CLARION 5

Enterprise Tools

COPYRIGHT 1998, 1999 by TopSpeed Corporation
All rights reserved.

This publication is protected by copyright and all rights are reserved by TopSpeed Corporation. It may not, in whole or part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from TopSpeed Corporation.

This publication supports Clarion Enterprise Edition. It is possible that it may contain technical or typographical errors. TopSpeed Corporation provides this publication as is, without warranty of any kind, either expressed or implied.

TopSpeed Corporation
150 East Sample Road
Pompano Beach, Florida 33064
(954) 785-4555

Trademark Acknowledgments:

TopSpeed® is a registered trademark of TopSpeed Corporation.
Clarion™ is a trademark of TopSpeed Corporation.
Wise™ is a trademark of Wise Solutions, Inc.
All other products and company names are trademarks of their respective owners.

Printed in the United States of America (0699)

CONTENTS

PART I

BUSINESS MATH LIBRARY 13

1 - INTRODUCTION 15

Documentation Conventions **15**
Typeface Conventions 15
Keyboard Conventions 15
Usage Conventions and Symbols 16
Reference Item Format 16

About this Part **18**

About the Business Math Library **18**

Installing the Business Math Library **19**
Business Math Library Template Support 19
Business Math Library Hand-Code Support 20

2 - FINANCE LIBRARY 23

Overview **23**
Finance Library Components 23
Finance Library Conventions 24

Procedures **26**
AMORTIZE (amortize loan for specific number of payments) 26
APR (annual percentage rate) 28
COMPINT (compound interest) 29
CONTINT (continuous compounding interest) 30
DAYS360 (days difference based on 360-day year) 31
FV (future value) 32
PREFV (future value with prepayment) 33
IRR (internal rate of return) 34
NPV (net present value) 36

PERS (periods of annuity) 38
PREPERS (periods of annuity with prepayment) 40
PMT (payment of annuity) 42
PREPMT (payment of annuity with prepayment) 43
PV (present value) 44
PREPV (present value with prepayment) 45
RATE (rate of annuity) 46
PRERATE (rate of annuity with prepayment) 47
SIMPINT (simple interest) 48

3 - Business Statistics Library 49

Overview **49**
Business Statistics Library Components 49

Procedures **51**
FACTORIAL (factorial of a number) 51
FREQUENCY (frequency count of an item in a set) 52
LOWERQUARTILE (lower quartile value of a set) 53
MEAN (mean of a set) 54
MEDIAN (median of a set) 55
MIDRANGE (midrange of a set) 56
MODE (mode of a set) 57
PERCENTILE (pth percentile value of a set) 58
RANGEOFSET (range of a set) 59
RVALUE (linear regression correlation coefficient) 60
SS (sum of squares) 61
SSXY (sum of squares for x and y) 62
ST1 (student’s t for a single mean) 63
SDEVIATIONP (standard deviation of a population) 64
SDEVIATIONS (standard deviation of a sample) 65
SUMM (summation of a set) 66
UPPERQUARTILE (upper quartile value of a set) 67
VARIANCEP (variance of a population) 68
VARIANCES (variance of a sample) 69

4 - BUSINESS MATH TEMPLATES 71

Overview **71**
Template Components ... 71
Registering the Template Classes ... 74
Adding Extension Templates to Your Application ... 75
Embedding Code Templates in Your Application ... 76

Finance Code Templates **77**
AMORTIZE ... 77
APR ... 79
COMPINT ... 80
CONTINT ... 81
DAYS360 ... 82
FV ... 83
IRR ... 84
NPV ... 85
PERS ... 86
PMT ... 87
PV ... 88
RATE ... 89
SIMPINT ... 90

Business Statistics Code Templates **91**
FACTORIAL ... 91
FREQUENCY ... 92
LOWERQUARTILE ... 93
MEAN ... 94
MEDIAN ... 95
MIDRANGE ... 96
MODE ... 97
PERCENTILE ... 98
RANGEOFSET ... 99
rVALUE ... 100
SDEVIATION ... 101
SS ... 102
SSxy ... 103
ST1 ... 104
SUMM ... 105
UPPERQUARTILE ... 106
VARIANCE ... 107

5 - Business Math Examples 109

Overview 109
Exploring the Example Application 109

Part II

Development Tools 111

6 - Dictionary Synchronizer 113

Overview 113
What the Synchronizer Does 113
Synchronizer Servers 114
Running the Synchronizer 115

Dictionary Synchronizer Tutorial 116
Tutorial Objectives 116
Consolidate Two Clarion Data Dictionaries 116
Review Dictionary Synchronizer Log 126
Convert Existing Data to New Format 127
Create Btrieve (or SQL) Database from a Clarion Dictionary 129
Create a Clarion Data Dictionary from an SQL Database 134

Synchronizer Wizard 138
Starting the Synchronizer 138
Select the Synchronizer Mode 138
Select the Other Dictionary 139
Set the Dictionary to Synchronize (source and target) 140
Set the Synchronizer Options 141
Set Automatic Matching Rules 141
Specify Files or Tables to Synchronize 143

Synchronize Dictionaries Dialog 145
Overview 145
Configuring the Synchronize Dictionaries Dialog 146
Status Indicators 146
Navigating the Synchronizer Tree 148
Matching Unmatched Items 148
Resolving Differences Between Dictionaries 149
Resolving Invalid Proposals 151

Implementing the Changes **152**
Running the Conversion Program ... 152
Running the SQL Script ... 153
Dictionary Synchronizer Options **154**
Options ... 154
Colors ... 156
Batch Synchronization **157**
7 - TopSpeed Version Control System 159
Overview **159**
What Is Version Control? ... 159
Terminology ... 159
Version Control for all your Workfiles ... 160
Team Development Facilities ... 160
All within the Clarion Development Environment ... 161
Version Control Tutorial **162**
The Archive Menu ... 162
Configure Archive Directories ... 163
Create archives ... 163
Retrieving Workfiles from Archives ... 166
Archive Menu (with files open) ... 168
Adding more procedure locks ... 169
Checking out other files ... 172
Reports ... 172
Utilities ... 173
Creating Project Definition files ... 175
Working with Project Definition files ... 176
Template Definition files ... 177
Summary ... 178
TopSpeed Version Control System (VCS) **179**
Basics ... 179
Version Labels ... 179
Version Control System Projects ... 180
Reports and Utilities ... 180
Shared Archives ... 180
Code Overwrite ... 180
Locks—Show other users when a file is in use ... 180
Sharing an APP file ... 181

User ID 181
Components 181

Configuring the Version Control System 182
Configuration Files 182
Archive Directories 183
Advanced Configuration 184

Using the VCS Administrator 186
First Time in the Administrator 186
What can you do in the Administrator? 188
Archive Directories 188
Configuration 189
Event Triggers 192
Revision Storage Options 195
Security 196
Advanced 196

Configuration Strategies 197
Single developer, non-networked environment 197
Single developer, networked environment 197
Multi-developer, single-project environment 197
Multi-developer, multi-project environment 198

Customizing the Version Control System 199
Setting your Version Control User ID 199
File Masks 199
Security 199

Using VCS Status 200

Using the Version Control System 201
Archive Menu 201

Basic Tasks 203
Check In 203
Check Out 207
Undo a Check Out 211
Undo a Check In 212
Reports 213

Advanced Tasks 215
Project Definition Files 215
Utilities 218

Security **220**
Security and Team Development 220
Security Strategies 220
Setting up Security in VCS 221

Problem Solving / Troubleshooting **225**
General 225
Check Out 225
Check In 226
Recovery from a Network Crash 226

Version Manager Users **227**

8 - DATA MODELLER 229

Overview **229**

Design Pad **231**
File Popup Menu 231
Key Popup Menu 233
Field Popup Menu 234
Relationship Popup Menu 235
Open Area Popup Menu 236

Data Pools **238**
Field Pool 238
Field Pool Popup Menu 238
Key Pool 240
Key Pool Popup Menu 241
Type Pool 241
Type Pool Popup Menu 242

The Data Modeller Menus **243**
File Menu 243
Edit Menu 244
Dictionary Menu 244
View Menu 245
Zoom Menu 247
Tools Menu 247
Print Menu 251
Setup Menu 252
Help Menu 253

User Preferences **254**
Color Settings 254
Other Settings 255
Advanced Settings 256
Cascade Edits 257
Database Drivers 257
File Aliases **258**
File Relationships **259**
SQL Script Generation **261**
Large Dictionaries **263**
Duplicate Idents **264**

PART III

WISE FOR CLARION GETTING STARTED GUIDE 265

9 - WISE INTRODUCTION AND SETUP **267**
Introduction **267**
Wise for Clarion Functions 268
About Wise Solutions, Inc. 269
Ease of Use 269
System Requirements **270**
Hardware 270
Operating System 270
Installing the Wise Software **271**
Running the Setup Program 271
Launching the Software 271
Registering Your Software 272
Getting Help **273**
Available Support Methods 273
Using Wise for Clarion Documentation & Online Help 273
Overview of Wise for Clarion **275**
Building Installs – Overview 275

Using the Installation Expert **278**
Getting Prepared ... 278
Opening the Installation Expert ... 279
Saving the Installation File ... 279
Navigating the Step Buttons ... 279

Files and Components **281**
Assigning Files to a Destination Folder ... 281
Removing Files from the Destination ... 283
File Details ... 283
Selecting Components ... 284
Creating Components ... 285
Uninstall ... 286

Assigning Program Groups & Icons **288**
Assigning Icons ... 288
Making Registry Entries ... 290
Creating Registry Entries from Scratch ... 291
Adding INI Entries ... 292
Autoexec.bat ... 294

User System Checks **297**
Configuration Checks ... 297

Wizard Appearance **299**
Entering the Software Title & Default Program Directory ... 299
Selecting Dialogs and Installation Options ... 300
Default Dialogs ... 301
Editing Dialog Templates ... 302
Screen ... 304
Billboards ... 305
Multiple Language Support ... 308

Choosing Media and Global Options **310**
Media ... 310
Choosing Global Options ... 311

Compiling & Testing Your Installation **314**
Compiling ... 314
Testing Your Installation Script ... 314
Running Your Installation Script ... 315
Distributing Your Installation Program ... 316
Application and Watch Wizard ... 318

Customizing the Environment **320**
Customizing the Dialog Templates 320
Wise Installation Script Template **321**
How to Use 321
INDEX **323**

PART I

BUSINESS MATH LIBRARY

1 - INTRODUCTION

Documentation Conventions

The documentation uses the typeface and keyboard conventions which appear below.

Typeface Conventions

Italics	Indicates what to type at the keyboard, such as *Type This*.
SMALL CAPS	Indicates keystrokes to enter at the keyboard, such as ENTER or ESCAPE.
ALL CAPS	Indicates Clarion Language keywords. For more information on these keywords, see the Language Reference.
Boldface	Indicates commands or options from a pull down menu or text in a dialog window. Note: this style also uses a different typeface to match the helvetica bold face which Windows uses as the system font.
`LETTER GOTHIC`	Used for diagrams, source code listings, to annotate examples, and for examples of the usage of source statements.

Tip: (or Note:)— information that is not immediately evident from the topic explanation.

Indicates critical information. If you read nothing else in this chapter, please read this.

Keyboard Conventions

F1	Indicates a keystroke. Press and release the F1 key.
ALT+X	Indicates a combination of keystrokes. Hold down the ALT key and press the X key, then release both keys.

Usage Conventions and Symbols

Symbols are used in the syntax diagrams as follows:

Symbol	Meaning
[ ]	Brackets enclose an optional (not required) attribute or parameter.
( )	Parentheses enclose a parameter list.
| |	Vertical lines enclose parameter lists, where one, but only one, of the parameters is allowed.

Coding example conventions used throughout this manual:

```
IF NOT SomeDate              !IF and NOT are keywords
  SomeDate = TODAY()       !SomeDate is a data name
END                          !TODAY and END are keywords
```

CLARION LANGUAGE KEYWORDS	Any word in “All Caps” is a Clarion Language keyword
DataNames	Use mixed case with caps for readability
Comments	Predominantly lower case

The purpose of these conventions is to make the code examples readable and clear.

Reference Item Format

Each procedure referenced in this manual is printed in UPPER CASE letters and are documented with a syntax diagram, a detailed description, and source code examples.

The documentation format used in this book is illustrated in the syntax diagram on the following page.

KEYWORD (short description of intended use)

[label] **KEYWORD(** | *parameter1* | [*parameter2*] **)**
| *alternate* |
| *parameter* |
| *list* |

KEYWORD A brief statement of what the KEYWORD does.

parameter1 A complete description of parameter1, along with how it relates to parameter2 and the KEYWORD.

parameter2 A complete description of parameter2, along with how it relates to parameter1 and the KEYWORD. Because it is enclosed in brackets, [ ], it is optional, and may be omitted.

alternate parameter list
A complete description of alternates to parameter1, along with how they relate to parameter2 and the KEYWORD.

A concise description of what the **KEYWORD** does.

Return Data Type: The data type returned if the KEYWORD is a procedure.

Internal Formulas: The formulas used within the procedure or procedure.

Example:

```
FieldOne = FieldTwo + FieldThree        !Source code example
FieldThree = KEYWORD(FieldOne,FieldTwo)   !Comments follow the “!”
```

About this Part

Introduction—this chapter—provides a general description of the Clarion Business Math Library, and the procedure for installing it.

The *Finance Library* chapter provides a detailed discussion of the specific usage of each procedure in the Finance Library.

The *Business Statistics Library* chapter provides a detailed discussion of the specific usage of each procedure in the Business Statistics Library.

The *Business Math Templates* chapter tells you how to use the code templates provided with the Business Math Library.

About the Business Math Library

The Clarion Business Math Library contains two components, the Finance Library and the Business Statistics Library.

The Finance Library contains procedures that allow you to perform financial operations including amortization, cash flow analysis, interest calculations, and time value of money computations.

The Business Statistics Library contains procedures that allow you to perform statistical operations on numeric data sets including such computations as mean, median, mode, variance, standard deviation, linear regression, etc.

Each component comes with templates and prototypes which make the business math procedures easy to implement within your Clarion applications.

In addition, an example application and dictionary is provided. The example application demonstrates practical implementations of each of the procedures in the Business Math Library.

Installing the Business Math Library

Simply run the setup.exe program on your installation disk. Be sure to read the ReadMe file for the latest information on the installation process.

Business Math Library Template Support

Global Extension Templates automate the process of adding required MAP and Project System entries to an application. Complete Code template support for all of the Business Math procedures are provided. In addition, the Code templates self-document the use of the library operations.

Registering the Template Classes

To use the Finance Code templates, the ..\TEMPLATE\FINANCE.TPL file must be registered in the Clarion Template Registry. FINANCE.TPL contains the Finance Template Class. To use the Business Statistics Code Templates, the ..\TEMPLATE\STATISTC.TPL file must be registered in the Template Registry. STATISTC.TPL contains the Statistics Template Class.

These template classes are automatically registered when you install the Business Math Library with the setup program. See the *Business Math Templates* chapter for information on registering the templates manually.

Adding Extension Templates to Your Application

Once the template classes are registered, you should add the business math extensions to your application's Global Extensions. This enables access to the associated Code templates from any embed point throughout the application. It also adds the appropriate procedure prototypes to the application's MAP and adds appropriate library entries to the application's project. See the *Business Math Templates* chapter for information on adding the global extensions to your application.

Ship List DLLs

One of the following DLLs needs to be shipped with your application whenever the 'Standalone (C5RUNx.DLL)' *Run-time Library* is selected in the application's project. The appropriate DLL is based upon the application's *Target OS* (ie. 16 or 32 bit).

C5FIN.DLL - Finance Library (16-bit Standalone DLL).
C5FINx.DLL - Finance Library (32-bit Standalone DLL).

Note: If 'Local' *Run-time Library* is selected in a project then no DLLs associated with the Finance Library need to be shipped.

Business Math Library Hand-Code Support

While the #EXTENSION templates automatically handle project system and prototytpe entries for your Application Generator projects, you must handle these entries for hand-coded Clarion programs. First, include appropriate procedure prototypes in your program's MAP structure. Second, add the appropriate library entries to your program's project.

The prototypes for the Finance Library procedures are in the CWFINPRO.CLW file. The prototypes for the Business Statistics Library procedures are in the CWSTATPR.CLW file. Both files are installed in the ..\LIBSRC subdirectory.

Each of these files should be included in your program's MAP as follows:

```
MAP
  ...                       ! other map entries
  INCLUDE('CWFINPRO.CLW') ! include Finance Library prototypes
  INCLUDE('CWSTATPR.CLW') ! include Business Statistics Library prototypes
  ...                       ! other map entries
END
```

The Finance Library file set contains the following files:

C5FIN.LIB	16-bit Standalone (Deploy with C5FIN.DLL)
C5FINL.LIB	16-bit Local Link
C5FINx.LIB	32-bit Standalone (Deploy with C5FINx.DLL)
C5FINxL.LIB	32-bit Local Link

The Business Statistics Library file set contains the following files:

C5STAT.LIB	16-bit Standalone (Deploy with C5STAT.DLL)
C5STATL.LIB	16-bit Local Link
C5STATx.LIB	32-bit Standalone (Deploy with C5STATx.DLL)
C5STATxL.LIB	32-bit Local Link

The .LIB files are installed in the ..\LIB subdirectory; the .DLL files are installed in the ..\BIN subdirectory.

The BUSMATH.PR file (in the ..\LIBSRC subdirectory) automatically includes the correct business math libraries in your hand-coded project when added to your hand-coded project as a *Project to include*. The BUSMATH.PR project includes the appropriate Business Math Libraries based on the parent project's Target OS and Run-Time Library settings.

Using the Project Editor, the BUSMATH.PR file should be added to a program's project under the *Projects to include* section as follows:

Including the BUSMATH.PR file in a Project

1. Load a project into the Project Editor.
2. Highlight *Projects to include*.
3. Press the **Add File** button.
4. Select the BUSMATH.PR file from the ..\LIBSRC subdirectory.

2 - FINANCE LIBRARY

Overview

This chapter provides a discussion of the purpose, components, and conventions of the Finance Library, as well as a discussion and an example of the specific usage of each procedure in the Finance Library.

The Finance Library contains procedures that allow you to perform financial operations including amortization, cash flow analysis, interest calculations, and time value of money computations.

Finance Library Components

Provided with the Finance Library are prototypes, code templates, and examples that simplify the implementation of the Finance procedures into your application or project. Follow the procedures described in *Installing the Business Math Library* in Chapter 1.

The Finance Library components (files) are:

...\LIB\C5FINL.LIB
: 16-bit Finance objects that link into your executable.

...\BIN\C5FIN.DLL
: 16-bit Finance procedures.

...\LIB\C5FIN.LIB
: 16-bit Finance stub file for resolving link references to procedures in C5FIN.DLL.

...\LIB\C5FINxL.LIB
: 32-bit Finance objects that link into your executable.

...\BIN\C5FINx.DLL
: 32-bit Finance procedures.

...\LIB\C5FINx.LIB
: 32-bit Finance stub file for resolving link references to procedures in C5FINx.DLL.

...\TEMPLATE\FINANCE.TPL
Finance Library templates. The templates self-document the use of the library procedures. The templates must be registered before they can be used in your application. See *Installing the Business Math Libraries* in Chapter 1.

...\LIBSRC\CWFINPRO.CLW
Prototypes for the Finance procedures.

Finance Library Conventions

Parameters

Even though most of the Finance procedure parameters are declared as data types other than DECIMAL (see CWFINPRO.CLW), the parameter values are immediately assigned internally to DECIMAL variables of the appropriate magnitude. All calculations within the Finace Library use DECIMAL variables, most of which are defined as DECIMAL(31,15) to provide an equal bias to both sides of the decimal point.

All Finance procedure *return values* are DECIMAL values passed back through a REAL (return declaration) resulting in no loss of precision if directly assigned to a DECIMAL variable. Whereever return parameters are passed into the procedures (AMORTIZE, IRR, NPV, etc.) they are declared as DECIMAL.

This prototype strategy is applied to enable flexible use of the Finance operations while maintaining the desired feature of using Binary Coded Decimal (Base 10) math (also called BCD math).

Rounding

With procedures that return REALs, we recommend that you round off these returned values to the desired magnitude. In Clarion, REAL values can be rounded several ways.

1. Use DECIMAL variables declared to the required precision when performing finance operations, and rounding is automatic.
2. Use Clarion's ROUND procedure.
3. Use Clarion’s built in data type conversion.
4. Move the REAL to a generic string variable, then format the string. Again, rounding occurs automatically.

Sign Conventions

In the time value of money procedures, the amortization procedure, and the cash flow analysis procedures that follow, plus signs (+) or minus signs (-) are required to indicate payment receipts and payment outlays respectively.

The need for a sign convention arises because the procedures are used for both loan repayment and savings scenarios. When considering this, the following rules apply:

> If the present value is less than the future value, payments are positive, and conversely, if the present value is greater than the future value, payments are negative.

Of course, how you display these values is up to you. For example,in the sample program for amortization, the loan amounts and payment amounts are displayed as entered—with prepended plus (+) or minus (-) signs. You may want to display negative values in red, enclosed in parentheses, or with no indication of the sign at all.

