ExampleFunction (X)
```

The value of Z would be the same as in the previous example. The only difference is that the result of the function has not been stored in a separate variable (Y in the previous example).

As well as using a function within the calculation of a variable it can also be used within other DCS statements as though it was a variable. For example, the following code checks if the function return value is greater than 10 and executes the code within the IF statement if this is true:

```
Function ExampleFunction (Byval viVar As Short) As Short

      Function Statements

EndFunction
X = X + 1
If ExampleFunction (X) > 10 Then
      IF Statements
EndIf
```

Statements within procedures and functions

Within the body of a procedure statements can be made in the same way as statements are made within standard DCS code.

Any variables declared in the main DCS script (a module variable) can also be accessed directly within a procedure provided that it is declared before the procedure declaration.

Note that any variables declared in the procedure declaration are only available within the procedure itself and not outside it i.e. the variables are local to the procedure.

Module versus Local Variables

You can have a variable with the same name at a different scope. For example, you could have a module variable named Temp and then, within a procedure, declare a local variable named Temp. References to the name Temp within the procedure would access the local variable. References to Temp outside the procedure would access the module variable.

For example if we have the code:

```
Temp As Integer
Proc ExampleProc()
      Dim Temp As Integer
      Temp = 2     ;i.e. Temp has a value of 2 only within the
procedure.
EndProc
Temp = 1
Call ExampleProc
```

then the variable Temp has a value of 1 prior to the procedure call. When the procedure is called Temp is redeclared within the procedure. This definition of the variable will be used and so Temp has a value of 2 within the procedure. After the procedure has been executed and control has been returned to the main code the value of Temp returns to 1.

In general, when variables have the same name but different scope, the more local variable always shadows (that is, it is accessed in preference to) less local variables. So if you had a procedure-level variable named Temp, it would shadow the module variable Temp within that module.

Further Examples

A procedure can use a variable that is not explicitly declared in the procedure if it is declared at a higher level. The following example illustrate this:

```
miValue As Short
Proc ExampleProc(Byval viVar As Short)

      miValue = miValue + viVar

EndProc
miValue =10
Call ExampleProc(20)
```

then the value of miValue after the procedure is called would be 30. Note that the module variable miValue is declared outside the procedure and is therefore defined at a higher level. If required, it can therefore be accessed directly within the procedure.

Alternatively the same result could also be obtained by passing miValue by reference into the procedure as follows:

```
miValue As Short
Proc ExampleProc(Byref riReturn As Short, Byval viVar As Short)

      riReturn = riReturn + viVar

EndProc
miValue =10
Call ExampleProc(miValue ,20)
```

Note that now the procedure is self contained but achieves the same result after the procedure has been called since miValue was passed into the procedure by reference.

If an argument has the same name as a variable declared at a higher level (i.e. a module variable) then the variable within the procedure or function will only be local within it i.e. changing its value will not change the value of it at the higher level as the local definition will take precedence. For example if the code was as follows:

```
miValue As Short
Proc ExampleProc(Byval miValue As Short)