Procedures

AMORTIZE (amortize loan for specific number of payments)

AMORTIZE(*balance,rate,payment,totalpayments,principal,interest,endbalance*)

AMORTIZE	Calculates principal, interest, and remaining balance for a payment or payments.
balance	A numeric constant or variable containing the loan balance.
rate	A numeric constant or variable containing the *periodic interest rate* applied for a single period.
payment	A numeric constant or variable containing the desired payment (a negative number, see *Sign Conventions* above).
totalpayments	A numeric constant or variable containing the number of payments to amortize.
principal	The label of a DECIMAL variable to receive the portion of the payment(s) applied to pay back the loan (a negative number, see *Sign Conventions* above).
interest	The label of a DECIMAL variable to receive the portion of the payment(s) applied towards loan interest (a negative number, see *Sign Conventions* above).
endbalance	The label of a DECIMAL variable to receive the remaining loan balance.

The **AMORTIZE** procedure shows precisely which portion of a loan payment, or payments, constitutes interest and which portion constitutes repayment of the principal amount borrowed. The computed amounts are based upon a loan balance (*balance*), a periodic interest rate (*rate*), the payment amount (*payment*) and the number of payments (*totalpayments*). The remaining balance (*endbalance*) is also calculated.

Periodic rate may be calculated as follows:

```
PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)
```

NOTE: The return parameters principal, interest, and endbalance must be DECIMAL values (passed by value).

Internal Formulas:

```
PRINCIPAL = payment + (balance * rate)
INTEREST = payment - principal
ENDINGBALANCE = balance + principal
```

Example:

```
Principal       DECIMAL(18,2)
Interest        DECIMAL(18,2)
EndingBalance   DECIMAL(18,2)

CODE

BeginningBalance = LoanAmount                          !Set first beginning balance
Period# = 1                                            !Begin with the first period
LOOP Ptr# = Period# TO TotalPeriods                    !Loop through the periods
  AMORTIZE(BeginningBalance,MonthlyRate,Payment,1, |
  Principal,InterestAmount,EndingBalance)              !Amortize 1 payment
  Q:Balance = BeginningBalance                        !Show the beginning balance
  Q:Payment = Payment * (-1)                          !..the payment amount
  Q:Principal = Principal * (-1)                      !..amount applied to principal
  Q:Interest = InterestAmount * (-1)                  !..amount applied to interest
  Q:NewBalance = EndingBalance                        !...ending balance
  IF Ptr# = TotalPeriods|                              !If last period
    AND EndingBalance < 0                              !and balance went negative
    Q:Principal += EndingBalance                      !adjust principal downard
    Q:Payment += EndingBalance                        !and also the payment
    Q:NewBalance = 0.0                                !and make the balance zero.
  END
  EndingBalance = Q:NewBalance                        !Save the ending balance
  ADD(AmortizeQue)                                     !Add all period values to Q
  BeginningBalance = EndingBalance                    !Make a new beginning balance
  TotalInterest += Q:Interest                          !Add up total interest
END                                                    !End loop
```

APR (annual percentage rate)

APR(*rate,periods*)

APR	Returns the effective annual interest rate.
rate	A numeric constant or variable containing the *contracted* interest rate.
periods	A numeric constant or variable containing the number of compounding periods per year.

The **APR** procedure determines the effective annual rate of interest based upon the contracted interest rate (*rate*) and the number of compounding periods (*periods*) per year. For example, periods = 2 results in semi-annual compounding. The contracted interest rate is a "non-compounded annual interest rate."

Return DataType: DECIMAL

Internal Formulas:

$$APR = \left(1 + \frac{rate}{periods}^{periods}\right) - 1$$

Example:

```
PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)! Setup Variables
RealRate     = InterestRate / 100
AnnualRate = APR(RealRate,PeriodsPerYear)                  ! Call APR
AnnualRate *= 100                                          ! Normalize Results
```

COMPINT (compound interest)

COMPINT(*principal,rate,periods*)

COMPINT	Computes total compounded interest plus principal.
principal	A numeric constant or variable containing the beginning balance, initial deposit, or loan.
rate	A numeric constant or variable containing the *applied interest rate* for the given time frame.
periods	A numeric constant or variable containing the number of compounding periods per year.

The **COMPINT** procedure computes total interest based on a principal amount (*principal*), an applied interest rate (*rate*), plus the number of compounding periods (*periods*). The computed amount includes the original principal plus the compound interest earned. *Periods* specifies the number of compounding periods. For example, periods = 2 results in semi-annual compounding.

Applied interest rate may be calculated as follows:

```
PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)
AppliedRate   = PeriodicRate * TotalPeriods
```

Return Data Type: DECIMAL

Internal Formulas:

$$\text{COMPOUND INTEREST} = \text{Principal} * \left(1 + \frac{\text{rate}}{\text{periods}}\right)^{\text{periods}}$$

Example:

```
CASE FIELD()
OF ?OK
  CASE EVENT()
  OF EVENT:Accepted
    PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)  ! Setup Variables
    ActualRate  = PeriodicRate * TotalPeriods
    CompoundInterest = COMPINT(Principal,ActualRate,TotalPeriods) ! Call COMPINT
    DO SyncWindow
  END
```

CONTINT (continuous compounding interest)

CONTINT(*principal*,*rate*)

CONTINT	Computes total continuously compounded interest.
principal	A numeric constant or variable containing the beginning balance, initial deposit, or loan.
rate	A numeric constant or variable containing the *applied interest rate* for the given time frame.

CONTINT computes total continuously compounded interest based on a principal amount (*principal*) and an applied interest rate (*rate*). The returned amount includes the original principal plus the interest earned. Applied interest rate may be calculated as follows:

```
PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)
AppliedRate  = PeriodicRate * TotalPeriods
```

Return Data Type: DECIMAL

Internal Formulas:

$$\text{CONTINUOUS COMPOUND INTEREST} = \text{Principal} * e^{\text{rate}}$$

where e = base of natural logarithm (2.71828...).

Example:

```
PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)! Setup Variables
ActualRate  = PeriodicRate * TotalPeriods
ContinuousInterestAmount = CONTINT(Principal,ActualRate)  ! Call CONTINT
```

DAYS360 (days difference based on 360-day year)

DAYS360(*startdate*,*enddate*)

DAYS360	Computes the difference in days, between two given dates.
startdate	A numeric constant or variable containing the beginning date.
enddate	A numeric constant or variable containing the ending date.

DAYS360 determines the number of days difference between a beginning date (*startdate*) and an ending date (*enddate*), based on a 360-day year (30 day month). Both date parameters MUST contain Clarion standard date values.

Return Data Type: LONG

Internal Formulas:

```
DAYS DIFFERENCE = ending date - beginning date
```

where:

```
ending date = 360(year) + 30(month) + z
z = 30 if ending date = 31 and beginning date > 29
z = ending date if ending date <> 31 and beginning date < 29
```

and:

```
beginning date = 360(year) + 30(month) + z
z = 30 if beginning date = 31
z = beginning date if beginning date <> 31
```

Example:

```
DaysDifference = DAYS360(StartDate,EndDate)
```

FV (future value)

FV(*presentvalue*,*periods*,*rate*,*payment*)

FV	Computes the future value of an investment plus an income stream.
presentvalue	A numeric constant or variable containing the present value of the investment.
periods	A numeric constant or variable containing the number of periods in which a cash flow occurred.
rate	A numeric constant or variable containing the *periodic* interest rate.
payment	A numeric constant or variable containing the periodic payment.

FV and **PREFV** determine the future value of an initial amount (*presentvalue*) plus an income stream. The income stream is defined as the total number of periods (*periods*), the periodic interest rate (*rate*), and a payment amount (*payment*). If payments occur at the beginning of each period, use the PREFV procedure, which takes into account the added interest earned on each period's payment.

Periodic rate may be calculated as follows:

```
PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)
```

Return Data Type: DECIMAL

Internal Formulas:

$$\text{FV} = \text{PresentValue}(1 + \text{rate})^{\text{periods}} + \text{payment}\,\frac{(1 + \text{rate})^{\text{int(periods)}} - 1}{\text{rate}}$$

where int(periods) is the integer portion of the *periods* parameter.

Example:

```
PeriodicRate = AnnualRate / (PeriodsPerYear * 100)
IF TimeOfPayment = 'Beginning of Periods'
  FutureValue = PREFV(PresentValue,TotalPeriods,PeriodicRate,Payment)
ELSE
  FutureValue = FV(PresentValue,TotalPeriods,PeriodicRate,Payment)
END
```

PREFV (future value with prepayment)

PREFV(*presentvalue*,*periods*,*rate*,*payment*)

PREFV	Computes the future value of an investment plus an income stream.
presentvalue	A numeric constant or variable containing the present value of the investment.
periods	A numeric constant or variable containing the number of periods in which a cash flow occurred.
rate	A numeric constant or variable containing the *periodic* interest rate.
payment	A numeric constant or variable containing the periodic payment.

FV and **PREFV** determine the future value of an initial amount (*presentvalue*) plus an income stream. The income stream is defined as the total number of periods (*periods*), the periodic interest rate (*rate*), and a payment amount (*payment*). If payments occur at the beginning of each period then use the PREFV procedure, which takes into account the added interest earned on each period's payment.

Periodic rate may be calculated as follows:

```
PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)
```

Return Data Type: DECIMAL

Internal Formulas:

$$\text{PREFV} = \text{PresentValue}(1 + \text{rate})^{\text{periods}} + \text{payment}(1+\text{rate})\,\frac{(1 + \text{rate})^{\text{int(periods)}} - 1}{\text{rate}}$$

where int(periods) is the integer portion of the *periods* parameter.

Example:

```
PeriodicRate = AnnualRate / (PeriodsPerYear * 100)
IF TimeOfPayment = 'Beginning of Periods'
  FutureValue = PREFV(PresentValue,TotalPeriods,PeriodicRate,Payment)
ELSE
  FutureValue = FV(PresentValue,TotalPeriods,PeriodicRate,Payment)
END
```

IRR (internal rate of return)

IRR(*investment,cashflows,periods,rate*)

IRR	Computes the internal rate of return on an investment.
investment	A numeric constant or variable containing the initial cost of the investment (a negative number).
cashflows[]	A single dimensioned DECIMAL array containing the amounts of money paid out and received during discrete accounting periods.
periods[]	A single dimensioned DECIMAL array containing period numbers identifying the discrete accounting periods in which cash flows occurred.
rate	A numeric constant or variable containing the desired *periodic* rate of return.

IRR determines the rate of return on an investment (*investment*). The result is computed by determining the rate that equates the present value of future cash flows to the cost of the initial investment. The *cashflows* parameter is an array of money paid out and received during the course of the investment. The *periods* parameter is an array which contains the period number in which a corresponding cash flow occurred. The desired periodic rate of return (*rate*) for the investment is also specified. *Investment* should contain a negative number (a cost).

Periodic rate may be calculated as follows:

```
PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)
```

> ***Note:*** ***Cashflows*** **and** ***periods*** **MUST both be DECIMAL arrays of single dimension and equal size. The last element in the** ***periods*** **array MUST contain a zero to terminate the IRR analysis.**

Return Data Type: DECIMAL

Internal Formulas:

$$0 = \frac{\text{cash flows[1]}}{(1 + \text{rate})^{\text{periods[1]}}} + \frac{\text{cash flows[2]}}{(1 + \text{rate})^{\text{periods[2]}}} + \ldots + \frac{\text{cash flows[n]}}{(1 + \text{rate})^{\text{periods[n]}}} + \text{investment}$$

The IRR procedure performs binary search iterations to home in on the return value. If more than 50 such iterations are required, a value of zero is returned.

Example:

```
CashFlows           DECIMAL(18,2),DIM(1000)
CashFlowPeriods     DECIMAL(4),DIM(1000)
InvestmentAmount    DECIMAL(18,2)
DesiredInterestRate DECIMAL(31,15)
Return              DECIMAL(10,6)

InvestmentTransactions  FILE,DRIVER('TOPSPEED'),|
                        NAME('busmath\!InvestmentTransactions'),PRE(INV),CREATE,THREAD
KeyIdPeriodNumber         KEY(INVTRN:Id,INVTRN:PeriodNumber),NOCASE,PRIMARY
Notes                     MEMO(1024)
Record                    RECORD,PRE()
Id                          DECIMAL(8)
PeriodNumber                DECIMAL(8)
Date                        ULONG
Type                        STRING(20)
Amount                      DECIMAL(16,2)
                          END
                        END

CODE

    CLEAR(CashFlows)                                  !initialize queue
    CLEAR(CashFlowPeriods)                            !initialize queue
    CLEAR(InvestmentAmount)
    CLEAR(TRANSACTION:RECORD)                         !initialize record buffer
    DesiredInterestRate = INV:NominalReturnRate / (100 * INV:PeriodsPerYear)
    transcnt# = 0
    TRANSACTION:Id = INV:Id                           !prime record buffer
    TRANSACTION:PeriodNumber = 0
    SET(TRANSACTION:KeyIdPeriodNumber,TRANSACTION:KeyIdPeriodNumber) !position file
    LOOP                                              !loop for all transactions
      NEXT(InvestmentTransactions)                    !read next record
      IF ERRORCODE() OR TRANSACTION:Id NOT = INV:Id   !if no more transactions
        BREAK                                         !break from loop
      ELSE                                            !else process transaction
        transcnt# += 1
        CashFlows[transcnt#] = TRANSACTION:Amount
        CashFlowPeriods[transcnt#] = TRANSACTION:PeriodNumber
      END                                             !endelse process transaction
    END                                               !endloop all transactions
    Return = IRR(InvestmentAmount,CashFlows,CashFlowPeriods,DesiredInterestRate)
    Return *= (100 * INV:PeriodsPerYear)            !normalize IRR to percent
```

NPV (net present value)

NPV(*investment,cashflows,periods,rate*)

NPV	Computes net present value of an investment.
investment	A numeric constant or variable containing the initial cost of the investment (a negative number).
cashflows[]	A single dimensioned DECIMAL array containing the amounts of money paid out and received during discrete accounting periods.
periods[]	A single dimensioned DECIMAL array containing period numbers identifying the discrete accounting periods in which cash flows occurred.
rate	A numeric constant or variable containing the desired *periodic* rate of return.

NPV determines the viability of an investment proposal by calculating the present value of future returns, discounted at the marginal cost of capital minus the cost of the investment (*investment*). The *cashflows* parameter is an array of money paid out and received during the course of the investment. The *periods* parameter is an array which contains the period number in which a corresponding cash flow occurred. The desired periodic rate of return (*rate*) for the investment is also specified. *Investment* should contain a negative number (a cost).

Periodic rate may be calculated as follows:

```
PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)
```

> ***Note:*** ***Cashflows*** **and** ***periods*** **MUST both be DECIMAL arrays of single dimension and equal size. The last element in the** ***periods*** **array MUST contain a zero to terminate the NPV analysis.**

Return Data Type: DECIMAL

Internal Formulas:

$$NPV = \frac{\text{cash flows[1]}}{(1 + \text{rate})^{\text{periods[1]}}} + \frac{\text{cash flows[2]}}{(1 + \text{rate})^{\text{periods[2]}}} + \ldots + \frac{\text{cash flows[n]}}{(1 + \text{rate})^{\text{periods[n]}}} + \text{investment}$$

Example:

```
CashFlows           DECIMAL(18,2),DIM(1000)
CashFlowPeriods     DECIMAL(4),DIM(1000)
InvestmentAmount    DECIMAL(18,2)
DesiredInterestRate DECIMAL(31,15)
NetValue            DECIMAL(18,2)

InvestmentTransactions  FILE,DRIVER('TOPSPEED'),|
                        NAME('busmath\!InvestmentTransactions'),PRE(INV),CREATE,THREAD
KeyIdPeriodNumber         KEY(INVTRN:Id,INVTRN:PeriodNumber),NOCASE,PRIMARY
Notes                     MEMO(1024)
Record                    RECORD,PRE()
Id                          DECIMAL(8)
PeriodNumber                DECIMAL(8)
Date                        ULONG
Type                        STRING(20)
Amount                      DECIMAL(16,2)
                          END
                        END

CODE

    CLEAR(CashFlows)                                  !initialize queue
    CLEAR(CashFlowPeriods)                            !initialize queue
    CLEAR(InvestmentAmount)
    CLEAR(TRANSACTION:RECORD)                         !initialize record buffer
    DesiredInterestRate = INV:NominalReturnRate / (100 * INV:PeriodsPerYear)
    transcnt# = 0
    TRANSACTION:Id = INV:Id                           !prime record buffer
    TRANSACTION:PeriodNumber = 0
    SET(TRANSACTION:KeyIdPeriodNumber,TRANSACTION:KeyIdPeriodNumber) !position file
    LOOP                                              !loop for all transactions
      NEXT(InvestmentTransactions)                    !read next record
      IF ERRORCODE() OR TRANSACTION:Id NOT = INV:Id   !if no more transactions
        BREAK                                         !break from loop
      ELSE                                            !else process transaction
        transcnt# += 1
        CashFlows[transcnt#] = TRANSACTION:Amount
        CashFlowPeriods[transcnt#] = TRANSACTION:PeriodNumber
      END                                             !endelse process transaction
    END                                               !endloop all transactions
    NetValue = NPV(InvestmentAmount,CashFlows,CashFlowPeriods,DesiredInterestRate)
```

PERS (periods of annuity)

PERS(*presentvalue,rate,payment,futurevalue*)

PERS	Computes the number of periods required to reach a targeted future value.
presentvalue	A numeric constant or variable containing the present value of the investment.
rate	A numeric constant or variable containing the *periodic* rate of return.
payment	A numeric constant or variable containing the periodic payment.
futurevalue	A numeric constant or variable containing the amount of the desired or targeted future value of the investment.

PERS determines the number of periods required to reach a desired amount (*futurevalue*) based upon a starting amount (*presentvalue*), a periodic interest rate (*rate*), and a payment amount (*payment*). If payments occur at the beginning of each period, use the PREPERS procedure, which takes into account the added interest earned on each period's payment.

Periodic rate may be calculated as follows:

```
PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)
```

> **Note: If the present value is less than the future value (annuities), payments are positive, and conversely, if the present value is greater than the future value (loans), payments are negative.**

Return Data Type: DECIMAL

Internal Formulas:

$$0 = \text{presentvalue}(1+\text{rate})^{\text{frac(periods)}} + \text{payment}\,\frac{1-(1+\text{rate})^{\text{int(-periods)}}}{\text{rate}} - \text{futurevalue}(1+\text{rate})^{\text{int(-periods)}}$$

where frac(periods) is the fractional portion of the *periods* parameter and where int(periods) is the integer portion of the *periods* parameter.

If the *payment* parameter is omitted or 0, then

$$\text{PERS} = \frac{\log(\text{futurevalue}/\text{presentvalue})}{\log(1 + \text{rate})}$$

If the *futurevalue* parameter is omitted or 0, then

$$\text{PERS} = \frac{\log(1 - (\text{rate} * \frac{\text{presentvalue}}{-\text{payment}}))}{\log(1 + \text{rate})}$$

If the *presentvalue* parameter is omitted or 0, then

$$PERS = \frac{\log\left(rate * \frac{presentvalue}{payment}\right)}{\log(1 + rate)}$$

The PERS procedure performs binary search iterations to home in on the periods value. If more than 50 such iterations are required, a value of zero is returned.

Example:

```
PeriodicRate = AnnualRate / (PeriodsPerYear * 100)
IF TimeOfPayment = 'Beginning of Periods'
  TotalPeriods = PREPERS(PresentValue,PeriodicRate,Payment,FutureValue)
ELSE
  TotalPeriods = PERS(PresentValue,PeriodicRate,Payment,FutureValue)
END
```

PREPERS (periods of annuity with prepayment)

PREPERS(*presentvalue,rate,payment,futurevalue*)

PREPERS	Computes the number of periods required to reach a targeted future value.
presentvalue	A numeric constant or variable containing the present value of the investment.
rate	A numeric constant or variable containing the *periodic* rate of return.
payment	A numeric constant or variable containing the periodic payment.
futurevalue	A numeric constant or variable containing the amount of the desired or targeted future value of the investment.

PREPERS determines the number of periods required to reach a desired amount (*futurevalue*) based upon a starting amount (*presentvalue*), a periodic interest rate (*rate*), and a payment amount (*payment*). If payments occur at the end of each period, use the PERS procedure, which calculates interest accordingly.

Periodic rate may be calculated as follows:

```
PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)
```

Note: If the present value is less than the future value (annuities), payments are positive, and conversely, if the present value is greater than the future value (loans), payments are negative.

Return Data Type: DECIMAL

Internal Formulas:

$$0 = \text{presentvalue}(1+\text{rate})^{\text{frac(periods)}} + \text{payment}(1+\text{rate})^{\text{frac(periods)}}\frac{1-(1+\text{rate})^{\text{int(-periods)}}}{\text{rate}} - \text{futurevalue}(1+\text{rate})^{\text{int(-periods)}}$$

where frac(periods) is the fractional portion of the *periods* parameter and where int(periods) is the integer portion of the *periods* parameter.

If the *payment* parameter is omitted or 0, then

$$\text{PREPERS} = \frac{\log(\text{futurevalue}/\text{presentvalue})}{\log(1 + \text{rate})}$$

If the *futurevalue* parameter is omitted or 0, then

$$\text{PREPERS} = \frac{\log\left(1 - \left(\frac{\text{rate}}{(1+\text{rate})} * \frac{\text{presentvalue}}{-\text{payment}}\right)\right)}{\log(1 + \text{rate})}$$

If the *presentvalue* parameter is omitted or 0, then

$$\text{PREPERS} = \frac{\log\left(\frac{\text{rate}}{(1+\text{rate})}\right) * \left(\frac{\text{futurevalue}}{\text{payment}} + 1\right)}{\log(1 + \text{rate})}$$

The PREPERS procedure performs binary search iterations to home in on the periods value. If more than 50 such iterations are required, a value of zero is returned.

Example:

```
PeriodicRate = AnnualRate / (PeriodsPerYear * 100)
IF TimeOfPayment = 'Beginning of Periods'
  TotalPeriods = PREPERS(PresentValue,PeriodicRate,Payment,FutureValue)
ELSE
  TotalPeriods = PERS(PresentValue,PeriodicRate,Payment,FutureValue)
END
```

PMT (payment of annuity)

PMT(*presentvalue,periods,rate,futurevalue*)

PMT	Computes the payment required to reach a targeted future value.
presentvalue	A numeric constant or variable containing the amount of the present value of the investment.
periods	A numeric constant or variable containing the number of periods in which a payment is made.
rate	A numeric constant or variable containing the *periodic* rate of return.
futurevalue	A numeric constant or variable containing the amount of the desired or targeted future value of the investment.

PMT determines the payment required to reach a desired amount (*futurevalue*) based upon a starting amount (*presentvalue*), a total number of periods (*periods*), and a periodic interest rate (*rate*). If payments occur at the beginning of each period then use the PREPMT procedure, which takes into account the added interest earned on each period's payment.

Periodic rate may be calculated as follows:

```
PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)
```

Note: If the present value is less than the future value (annuities), payments are positive, and conversely, if the present value is greater than the future value (loans), payments are negative.

Return Data Type: DECIMAL

Internal Formulas:

$$PMT = \frac{futurevalue(1+rate)^{int(-periods)} - presentvalue(1+rate)^{frac(periods)}}{\frac{1-(1+rate)^{int(-periods)}}{rate}}$$

where frac(periods) is the fractional portion of the *periods* parameter and where int(periods) is the integer portion of the *periods* parameter.

Example:

```
PeriodicRate = AnnualRate / (PeriodsPerYear * 100)
IF TimeOfPayment = 'Beginning of Periods'
  Payment = PREPMT(PresentValue,TotalPeriods,PeriodicRate,FutureValue)
ELSE
  Payment = PMT(PresentValue,TotalPeriods,PeriodicRate,FutureValue)
END
```

PREPMT (payment of annuity with prepayment)

PREPMT(*presentvalue,periods,rate,futurevalue*)

PREPMT	Computes the payment required to reach a targeted future value.
presentvalue	A numeric constant or variable containing the amount of the present value of the investment.
periods	A numeric constant or variable containing the number of periods in which a payment is made.
rate	A numeric constant or variable containing the *periodic* rate of return.
futurevalue	A numeric constant or variable containing the amount of the desired or targeted future value of the investment.

PREPMT determines the payment required to reach a desired amount (*futurevalue*) based upon a starting amount (*presentvalue*), a total number of periods (*periods*), and a periodic interest rate (*rate*). If payments occur at the end of each period then use the PMT procedure, which calculates interest accordingly.

Periodic rate may be calculated as follows:

```
PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)
```

Note: If the present value is less than the future value (annuities), payments are positive, and conversely, if the present value is greater than the future value (loans), payments are negative.

Return Data Type: DECIMAL

Internal Formulas:

$$\text{PREPMT} = \frac{\text{futurevalue}(1+\text{rate})^{\text{int}(-\text{periods})} - \text{presentvalue}(1+\text{rate})^{\text{frac}(\text{periods})}}{(1+\text{rate})\,\frac{1-(1+\text{rate})^{\text{int}(-\text{periods})}}{\text{rate}}}$$

where frac(periods) is the fractional portion of the *periods* parameter and where int(periods) is the integer portion of the *periods* parameter.

Example:

```
PeriodicRate = AnnualRate / (PeriodsPerYear * 100)
IF TimeOfPayment = 'Beginning of Periods'
  Payment = PREPMT(PresentValue,TotalPeriods,PeriodicRate,FutureValue)
ELSE
  Payment = PMT(PresentValue,TotalPeriods,PeriodicRate,FutureValue)
END
```

PV (present value)

PV(*periods,rate,payment,futurevalue*)

PV	Computes the present value required to reach a targeted future value.
periods	A numeric constant or variable containing the number of periods in which a cash flow occurred.
rate	A numeric constant or variable containing the *periodic* rate of return.
payment	A numeric constant or variable containing the periodic payment amount.
futurevalue	A numeric constant or variable containing the amount of the desired or targeted future value of the investment.

PV determines the present value required today to reach a desired amount (*futurevalue*) based upon the total number of periods (*periods*), a periodic interest rate (*rate*), and a payment amount (*payment*). If payments occur at the beginning of each period then use the **PREPV** procedure, which takes into account the added interest earned on each period's payment.

Periodic rate may be calculated as follows:

```
PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)
```

Note: If the present value is less than the future value (annuities), payments are positive, and conversely, if the present value is greater than the future value (loans), payments are negative.

Return Data Type: DECIMAL

Internal Formulas:

$$PV = futurevalue(1+rate)^{-periods} - payment\frac{(1+rate)^{frac(-periods)} - (1+rate)^{-periods}}{rate}$$

where frac(periods) is the fractional portion of the *periods* parameter.

Example:

```
PeriodicRate = AnnualRate / (PeriodsPerYear * 100)
IF TimeOfPayment = 'Beginning of Periods'
  PresentValue = PREPV(TotalPeriods,PeriodicRate,Payment,FutureValue)
ELSE
  PresentValue = PV(TotalPeriods,PeriodicRate,Payment,FutureValue)
END
```

PREPV (present value with prepayment)

PREPV(*periods,rate,payment,futurevalue*)

PREPV	Computes the present value required to reach a targeted future value.
periods	A numeric constant or variable containing the number of periods in which a cash flow occurred.
rate	A numeric constant or variable containing the *periodic* rate of return.
payment	A numeric constant or variable containing the periodic payment amount.
futurevalue	A numeric constant or variable containing the amount of the desired or targeted future value of the investment.

PREPV determines the present value required today to reach a desired amount (*futurevalue*) based upon the total number of periods (*periods*), a periodic interest rate (*rate*), and a payment amount (*payment*). If payments occur at the end of each period then use the **PV** procedure, which calculates interest accordingly.

Periodic rate may be calculated as follows:

```
PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)
```

Note: If the present value is less than the future value (annuities), payments are positive, and conversely, if the present value is greater than the future value (loans), payments are negative.

Return Data Type: DECIMAL

Internal Formulas:

$$\text{PREPV} = \text{futurevalue}(1+\text{rate})^{-\text{periods}} - \text{payment}(1+\text{rate})\frac{(1+\text{rate})^{\text{frac}(-\text{periods})} - (1+\text{rate})^{-\text{periods}}}{\text{rate}}$$

where frac(periods) is the fractional portion of the *periods* parameter.

Example:

```
PeriodicRate = AnnualRate / (PeriodsPerYear * 100)
IF TimeOfPayment = 'Beginning of Periods'
  PresentValue = PREPV(TotalPeriods,PeriodicRate,Payment,FutureValue)
ELSE
  PresentValue = PV(TotalPeriods,PeriodicRate,Payment,FutureValue)
END
```

RATE (rate of annuity)

RATE(*presentvalue,periods,payment,futurevalue*)

RATE	Computes the periodic interest rate required to reach a targeted future value.
presentvalue	A numeric constant or variable containing the present value of the investment.
periods	A numeric constant or variable containing the number of periods in which a cash flow occurred.
payment	A numeric constant or variable containing the periodic payment amount.
futurevalue	A numeric constant or variable containing the amount of the desired or targeted future value of the investment.

RATE determines the periodic interest rate required to reach a desired amount (*futurevalue*) based upon a starting amount (*presentvalue*), the total number of periods (*periods*), and a payment amount (*payment*). If payments occur at the beginning of each period then use the PRERATE procedure, which takes into account the added interest earned on each period's payment.

> **Note: If the present value is less than the future value (annuities), payments are positive, and conversely, if the present value is greater than the future value (loans), payments are negative.**

Return Data Type: DECIMAL

Internal Formulas:

$$0=\text{presentvalue}(1+\text{rate})^{\text{frac(periods)}} + \text{payment}\ \frac{1-(1-\text{rate})^{\text{int(-periods)}}}{\text{rate}} - \text{futurevalue}(1+\text{rate})^{\text{int(-periods)}}$$

where frac(periods) is the fractional portion of the *periods* parameter and where int(periods) is the integer portion of the *periods* parameter.

If the *payment* parameter is omitted or 0, then

$$\text{RATE} = \left(\frac{\text{futurevalue}}{\text{presentvalue}}\right)^{1/\text{periods}} - 1$$

The RATE procedure performs binary search iterations to home in on the interest rate value. If more than 50 such iterations are required, RATE returns a value of zero.

Example:

```
IF TimeOfPayment = 'Beginning of Periods'
  AnnualRate = PRERATE(PresentValue,TotalPeriods,Payment,FutureValue)
  AnnualRate *= (PeriodsPerYear * 100)
ELSE
  AnnualRate = RATE(PresentValue,TotalPeriods,Payment,FutureValue)
  AnnualRate *= (PeriodsPerYear * 100)
END
```

PRERATE (rate of annuity with prepayment)

PRERATE(*presentvalue,periods,payment,futurevalue*)

PRERATE	Computes the periodic interest rate required to reach a targeted future value.
presentvalue	A numeric constant or variable containing the present value of the investment.
periods	A numeric constant or variable containing the number of periods in which a cash flow occurred.
payment	A numeric constant or variable containing the periodic payment amount.
futurevalue	A numeric constant or variable containing the amount of the desired or targeted future value of the investment.

PRERATE determines the periodic interest rate required to reach a desired amount (*futurevalue*) based upon a starting amount (*presentvalue*), the total number of periods (*periods*), and a payment amount (*payment*). If payments occur at the end of each period then use the RATE procedure, which calculates interest accordingly.

> **Note: If the present value is less than the future value (annuities), payments are positive, and conversely, if the present value is greater than the future value (loans), payments are negative.**

Return Data Type: DECIMAL

Internal Formulas:

$$0=\text{presentvalue}(1+\text{rate})^{\text{frac(periods)}}+\text{payment}(1+\text{rate})^{\text{frac(periods)}}\frac{1-(1+\text{rate})^{\text{int(-periods)}}}{\text{rate}}-\text{futurevalue}(1+\text{rate})^{\text{int(-periods)}}$$

where frac(periods) is the fractional portion of the *periods* parameter and where int(periods) is the integer portion of the *periods* parameter.

If the *payment* parameter is omitted or 0, then

$$\text{PRERATE} = \left(\frac{\text{futurevalue}}{\text{presentvalue}}\right)^{1/\text{periods}} - 1$$

The PRERATE procedure performs binary search iterations to home in on the interest rate value. If more than 50 such iterations are required, RATE returns a value of zero.

Example:

```
IF TimeOfPayment = 'Beginning of Periods'
  AnnualRate = PRERATE(PresentValue,TotalPeriods,Payment,FutureValue)
  AnnualRate *= (PeriodsPerYear * 100)
ELSE
  AnnualRate = RATE(PresentValue,TotalPeriods,Payment,FutureValue)
  AnnualRate *= (PeriodsPerYear * 100)
END
```

SIMPINT (simple interest)

SIMPINT(*principal,rate*)

SIMPINT	Returns simple (uncompounded) interest.
principal	A numeric constant or variable containing the beginning balance, initial deposit, or loan.
rate	A numeric constant or variable containing the simple or contract interest rate.

SIMPINT determines an interest amount based solely on a given amount (*principal*) and the simple interest rate (*rate*). The amount returned ONLY reflects interest earned.

Return Data Type: DECIMAL

Internal Formulas:

```
SIMPLE INTEREST = principal * rate
```

Example:

```
PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)!Set up variables
ActualRate = PeriodicRate * TotalPeriods
SimpleInterestAmount = SIMPINT(Principal,ActualRate)      !Call SIMPINT
SimpleInterestAmount += Principal                           !Add in principal
```

3 - BUSINESS STATISTICS LIBRARY

Overview

This chapter provides a discussion of the purpose and components of the Business Statistics Library, as well as a discussion and an example of the specific usage of each procedure in the Business Statistics Library.

The Business Statistics Library contains procedures which allow you to perform statistical operations including the calculation of means, medians, standard deviations, factorials, etc.

Business Statistics Library Components

Provided with the Business Statistics Library are prototypes, code templates, and examples that simplify the implementation of the Business Statistics procedures into your application or project. Follow the procedures described in *Installing the Business Math Library* in Chapter 1.

The Business Statistics Library components (files) are:

...\LIB\C5STATL.LIB
: 16-bit Business Statistics objects that link into your executable.

...\BIN\C5STAT.DLL
: 16-bit Business Statistics procedures.

...\LIB\C5STAT.LIB
: 16-bit Business Statistics stub file for resolving link references to procedures in C5STAT.DLL.

...\LIB\C5STATxL.LIB
: 32-bit Business Statistics objects that link into your executable.

...\BIN\C5STATx.DLL
: 32-bit Business Statistics procedures.

...\LIB\C5STATx.LIB
: 32-bit Business Statistics stub file for resolving link references to procedures in C5STATx.DLL.

...\TEMPLATE\STATISTC.TPL
: Business Statistics Library templates. The templates self-document the use of the library procedures. The templates must be registered before they can be used in your application. See *Installing the Business Math Libraries* in Chapter 1.

...\LIBSRC\CWSTATPR.CLW
: Prototypes for the Business Statistics procedures.

...\LIBSRC\CWSTATDT.CLW
: TYPE definitions for QUEUES passed to Business Statistics procedures.

Procedures

FACTORIAL (factorial of a number)

FACTORIAL(*number*)

FACTORIAL	Computes the factorial of a number.
number	A numeric constant or variable containing a positive integer value.

The **FACTORIAL** procedure implements the standard factorial formula. For example, if the *number* provided is 5 then the procedure returns the value of: (1 x 2 x 3 x 4 x 5) = 120.

Return DataType: REAL

Example:

```
X = 5
FactorialOfX = FACTORIAL(X)                    !call FACTORIAL
```

FREQUENCY (frequency count of an item in a set)

FREQUENCY(*dataset,searchvalue*)

FREQUENCY	Counts the number of instances of a particular value within a numeric set.
dataset	The label of a QUEUE with its first component field defined as a REAL.
searchvalue	A numeric constant or variable containing the value to search for in the QUEUE.

FREQUENCY counts the number of instances of a particular value (*searchvalue*) within a numeric set (*dataset*). The procedure operates on the numeric set defined by all the entries in the first component of the designated QUEUE (*dataset*).

For example, given the data set: [1,2,2,3,4,5] and the search value 2, the procedure would return a frequency count of 2.

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

Return DataType: REAL

Example:

```
StatSetX    QUEUE,PRE()
X             REAL
            END

FrequencyOfX  REAL
SearchValue   REAL(1)

CODE
    FREE(StatSetX)                                    !free the QUEUE
    CLEAR(STADAT:RECORD)                             !clear the record buffer
    STADAT:Id = STA:Id                                !prime the record buffer
    STADAT:ItemNumber = 1
    SET(STADAT:KeyIdItemNumber,STADAT:KeyIdItemNumber !position file pointer
    LOOP                                              !load the QUEUE
      NEXT(StatisticsData)                            !read next record
      IF ERRORCODE() OR STADAT:Id NOT = STA:Id
        BREAK
      ELSE
        StatSetX:X = STADAT:X                         !load the QUEUE buffer
        ADD(StatSetX)                                 !add to the QUEUE
      END
    END
    FrequencyOfX = FREQUENCY(StatSetX,SearchValue)  !call FREQUENCY to search Q
```

LOWERQUARTILE (lower quartile value of a set)

LOWERQUARTILE(*dataset*)

LOWERQUARTILE
Returns the median of the lower half of an ordered numeric data set.

dataset The label of a QUEUE with its first component field defined as a REAL.

LOWERQUARTILE computes a value such that at most 25% of a numeric set's values are less than the computed value, and at most 75% of the set's values are greater than the computed value. The procedure operates on the numeric set defined by all the entries in the first component of the designated QUEUE (*dataset*). For example, given the data set: [1,2,3,4,5,6,7,8] the lower quartile value is 2.5.

In general, the LOWERQUARTILE procedure is only meaningful when the number of elements in the data set is large (ie. greater than 20). See also *PERCENTILE, UPPERQUARTILE*.

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

Return DataType: REAL

Example:

```
StatSetX    QUEUE,PRE()
X             REAL
            END

LowerQuartileOfSet  REAL

CODE
    FREE(StatSetX)                                    !free the QUEUE
    CLEAR(STADAT:RECORD)                             !clear the record buffer
    STADAT:Id = STA:Id                                !prime the record buffer
    STADAT:ItemNumber = 1
    SET(STADAT:KeyIdItemNumber,STADAT:KeyIdItemNumber !position file pointer
    LOOP                                              !load the QUEUE
      NEXT(StatisticsData)                            !read next record
      IF ERRORCODE() OR STADAT:Id NOT = STA:Id
        BREAK
      ELSE
        StatSetX:X = STADAT:X                         !load the QUEUE buffer
        ADD(StatSetX)                                 !add to the QUEUE
      END
    END
    LowerQuartileOfSet = LOWERQUARTILE(StatSetX)    !call LOWERQUARTILE
```

MEAN (mean of a set)

MEAN(*dataset*)

MEAN	Returns the mean or average of a set of numbers.
dataset	The label of a QUEUE with its first component field defined as a REAL.

MEAN computes the arithmetic average of a set. The arithmetic average is the sum of a set's values divided by the number of elements in the set. The procedure operates on the numeric set defined by all the entries in the first component of the designated QUEUE (*dataset*).

For example, if the set contains 5 values: [1,2,3,4,5] then the mean is (1+2+3+4+5) / 5 = 3.

> **Note:** **The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.**

Return DataType: REAL

Example:

```
StatSetX    QUEUE,PRE()
X             REAL
            END

MeanOfSet   REAL

CODE
    FREE(StatSetX)                                    !free the QUEUE
    CLEAR(STADAT:RECORD)                             !clear the record buffer
    STADAT:Id = STA:Id                                !prime the record buffer
    STADAT:ItemNumber = 1
    SET(STADAT:KeyIdItemNumber,STADAT:KeyIdItemNumber !position file pointer
    LOOP                                              !load the QUEUE
      NEXT(StatisticsData)                            !read next record
      IF ERRORCODE() OR STADAT:Id NOT = STA:Id
        BREAK
      ELSE
        StatSetX:X = STADAT:X                         !load the QUEUE buffer
        ADD(StatSetX)                                 !add to the QUEUE
      END
    END
    MeanOfSet = MEAN(StatSetX)                       !call MEAN
```

MEDIAN (median of a set)

MEDIAN(*dataset*)

MEDIAN	Returns the middle value of an ordered numeric data set.
dataset	The label of a QUEUE with its first component field defined as a REAL.

MEDIAN finds the value which is exactly in the middle when all of the data is in (sorted) order from low to high, or the mean of the two data values which are nearest the middle. The procedure operates on the numeric set defined by all the entries in the first component of the designated QUEUE (*dataset*).

For example, given the data set: [1,2,3,4,5] the median is 3, or given the data set: [1,2,4,5] the median is (2 + 4) / 2 = 3.

> **Note:** **The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.**

Return DataType: REAL

Example:

```
StatSetX    QUEUE,PRE()
X             REAL
            END

MedianOfSet REAL

CODE
    FREE(StatSetX)                                    !free the QUEUE
    CLEAR(STADAT:RECORD)                             !clear the record buffer
    STADAT:Id = STA:Id                                !prime the record buffer
    STADAT:ItemNumber = 1
    SET(STADAT:KeyIdItemNumber,STADAT:KeyIdItemNumber !position file pointer
    LOOP                                              !load the QUEUE
      NEXT(StatisticsData)                            !read next record
      IF ERRORCODE() OR STADAT:Id NOT = STA:Id
        BREAK
      ELSE
        StatSetX:X = STADAT:X                         !load the QUEUE buffer
        ADD(StatSetX)                                 !add to the QUEUE
      END
    END
    MedianOfSet = MEDIAN(StatSetX)                   !call MEDIAN
```

MIDRANGE (midrange of a set)

MIDRANGE(*dataset*)

MIDRANGE	Returns the average of the highest and lowest value in a numeric set.
dataset	The label of a QUEUE with its first component field defined as a REAL.