      miValue = miValue + 10

EndProc
miValue =10
Call ExampleProc(miValue)
```

then the value of miValue after the procedure call will still be 10 since miValue was passed into the procedure by value. miValue within the procedure is local and any changes to it will be lost after the procedure finishes.

Modules

Simple DCS programs can consist of just a single DCS code module. This is the Main code module. Code in the main module is stored within the DCS file and is the code that gets run when a DCS program has been compiled.

As your DCS programs get larger and more sophisticated, you can add separate code modules. You might find that there is common code you want to execute in several different DCS programs. You don't want to duplicate the code in every program so you can create separate modules containing procedures and functions that implement the common code. These separate modules are known as DCS code modules and are stored in files with a DCM extension. DCM files are simply ASCII files containing DCS code that can be used in the main DCS code or other code modules. Over time, you can build up a library of DCS code modules containing shared procedures and functions.

A DCS code module contains the code that can be accessed by DCS programs and other DCS code modules. You create, save and manage all DCS code modules using the DCS interface.

Each DCS code module can contain:

- Declarations - You can place variable declarations at the module level. In addition any other modules used by the module must also be declared.
- Procedures and functions - Proc or Function definitions that contain pieces of code that can be executed as a unit.

DCS code modules can be managed in two ways:

- As part of an existing DCS program
- Stand alone

When DCS code modules are built as stand alone they cannot be compiled. You would generally manage modules as part of a DCS program for which the modules are to be used.

DCS Main code

The main DCS code module forms the foundation of DCS programs. This would consist of a block of code that gets executed when the DCS program is run. In addition the main DCS code can also contain procedures and functions. The code that you write in the main DCS code module is specific to the particular DCS program. It can also reference other DCS code modules.

DCS Code Modules

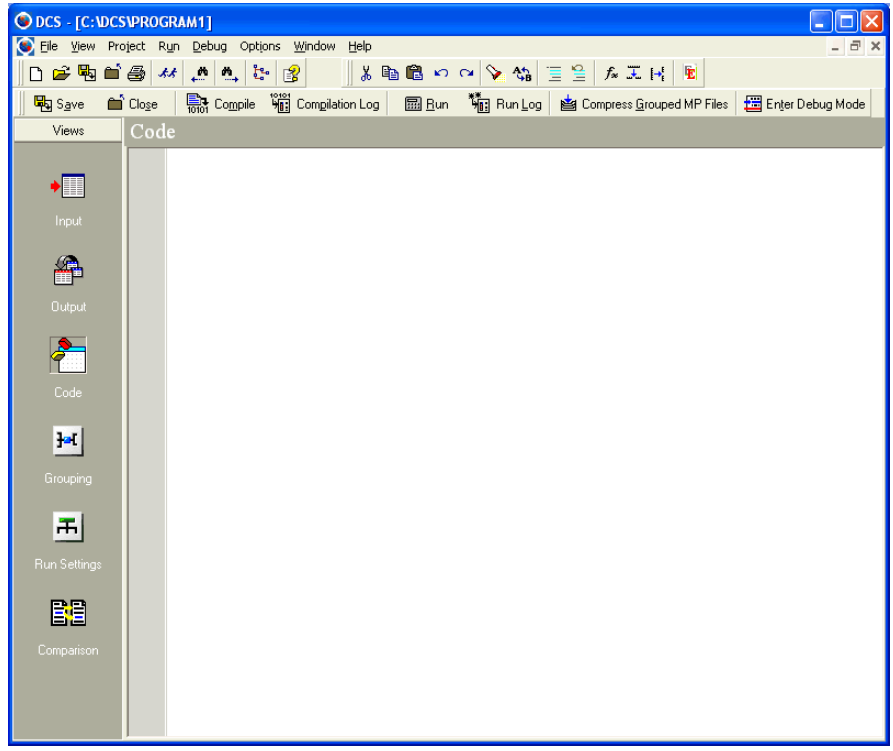
DCS code modules (.DCM file name extension) are containers for procedures and functions commonly accessed by DCS programs and other DCS code modules. DCS code modules contain module-level declarations of variables, procedures and functions. The code that you write in a DCS code module isn't necessarily tied to a particular DCS program and can therefore be reused in many different DCS programs.

Managing DCS code modules and locations

DCS code module management within a DCS program

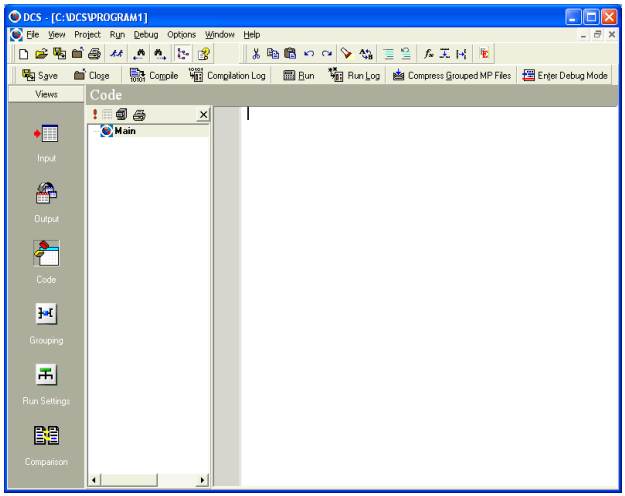
Viewing modules in the stand alone method outlined later on can be useful if you want to look at specific code. In general however modules would be managed whilst a DCS program is open as code modules can only be used and compiled by DCS programs.