MIDRANGE computes the mean of the smallest and largest values in a data set. The procedure operates on the numeric set defined by all the entries in the first component of the designated QUEUE (*dataset*).

For example, given the data set: [1,2,3,4,5] the midrange is (1 + 5) / 2 = 3.

> **Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.**

Return DataType: REAL

Example:

```
StatSetX      QUEUE,PRE()
X               REAL
              END

MidrangeOfSet   REAL

CODE
    FREE(StatSetX)                                    !free the QUEUE
    CLEAR(STADAT:RECORD)                             !clear the record buffer
    STADAT:Id = STA:Id                                !prime the record buffer
    STADAT:ItemNumber = 1
    SET(STADAT:KeyIdItemNumber,STADAT:KeyIdItemNumber !position file pointer
    LOOP                                              !load the QUEUE
      NEXT(StatisticsData)                            !read next record
      IF ERRORCODE() OR STADAT:Id NOT = STA:Id
        BREAK
      ELSE
        StatSetX:X = STADAT:X                         !load the QUEUE buffer
        ADD(StatSetX)                                 !add to the QUEUE
      END
    END
    MidrangeOfSet = MIDRANGE(StatSetX)               !call MIDRANGE
```

MODE (mode of a set)

MODE(*dataset*,*modeset*)

MODE	Identifies the value(s) that occurs most often in a numeric set and returns the number of times it occurs.
dataset	The label of a QUEUE with its first component field defined as a REAL.
modeset	The label of a QUEUE with its first component field defined as a REAL.

MODE determines which value or values within a numeric set occurs most often. The value or values are then stored in the passed QUEUE (*modeset*), and the procedure returns the number of occurrences (mode count).

The procedure operates on the numeric set defined by all the entries in the first component of the designated QUEUE (*dataset*). *Modeset* must be a QUEUE with the first component defined as a REAL variable. The contents of the *modeset* QUEUE are freed upon entry to the procedure.

For example, given the data set: [5,5,7,7,9] the returned mode count is 2 and the *modeset* QUEUE would contain two separate entries: 5 and 7.

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

Return DataType: REAL

Example:

```
StatSetX      QUEUE,PRE()
X               REAL
              END
ModeSet       QUEUE,PRE()
ModeValue       REAL
              END
ModeCount       REAL
CODE
    FREE(StatSetX)                                    !free the QUEUE
    CLEAR(STADAT:RECORD)                             !clear the record buffer
    STADAT:Id = STA:Id                                !prime the record buffer
    STADAT:ItemNumber = 1
    SET(STADAT:KeyIdItemNumber,STADAT:KeyIdItemNumber !position file pointer
    LOOP                                              !load the QUEUE
      NEXT(StatisticsData)                            !read next record
      IF ERRORCODE() OR STADAT:Id NOT = STA:Id
        BREAK
      ELSE
        StatSetX:X = STADAT:X                         !load the QUEUE buffer
        ADD(StatSetX)                                 !add to the QUEUE
      END
    END
    ModeCount = MODE(StatSetX,ModeSet)               !call MODE
```

PERCENTILE (pth percentile value of a set)

PERCENTILE(*dataset,percentile*)

PERCENTILE	Returns a value that divides an ordered numeric data set into two portions of specified relative size.
dataset	The label of a QUEUE with its first component field defined as a REAL.
percentile	A numeric constant or variable containing an integer value in the range: 1 >= p <= 99 .

PERCENTILE computes a value such that at most pth% of the set's values are less than the amount, and at most 100 - pth% of the set's values are greater than the amount. The procedure operates on the numeric set defined by all the entries in the first component of the designated QUEUE (*dataset*).

For example, given the data set: [1,2,3,4,5,6], the 50th Percentile value is 3.5.

In general, the PERCENTILE procedure is only meaningful when the number of elements in the data set is large (ie. greater than 20). See also *LOWERQUARTILE*, *UPPERQUARTILE*.

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

Return DataType: REAL

Example:

```
StatSetX      QUEUE,PRE()
X               REAL
              END

PercentileOfX   REAL
DividePoint     REAL(90)

CODE
    FREE(StatSetX)                                      !free the QUEUE
    CLEAR(STADAT:RECORD)                               !clear the record buffer
    STADAT:Id = STA:Id                                  !prime the record buffer
    STADAT:ItemNumber = 1
    SET(STADAT:KeyIdItemNumber,STADAT:KeyIdItemNumber !position file pointer
    LOOP                                                !load the QUEUE
      NEXT(StatisticsData)                              !read next record
      IF ERRORCODE() OR STADAT:Id NOT = STA:Id
        BREAK
      ELSE
        StatSetX:X = STADAT:X                           !load the QUEUE buffer
        ADD(StatSetX)                                   !add to the QUEUE
      END
    END
    PercentileOfX = PERCENTILE(StatSetX,DividePoint)!call PERCENTILE
```

RANGEOFSET (range of a set)

RANGEOFSET(*dataset*)

RANGEOFSET	Returns the difference between the largest and smallest values in a numeric set.
dataset	The label of a QUEUE with its first component field defined as a REAL.

RANGEOFSET computes the difference between the largest and smallest values in a numeric set. The procedure operates on the numeric set defined by all the entries in the first component of the designated QUEUE (*dataset*).

For example, given the data set: [1,2,3,4,5], the range is (5 - 1) = 4.

> **Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.**

Return DataType: REAL

Example:

```
StatSetX      QUEUE,PRE()
X               REAL
              END

RangeOfSetX   REAL

CODE
    FREE(StatSetX)                                    !free the QUEUE
    CLEAR(STADAT:RECORD)                             !clear the record buffer
    STADAT:Id = STA:Id                                !prime the record buffer
    STADAT:ItemNumber = 1
    SET(STADAT:KeyIdItemNumber,STADAT:KeyIdItemNumber !position file pointer
    LOOP                                              !load the QUEUE
      NEXT(StatisticsData)                            !read next record
      IF ERRORCODE() OR STADAT:Id NOT = STA:Id
        BREAK
      ELSE
        StatSetX:X = STADAT:X                         !load the QUEUE buffer
        ADD(StatSetX)                                 !add to the QUEUE
      END
    END
    RangeOfSetX = RANGEOFSET(StatSetX)               !call RANGEOFSET
```

RVALUE (linear regression correlation coefficient)

RVALUE(*dataset* [,*meanX*] [,*meanY*])

RVALUE	Returns the correlation coefficient of the linear relationship between the paired data points in the data set.
dataset	The label of a QUEUE with its first two component fields defined as REAL. The first component contains the set of X values and the second component contains the set of Y values.
meanX	A numeric constant or variable containing the average of the X values (optional).
meanY	A numeric constant or variable containing the average of the Y values (optional).

RVALUE computes the correlation coefficient of the linear relationship between the paired data points in the data set. The procedure operates on the numeric sets defined by all the entries in the first two components of the designated QUEUE (*dataset*).

The optional parameters help to optimize the procedure. If not provided, the value(s) are computed internally by the procedure.

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

Return DataType: REAL

Example:

```
StatSetXY     QUEUE,PRE()
X               REAL
Y               REAL
              END
CorrelationCoefficient  REAL

CODE
    FREE(StatSetXY)                                    !free the QUEUE
    CLEAR(STADAT:RECORD)                              !clear the record buffer
    STADAT:Id = STA:Id                                 !prime the record buffer
    STADAT:ItemNumber = 1
    SET(STADAT:KeyIdItemNumber,STADAT:KeyIdItemNumber !position file pointer
    LOOP                                               !load the QUEUE
      NEXT(StatisticsData)                             !read next record
      IF ERRORCODE() OR STADAT:Id NOT = STA:Id
        BREAK
      ELSE
        StatSetXY:X = STADAT:X                         !load the QUEUE buffer
        StatSetXY:Y = STADAT:Y                         !load the QUEUE buffer
        ADD(StatSetXY)                                 !add to the QUEUE
      END
    END
    CorrelationCoefficient = RVALUE(StatSetXY)      !call RVALUE
```

SS (sum of squares)

SS(*dataset* [,*meanofset*])

SS	Returns the sum of the squared differences between each data set value and the data set's mean.
dataset	The label of a QUEUE with its first component field defined as a REAL.
meanofset	A numeric constant or variable representing the average of the data set's values (optional).

SS computes the sum of the squared differences between each data set value and the data set's mean. The procedure operates on the numeric set defined by all the entries in the first component of the designated QUEUE (*dataset*). The computation is a significant factor in several statistical formulas.

The optional parameter helps to optimize the procedure. If not provided, the mean value is computed internally by the procedure.

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

Return DataType: REAL

Example:

```
StatSetX      QUEUE,PRE()
X               REAL
              END

SumOFSquaresX   REAL

CODE
    FREE(StatSetX)                                    !free the QUEUE
    CLEAR(STADAT:RECORD)                             !clear the record buffer
    STADAT:Id = STA:Id                                !prime the record buffer
    STADAT:ItemNumber = 1
    SET(STADAT:KeyIdItemNumber,STADAT:KeyIdItemNumber !position file pointer
    LOOP                                              !load the QUEUE
      NEXT(StatisticsData)                            !read next record
      IF ERRORCODE() OR STADAT:Id NOT = STA:Id
        BREAK
      ELSE
        StatSetX:X = STADAT:X                         !load the QUEUE buffer
        ADD(StatSetX)                                 !add to the QUEUE
      END
    END
    SumOFSquaresX = SS(StatSetX)                     !call SS
```

SSXY (sum of squares for x and y)

SSXY(*dataset* [,*meanX*] [,*meanY*])

SSXY	Returns the sum of the products of the differences between each data point and its associated (either X or Y) mean value.
dataset	The label of a QUEUE with its first two component fields defined as REAL. The first component contains the set of X values and the second component contains the set of Y values.
meanX	A numeric constant or variable containing the average of the X values (optional).
meanY	A numeric constant or variable containing the average of the Y values (optional).

SSXY computes the sum of the products of the differences between each data point and its associated (either X or Y) mean value. The procedure operates on the numeric sets defined by all the entries in the first two components of the designated QUEUE (*dataset*). The computation is a significant factor in linear regression formulas.

The optional parameters help to optimize the procedure. If not provided, the value(s) will be computed internally by the procedure.

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

Return DataType: REAL

Example:

```
StatSetXY     QUEUE,PRE()
X               REAL
Y               REAL
              END
SumOfSquares    REAL
CODE
    STADAT:Id = STA:Id                                  !prime the record buffer
    STADAT:ItemNumber = 1
    SET(STADAT:KeyIdItemNumber,STADAT:KeyIdItemNumber  !position file pointer
    LOOP                                                !load the QUEUE
      NEXT(StatisticsData)                              !read next record
      IF ERRORCODE() OR STADAT:Id NOT = STA:Id
        BREAK
      ELSE
        StatSetXY:X = STADAT:X                          !load the QUEUE buffer
        StatSetXY:Y = STADAT:Y                          !load the QUEUE buffer
        ADD(StatSetXY)                                  !add to the QUEUE
      END
    END
    SumOfSquares = SSXY(StatSetXY)                   !call SSXY
```

ST1 (student's t for a single mean)

ST1(*dataset*,*hypothesizedmean*)

ST1	Returns the Student's t value for a given data set and hypothesized mean value.
dataset	The label of a QUEUE with its first component field defined as a REAL.
hypothesizedmean	A numeric constant or variable representing a hypothesized mean value associated with the statistical test.

ST1 computes the Student's t value for a given data set and hypothesized mean value. The procedure operates on the numeric set defined by all the entries in the first component of the designated QUEUE (*dataset*). The returned t value is used in a statistical test of an hypothesis about a normally distributed population based on a small sample.

> **Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.**

Return DataType: REAL

Example:

```
StatSetXY     QUEUE,PRE()
X               REAL
              END
TValue        REAL

CODE
    FREE(StatSetX)                                      !free the QUEUE
    CLEAR(STADAT:RECORD)                               !clear the record buffer
    STADAT:Id = STA:Id                                  !prime the record buffer
    STADAT:ItemNumber = 1
    SET(STADAT:KeyIdItemNumber,STADAT:KeyIdItemNumber !position file pointer
    LOOP                                                !load the QUEUE
      NEXT(StatisticsData)                              !read next record
      IF ERRORCODE() OR STADAT:Id NOT = STA:Id
        BREAK
      ELSE
        StatSetX:X = STADAT:X                           !load the QUEUE buffer
        ADD(StatSetX)                                   !add to the QUEUE
      END
    END
    TValue = ST1(StatSetX,HypothesizedMean)             !call ST1
```

SDEVIATIONP (standard deviation of a population)

SDEVIATIONP(*dataset* [,*meanofset*])

SDEVIATIONP	Returns the standard deviation of the entire population of a numeric set.
dataset	The label of a QUEUE with its first component field defined as a REAL.
meanofset	A numeric constant or variable representing the average of the data set's values (optional).

SDEVIATIONP computes the standard deviation of a given population data set. SDEVIATIONS computes the standard deviation of a given sample data set. The 'Population' procedure call should be used only if the data set contains all of a population's values. The procedure operates on the numeric set defined by all the entries in the first component of the designated QUEUE (*dataset*).

The optional parameter helps to optimize the procedure. If not provided, the value is computed internally by the procedure.

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

Return DataType: REAL

Example:

```
StatSetXY             QUEUE,PRE()
X                       REAL
                      END
StandardDeviation       REAL

CODE
    FREE(StatSetX)                                      !free the QUEUE
    CLEAR(STADAT:RECORD)                               !clear the record buffer
    STADAT:Id = STA:Id                                  !prime the record buffer
    STADAT:ItemNumber = 1
    SET(STADAT:KeyIdItemNumber,STADAT:KeyIdItemNumber !position file pointer
    LOOP                                                !load the QUEUE
      NEXT(StatisticsData)                              !read next record
      IF ERRORCODE() OR STADAT:Id NOT = STA:Id
        BREAK
      ELSE
        StatSetX:X = STADAT:X                           !load the QUEUE buffer
        ADD(StatSetX)                                   !add to the QUEUE
      END
    END
    StandardDeviation = SDEVIATIONP(StatSetX)           !call SDEVIATIONP
```

SDEVIATIONS (standard deviation of a sample)

SDEVIATIONS(*dataset* [,*meanofset*])

SDEVIATIONS	Returns the standard deviation of a sample of a numeric set.
dataset	The label of a QUEUE with its first component field defined as a REAL.
meanofset	A numeric constant or variable representing the average of the data set's values.

SDEVIATIONP computes the standard deviation of a given population data set. SDEVIATIONS computes the standard deviation of a given sample data set. The 'Sample' procedure call should be used only if the data set contains less than all of a population's values. The procedure operates on the numeric set defined by all the entries in the first component of the designated QUEUE (*dataset*).

The optional parameter helps to optimize the procedure. If not provided, the value is computed internally by the procedure.

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

Return DataType: REAL

Example:

```
StatSetXY             QUEUE,PRE()
X                       REAL
                      END
StandardDeviation       REAL

CODE                                  !free the QUEUE
    CLEAR(STADAT:RECORD)                            !clear the record buffer
    STADAT:Id = STA:Id                                !prime the record buffer
    STADAT:ItemNumber = 1
    SET(STADAT:KeyIdItemNumber,STADAT:KeyIdItemNumber !position file pointer
    LOOP                                              !load the QUEUE
      NEXT(StatisticsData)                            !read next record
      IF ERRORCODE() OR STADAT:Id NOT = STA:Id
        BREAK
      ELSE
        StatSetX:X = STADAT:X                         !load the QUEUE buffer
        ADD(StatSetX)                                 !add to the QUEUE
      END
    END
    StandardDeviation = SDEVIATIONS(StatSetX)         !call SDEVIATIONS
```

SUMM (summation of a set)

SUMM(*dataset*)

SUMM Returns the summation of all of a data set's values.

dataset The label of a QUEUE with its first component field defined as a REAL.

SUMM computes the summation of all of a data set's values. The procedure operates on the numeric set defined by all the entries in the first component of the designated QUEUE (*dataset*). The computation is a significant factor in several statistical formulas.

> **Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.**

Return DataType: REAL

Example:

```
StatSetXY             QUEUE,PRE()
X                       REAL
                      END
SummationOfSet          REAL

CODE
    FREE(StatSetX)                                      !free the QUEUE
    CLEAR(STADAT:RECORD)                               !clear the record buffer
    STADAT:Id = STA:Id                                  !prime the record buffer
    STADAT:ItemNumber = 1
    SET(STADAT:KeyIdItemNumber,STADAT:KeyIdItemNumber !position file pointer
    LOOP                                                !load the QUEUE
      NEXT(StatisticsData)                              !read next record
      IF ERRORCODE() OR STADAT:Id NOT = STA:Id
        BREAK
      ELSE
        StatSetX:X = STADAT:X                           !load the QUEUE buffer
        ADD(StatSetX)                                   !add to the QUEUE
      END
    END
    SummationOfSet = SUMM(StatSetX)                     !call SUMM
```

UPPERQUARTILE (upper quartile value of a set)

UPPERQUARTILE(*dataset*)

UPPERQUARTILE
Returns the median of the upper half of an ordered numeric data set.

dataset The label of a QUEUE with its first component field defined as a REAL.

UPPERQUARTILE computes a value such that at most 75% of the set's values are less than the amount, and at most 25% of the set's values are greater than the amount. The procedure operates on the numeric set defined by all the entries in the first component of the designated QUEUE (*dataset*).

For example, given the data set: [1,2,3,4,5,6,7,8] the upper quartile value is 6.5.

In general, the UPPERQUARTILE procedure is only meaningful when the number of elements in the data set is large (ie. greater than 20). See also *LOWERQUARTILE, PERCENTILE*.

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

Return DataType: REAL

Example:

```
StatSetXY             QUEUE,PRE()
X                       REAL
                      END
UpperQuartileOfSet      REAL

CODE
    FREE(StatSetX)                                      !free the QUEUE
    CLEAR(STADAT:RECORD)                               !clear the record buffer
    STADAT:Id = STA:Id                                  !prime the record buffer
    STADAT:ItemNumber = 1
    SET(STADAT:KeyIdItemNumber,STADAT:KeyIdItemNumber  !position file pointer
    LOOP                                                !load the QUEUE
      NEXT(StatisticsData)                              !read next record
      IF ERRORCODE() OR STADAT:Id NOT = STA:Id
        BREAK
      ELSE
        StatSetX:X = STADAT:X                           !load the QUEUE buffer
        ADD(StatSetX)                                   !add to the QUEUE
      END
    END
    UpperQuartileOfSet = UPPERQUARTILE(StatSetX)    !call UPPERQUARTILE
```

VARIANCEP (variance of a population)

VARIANCEP(*dataset* [,*meanofset*])

VARIANCEP	Returns the variance of the entire population of a numeric set.
dataset	The label of a QUEUE with its first component field defined as a REAL.
meanofset	A numeric constant or variable representing the average of the data set's values (optional).

VARIANCEP computes the variance of a given population data set. VARIANCES computes the variance of a given sample data set. The 'Population' procedure call should be used only if the data set contains all of a population's values. The procedure operates on the numeric set defined by all the entries in the first component of the designated QUEUE (*dataset*).

The optional parameter helps to optimize the procedure. If not provided, the value is computed internally by the procedure.

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

Return DataType: REAL

Example:

```
StatSetXY             QUEUE,PRE()
X                       REAL
                      END
VarianceOfSet         REAL

CODE
    FREE(StatSetX)                                      !free the QUEUE
    CLEAR(STADAT:RECORD)                               !clear the record buffer
    STADAT:Id = STA:Id                                  !prime the record buffer
    STADAT:ItemNumber = 1
    SET(STADAT:KeyIdItemNumber,STADAT:KeyIdItemNumber !position file pointer
    LOOP                                                !load the QUEUE
      NEXT(StatisticsData)                              !read next record
      IF ERRORCODE() OR STADAT:Id NOT = STA:Id
        BREAK
      ELSE
        StatSetX:X = STADAT:X                           !load the QUEUE buffer
        ADD(StatSetX)                                   !add to the QUEUE
      END
    END
    VarianceOfSet = VARIANCEP(StatSetX)                 !call VARIANCEP
```

VARIANCES (variance of a sample)

VARIANCES(*dataset* [,*meanofset*])

VARIANCES	Returns the variance of a sample of a numeric set.
dataset	The label of a QUEUE with its first component field defined as a REAL.
meanofset	A numeric constant or variable representing the average of the data set's values (optional).

VARIANCES computes the variance of a given sample data set. VARIANCEP computes the variance of a given population data set. The 'Sample' procedure call should be used only if the data set contains less than all of a population's values. Both procedures operate on on the numeric set defined by all the entries in the first component of the designated QUEUE (*dataset*).

The optional parameter helps to optimize the procedure. If not provided, the value is computed internally by the procedure.