When a DCS program is loaded you can switch to the Code view by clicking on the **Code** button in the views bar. The Code view of the DCS program will then appear as follows:



To work with modules you need to activate the project explorer pane so that the project code files can be seen. This can be done by choosing **Project Explorer** in the **View** menu or by using the Project Explorer toolbar button.

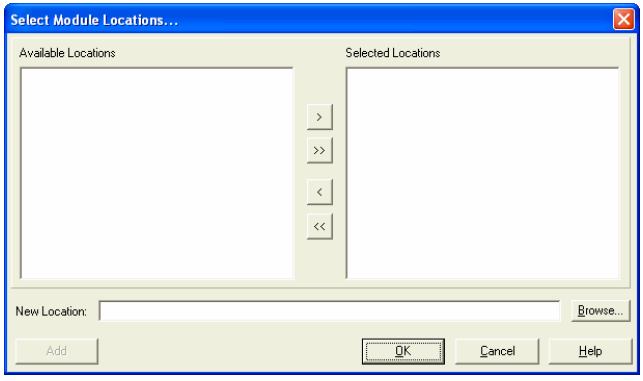
The project explorer pane will then appear and looks similar to the following:



If no modules are referenced by the DCS program a node labelled **Main** will be displayed in the project explorer pane as shown above. The code window that is active will be the main DCS code module that is available as standard with a DCS program. Code can be entered here in the usual way.

DCS module locations

If code modules are required then a reference to a module location needs to be added. Module locations for a DCS program are managed using the Select Module Locations dialog. To manage the module locations choose **Module Location(s)…** in the **Project** menu (this menu item also appears as a toolbar button on the project explorer pane toolbar). The select Module Locations dialog will then appear and looks similar to the following:



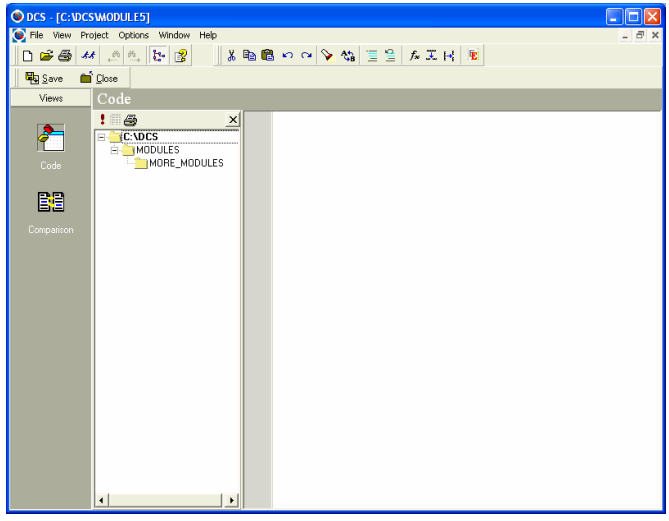
Project explorer pane

The project explorer pane displays a root node for the main DCS program and each module location that has been specified.

If a module location is relative to the DCS program then it will only display the path relative to the DCS program file. In addition the project explorer pane will also display any sub folders within that module location as a sub folder of the module location. For example if:

- The main DCS program file is located in C:\DCS
- Within the main DCS folder C:\DCS there is a folder called MODULES
- Within the MODULES folder there is another folder called MORE_MODULES

then if the location C:\DCS\MODULES is added via the module locations dialog the project explorer pane would look as follows:



You will see that the project explorer has:

- A root node at the top of the project explorer pane called Main. This is the main DCS program code window
- A root node for each module location that has been added using the Select Module Locations dialog.

- Using the above example you can see that a root node called MODULES is displayed. This is the MODULES sub folder relative to the main DCS program file
- For each module location node the folder structure within that location is also displayed.

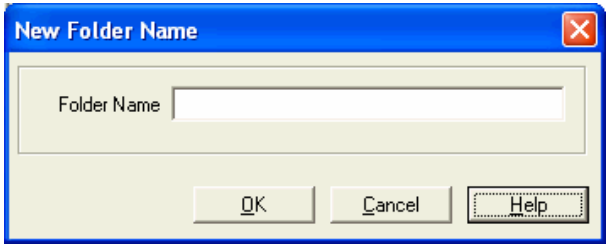
  Using the above example a sub folder is displayed under the MODULES node as this is a sub folder within the MODULES module location added.
- The project explorer also displays the procedures and functions that are defined in the main DCS code or DCS code module as separate nodes. This is covered in more detail later on in the section 'Creating code within a module'.

It should be noted that when a module location is added all the sub folders within the location are added in the project explorer pane. If any of the folders within the module location contain a DCS code module (file with a DCM extension) then these are also displayed. This is explained further below.

Creating new DCS code modules

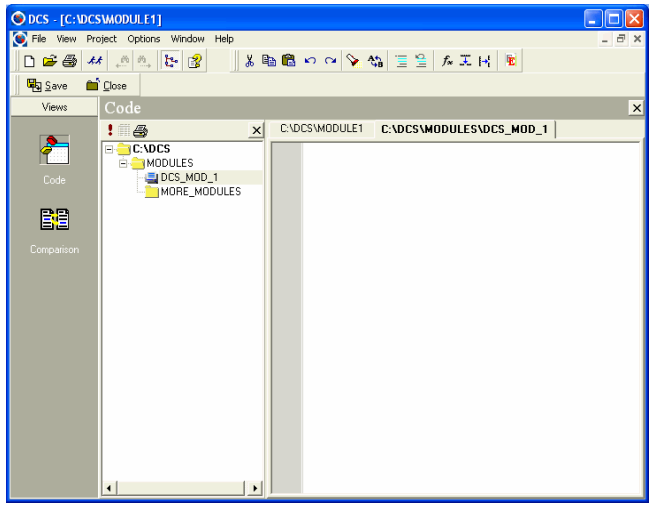
To create a new DCS code module you need to:

- Click the folder in the project explorer pane where the module is to be created.
- Choose **Insert Module File…** in the **Project** menu. A dialog will then appear requesting the name of the module. The dialog appears similar to the following:



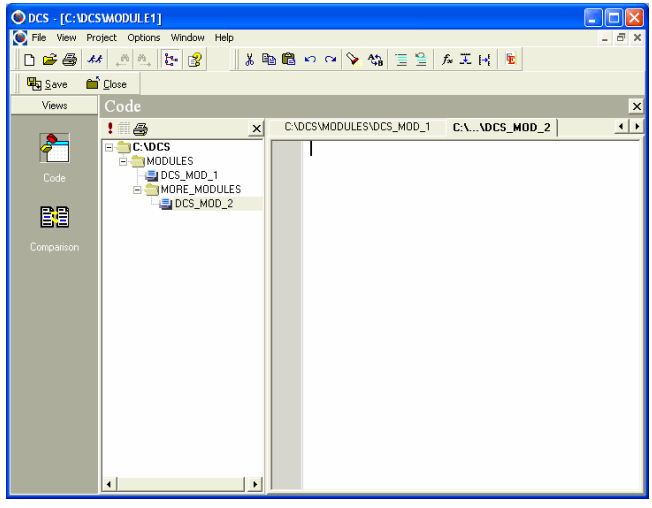
- Type the name of the module in the Module Name text box.
- Click **OK** to create the module.

For example, if we created a module called DCS_MOD_1 then the project explorer would appear similar to the following:



You can see that a new node has been created under the MODULES location node called DCS_MOD_1. This is a DCS code module file that is located in C:DCS\MODULES. The DCS code module file is called DCS_MOD_1.DCM.

If we create another DCS code module in the MORE_MODULES sub folder called DCS_MOD_2 then the project explorer would appear similar to the following:



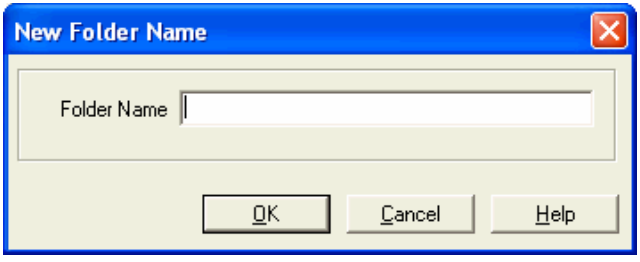
You can see that a new node has been created under the MORE_MODULES sub folder of the MODULES location called DCS_MOD_2. This is a DCS code module file that is located in C:\DCS\MODULES\MORE_MODULES. The DCS code module file is called DCS_MOD_2.DCM.

Managing DCS code module folders

As well as being able to insert and delete DCS code modules you can also insert and delete sub folders within a module location.

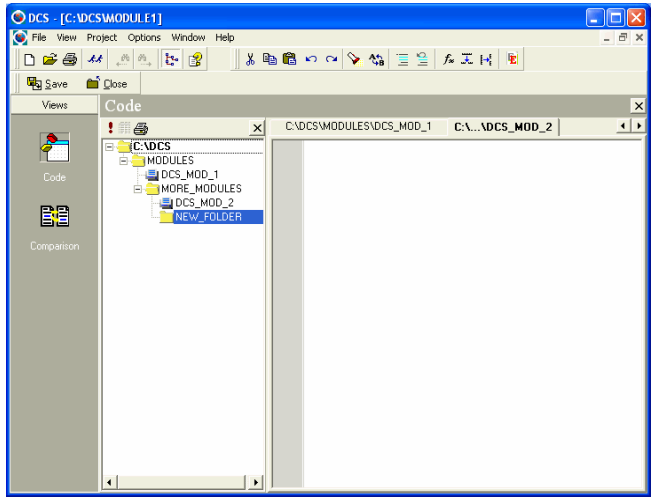
To create a new sub folder within a module location you need to:

- Click the folder in the project explorer pane where the sub folder is to be created.
- Choose **Insert Module Folder…** in the **Project** menu. A dialog will then appear requesting the name of the folder to create. The dialog appears similar to the following:



- Type the name of the folder in the folder name text box.
- Click **OK** to create the folder.

For example, if we created a folder called NEW_FOLDER in the MORE_MODULES folder then the project explorer would appear similar to the following:



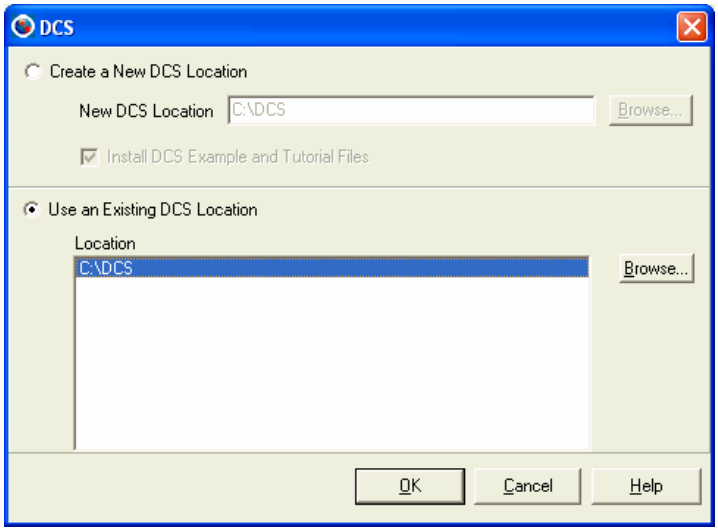
Note that DCS has created a new folder in the location C:\DCS\MODULES\MORE_MODULES called NEW_FOLDER.

Note: This could have been done manually using Windows Explorer. However, if you had done it in Windows Explorer the project explorer pane would not have been updated. To update the project explorer you can choose **Refresh** in the **Project** menu or click the refresh button on the project explorer toolbar. The project explorer would then be refreshed with the latest folder structure. Whenever a DCS program or module location is loaded the module locations will be rescanned to reflect the current structure.

Stand alone DCS code module management

As well as managing your DCS code modules within a DCS program you can also load DCS code modules without a DCS program being loaded.

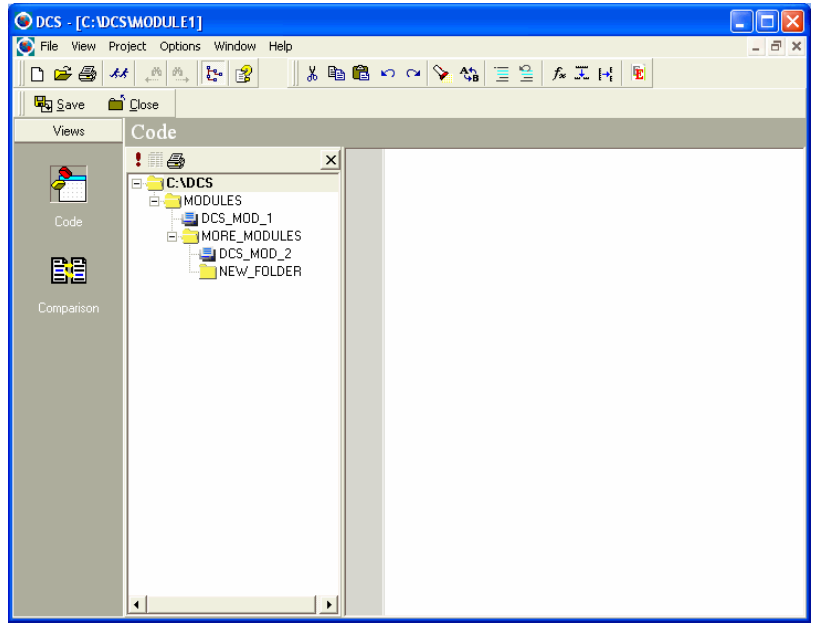
When you start DCS the following dialog appears. It allows you to select the location in which your DCS programs are held:



To open an existing location select **Use an Existing DCS Location** and then either select the location from the list or use the **Browse** button. To create a new location select **Create a New DCS Location**.