> **Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.**

Return DataType: REAL

Example:

```
StatSetXY             QUEUE,PRE()
X                       REAL
                      END
VarianceOfSet         REAL

CODE
    FREE(StatSetX)                                      !free the QUEUE
    CLEAR(STADAT:RECORD)                               !clear the record buffer
    STADAT:Id = STA:Id                                  !prime the record buffer
    STADAT:ItemNumber = 1
    SET(STADAT:KeyIdItemNumber,STADAT:KeyIdItemNumber !position file pointer
    LOOP                                                !load the QUEUE
      NEXT(StatisticsData)                              !read next record
      IF ERRORCODE() OR STADAT:Id NOT = STA:Id
        BREAK
      ELSE
        StatSetX:X = STADAT:X                           !load the QUEUE buffer
        ADD(StatSetX)                                   !add to the QUEUE
      END
    END
    VarianceOfSet = VARIANCES(StatSetX)                 !call VARIANCES
```

4 - BUSINESS MATH TEMPLATES

Overview

This chapter provides a look at all of the templates supplied with the Clarion Business Math Library and demonstrates how to use each one. The templates make it easy to implement Business Math procedures into your application by:

- Adding the appropriate procedure prototypes to the application's MAP.
- Adding appropriate library entries to the application's project.
- Enabling access to the associated Code templates from any embed point throughout the application.
- Prompting for necessary parameters and generating syntactically correct calls to the procedure or procedure.
- Providing a self documenting method of calling each procedure or procedure.

Template Components

Finance and Business Statistics each has its own template class. Each of these template classes includes an #EXTENSION template plus several #CODE templates. Class Finance is in the FINANCE.TPL file, and Class Statistics is in the STATISTC.TPL file.

Class Finance

#EXTENSION Templates

FinanceLibrary	Enables access to the associated Code templates from any embed point throughout the application. It also adds the appropriate procedure prototypes to the application's MAP and adds appropriate library entries to the application's project.

#CODE Templates

AMORTIZE	Amortize Loan for Specific Number of Payments
APR	Annual Percentage Rate
COMPINT	Compound Interest
CONTINT	Continuous Compounding Interest
DAYS360	Days Difference Based on 360-day Year
FV	Future Value
IRR	Internal Rate of Return
NPV	Net Present Value
PERS	Periods of Annuity (with/without Prepayment)
PMT	Payment of Annuity (with/without Prepayment)
PV	Present Value (with/without Prepayment)
RATE	Rate of Annuity (with/without Prepayment)
SIMPINT	Simple Interest

Class Statistics

#EXTENSION Templates

StatisticsLibrary	Enables access to the associated Code templates from any embed point throughout the application. It also adds the appropriate procedure prototypes to the application's MAP and adds appropriate library entries to the application's project.

#CODE Templates

FACTORIAL	Factorial of a Number
FREQUENCY	Frequency Count of an Item in a Set
LOWERQUARTILE	Lower Quartile Value of a Set
MEAN	Mean of a Set
MEDIAN	Median of a Set
MIDRANGE	Midrange of a Set
MODE	Mode of a Set
PERCENTILE	pth Percentile Value of a Set
RANGEOFSET	Range of a Set
rVALUE	Linear Regression Correlation Coefficient
SS	Sum of Squares
SSxy	Sum of Squares for X and Y
ST1	Student's t (single mean)
SDEVIATION	Standard Deviation of a Set
SUMM	Summation of a Set
UPPERQUARTILE	Upper Quartile Value of a Set
VARIANCE	Variance of a Set

Registering the Template Classes

To use the Finance Code templates, register the FINANCE.TPL file in the in the Clarion Template Registry. To use the Business Statistics Code Templates, register the STATISTC.TPL file. Both files are installed in the ..\TEMPLATE subdirectory.

These template classes are automatically registered when you install the Business Math Library with the setup program. However, should you need to re-register the templates for any reason, here are the steps to follow.

❑ *To register the business math template classes:*

1. Choose **Setup ➤ Template Registry** from the Clarion Environment menubar.
2. In the **Template Registry** dialog, press the **Register** button.
3. In the **Template File** dialog, select the FINANCE.TPL file in the ..\TEMPLATE subdirectory, then press **OK.**
4. In the **Template Registry** dialog, press the **Register** button.
5. In the **Template File** dialog, select the STATISTC.TPL file in the ..\TEMPLATE subdirectory, then press **OK.**
6. In the **Template Registry** dialog, press the **Close** button.

Note: **You may register either of these template classes individually. That is, you may register only the Finance Class or only the Business Statistics Class.**

Adding Extension Templates to Your Application

Once the template classes have been registered, you should add the business math extensions to your application's Global Extensions. This enables access to the associated Code templates from any embed point throughout the application. It also adds the appropriate procedure prototypes to the application's MAP and adds appropriate library entries to the application's project.

❑ *To add the business math extension templates to an application:*

1. Load the application into Clarion.
2. In the **Application Tree** dialog, press the **Global** button.
3. In the **Global Properties** dialog, press the **Extensions** button.
4. In the **Extensions and Control Templates** dialog, press the **Insert** button.
5. In the **Select Extension** dialog, highlight the *Clarion Finance Library* extension, and press the **Select** button.
6. In the **Extensions and Control Templates** dialog, press the **Insert** button.
7. In the **Select Extension** dialog, highlight the *Clarion Business Statistics Library* extension, and press the **Select** button.
8. In the **Extensions and Control Templates** dialog, press **OK**.
9. In the **Global Properties** dialog, press **OK.**

Embedding Code Templates in Your Application

Once the templates are registered and the extension template is added to the application's global extensions, the Code templates are available from any embed point in the application.

❑ *To embed a Code template in your application:*

1. In any dialog that has one (eg. Procedure Properties, List Properties), press the **Embeds** button.

   The **Embedded Source** dialog appears. The **Embedded Source** dialog is also accessible from the popup menus associated with procedures, windows, and controls.

2. Highlight an embed point and press the **Insert** button.

   The **Select Embed Type** dialog appears. From here you can see the registered template classes, including Class Finance and Class Statistics.

3. Highlight the desired Code template (eg. PV), and press the **Select** button.

   The **Prompts for ...** dialog appears.

4. Fill in the prompts according to the instructions, then press the **OK** button.

Finance Code Templates

Code templates generate executable code. Their purpose is to make customization—adding embedded source code fragments that do exactly what you want—easier. Each Code template has one well-defined task. For example, the APR Code template calculates an annual percentage rate, and nothing more. Typically, Code templates have a dialog box with options and instructions.

> **Tip:** **Pressing the ellipsis (...) button in the Prompts for ... dialog opens the Select Field dialog where you may choose the label of a data dictionary field or memory variable for the corresponding prompt, or you may define new fields or variables as needed.**

AMORTIZE

This Code template generates code to call the AMORTIZE procedure. AMORTIZE calculates which portion of a loan payment, or payments, constitutes interest and which portion constitutes repayment of the principal amount borrowed. The procedure also calculates the remaining loan balance after the payment or payments are applied.

The template prompts for the following parameters:

Current Balance The label of a variable containing the amount of the current loan balance.

Rate (Periodic) The label of a variable containing the amount of the *periodic interest rate* applied for a single period. Periodic interest rate may be calculated as:

```
PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)
ActualRate = PeriodicRate * TotalPeriods
```

Payment The label of a variable containing the amount of the desired payment (a negative number).

Number of Payments
The label of a variable containing the amount of the number of payments to amortize.

Principal The label of a DECIMAL variable to receive the portion of the payment(s) applied to pay back the loan (a negative number).

Interest The label of a DECIMAL variable to receive the portion of the payment(s) applied towards loan interest (a negative number).

Ending Balance The label of a DECIMAL variable to receive the remaining loan balance.

Display Return Values
Checking this box generates a DISPLAY statement for each of the return values.

Example code generated by the AMORTIZE Code template:

```
AMORTIZE(Balance,PeriodicRate,Payment,TotalPeriods,|
PrincipalAmount,InterestAmount,EndingBalance)
DISPLAY(?PrincipalAmount)
DISPLAY(?InterestAmount)
DISPLAY(?EndingBalance)
```

APR

This Code template generates code to call the APR procedure. APR determines the effective annual rate of interest based upon the contracted interest rate and the number of compounding periods per year. The contracted interest rate is a “non-compounded annual interest rate.”

The template prompts for the following parameters:

Rate The label of a variable containing the amount of the *contracted* interest rate.

Periods The label of a variable containing the amount of the number of compounding periods per year.

Annual Percentage Rate
The label of a DECIMAL variable to receive the calculated annual percentage rate.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Example code generated by the APR Code template:

```
AnnualRate = APR(RealRate,PeriodsPerYear)
DISPLAY(?AnnualRate)
```

COMPINT

This Code template generates code to call the COMPINT procedure. COMPINT computes total interest based on a principal amount, an applied interest rate, plus the number of compounding periods. The computed amount includes the original principal plus the compound interest earned.

The template prompts for the following parameters:

Principal The label of a variable containing the amount of the beginning balance, initial deposit, or loan.

Rate (Applied) The label of a variable containing the amount of the *applied interest rate* for the given time frame. Applied interest rate may be calculated as:

```
PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)
AppliedRate = PeriodicRate * TotalPeriods
```

Periods The label of a variable containing the number of compounding periods per year.

Ending Balance The label of a DECIMAL variable to receive the new balance.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Example code generated by the COMPINT Code template:

```
CompoundInterestAmount = COMPINT(Principal,ActualRate,TotalPeriods)
DISPLAY(?CompoundInterestAmount)
```

CONTINT

This Code template generates code to call the CONTINT procedure. CONTINT computes total continuously compounded interest based on a principal amount and an applied interest rate. The returned ending balance includes the original principal plus the interest earned.

The template prompts for the following parameters:

Principal The label of a variable containing the amount of the beginning balance, initial deposit, or loan.

Rate (Applied) The label of a variable containing the amount of the *applied interest rate* for the given time frame. Applied interest rate may be calculated as:

```
PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)
ActualRate = PeriodicRate * TotalPeriods
```

Ending Balance The label of a DECIMAL variable to receive the new balance.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Example code generated by the CONTINT Code template:

```
ContinuousInterestAmount = CONTINT(Principal,ActualRate)
DISPLAY(?ContinuousInterestAmount)
```

DAYS360

This Code template generates code to call the DAYS360 procedure. DAYS360 determines the number of days difference between a beginning date and an ending date, based on a 360-day year (30 day month). Both date parameters MUST contain Clarion standard date values.

The template prompts for the following parameters:

Start Date The label of a variable containing the beginning date.

End Date The label of a variable containing the ending date.

Number of Days The label of a DECIMAL variable to receive the difference between the start and end dates.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Example code generated by the DAYS360 Code template:

```
NumberOfDays = DAYS360(BeginningDate,EndingDate)
DISPLAY(?NumberOfDays)
```

FV

This Code template generates code to call the FV procedure or the PREFV procedure. Both procedures determine the future value of an initial amount plus an income stream. The income stream is defined by the total number of periods, the periodic interest rate, and a payment amount.

The template prompts for the following parameters:

Present Value The label of a variable containing the present value of the investment.

Periods The label of a variable containing the number of periods in which a cash flow occurred.

Rate (Periodic) The label of a variable containing the *periodic* interest rate.Periodic rate may be calculated as:

```
PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)
```

Payment The label of a variable containing the periodic payment amount (optional).

Future Value The label of a DECIMAL variable to receive the calculated future value.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Beginning of Periods
Checking this box generates a call to PREFV. PREFV assumes payments are made at the beginning of the compounding period and calculates interest accordingly.

Example code generated by the FV Code template:

```
FutureValue = FV(PresentValue,TotalPeriods,PeriodicRate,Payment)
DISPLAY(?FutureValue)
```

IRR

This Code template generates code to call the IRR procedure. IRR determines the internal rate of return on an investment. The result is computed by determining the rate that equates the present value of future cash flows to the cost of the initial investment.

The template prompts for the following parameters:

Investment The label of a variable containing the initial cost of the investment (a negative number).

Cash Flows The label of a single dimensioned DECIMAL array containing values in the amount of monies paid out and received during discrete accounting periods.

Periods The label of a single dimensioned DECIMAL array containing period numbers identifying the discrete accounting period in which a corresponding cash flow occurred. *The last element in this array MUST contain a zero to terminate the IRR analysis.*

Rate (Periodic) The label of a variable containing the desired *periodic* rate of return.

Rate of Return The label of a DECIMAL variable to receive the calculated internal rate of return.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Example code generated by the IRR Code template:

```
InternalRateOfReturn = IRR(InvestmentAmount,CashFlows,      |
                           CashFlowPeriods,PeriodicRate)
DISPLAY(?InternalRateOfReturn)
```

NPV

This Code template generates code to call the NPV procedure. NPV determines the present value of future returns, discounted at the marginal cost of capital minus the cost of the investment (*investment*).

The template prompts for the following parameters:

Investment The label of a variable containing the initial cost of the investment (a negative number).

Cash Flows The label of a single dimensioned DECIMAL array containing values in the amount of monies paid out and received during discrete accounting periods.

Periods The label of a single dimensioned DECIMAL array containing period numbers identifying the discrete accounting period in which a corresponding cash flow occurred. *The last element in this array MUST contain a zero to terminate the IRR analysis.*

Rate (Periodic) The label of a variable containing the desired *periodic* rate of return.

Net Present Value The label of a DECIMAL variable to receive the calculated net present value.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Example code generated by the NPV Code template:

```
NetPresentValue = NPV(InvestmentAmount,CashFlows,          |
                      CashFlowPeriods,PeriodicRate)
DISPLAY(?NetPresentValue)
```

PERS

This Code template generates code to call the PERS procedure or the PREPERS procedure. Both procedures determine the number of periods required to reach a desired future value based upon a starting amount, a periodic interest rate, and a periodic payment.

The template prompts for the following parameters:

Present Value The label of a variable containing the present value of the investment.

Rate (Periodic) The label of a variable containing the *periodic* interest rate.Periodic rate may be calculated as:

```
PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)
```

Payment The label of a variable containing the periodic payment amount (optional).

Future Value The label of a variable containing the desired or targeted future value of the investment.

Number of Periods The label of a DECIMAL variable to receive the calculated number of periods.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Beginning of Periods
Checking this box generates a call to PREPERS. PREPERS assumes payments are made at the beginning of the compounding period and calculates interest accordingly.

Example code generated by the PERS Code template:

```
TotalPeriods = PERS(PresentValue,PeriodicRate,Payment,FutureValue)
DISPLAY(?TotalPeriods)
```

PMT

This Code template generates code to call the PMT procedure or the PREPMT procedure. Both procedures determine the periodic payment required to reach a desired future value based upon a starting amount, a total number of periods, and a periodic interest rate.

The template prompts for the following parameters:

Present Value The label of a variable containing the present value of the investment.

Periods The label of a variable containing the number of periods in which a payment is made.

Rate (Periodic) The label of a variable containing the *periodic* rate of return. Periodic rate may be calculated as:

```
PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)
```

Future Value The label of a variable containing the desired or targeted future value of the investment.

Payment Amount The label of a DECIMAL variable to receive the calculated payment amount.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Beginning of Periods
Checking this box generates a call to PREPMT. PREPMT assumes payments are made at the beginning of the compounding period and calculates interest accordingly.

Example code generated by the PMT Code template:

```
Payment = PMT(PresentValue,TotalPeriods,PeriodicRate,FutureValue)
DISPLAY(?Payment)
```

PV

This Code template generates code to call the PV procedure or the PREPV procedure. Both procedures determine the present value required today to reach a desired future value based upon the total number of periods, a periodic interest rate, and a periodic payment amount.

The template prompts for the following parameters:

Periods The label of a variable containing the number of periods in which a payment is made.

Rate (Periodic) The label of a variable containing the *periodic* rate of return. Periodic rate may be calculated as:

```
PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)
```

Payment The label of a variable containing the periodic payment amount (optional).

Future Value The label of a variable containing the desired or targeted future value of the investment.

Present Value The label of a DECIMAL variable to receive the calculated present value.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Beginning of Periods
Checking this box generates a call to PREPV. PREPV assumes payments are made at the beginning of the compounding period and calculates interest accordingly.

Example code generated by the PV Code template:

```
PresentValue = PV(TotalPeriods,PeriodicRate,Payment,FutureValue)
DISPLAY(?PresentValue)
```

RATE

This Code template generates code to call the RATE procedure or the PRERATE procedure. Both procedures determine the periodic interest rate required to reach a desired future value based upon a starting amount, the total number of periods, and a payment amount.

The template prompts for the following parameters:

Present Value The label of a variable containing the present value of the investment.

Periods The label of a variable containing the number of periods in which a payment is made.

Payment The label of a variable containing the periodic payment amount (optional).

Future Value The label of a variable containing the desired or targeted future value of the investment.

Rate (Periodic) The label of a DECIMAL variable to receive the calculated periodic rate.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Beginning of Periods
Checking this box generates a call to PRERATE. PRERATE assumes payments are made at the beginning of the compounding period and calculates interest accordingly.

Example code generated by the RATE Code template:

```
PeriodicRate = RATE(PresentValue,TotalPeriods,Payment,FutureValue)
DISPLAY(?PeriodicRate)
```

SIMPINT

This Code template generates code to call the SIMPINT procedure. SIMPINT determines an interest amount based solely on a given amount and the simple interest rate.

The template prompts for the following parameters:

Principal The label of a variable containing the beginning balance, initial deposit, or loan.

Rate (Applied) The label of a variable containing the simple or contract interest rate.

Simple Interest Amount
The label of a DECIMAL variable to receive the calculated simple interest amount.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Example code generated by the SIMPINT Code template:

```
SimpleInterestAmount = SIMPINT(Principal,ActualRate)
DISPLAY(?SimpleInterestAmount)
```

Business Statistics Code Templates

Code templates generate executable code. Their purpose is to make customization—adding embedded source code fragments that do exactly what you want—easier. Each Code template has one well-defined task. For example, the FACTORIAL Code template calculates a factorial, and nothing more. Typically, Code templates have a dialog box with options and instructions.

> **Tip:** **Pressing the ellipsis (...) button opens the Select Field dialog where you may choose the label of a data dictionary field or memory variable for the corresponding prompt, or you may define new fields or variables as needed.**

FACTORIAL

This Code template generates code to call the FACTORIAL procedure. FACTORIAL implements the standard factorial formula. For example, if the number provided is 5 then the procedure returns the value of: (1 x 2 x 3 x 4 x 5) = 120.

The template prompts for the following parameters:

Number The label of a variable containing the number to factorialize.

Factorial Result The label of a REAL variable to receive the calculated factorial amount.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Example code generated by the FACTORIAL Code template:

```
FactorialOfN = FACTORIAL(N)
DISPLAY(?FactorialOfN)
```

FREQUENCY

This Code template generates code to call the FREQUENCY procedure. FREQUENCY counts the number of instances of a numeric value in a numeric set.

The template prompts for the following parameters:

Data Set The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to search.

Search Value The label of a variable containing the particular value to count.

Frequency Count The label of a REAL variable to receive the number of instances counted.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Example code generated by the FREQUENCY Code template:

```
FrequencyOfX = FREQUENCY(StatSetX,SearchValue)
DISPLAY(?FrequencyOfX)
```

LOWERQUARTILE

This Code template generates code to call the LOWERQUARTILE procedure. LOWERQUARTILE returns the median of the lower half of an ordered numeric data set. In other words, it returns a value such that at most 25% of a numeric set's values are less than the computed value, and at most 75% of the set's values are greater than the computed value.

In general, the LOWERQUARTILE procedure is only meaningful when the number of elements in the data set is large (ie. greater than 20). See also *PERCENTILE* and *UPPERQUARTILE*.

The template prompts for the following parameters:

Data Set The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to analyze.

Lower Quartile Value
The label of a REAL variable to receive the calculated value.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Example code generated by the LOWERQUARTILE Code template:

```
LowerQuartileOfSet = LOWERQUARTILE(StatSetX)
DISPLAY(?LowerQuartileOfSet)
```

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

MEAN

This Code template generates code to call the MEAN procedure. MEAN computes the arithmetic average of a set. The arithmetic average is the sum of a set's values divided by the number of elements in the set. For example, if the set contains 5 values: [1,2,3,4,5] then the mean is (1+2+3+4+5) / 5 = 3.

The template prompts for the following parameters:

Data Set The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to analyze.

Mean Value The label of a REAL variable to receive the calculated value.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Example code generated by the MEAN Code template:

```
MeanValue = MEAN(StatSetX)
DISPLAY(?MeanValue)
```

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

MEDIAN

This Code template generates code to call the MEDIAN procedure. MEDIAN returns the value which occurs in the middle when all of the data is in (sorted) order from low to high, or the mean of the two data values which are nearest the middle. For example, given the data set: [1,2,3,4,5] the median is 3, or given the data set: [1,2,4,5] the median is (2 + 4) / 2 = 3.

The template prompts for the following parameters:

Data Set The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to analyze.

Median Value The label of a REAL variable to receive the calculated value.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Example code generated by the MEDIAN Code template:

```
MedianValue = MEDIAN(StatSetX)
DISPLAY(?MedianValue)
```

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

MIDRANGE

This Code template generates code to call the MIDRANGE procedure. MIDRANGE computes the mean of the smallest and largest values in a data set. For example, given the data set: [1,2,3,4,5] the midrange is (1 + 5) / 2 = 3.

The template prompts for the following parameters:

Data Set The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to analyze.

Midrange Value The label of a REAL variable to receive the calculated value.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Example code generated by the MIDRANGE Code template:

```
MidrangeValue = MIDRANGE(StatSetX)
DISPLAY(?MidrangeValue)
```

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

MODE

This Code template generates code to call the MODE procedure. MODE identifies the value(s) that occurs most often in a numeric set and returns the number of times it occurs. For example, given the data set: [5,5,7,7,9] the returned mode count is 2 and the Mode Set QUEUE would contain two separate entries: 5 and 7.

The template prompts for the following parameters:

Data Set The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to analyze.

Mode Set The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE receives the value or values that occur most often in the data set.

Mode Frequency Count
The label of a REAL variable to receive the calculated value.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Example code generated by the MODE Code template:

```
ModeCount = MODE(StatSetX,ModeSet)
DISPLAY(?ModeCount)
```

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

PERCENTILE

This Code template generates code to call the PERCENTILE procedure. PERCENTILE Returns a value that divides an ordered numeric data set into two portions of specified relative size. In other words, PERCENTILE computes a value such that at most pth% of the set's values are less than the amount, and at most 100 - pth% of the set's values are greater than the amount. For example, given the data set: [1,2,3,4,5,6], the 50th Percentile value is 3.5. See also *UPPERQUARTILE* and *LOWERQUARTILE*.

The template prompts for the following parameters:

Data Set The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to analyze.

Percentile The label of a variable containing the division point expressed as a percentage.

Percentile Value The label of a REAL variable to receive the calculated value.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Example code generated by the PERCENTILE Code template:

```
Percentile = PERCENTILE(StatSetX,PercentileOfSet)
DISPLAY(?Percentile)
```

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

RANGEOFSET

This Code template generates code to call the RANGEOFSET procedure. RANGEOFSET Returns the difference between the largest and smallest values in a numeric set. For example, given the data set: [1,2,3,4,5], the range is (5 - 1) = 4.

The template prompts for the following parameters:

Data Set The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to analyze.

Range Value The label of a REAL variable to receive the calculated value.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Example code generated by the RANGEOFSET Code template:

```
RangeOfSet = RANGEOFSET(StatSetX)
DISPLAY(?RangeOfSet)
```

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

rVALUE

This Code template generates code to call the RVALUE procedure. RVALUE Returns the correlation coefficient of the linear relationship between the paired data points in the data set.

The template prompts for the following parameters:

Data Set The label of a QUEUE whose first two component fields are REAL. The first component of the QUEUE comprises the set of X values to analyze. The second component of the QUEUE comprises the set of Y values to analyze. The two components are treated as x-y coordinate pairs.

Mean of X Data The label of a variable containing the average of the X values (optional).

Mean of Y Data The label of a variable containing the average of the Y values (optional).

Correlation Coefficient
The label of a REAL variable to receive the calculated value.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Example code generated by the rVALUE Code template:

```
CorrelationCoefficient = RVALUE(StatSetX)
DISPLAY(?CorrelationCoefficient)
```

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

SDEVIATION

This Code template generates code to call the SDEVIATIONP procedure or the SDEVIATIONS procedure. SDEVIATIONP computes the standard deviation of an *entire population*. SDEVIATIONS computes the standard deviation of a *population sample*.

The template prompts for the following parameters:

Type Choose ‘Sample’ or ‘Population.’ ‘Sample’ is the default.

Data Set The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to analyze.

Mean of Data Set The label of a variable containing the average of the set’s values (optional).

Standard Deviation
The label of a REAL variable to receive the calculated value.

Display Return Value

Example code generated by the SDEVIATION Code template:

```
StandardDeviation = SDEVIATIONP(StatSetX,MeanValue)
DISPLAY(?StandardDeviation)
```

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

SS

This Code template generates code to call the SS procedure. SS returns the sum of the squared differences between each data set value and the data set's mean.

The template prompts for the following parameters:

Data Set The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to analyze.

Mean of Data Set The label of a variable containing the average of the set's values (optional).

Sum of Squares Value
The label of a REAL variable to receive the calculated value.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Example code generated by the SS Code template:

```
SumOfSquaresForX = SS(StatSetX)
DISPLAY(?SumOfSquaresForX)
```

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

SSxy

This Code template generates code to call the SSXY procedure. SSXY returns the sum of the products of the differences between each data point and its associated (either X or Y) mean value.

The template prompts for the following parameters:

Data Set The label of a QUEUE whose first two component fields are REAL. The first component of the QUEUE comprises the set of X values to analyze. The second component of the QUEUE comprises the set of Y values to analyze. The two components are treated as x-y coordinate pairs.

Mean of X Data The label of a variable containing the average of the X values (optional).

Mean of Y Data The label of a variable containing the average of the Y values (optional).

Sum of Squares for X and Y
The label of a REAL variable to receive the calculated value.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Example code generated by the SSxy Code template:

```
SumOfSquaresForXandY = SSXY(StatSetXY)
DISPLAY(?SumOfSquaresForXandY)
```

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

ST1

This Code template generates code to call the ST1 procedure. ST1 Returns the Student's t value for a given data set and hypothesized mean value.

The template prompts for the following parameters:

Data Set The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to analyze.

Hypothesized Mean Value
The label of a REAL variable containing a hypothesized mean value associated with the statistical test.

Student's t Value The label of a REAL variable to receive the calculated value.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Example code generated by the ST1 Code template:

```
StudentsT = ST1(StatSetX,HypothesisMeanValue)
DISPLAY(?StudentsT)
```

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

SUMM

This Code template generates code to call the SUMM procedure. SUMM computes the sum of all of a data set's values.

The template prompts for the following parameters:

Data Set The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to analyze.

Summation Value The label of a REAL variable to receive the calculated value.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Example code generated by the SUMM Code template:

```
SummationValue = SUMM(StatSetX)
DISPLAY(?SummationValue)
```

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

UPPERQUARTILE

This Code template generates code to call the UPPERQUARTILE procedure. UPPERQUARTILE returns the median of the upper half of an ordered numeric data set. In other words, it returns a value such that at most 75% of a numeric set's values are less than the computed value, and at most 25% of the set's values are greater than the computed value.

In general, the UPPERQUARTILE procedure is only meaningful when the number of elements in the data set is large (ie. greater than 20). See also *PERCENTILE* and *LOWERQUARTILE*.

The template prompts for the following parameters:

Data Set The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to analyze.

Upper Quartile Value
The label of a REAL variable to receive the calculated value.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Example code generated by the UPPERQUARTILE Code template:

```
UpperQuartileOfSet = UPPERQUARTILE(StatSetX)
DISPLAY(?UpperQuartileOfSet)
```

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

VARIANCE

This Code template generates code to call the VARIANCEP procedure or the VARIANCES procedure. VARIANCEP computes the variance of an *entire population*. VARIANCES computes the variance of a *population sample*.

The template prompts for the following parameters:

Type Choose ‘Sample’ or ‘Population.’ ‘Sample’ is the default.

Data Set The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to analyze.

Mean of Data Set The label of a variable containing the average of the set’s values (optional).

Variance of Set The label of a REAL variable to receive the calculated value.

Display Return Value
Checking this box generates a DISPLAY statement for the return value.

Example code generated by the VARIANCE Code template:

```
VarianceOfSample = VARIANCES(StatSetX)
DISPLAY(?VarianceOfSample)
```

Note: The passed data set does not have to be sorted. The procedure copies the passed set. The passed data set is unchanged.

5 - BUSINESS MATH EXAMPLES

Overview

This chapter provides a look at the example application supplied with the Clarion Business Math Library. The example application demonstrates a real world implementation of many of the Business Math procedures. Some implementations are done with Code templates and others are done by hand code.

Exploring the Example Application

The example Business Math application is installed in ..\CLARION5\EXAMPLES\BUSMATH. Files comprising the example are:

BUSMATH.APP	Application File
BUSMATH.DCT	Dictionary File
BUSMATH.EXE	Executable Program
BUSMATH.TPS	Sample Data File

To run the example program, DOUBLE-CLICK on the BUSMATH.EXE file. Look at the options under the **Finance** menu and the **Statistics** menu. You will see the following business procedures:

Finance

Value of Money
This procedure presents a time value of money equation and solves for the variable you specify. This solution is implemented with conditional hand-coded calls to PV &PREPV, PERS & PREPERS, RATE & PRERATE, PMT & PREPMT, and FV & PREFV.

Interest Calculations
This procedure presents various types of interest calculations including simple interest, compound interest, continuously compounding interest, and annual percentage rate (APR). This solution is implemented using embedded Coded templates for each calculation.

Loan Amortization

This procedure presents loan terms (principal amount, interest rate, number of years, etc.) and generates a loan payment schedule, breaking down each payment into principal and interest components. This solution is implemented with hand coded repeating calls to the AMORTIZE procedure.

Cash Flow Analysis (BrowseInvestments)

This procedure presents calculates the internal rate of return and the net present value of an initial investment and its resulting income stream. This solution is implemented with hand-coded intitialization code followed by Code template calls to the NPV (Net Present Value) and IRR (Internal Rate of Return) procedures.

Statistics

Linear Regression Analysis (BrowseStatisticsPairedDataItems)

This procedure presents a linear regression analysis of paired coordinates, calculating the correlation coefficient and the sum of squares for the X coordinate, the Y coordinate, and for both. This solution is implemented with Code template calls to procedures SS, SSXY, and to RVALUE.

Data Set Analysis (BrowseStatisticsSingleDataItems)

This procedure presents various statistical indicators for a data set including mean, median, midrange, mode, standard deviation, variance, etc. This solution is implemented with Code template calls to the respective statistical procedures.

Classroom Data Example

This procedure presents the same information as the **Data Set Analysis** procedure as applied to a typical classroom grade book. This solution is also implemented with Code template calls to the respective statistical procedures.

PART II

DEVELOPMENT TOOLS

6 - DICTIONARY SYNCHRONIZER

Overview

The Dictionary Synchronizer synchronizes the open Clarion data dictionary with another Clarion data dictionary (*.DCT), a Clarion text dictionary (*.TXD), or a non-Clarion database such as Scalable SQL, ORACLE, MSSQL, or Btrieve.

This synchronization process saves time in several development contexts: developing new applications to serve legacy databases, updating existing applications to conform to a modified database, and sharing application development among several developers with incremental modifications to the database definition.

What the Synchronizer Does

Generally

Generally speaking, the Dictionary Synchronizer

- automates the conversion of existing data between different versions of your software;
- automates the synchronization of the “master” data dictionary or database with other project data dictionaries and databases and vice versa;
- creates a complete Clarion data dictionary, including relationships, from an existing database (SQL, Btrieve, etc.) in a single pass.

Specifically

More specifically, the Dictionary Synchronizer

- compares a Clarion data dictionary with another Clarion data dictionary or with a database (SQL or Btrieve) to identify any differences between them,
- resolves the differences (with your interactive input, or according to a prior synchronization) by proposing changes to the synchronized (target) dictionary or database,
- validates the proposed changes,
- backs up the synchronized (target) dictionary or data definition (Btrieve or Scalable .ddf) files,

- implements the proposed changes directly to the target dictionary (.dct, .txd, or .ddf), or generates an SQL script to implement the new SQL database,
- generates a Clarion program to implement the proposed changes to any existing data,
- saves pertinent information about the process for subsequent reuse.

This aids team development by automating the synchronization of a “master” dictionary or database with other project dictionaries and databases. It lets you automatically convert or upgrade existing data to higher or later versions of your software, which is a major benefit to both team and individual developers. Finally, the power of Dictionary Synchronizer applies not only to Clarion data dictionaries, but also to non-Clarion file systems and databases such as Oracle, MSSQL, Scalable SQL, Btrieve, etc., so you can create complete Clarion data dictionaries from existing databases in a single pass.

Synchronizer Servers

The Dictionary Synchronizer uses Synchronizer Servers to access data dictionaries and databases. A Synchronizer Server is a .dll that communicates with a database or file system such as ORACLE, MSSQL, Btrieve, etc.

Generally speaking, the Synchronizer Server does two things:

- During the dictionary comparison stage, it queries the dictionary or database and collects a complete, current description of the database’s files or tables, keys, views, relationships, objects, and properties.
- During the implementation stage, it generates and optionally executes the programs and database commands necessary to carry out any proposed changes.

There is a built in Synchronizer Server for Clarion data dictionaries, plus several separate Synchronizer Servers for various SQL and Btrieve databases. The Clarion Dictionary server handles all Clarion data dictionaries (regardless of the file driver), and need not be registered.

To register a Synchronizer Server, simply register the corresponding file driver with the Database Driver Registry. See *Clarion’s Development Environment—Database Driver Registry* in the *User’s Guide* for more information.

Tip: **Each Synchronizer Server calls its corresponding database driver to collect information from the backend database. Therefore, you can enable database driver logging to trace the Synchronizer Server calls. See *Debugging File I/O* in the *Programmer’s Guide.***

Running the Synchronizer

To run the Dictionary Synchronizer, first, be sure to register the appropriate database driver (see *Synchronizer Servers* for more information). Next, open your Clarion data dictionary, then press the **Synchronize** button. This starts the Synchronizer Wizard which leads you step-by-step through the synchronization process. The process requires that you

- identify the *other* dictionary or database with which to synchronize,

> **Tip:** **To synchronize an SQL database, you must have read privileges for the SQL database's system tables—except for security or access tables (user ids, passwords, privileges, etc.)**

- specify dictionary to synchronize (the target dictionary),
- tell the synchronizer *what* to synchronize (all files or some subset of files),
- tell the synchronizer *how to match* files, fields, keys, and relationships (by name, by order, manually, or the same as the last time you synchronized these two dictionaries),
- tell the synchronizer *how to resolve differences* between the dictionaries (add, change, delete, ignore, etc.).

Dictionary Synchronizer Tutorial

This tutorial introduces you to the Clarion Dictionary Synchronizer. It takes you through all the steps needed to synchronize two databases—both the data and the data definitions (.dct, .ddf, or SQL script).

This synchronization process saves time in several development contexts: developing new applications to serve legacy databases, updating existing applications to conform to a modified database, and sharing application development among several developers with incremental modifications to the database definition.

Tutorial Objectives

This tutorial introduces you to the Clarion Dictionary Synchronizer. When you have finished the tutorial you should be able to

- Consolidate two Clarion data dictionaries
- Run the Synchronizer Wizard
- Convert existing data to a new format
- Create and review a Dictionary Synchronizer log
- Create a Clarion Data Dictionary from a Btrieve or SQL database

Note: To complete the tutorial you will need a copy of the files from the Clarion ..\Tutor\ SQL\ folder. Please copy the files to a separate folder called \SyncTutr.

Consolidate Two Clarion Data Dictionaries

In a team development environment, the Dictionary Synchronizer automates the synchronization of the “master” dictionary or database with other project dictionaries and databases.

This section of the tutorial shows how to consolidate two similar but different Clarion data dictionaries into a single dictionary. In this example, the \Tutorial.dct represents the “master” dictionary and the Qwktutor.dct represents another dictionary used by the development team. The two dictionaries must be consolidated into a single dictionary that all team members can use to continue to develop their application components.

By using the Dictionary Synchronizer to update the Qwktutor.dct (as opposed to simply replacing it with a copy of the Tutorial.dct), you are able to see exactly which dictionary items changed and you can intelligently manage those changes; if both dictionaries have changed, you can incorporate the changes from *both* dictionaries by synchronizing each

dictionary with the other; and finally, the Dictionary Synchronizer can generate a Clarion program to convert your existing data to the new format.

Note: **Whenever you consolidate two Clarion dictionaries into a single dictionary, you risk damaging the relationship between the replaced dictionary and its corresponding applications, unless you first export those applications to text (.TXA) format. Using Version Control accomplishes this without the need of explicitly creating a .TXA file, but this tutorial shows you how to accomplish the same purpose without Version Control.**

Make a Tutorial.TXA File

Make a Tutorial.TXA file that you can use to create a Tutorial.app that will work with the consolidated dictionary.

Starting Point:
The Clarion5\Synctutr\Tutorial.app file should be open.

1. Choose **File ➤ Export Text**, then press the **Save** button to create Tutorial.txa.
2. Press the **OK** button to close the \SyncTutr\Tutorial.app.

   This makes the \SyncTutr folder the current working directory. This has the benefit of making it the default folder for file selections and for creating new files such as new .TXDs and new .DCTs.

Run the Synchronizer Wizard

The Synchronizer Wizard leads you step-by-step through the synchronization process. The wizard lets you select the Synchronizer Server, the data dictionaries, the dictionary to synchronize (identify source and target), and set rules for matching items (files, fields, keys, etc.) between the dictionaries.

1. Open *Qwktutor.dct*.

   Every synchronization requires an open Clarion data dictionary (although the dictionary may be a newly created one that is completely empty).

   Tip: **When synchronizing two Clarion data dictionaries, it doesn't matter which dictionary you open first.**

2. Press the **Synchronize** button to start the Synchronizer Wizard (save the changes).
3. Move the **Prompt Detail** slider to the **More** position.

Select the Synchronizer Mode

1. CLICK on the **Next >** button to start a new synchronization.

Tip: **This page lets you rerun a prior synchronization.**

Tip: **This page lets you generate a data conversion program without modifying a data dictionary.**

Select the Synchronizer Server

The Clarion Dictionary Server is the default (**Synchronize With**) server. A Synchronizer Server is a .dll that lets the Dictionary Synchronizer access a Clarion data dictionary or a database/file system such as ORACLE, MSSQL, Btrieve, etc. The Clarion Dictionary server handles all Clarion data dictionaries (regardless of the file driver).

Select the Other Clarion Dictionary

1. Press the ellipsis (...) button to select the “other” database or dictionary with the Windows file dialog.

Note: **If you choose an SQL or Btrieve Synchronizer Server, the Synchronizer Wizard prompts you for the logon information needed to connect to the SQL database or for the folder containing the Btrieve .DDF files.**

2. Select the Tutorial.dct, then press the **Open** button.

Set the Synchronizer Log File

The Synchronizer saves details about each synchronization so you can automatically repeat a synchronization as often as you need to. You can use the **Log Synchronization To** field to name the log file that stores the

synchronization information. The default log pathname is ..*currentdirectory**currentdictionary*.log. For now, simply accept the default pathname.

3. CLICK on the **Next** button to continue.

Set the Dictionary to Synchronize (source and target)

1. CLICK on the **From ..\Tutorial.dct To Qwktutor.dct** radio button to make Qwktutor.dct the target dictionary.

   The Dictionary Synchronizer only modifies the target dictionary. Specify which dictionary is the source and which is the target by CLICKING the appropriate radio button.

   The warning is simply a reminder that for any applications that use a dictionary that will be replaced (copied over), you should export those applications to text (.txa) format before you replace (copy over) the

dictionary. A .txa is not necessary if you are simply synchronizing two dictionaries, both of which will survive the synchronization.

2. CLICK on the **Next** button to continue.

Select the Source Files to Synchronize

Select the files, views, and tables within the Clarion data dictionary to include in the synchronization process. The Synchronizer Wizard provides two file lists for the dictionary: on the left is a list of files *not* synchronized and on the right is a list of files *to* synchronize. Depending on the database server, the list may show filenames, table names, owner names, etc.

1. CLICK on the **>** button to select the Customer file *and related files*.

2. CLICK on the **Next** button.

3. CLICK on the **>>** button to move all the files from the left to the right.

This synchronize all file from Tutorial.DCT to QwkTutor.DCT.

4. CLICK on the **Next** button to continue.

Set the Synchronizer Options

1. Clear the **Generate Data Conversion Program** box.

   The Dictionary Synchronizer options control how the Synchronizer *begins* the synchronization. You may change these settings later on in the process (in the **Synchronize Dictionaries** dialog). See *Dictionary Synchronizer Options* for more information on these settings.

Tip: This page lets you request a data conversion program to convert existing data to the new format.

Tip: The Matchings button sets rules for matching dictionary items. The default rules match primarily on file and field names.

2. CLICK on the **Finish** button to accept the defaults and continue.

Synchronize Dictionaries Dialog

When you “finish” the **Synchronizer Wizard**, it loads, analyzes, and compares the two dictionaries, then opens the **Synchronize Dictionaries** dialog where you complete the synchronization process.

The **Synchronize Dictionaries** dialog compares the two data dictionaries, item by item and lets you resolve any differences between them.

1. CLICK the *Dictionary* item (first item in the list) to select it.

   Actions you apply to the selected item cascade to all its children.

   The **Synchronize Dictionaries** dialog uses a "file centric" hierarchical list to compare the dictionaries. Beneath each item in the list, the dialog nests the item's properties and components (files, fields, keys, relationships, and aliases).

   The **Synchronize Dictionaries** dialog uses **color** to provide information about the synchronization process:

   Blue
   indicates no action is applied, and the item is either different than its matching item or it has no matching item.

   Black
   indicates an action is applied and the resulting target item is valid, or no action is applied but none is needed because the items already match and the target item is valid.