Click **OK** and the DCS Program Selection dialog appears from which you can select an existing DCS program. Change the file type drop down to 'DCS Module (*.DCM)' and select the DCS code module file to load. Alternatively you can click **Cancel** and choose **New** in the **File** menu and then select **DCS Module** to create a new DCS code module.

The Code view of the DCS program window for modules then appears as follows:



From this view you can:

- Add, delete and modify DCS code module files
- Enter the DCS module code for a specific module in the Code view
- Carry out a comparison with another DCS code module in the Comparison view so that you can see the differences between the two code modules.

Files can be saved by clicking the **Save** button. When you click save on a new module you will be presented with a save dialog which will allow you to save the module in a specific location.

A new DCS code module can be created at any time by selecting **New** in the **File** menu and then select **DCS Module**.

If DCS is already open you can open a different module location that will allow you to load any of the modules in that specified location and any sub folders within that location. This can be done at any time by selecting **Open Module Location** in the **File** menu and then selecting the location to be opened. A new DCS window will be created listing all the sub folders and modules within that location in the code explorer.

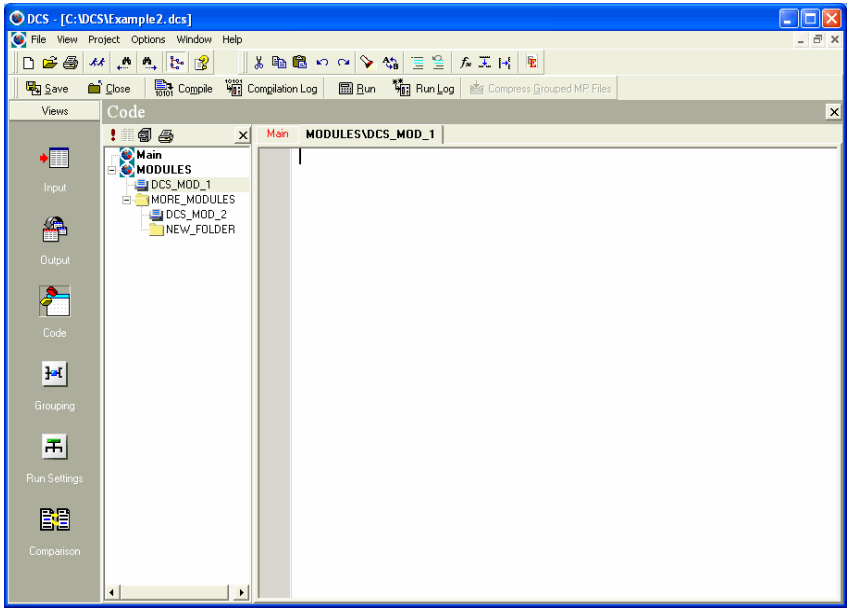
Module Code windows

When modules are present the code window in DCS behaves differently since it is possible to view more than just the main code window.

Loading a code module

A module can be loaded in the code window by selecting the module in the project explorer pane and then choosing **View Code** in the **Project** menu (this menu item also appears as a toolbar button on the project explorer pane toolbar). Alternatively you can double click the DCS code module to be loaded in the project explorer pane.

When a module is loaded in the code window it will be selected in the project explorer pane. In addition the code window will also have tabs along the top. The active tab is the current active module. For example, if we load the DCS_MOD_1 module then the code window will appear similar to the following:



A tab called ‘Main’ is always displayed as this represents the code for the DCS program file. This is always the first item displayed in red to distinguish it from the DCS code modules that may also be loaded.

For each DCS code module that is loaded in the code window a separate tab will appear with the name of the DCS code module. To switch to a DCS code module that is loaded in the code window you can click on the tab to activate it.

Closing a code module

To close a loaded DCS code module you can right click on the tab for the loaded DCS code module and select **Close Module Window**.

As well as closing individual DCS code modules you can close all the loaded DCS code modules. To do this choose **Close All Module Windows** in the **Window** menu.

Note: The main DCS code module is never closed.

Creating code within a module