   Red
   indicates an action is applied and the resulting target item is not valid—the target does not support the applied property. Red may also indicate an invalid condition in the source dictionary—you can clean up the source dictionary and resynchronize, or you can ignore the invalid item and complete the current synchronization.

   Gray
   The item is not supported by the target dictionary.

Tip: **Use the navigation buttons and the corresponding navigation menu items to quickly locate new, changed, or invalid dictionary components.**

The **Synchronize Dictionaries** dialog uses **action indicators** to show the action applied to each item. Action indicators are at the left side of the **Destination** (target) column. The question mark (?) indicates no decision as yet.

> **Tip:** **A gray action indicator indicates a cascaded (or inherited) action—the action was originally applied to a parent (full color indicator) and the child has inherited the action (gray indicator).**

2. Choose **Synchronize ➤ Copy to** from the menu to apply the action to the selected items.

   The action cascades to all children of the selected item—that is, to all items in the dictionary.

   When you select **Copy to**, the Action Indicator changes to a right arrow (➡) for matched items, and to an exclamation point (!) for unmatched items.

   The right arrow (➡) indicates the synchronizer will *copy the item's properties* from the source to the target.

   The exclamation point (!) indicates the synchronizer will *ignore* the suggested action for unmatched items. This is because the **Copy to** action *only changes existing items* in the target dictionary—it does not add new items.

3. Choose **Edit ➤ Undo** from the menu to undo the **Copy to** action.

   The Action Indicator reverts to the question mark (?) — indicating no decision.

> **Tip:** **Undo may be applied multiple times. It undoes actions in LIFO (Last In First Out) order.**

4. RIGHT-CLICK the *Dictionary* item (first item in the list), then choose **Merge Into** from the popup menu.

You can use either the **Synchronize** menu or the popup menu to apply synchronizer actions.

When you select **Merge Into**, the Action Indicator changes to a right arrow (➡) for matched items, and to a plus sign (✚) for unmatched items. A multi-item indicator on a parent item ( ) indicates different actions for different children. For example, a file may contain both matching fields (right arrow) and new fields (plus sign).

The right arrow (➡) indicates the synchronizer will *copy the item's properties* from the source to the target.

The plus sign (✚) indicates the synchronizer will *add the marked items* to the target dictionary. This is because the **Merge Into** action not only changes existing items in the target dictionary, it also *adds items*.

Tip: **A gray action indicator indicates a cascaded (or inherited) action—the action was originally applied to a parent (full color indicator) and the child has inherited the action (gray indicator).**

5. Press the **OK** button to complete the synchronization and update the Qwktutor.dct.

   The Dictionary Editor dialog shows the updated Qwktutor.dct.

6. Press the button to save the Qwktutor.dct.

   The Qwktutor.dct now matches the Tutorial.dct.

7. Press the **Close** button to close the Qwktutor.dct.

   This Qwktutor.dct now contains all the same information (properties, files, fields, and relationships) as the Tutorial.dct; however, these two .dct files are not identical. The Qwktutor.dct will work with the Qwktutor.app, and the Tutorial.dct will work with the Tutorial.app, but the converse is not true—the dictionaries are not interchangeable because their internal file, field, and key numbers may be different.

Remake the Tutorial.app File

Starting Point:
The Clarion environment should be open.

1. Choose **File ➤ New ➤ Application**.
2. Select Tutorial.app from the file list, clear the **Use Quick Start** box, then press the **Save** button.
3. Press the **Yes** button when prompted to **Replace existing file?**.

   This opens the **Application Properties** dialog.
4. Press the **Dictionary File** ellipsis button (...) to select the updated Qwktutor.dct.
5. Clear the **Application Wizard** box, then press the **OK** button.

   This opens the **Application Tree** dialog.
6. Choose **File ➤ Import Text**, select *Tutorial.txa* from the list, then press the **Open** button.
7. Press the **Replace All** button when prompted for **Procedure name clash**.

NOTE: An Error Editor dialog warns you that the .TXA specifies a different dictionary than the application. Since we just synchronized the two DCT files, this warning can be safely ignored.

8. Press the **Close** button to ignore the warning.

   You can now make and run the Tutorial application using the synchronized Qwktutor.dct, and you can make and run the Qwktutor application with the same Qwktutor.dct!

Summary

When you replace a data dictionary, you must take the following steps to preserve the internal relationships between the application and the data dictionary: first, create a text application file (.TXA), then update the data dictionary, finally create a new application (.APP) from the text application file (.TXA).

By using the Dictionary Synchronizer to update the Qwktutor.dct (as opposed to simply replacing it with a copy of the Tutorial.dct), you are able to see exactly which items changed and you can intelligently manage those changes; if both dictionaries have changed, you can incorporate the changes from *both* dictionaries by synchronizing each dictionary with the other; finally, the Dictionary Synchronizer can generate a Clarion program to convert your existing data to the new data format.

Review Dictionary Synchronizer Log

The Dictionary Synchronizer creates a log to help with the debugging process and to allow batch synchronizations (see *Batch Synchronizations* for more information).

For every synchronization, the Dictionary Synchronizer creates a log file with the pathname you specify. This log serves two purposes: it can be used to rerun a synchronization in batch mode (see *Batch Synchronizations*), and it should be submitted with any bug reports along with the synchronized data dictionaries to aid in the debugging process.

The synchronization from the first part of this tutorial created a log file ..\SyncTutr\Qwktutor.log. If you browse this file you will see the following:

```
CMD(DCT(Clarion Dictionary-Qwktutor.dct))
CMD(DCT(Clarion Dictionary-C:\SyncTutr\tutorial.dct))
CMD(MATCH_Files(MatchByName))
CMD(MATCH_Fields(MatchByName))
CMD(MATCH_Keys(MatchByComponent))
CMD(MATCH_Alias(MatchByName))
CMD(SortFiles(TRUE))
CMD(SortFields(FALSE))
CMD(SortKeys(FALSE))
CMD(Hide(TRUE))
CMD(Convert(TRUE))
CMD(AutoCorrect(FALSE))
CMD(CWSource(FALSE))
CMD(StopAfterMatch(FALSE))
CMD(StopBeforeLastCmd(TRUE))
FILTER(CLARION)
Add            ,FullCluster    ,Customer
END_FILTER
FILTER(OTHER)
Add            ,FullCluster    ,Customer
END_FILTER
CMD(OP_Copy(FALSE,Dictionary,C:\SyncTutr\tutorial.dct,Object))
CMD(EVENT(Undo))
CMD(OP_Merge(FALSE,Dictionary,C:\SyncTutr\tutorial.dct,Object))
CMD(EVENT(OK))
```

Note: **For each synchronization that fails, the Clarion environment tries to create a log called C5log.txt in the current directory. Please submit this log, the synlog.txt file, and the synchronized data dictionaries and with your bug reports.**

Tip: **Each Synchronizer Server (Oracle, MSSQL, Pervasive, etc.) calls the corresponding database driver to collect information from the database. Therefore, you can also enable database driver logging to trace the Synchronizer Server calls. See *Debugging and Tracing File I/O* in the *Programmer's Guide*.**

Convert Existing Data to New Format

The Dictionary Synchronizer optionally generates a Clarion program to convert existing data to the new format.

Starting Point:
Open the \Synctutr\Tutorial.dct with the Clarion environment.

Add a Field to the Tutorial Dictionary

1. RIGHT-CLICK the Customer file, then choose **Fields/Keys** from the popup menu.
2. Select the *FirstName* field, then press the **Insert** button.
3. In the **Field Name** field, type *MiddleName*, then press the **OK** button.
4. Press the **Cancel** button to stop inserting fields.
5. Press the **Close** button to close the **Fields/Keys** dialog.
6. Press the button to save the Tutorial.dct.

Synchronize the Qwktutor Dictionary

1. Press the **Synchronize** button to start the Synchronizer Wizard.
2. Press the **Next** button to start a new synchronization.
3. Press the ellipsis (...) button to select the Qwktutor.dct, then press the **OK** button.
4. CLICK on the **Next** button to continue.
5. CLICK on the **Next** button to make Qwktutor.dct the target dictionary.

Set Options to Generate Conversion Program

1. Make sure the **Generate Data Conversion Program** box is checked.
2. CLICK on the **Next** button to continue.
3. CLICK on the **>** button to select all the Tutorial.dct files.
4. CLICK on the **Next** button to continue.
5. CLICK on the **>** button to select all the Qwktutor.dct files.
6. CLICK on the **Finish** button to open the **Synchronize Dictionaries** dialog.

Finish the Synchronization and Generate the Program

1. In the **Synchronize Dictionaries** dialog, RIGHT-CLICK the *Dictionary* item (first item in the list), then choose **Add to** from the popup menu.

   This adds all new items (MiddleName field) to the Qwktutor.dct.

2. Press the **OK** button to complete the synchronization, update the Qwktutor.dct, *and generate the data conversion program*.

   The Synchronizer generates the conversion program components into the working directory (c:\SyncTutr). The conversion program consists of the following components:

C5cvt___1.clw	file declarations, procedure definitions, field assignments
Convert.clw	program map, class declarations, initialization
Convert.prj	project file to compile and link the program

3. Press the **Close** button to close the Tutorial.dct.

Make and Run the Conversion Program

1. Open the Convert.prj (choose **File ➤ Open**, select *Project (*.prj)* from the **Files of type** drop list, then DOUBLE-CLICK Convert.prj.

2. Press the button to make and run the conversion program.

   The conversion program lets you select the specific data files to convert to the new format. The conversion program optionally backs up the files you select to the backup folder you specify.

   The conversion program handles new files as well as changed files. This gives you the framework to create and prime new files or to split one file into two new related files. You can edit the C5cvt___1.clw source code to prime records and make custom field assignments as needed.

3. Press the **Proceed** button to convert the Customer file, or press the **Cancel** button to abort the conversion.

Create Btrieve (or SQL) Database from a Clarion Dictionary

The Dictionary Synchronizer can create an entire Data Dictionary, including relationships, from an existing database in a single pass. Conversely, it can create a database from a Clarion Data Dictionary. This part of the tutorial shows you how to create a database from a Clarion Data Dictionary.

In the SQL case the Synchronizer generates a SQL script you can run to create the database; in the Btrieve case, the Synchronizer actually creates the ddf files that define the database.

Everything you need to complete this part of the tutorial using Btrieve comes on your CD.

If you have Windows NT or Netware 3.12, then you can also complete this section using Scalable SQL with only the components on your CD.

You can apply this part of the tutorial to your preferred SQL database (MSSQL, Oracle, SQLAnywhere, or AS/400) by replacing the Btrieve references with corresponding operations for you SQL server. However, these SQL databases require components that are not on your CD.

Create a Btrieve Database Definition

Starting Point:
The \SyncTutr\Tutorial.dct is open and the Btrieve database driver is registered.

1. Press the **Synchronize** button to start the Synchronizer Wizard.
2. CLICK on the **Next >** button to start a new synchronization.
3. Choose *Btrieve* from the **Synchronize With** drop list.

Set the Btrieve DDF Destination Folder

1. Press the **Select other dictionary** ellipsis button (...) to select the folder that will contain the Btrieve data definition files (.DDF).

   This opens the **Select Dictionary** dialog (a Windows file dialog).

   In the SQL case, specify the connection information in the **Host String, User Name, and Password** fields, then press the **OK** button.

2. In the **File name** field, type *dummy.ddf*, then press the **OK** button.

   This sets the folder name that will contain the Btrieve data definition files (.DDF). Alternatively, you may select an existing .DDF file in the destination folder, or simply type the pathname to use.

3. CLICK on the **Next** button to continue.

Set the Synchronizer Source and Target

1. CLICK on the **From ..\Tutorial.dct To \SYNCTUTR** radio button to make \SyncTutr the target folder for the .DDF files.

   Because there is no Btrieve database in the \SyncTutr folder, only one radio button is enabled.

2. CLICK on the **Next** button to continue.

Set the Synchronizer Options

3. CLICK on the **Next** button to accept the default synchronizer options and continue.

Set the Files to Synchronize

4. CLICK on the **>>** button to include all the Tutorial.dct files for synchronization.

5. CLICK on the **Next** button to continue.

6. CLICK on the **Finish** button to open the **Synchronize Dictionaries** dialog.

Add the .Dct Information to the .DDF Files

1. In the **Synchronize Dictionaries** dialog, RIGHT-CLICK the *Dictionary* item (first item in the list), then choose **Merge Into** from the popup menu.

   The Action Indicator changes to a plus sign (✚) for all items. The plus sign (✚) indicates the synchronizer will *add the marked items* to the target database.

5. Press the **OK** button to complete the synchronization and create the Btrieve database definition.

   In the Btrieve case, the Synchronizer writes .ddf files to the path you specified.

> **Tip:** **In the SQL case, the Synchronizer writes an SQL script (an .sql file instead of a .dct or .ddf file). The Synchronizer prompts you for the pathname of the script file. Type *Tutorial.sql* in the Filename field of the Windows save file dialog.**

The script (..\Synctutr\tutorial.sql) looks something like the following:

```
CREATE TABLE “Customer”(
   “CustNumber” INT(4),
   “Company” CHAR(20),
   “FirstName” CHAR(20),
   “MiddleName” CHAR(20),
   “LastName” CHAR(20),
   “Address” CHAR(20),
   “City” CHAR(20),
   “State” CHAR(2),
   “ZipCode” INT(4));
SET DEFAULT “Customer”.”State” = ‘’FL’’;
SET VALUES “Customer”.State = ‘AL’, ‘MS’, ‘FL’, ‘GA’, ‘LA’, ‘SC’;
CREATE UNIQUE INDEX “KeyCustNumber” ON “Customer”(“CustNumber” MOD NULL);
CREATE INDEX “KeyCompany” ON “Customer”(“Company” MOD);
CREATE INDEX “KeyZipCode” ON “Customer”(“ZipCode” MOD);

CREATE TABLE “FieldPool”(
   “CustNumber” INT(4),
   “OrderNumber” INT(2),
   “ProdNumber” INT(2));
SET VALUES “Customer”.State = ‘AL’, ‘MS’, ‘FL’, ‘GA’, ‘LA’, ‘SC’;

CREATE TABLE “Orders”(
   “CustNumber” INT(4),
   “OrderNumber” INT(2),
   “InvoiceAmount” DECIMAL(4,2),
   “OrderDate” INT(4),
   “OrderNote” CHAR(80),
   PRIMARY KEY (“OrderNumber”));
SET VALUES “Customer”.State = ‘AL’, ‘MS’, ‘FL’, ‘GA’, ‘LA’, ‘SC’;
CREATE INDEX “KeyCustNumber1” ON “Orders”(“CustNumber” MOD NULL);

CREATE TABLE “Detail”(
   “OrderNumber” INT(2),
   “ProdNumber” INT(2),
   “Quantity” INT(2),
   “ProdAmount” DECIMAL(3,2),
   “TaxRate” DECIMAL(1,2));
SET VALUES “Customer”.State = ‘AL’, ‘MS’, ‘FL’, ‘GA’, ‘LA’, ‘SC’;
CREATE INDEX “KeyProdNumber” ON “Detail”(“ProdNumber” MOD NULL);
CREATE INDEX “KeyOrderNumber” ON “Detail”(“OrderNumber” MOD NULL);
```

```
CREATE TABLE “Products”(
   “ProdNumber” INT(2),
   “ProdDesc” CHAR(25),
   “ProdAmount” DECIMAL(3,2),
   “TaxRate” DECIMAL(1,2),
   PRIMARY KEY (“ProdNumber”));
SET VALUES “Customer”.State = ‘AL’, ‘MS’, ‘FL’, ‘GA’, ‘LA’, ‘SC’;
CREATE INDEX “KeyProdDesc” ON “Products”(“ProdDesc” MOD NULL);

CREATE TABLE “Phonestry”(
   “PhoneID” INT(2),
   “CustNum” INT(4),
   “Area” INT(4),
   “Phone” INT(4),
   “Description” CHAR(20),
   PRIMARY KEY (“PhoneID”));
SET VALUES “Customer”.State = ‘AL’, ‘MS’, ‘FL’, ‘GA’, ‘LA’, ‘SC’;
CREATE INDEX “KeyCustNumber2” ON “Phonestry”(“CustNum” MOD NULL);

ALTER TABLE “Phonestry” ADD
   FOREIGN KEY “KeyCustNumber2” (“CustNum”)
   REFERENCES “Customer”;

ALTER TABLE “Orders” ADD
   FOREIGN KEY “KeyCustNumber1” (“CustNumber”)
   REFERENCES “Customer”;

ALTER TABLE “Detail” ADD
   FOREIGN KEY “KeyOrderNumber” (“OrderNumber”)
   REFERENCES “Orders”;

ALTER TABLE “Detail” ADD
   FOREIGN KEY “KeyProdNumber” (“ProdNumber”)
   REFERENCES “Products”;
```

Create a Clarion Data Dictionary from an SQL Database

The Dictionary Synchronizer can create an entire Data Dictionary from an existing database in a single pass. Conversely, it can create a database from a Clarion Data Dictionary. This part of the tutorial shows you how to create a Clarion Data Dictionary from an existing Oracle database.

Tip: **To synchronize an SQL database, you must have read privileges for the database system tables—except for security or access tables (user ids, passwords, privileges, etc.)**

Tip: **If you do not have access to an Oracle database, you can substitute another SQL database and driver instead (AS/400, MSSQL, Scalable SQL, SQLAnywhere, or ODBC datasource).**

Create an Empty Dictionary

Starting Point:
The Clarion Environment is open, the Oracle database driver is registered, and the Oracle database is started.

1. Choose **File ➤ New ➤ Dictionary**.

   This opens the Windows file dialog.

2. In the **File name** field, type OraDemo, then press the **Save** button.

   This opens the **Dictionary** dialog to the empty data dictionary.

3. Press the button to save the empty OraDemo.dct.
4. Press the **Synchronize** button to start the Synchronizer Wizard.
5. CLICK on the **Next >** button to start a new synchronization.
6. Choose *Oracle* from the **Synchronize With** drop list.

Select the Oracle Database

1. Press the ellipsis button to select the Oracle database.
2. Specify the connection information in the **Host String, User Name, and Password** fields, then press the **OK** button.

   This tutorial uses the Oracle Demo database supplied with Personal Oracle. The host is local (2:), the Username is DEMO, and the Password is DEMO.

Tip: **The host for later versions of Personal Oracle (7.3 and 8) is blank (not 2:).**

The default database filter excludes system tables from the synchronization process. Generally your applications do not need access to system tables.

3. CLICK on the **Next** button to continue.

Set the Synchronizer Source and Target

1. CLICK on the **From ..Demo@2 To ..\OraDemo.dct** radio button to make OraDemo.dct the target dictionary.
2. CLICK on the **Next** button to continue.
3. CLICK on the **Next** button to accept the synchronizer options and continue.

Select the Tables to Synchronize

1. CLICK on the **Next** button to select files and continue.
2. CLICK on DEMO.CUSTOMER, then CLICK on the **>** button to select the Customer table and all related tables.

3. CLICK on the **Finish** button to continue.

   The synchronizer collects information from the Demo database.

4. In the **Synchronize Dictionaries** dialog, RIGHT-CLICK the *Dictionary* item (first item in the list), then choose **Merge Into** from the popup menu.

The Action Indicator changes to a plus sign (✚) for all items. The plus sign (✚) indicates the synchronizer will *add the marked items* to the target data dictionary.

5. Press the **OK** button to complete the synchronization and complete the OraDemo.dct dictionary.

   The Dictionary dialog shows the completed OraDemo.dct dictionary, including file relationships.

6. Press the **Close** button to close and save the OraDemo.dct file.

7. Run the Application Wizard against the OraDemo.dct file to create a complete application to access the Oracle database.

Synchronizer Wizard

To run the Synchronizer Wizard, open your Clarion data dictionary, then press the Synchronize button. This starts the Synchronizer Wizard which leads you step-by-step through the synchronization process. This process requires that you

- identify the appropriate Synchronizer Server,
- identify the other dictionary or database with which to synchronize,
- identify the synchronizer file which contains information about the prior synchronization of these two dictionaries,
- specify the dictionary to synchronize (identify source and target),
- identify what portion of the dictionaries to synchronize (all files or some subset of files),
- tell the synchronizer how to identify matching files, fields, keys, and aliases (by name, by order, manually, or by history)

Starting the Synchronizer

To run the Synchronizer Wizard, open your Clarion data dictionary, then press the Synchronize button.

Every synchronization requires an open Clarion data dictionary (although the dictionary may be a newly created one that is completely empty).

Tip: When synchronizing two Clarion data dictionaries, it doesn't matter which dictionary you open.

Select the Synchronizer Mode

Start New Synchronization
Select this option to run the Synchronizer in interactive mode.

Rerun Previous Synchronization
Select this option to run the Synchronizer in a controlled batch mode using input saved from a prior synchronization.

Generate Conversion Program Only
Check this box to generate a conversion program without updating .dct or .ddf files.

Next
Press this button to proceed to the next Wizard page.

Select the Other Dictionary

Synchronize With
Choose the Synchronizer Server from the drop list.