DCS code can be written in modules in the same way as you create code in the main DCS code module but there are two main differences. DCS code modules consist of:

- Declarations - You can place variable declarations at the module level. In addition any other modules used by the module must also be declared.
- Procedures and functions - Proc or Function definitions that contain pieces of code that can be executed as a unit.

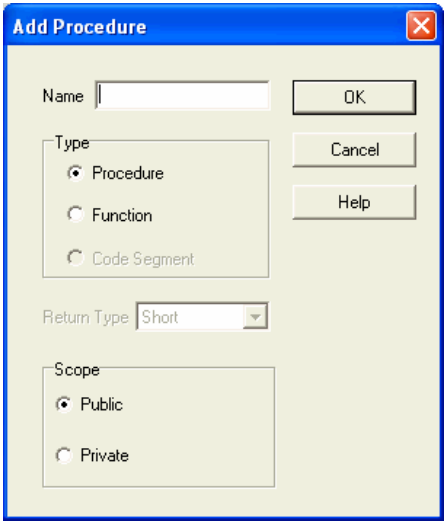
Unlike the main DCS code module all code must be defined within procedures and functions. Any code written in a DCS code module can then be called from the main DCS code or other DCS code modules.

Adding procedures and functions

Procedures and functions can be added to DCS code modules in two different ways. You can:

- Add procedures and functions by writing the code directly
- Add procedures and functions by using the 'Add Procedure' dialog.

To use the 'Add Procedure' dialog choose **Add Procedure…** in the **Project** menu to create a new procedure or function. The Add Procedure dialog will then appear and looks similar to the following:



In the dialog:

- Set the name of the procedure or function to be created.
- Select whether a procedure or function is required.
- If a function is required you also need to set the return type for the function.
- Set the scope of the procedure or function. Note that only public procedures or functions can be called in the main DCS code or other DCS code modules.
- Click **OK** and the procedure or function will be added to the code editor at the current insertion point.

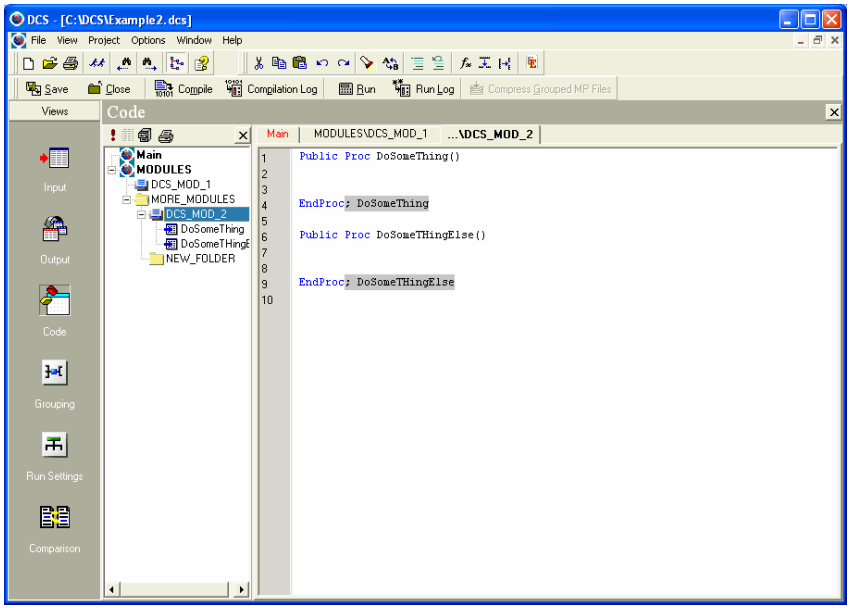
If the procedure or function requires parameters the declaration of the procedure or function will need editing. This was covered in detail earlier in this chapter.

Project explorer pane

The project explorer pane displays a node for each procedure and function that is added to a module. This makes it very easy to navigate the code in modules. To view the code for a procedure or function displayed in the project explorer pane you can:

- Select the procedure or function to view in the project explorer pane and choose **View Code** in the **Project** menu. Alternatively you can click the **View the code** button on the project explorer pane toolbar.
- Double click the procedure or function in the project explorer pane

If the DCS_MOD_2 module has the procedures DoSomething and DoSomethingElse the project explorer pane would look similar to the following:



If you manually enter a procedure or function the project explorer pane needs to be refreshed before they are displayed within it. If you use the Add Procedure dialog it is updated automatically for you. To refresh the project explorer pane choose **Refresh** in the **Project** menu or click the refresh button on the project explorer toolbar.

Using code defined in a module

Code written in a DCS module can be called from the main DCS code or other DCS modules if the procedure or function is defined as public.

When code from a DCS module is to be used in the main DCS code or another DCS module a declaration must appear in the module that will use this code so that the compiler knows where the code is located.

The syntax for this declaration is as follows:

```
USES [MODULE_PATH]
```

The USES statement must appear before the call to the procedure or function in another DCS module is made.

The [MODULE_PATH] is the path of the module to be used including the DCM file extension. The path of the module can be defined in two ways:

- A full path reference
- A relative location reference

USES clause with a full path reference

When a full path reference is used the DCS code module at the location specified will be compiled with the main DCS code. If the file cannot be found at compile time a compile error will occur.

Note that a DCS program can reference a code module using a full path reference even if the module is not loaded in the DCS program i.e. the location for the module does not need to be referenced in the module location(s) dialog.

Note: If a location is hard coded then care is needed if the DCS program is moved.

USES clause with a relative path reference

When a relative path reference is used to reference a DCS code module DCS needs to work out the path reference at compile time. DCS uses the module location(s) you have specified to set a path reference to the DCS code module. DCS does this as follows:

- For each module location specified checks if the relative path exists. The path is made up as follows:
- Module path = [MODULE_LOCATION] + "\" + [USES_PATH]
- where the [USES_PATH] is the relative path specified in the USES statement.
- If the path does not exist for any of the module location(s) specified a compile error will be generated since the module cannot be found.
- If more than one valid path exists for the relative USES reference a compile error will be generated as it is not unique.
- If more than one USES statement references the same path a compile error will be generated as it is not unique.

USES Clause Examples

For example if:

- Main DCS program path is C:\DCS\Program.DCS
- Code module exists with path C:\DCS\Modules\DCSCode.DCM

then the USES statement in the main DCS code to use the code module DCSCode.DCM using a full path references is:

USES “C:\DCS\Modules\DCSCode.DCM”

If a module location of C:\DCS\Modules was added then the USES statement for the DCSCode.DCM module could also be set using a relative reference of:

USES “DCSCode.DCM”

If instead a module location of C:\DCS was added then the relative reference would be:

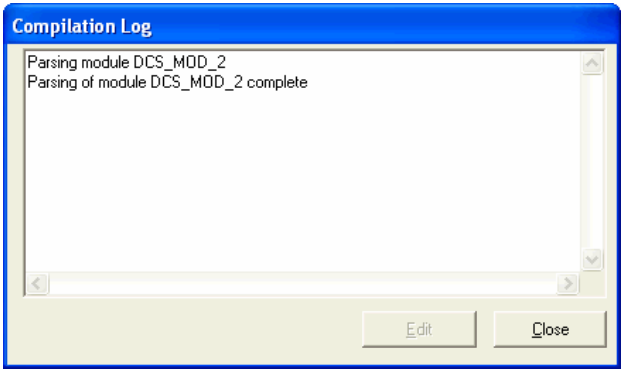
USES “Modules\DCSCode.DCM”

Note in both the USES statements above the concatenation of the module location and the USES path results in the full path to the module.

Checking the code in a module

A DCS code module cannot be compiled by itself because it is not a complete program. DCS, however, does include the command to compile a module which checks that the code can be parsed.

To check that a module can be compiled choose **Compile Module** in the **Project** menu. A compile dialog will then appear and looks similar to the following::



Printing a code module

To print a DCS code module you need to load it in the code window. Once the code window has been activated you can click on the print toolbar button provided on the project explorer pane.

Using a procedure or function defined in another DCS code module

When a USES statement has been declared in the main DCS code or DCS code module the public procedures and functions defined in that module can be called.

The syntax for calling a procedure defined in another module is as follows (where [ ] denote optional elements):

```
[Call] [MODULE_NAME.]ProcName [argumentlist]
```

The only difference between this declaration and the standard procedure declaration outlined earlier in the chapter is the reference to the module name. The reference to the module name is optional. However it should be noted that if you have more than one procedure with the same name in different modules that are being used by the calling module then a compile error will occur as the compiler will not know which procedure is to be used. You can therefore add the module name as a prefix to the procedure name so that the reference is defined.

For example if the uses statement is as follows:

```
USES "Modules\ModuleCode.DCM"
```

and the DCS code module ModuleCode.DCM has a public procedure defined called DoSomething then to call this you would write:

```
Call ModuleCode.DoSomething
```

the MODULE_NAME is therefore just name of the DCS code module file without the extension i.e. ModuleCode. Alternatively assuming the procedure DoSomething does not exist in other modules used by the module being coded you could also write:

```
Call DoSomething
```

The syntax for calling a function defined in another module is similar to calling a procedure. The syntax is as follows (where [ ] denote optional elements):

```
[ReturnValue=] [MODULE_NAME.]FunctionName [argumentlist]
```

Note that the only difference between this declaration and the standard function declaration outlined earlier in the chapter is the reference to the module name.

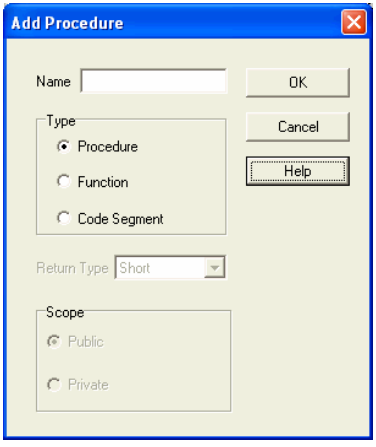
Code Segments

The code for the main DCS program can be broken down into segments which appear on separate tabs. Code segments tabs will appear in the order in which they will be run. Code segments can only be created in a DCS program for the main DCS code.

Code segments can be added to the main DCS code in two different ways. You can:

- Add a code segment by writing the code directly.
- Add a code segment by using the 'Add Procedure' dialog.

To use the 'Add Procedure' dialog choose **Add Procedure…** in the **Project** menu to create a new code segment. The Add Procedure dialog will then appear and looks similar to the following:



In the dialog:

- Set the name of the code segment to be created.
- Set the type to 'Code Segment'.
- Click **OK** and the code segment will be added to the code editor at the current insertion point.

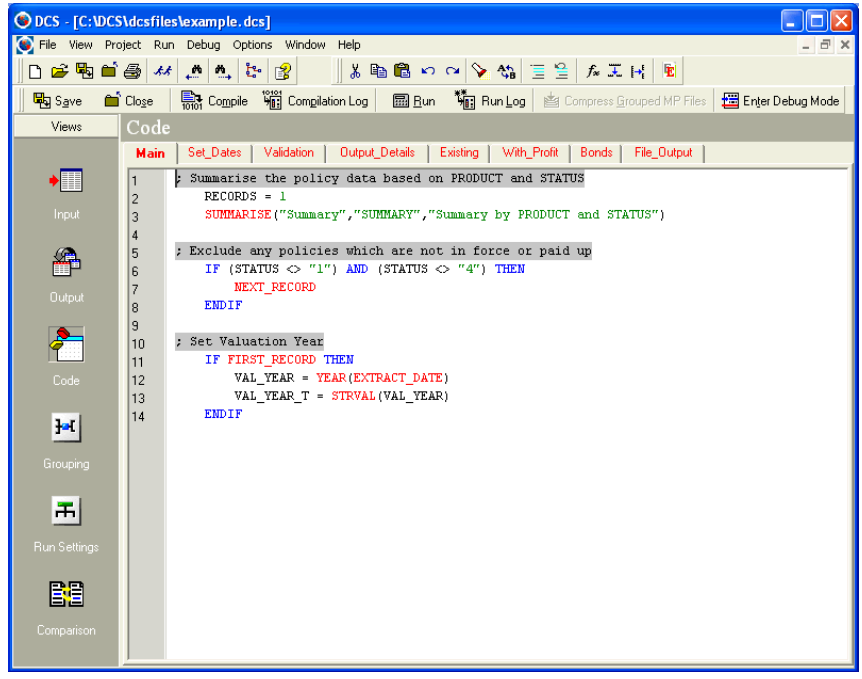
You can also type the code segment code directly to the main DCS code into segments by using the CODE_SEGMENT statement. The syntax for this statement is as follows:

```
CODE_SEGMENT CodeSegmentName
```

where CodeSegmentName is the name of the code segment that will appear on a separate tab. The name of the code segment must be alphanumeric without any spaces in it.

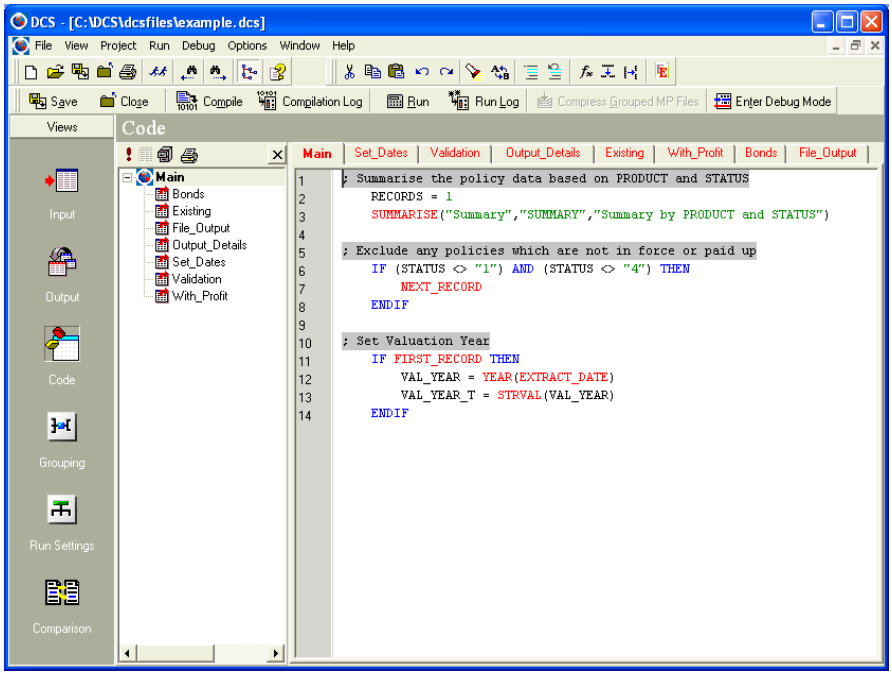
When a code segment is created the DCS code gets split at the point where the code segment statement appears and is displayed on a separate tab. Note that the Main tab will always exist and any code segments created will appear in order after this tab.

If the code segment is not added using the Add Procedure dialog the code only gets split into segments when the code window is refreshed. To refresh the code window choose **Refresh** in the **Project** menu or click the refresh button on the project explorer toolbar. The code window with code segments will look similar to the following:



As well as being displayed as tabs, code segments will also appear in the project explorer pane. Any code segments in the main DCS code appear under the Main node within the project explorer pane in alphabetical order.

The code window will look similar to the following if the project explorer pane is active:



INDEX

E

F

G