The Dictionary Synchronizer uses Synchronizer Servers to access data dictionaries and databases. A Synchronizer Server is a .dll that communicates with a database or file system such as ORACLE, MSSQL, Btrieve, etc.

There is a built in Synchronizer Server for Clarion data dictionaries, plus several separate Synchronizer Servers for SQL and Btrieve databases. The Clarion Dictionary server handles all Clarion data dictionaries (regardless of the file driver), and need not be registered.

To register a Synchronizer Server, simply register the corresponding file driver with the Database Driver Registry. See *Clarion's Development Environment—Database Driver Registry* in the *User's Guide* for more information.

Select other dictionary

Press the ellipsis button to select the “other” database or dictionary with the Windows file dialog. If the data dictionary resides in several files (such as a set of Btrieve .DDFs), then select the directory that contains the definition files.

For SQL databases (such as ORACLE, SQLAnywhere, or MSSQL), the Synchronizer Wizard prompts you for the logon information needed to connect to the SQL database.

Tip: To synchronize an SQL database, you must have read privileges for the SQL database’s system tables—except for security or access tables (user ids, passwords, privileges, etc.)

Log Synchronization To

The Previous Synchronization log (*.log) stores information about the synchronization of two particular dictionaries. Specifically, it stores the files to synchronize from the source dictionary, the files to synchronize from the target dictionary, and a list of matching items (files, fields, keys, and aliases) between the two dictionaries.

If you wish to save the synchronization information, or if you wish to reuse the information from a prior synchronization, you may specify a filename for the Previous Synchronization log.

Press the ellipsis button to select a Previous Synchronization log with the Windows file dialog, or type the Previous Synchronization log pathname.

Previous

Press this button to return to the prior Wizard page.

Cancel

Press this button to abandon the synchronization.

Set the Dictionary to Synchronize (source and target)

Directions

The Dictionary Synchronizer never modifies the source dictionary; it only modifies the target dictionary.

Specify whether the open data dictionary is the source or the target by CLICKING the appropriate radio button.

The warning is simply a reminder that for any applications that use a dictionary that will be replaced (copied over), you should export those applications to text (.txa) format before you replace (copy over) the dictionary. A .txa is not necessary if you are simply synchronizing two dictionaries, both of which will survive the synchronization.

Set the Synchronizer Options

These wizard synchronizer options operate exactly the same as the global synchronizer options described in the *Dictionary Synchronizer Options* section. Changing the settings here automatically updates the global settings and vice versa. See the *Dictionary Synchronizer Options* section for more information on these prompts.

CLICK on the **Matchings** button to tell the synchronizer how to match files, fields, keys, and aliases between the two dictionaries.

Tip: Proper matching of dictionary items is essential to producing a useful synchronization.

Set Automatic Matching Rules

Proper matching of dictionary items is essential to produce a useful synchronization. The synchronizer provides a range of automatic, intelligent

matching choices, which you can combine with manual matchings to quickly and accurately match your dictionary items.

Any items the synchronizer cannot match automatically are flagged within the **Synchronize Dictionaries** dialog so you can manually match them. See *Synchronize Dictionaries Dialog* for more information.

Historically
The synchronizer matches the items based on the Previous Synchronization log you specified earlier. This choice is disabled if you specified a new Previous Synchronization log or no Previous Synchronization log.

By Name Only
The synchronizer matches the items based on their labels and external names.

By Order Only
The synchronizer matches the items based on the order they appear in the two dictionaries.

This option is not available for SQL databases.

By Component Only
The synchronizer matches keys based on the number and labels or external names of the key components.

This option is not available for SQL databases.

Note: The availability of the matching rules depends on the particular database server.

Manually
The synchronizer does not match items. You must match the items with the **Synchronize Dictionaries** dialog.

CLICK on the **OK** button to return to the Options page, then CLICK on the **Next** button to continue.

> **Note:** **Pressing the finish button at this point matches the dictionaries according to these settings. For any manually matched items, all sub-items are matched using these settings. For example, if you match a file manually, all its fields are matched by name if By Name Only is specified.**

Specify Files or Tables to Synchronize

Select which files, views, and tables within the Clarion data dictionary to include in the synchronization process. The Synchronizer Wizard provides two file lists for the data dictionary: on the left is a list of files *not* synchronized and on the right is a list of files *to* synchronize. Depending on the database server, the list may show filenames, table names, owner names, etc.

It is to your advantage to include as few files as possible in the synchronization process because fewer files results in a faster process.

> Press this button to move the highlighted file *and all related files and aliases* to the **Files to Synchronize** list. Alternatively, DOUBLE-CLICK the file to synchronize.

>> Press this button to move *all* the files to the **Files to Synchronize** list.

< Press this button to move the highlighted file *and all related files* to the **Files not Synchronized** list.

<< Press this button to move *all* the files to the **Files not Synchronized** list.

Clustering
This option is only available if you chose **Multi-File Import**. It determines whether related files move together as a group.

None
Select this to manipulate all files individually.

Partial
Select this to treat files with a two-way relationship as a single entity—move one, move all. Files with a one-way relationship are moved individually.

Full
Select this to treat files with any relationship as a single entity—move one, move all.

Tip: **The Synchronizer Wizard lists the files in the order specified on the Options page, that is, alphabetically or according to the source dictionary.**

Next

CLICK on the **Next** button to select which files within the other database/dictionary to include in the synchronization process. This page works exactly like the previous file selection page, with one exception: if the other dictionary is a not a Clarion dictionary, loading and updating the file list takes significantly longer than for a Clarion dictionary.

Finish

Selecting the files to synchronize enables the **Finish** button.

The **Finish** button completes the wizard and opens the **Synchronize Dictionaries** dialog, which does the item by item comparison, identifies and (interactively) resolves any differences between the data dictionaries.

The **Finish** button does not implement changes to the target dictionary, it simply starts the comparison process.

Synchronize Dictionaries Dialog

When you “finish” the **Synchronizer Wizard**, it opens the **Synchronize Dictionaries** dialog.

The **Synchronize Dictionaries** dialog compares the two data dictionaries, item by item and lets you resolve any differences between them. It validates the proposed changes, then generates all the source code, scripts, and data definitions required to implement the changes. Finally, it gives you the opportunity to review and edit the generated source code or SQL scripts.

Overview

The **Synchronize Dictionaries** dialog uses a “file centric” hierarchical list to present the comparison. That is, beneath each file in the list, the **Synchronize Dictionaries** dialog nests the file’s properties and components (fields, keys, relationships, and aliases). Beneath each item in the list, the **Synchronize Dictionaries** dialog nests that item’s respective properties and components.

> **Tip:** CLICK **on the plus (+) sign to expand a list item;** CLICK **on the minus (-) sign to contract an item.**

For every file, field, key, relationship, alias, and property in the list, the **Synchronize Dictionaries** dialog shows the value from each database or dictionary, plus status indicators to show whether the values are different, and how the difference is resolved (for example, ignore the difference, copy from source to target, delete item from target, not yet resolved, etc.).

The **Synchronize Dictionaries** dialog either dims or omits database properties the target database/dictionary does not support, depending on the settings in the **Synchronizer Options** dialog (see *Dictionary Synchronizer Options*).

Generally, follow these steps to synchronize the dictionaries:

1. Configure the **Synchronize Dictionaries** dialog to sort dictionary items according to your preferences—alphabetically or according to the source dictionary (use the **View** menu).
2. Navigate to each unmatched item and make sure each one is properly matched to the corresponding item in the other dictionary by selecting a matching option from the **Synchronize** menu or the popup (RIGHT-CLICK) menu. Items may be left unmatched if there is no corresponding item in the other dictionary.
3. Navigate to each difference and resolve it by selecting an option from the **Synchronize** menu or the popup (RIGHT-CLICK) menu.
4. Press the **OK** button.

Configuring the Synchronize Dictionaries Dialog

By default, the **Synchronize Dictionaries** dialog displays files, fields, and keys in the order specified in the **Synchronizer Options** dialog. You can also set default colors and other behaviors with this dialog (see *Dictionary Synchronizer Options*). However, you can reset the various sort orders (files, fields, keys) at any time with the **View** menu.

Status Indicators

The **Synchronize Dictionaries** dialog uses status indicators as well as color to provide information about the current synchronization process.

Status Indicators

Status indicators are in the right-most column of the list. Status indicators include:

- 32-bit only. This indicates the file's record buffer exceeds 64k bytes making the file unsuitable for 16-bit applications.
- File Structure (field order) changed. This indicates a field's *position* has changed within the target dictionary. This indicator is especially useful when fields are sorted alphabetically and their physical order is not readily visible.
- Invalid proposal below. This indicates there is an invalid proposal further down in the hierarchy. Expand the tree or use the navigation controls to find the violation.

Invalid proposal. This indicates the proposed change cannot be accepted by the target server and disables the **OK** button. For example, number of characters not valid for current data type.

RIGHT-CLICK on this icon then choose **Display Validation Error** to see an explanation of the violation.

Note: If you have elected to resolve conflicts automatically (see *Dictionary Synchronizer Options*) and the conflict can be resolved automatically, then this status indicator is not set!

Warning below. This indicates there is an unsupported proposal further down in the hierarchy. Expand the tree or use the navigation controls to find the warning.

Warning. This indicates the proposed change is not supported by the target server, but can be ignored. This does not disable the **OK** button. For example, the backend does not support a ValidityCheck of BOOLEAN.

List Item Colors

Color indicators include:

Blue
indicates no action is applied, and the item is either different than its matching item or it has no matching item.

Black
indicates an action is applied and the resulting target item is valid, or no action is applied but none is needed because the items match and the target item is valid.

Red
indicates an action is applied and the resulting target item is not valid—the target does not support the applied property. You must resolve the invalid proposal before you can complete the synchronization.

Gray
The item is not supported by the target dictionary.

Action Indicator Colors

A full color action indicator indicates the action was applied directly to the dictionary item. A gray action indicator indicates a cascaded (or inherited) action—the action was originally applied to a parent (full color indicator) and the child has inherited the action (gray indicator).

Navigating the Synchronizer Tree

The **Synchronizer** provides the following navigation aids to let you quickly identify and resolve differences between the two data dictionaries:

Go to the previous difference—automatically expands the list as required. Alternatively choose **Edit ➤ Previous Difference**.

Go to the next difference—automatically expands the list as required. Alternatively choose **Edit ➤ Next Difference**.

Go to the previous invalid proposal—automatically expands the list as required. Alternatively choose **Edit ➤ Previous Invalid Proposal**.

Go to the next invalid proposal—automatically expands the list as required. Alternatively choose **Edit ➤ Next Invalid Proposal**.

Tip: Use the navigation buttons and the corresponding navigation menu items to quickly locate new, changed, or invalid dictionary components.

Matching Unmatched Items

The **Synchronize** menu (and the RIGHT-CLICK popup menu) contains all the procedures needed to match items between the two dictionaries.

Match
Associates the selected item with an unmatched item in the other dictionary. The Synchronizer moves the matched items onto the same row. RIGHT-CLICK or press ESC to abort the match.

Unmatch
This uncouples the selected items and moves them to separate rows so they can be rematched. This is for items that were matched inadvertently or which should not be matched.

Match by Name
Unmatches the selected item and rematches it by name.

Match by Order
Unmatches the selected item and rematches it by order of the source dictionary.

Match by Component
Unmatches the selected key and rematches it based on component fields.

Tip: DRAG **from a source column in one row and** DROP **on a target column in another row to match the two items. If either item was already matched, this operation first unmatches the original match.**

Tip: **Relationships are matched automatically, based on the files and keys the relationship uses.**

Resolving Differences Between Dictionaries

All differences between source and target dictionaries can be resolved by applying an appropriate operations to each item. The **Synchronize** menu (and the RIGHT-CLICK popup menu) contains all the operations needed to resolve differences between the two dictionaries.

The **Synchronize Dictionaries** dialog uses action indicators to provide information about the current synchronization process.

Synchronizer Actions and Action Icons

Action icons are between the source and target columns of the list. Applying an action to any item in the list affects the selected item *and its children*. The action applied to an item cascades to any nested items. The primary item's action icon is in full color; the cascaded action indicators are gray.

Tip: **Choose Edit ➤ Undo to undo the last action.**

The **Synchronize Dictionaries** dialog shows the proposed action for each item in the list. The action icons are in the column between the two dictionaries. Synchronizer actions and their corresponding action icons are:

Icon **Menu Selection** and explanation of synchronizer action.

? No decision. This is not a menu selection. It is the initial state of the **Synchronize Dictionaries** dialog before any actions are applied. This state does not change the target dictionary.

Add to
Add only. Adds the selected source dictionary items to the target dictionary. This applies only to source items with no matching target items. This action only adds new items in the target dictionary.

Copy to
Change only. Copies the properties of the selected items from the source to the target. This action only affects matched items—it ignores unmatched items. This action only changes existing items in the target dictionary.

Tip: **A gray action indicator indicates a cascaded (or inherited) action—the action was originally applied to a parent (full color indicator) and the child has inherited the action (gray indicator).**

Merge into
Add and change. For matching items, this action copies properties from source to target; it adds unmatched source items to the target dictionary; it ignores any unmatched target items. This action can add new items to, and change existing items in the target dictionary. This action makes the target dictionary a union of the source and target.

Replace
Add, change, and delete. For matching items, this action copies properties from source to target; it adds unmatched source items to the target dictionary; it **deletes unmatched target items.** This action can add new items to, change existing items in, and delete items from the target dictionary. This action changes the target item to exactly match the source item.

Delete from
Delete only. Deletes the selected items from the target dictionary. This applies only to target items with no matching source items.

Ignore Difference
Ignore. Ignore the selected items. This action does not change the target dictionary.

Set to Default
Parent. Applies the parent's action to the selected items. This action is available only if you have previously changed a child's default operation.

Tip: **Choose Edit ➤ Undo to undo the last menu selection.**

Resolving Invalid Proposals

Invalid proposed changes to the target dictionary can be resolved in two ways: from within the Synchronizer and from outside the synchronizer.

Within the Synchronizer, choose **Display Validation Error** from the menu to find out exactly what the problem is. **Display Validation Error** Provides information about the nature of the invalid proposal—describes why it is invalid.

If you don't need the offending item in the target dictionary/database, you can proceed with the synchronization by unmatching the item (RIGHT-CLICK the item then choose **UnMatch** from the popup menu), then deleting it from the target (RIGHT-CLICK the item then choose **Delete** from the popup menu).

If you do need the item in the target, you may be able to resolve the violation by changing the Synchronizer operation applied to the item. For example, if you copied the item from the source to the target (you chose **Copy to** from the menu), then all properties of the source item (including any unsupported properties) are applied to the target item, resulting in an invalid proposal. Changing the operation from "copy" to "ignore" should resolve the problem (RIGHT-CLICK the item then choose **Ignore Difference** from the popup menu) because the Synchronizer does not copy the source item properties to the target item.

Alternatively, you may resolve the violation by cancelling the synchronizer session, changing the offending data declaration in the source dictionary/database, then resynchronizing.

Implementing the Changes

When you have matched all relevant database items, and resolved all differences and invalid proposals in the **Synchronize Dictionaries** dialog, you can implement any proposed changes by pressing the **OK** button.

The **Synchronizer** creates the data conversion program, backs up the target dictionary, and generates the new target dictionary or database definition in the appropriate format—Clarion .DCT or .TXD file, Btrieve .DDF files, or SQL script.

You may execute the SQL script, or you may save the script, optionally edit it, and execute it later. The SQL script implements the new database structure.

The **Synchronizer** creates back up files by renaming the original target file—**Synchronizer** changes the first position of the file extension to a 'B.' So *.DCT is backed up to *.BCT, *.TXD is backed up to *.BXD, *.DDF is backed up to *.BDF, etc.

Running the Conversion Program

When you check the **Generate Data Conversion Program** box, the Synchronizer generates the conversion program components into the working directory. The conversion program lets you select the specific data files to convert to the new format. The conversion program optionally backs up the files you select to the backup folder you specify.

The conversion program handles new files as well as changed files. This gives you the framework to create and prime new files or to split one file into two new related files. You can edit the C5cvt___1.clw source code to prime records and make custom field assignments as needed.

The conversion program consists of the following components:

C5cvt___1.clw	file declarations, procedure definitions, field assignments
Convert.clw	program map, class declarations, initialization
Convert.prj	project file to compile and link the program

To make and run the conversion program, open the Convert.prj (choose **File ➤ Open**, select *Project (*.prj)* from the **Files of type** drop list, then DOUBLE-CLICK Convert.prj. Finally, press the button to make and run the conversion program.

Running the SQL Script

How you run the generated SQL script depends on the SQL backend. See your SQL database documentation for instructions on storing and running scripts.

Dictionary Synchronizer Options

The Dictionary Synchronizer is configurable. That is, to some extent you can determine how the synchronizer looks and acts. The main synchronizer elements you can configure are

- colors,
- sort sequences,
- automatic generation of data conversion program,
- automatic resolution of conflicts.

The default settings are suitable for many cases so you probably don't need to change these settings at first. However, at some point you may want to change the default settings.

To reconfigure the synchronizer, choose **Setup ➤ Dictionary Synchronizer Options**.

Tip: **You can change many of these settings with the View menu while the Synchronize Dictionaries dialog is open.**

Options

Sort files alphabetically

Checking this box displays files and aliases in alphabetical order on the **Synchronize Dictionaries** dialog. Clearing the box displays them in the order they are defined in the source database.

You may also specify the display order during synchronization by choosing **View ➤ Sort files alphabetically**.

Sort fields alphabetically

Checking this box displays fields in alphabetical order on the **Synchronize Dictionaries** dialog. Clearing the box displays them in the order they are defined in the source database.

You may also specify the display order during synchronization by choosing **View ➤ Sort fields alphabetically**.

Sort keys alphabetically

Checking this box displays keys in alphabetical order on the **Synchronize Dictionaries** dialog. Clearing the box displays them in the order they are defined in the source database.

You may also specify the display order during synchronization by choosing **View ➤ Sort keys alphabetically**.

Create conversion program

Checking this box tells the synchronizer to automatically generate a Clarion program to convert existing data to the new format of the target database.

Note: This box is disabled when the target is an SQL database, because no separate conversion program is required. Rather, the synchronizer automatically generates an SQL script to change the database definition to the new format.

Hide unsupported attributes

Checking this box tells the synchronizer to hide (omitted from the **Synchronize Dictionaries** dialog) any attributes of the database the synchronizer server does not support. Clearing the box tells the synchronizer to display the unsupported attributes, but as disabled (dimmed) items.

Correct validity violations automatically

Checking this box tells the synchronizer to automatically resolve conflicts where possible. Clearing the box tells the synchronizer to display the Invalid Proposal icon so you can resolve the problem manually.

For example, a field data type is record picture, but the target dictionary does not support record picture, so the synchronizer automatically resolves the violation by applying the same operation on record picture as on data type.

Note: The synchronizer cannot automatically resolve some conflicts, such as maximum number of MEMO fields exceeded.

Note: If you have elected to resolve conflicts automatically and the conflict can be resolved automatically, then the unresolved difference status indicator in the Synchronize Dictionaries dialog is not set!

Generate DBA instruction script

Checking this box tells the synchronizer generate instructions for *modifying* an existing SQL database. The instructions are an approximation of the SQL needed to achieve the change.

Colors

The **Synchronize Dictionaries** dialog uses color to provide information about the current synchronization process. You can use the default colors or specify your own colors.

Select a synchronizer element in the **Elements** list box, then CLICK on a color selection box. The sample text shows you how the selected element will appear in the **Synchronize Dictionaries** dialog.

Batch Synchronization

For every synchronization, the Dictionary Synchronizer creates a log called *dictionary*.log in the current directory. This log serves two purposes: you can use it to automatically rerun a synchronization, and you should submit it with any bug reports along with the synchronized data dictionaries to aid in the debugging process.

After you finish a synchronization (either successfully or unsuccessfully), you can use this log file to automatically rerun the synchronization in batch mode. This can save lots of time for large or repetitive synchronization tasks.

To Synchronize in batch mode:

1. Open the Clarion environment and close any application modal windows (the Pick dialog, the Window Formatter, etc).
2. Run the Resync example program (..\Examples\Utility\Resync).

   The program prompts you for the synchronizer log file and the Clarion dictionary that started the original synchronization—not necessarily the source dictionary.

   You may also supply these two pathnames as command line parameters to the Resync program.

   The Resync program starts the Synchronizer and repeats the steps logged in the *dictionary*.log file.

Note: If you pressed OK or Cancel at the end of a synchronize, you may want to delete those commands from the log file.

7 - TOPSPEED VERSION CONTROL SYSTEM

Overview

What Is Version Control?

Software developers work with files all the time—source code files, project files, executables, documentation files, etc. For many of these *workfiles*, tracking the changes as they are made is as important as protecting the files themselves.

To track and protect data, workfiles are stored in an *archive*. An archive can contain multiple versions or *revisions* of any type of workfile. The archive also contains information about who changed the file, what the change was, and when the change was made. This provides an evolutionary history of the file's development, and the ability to intelligently revert to any prior version of the file, or group of related files.

In addition to the archive, developers need an efficient, intelligent tool to store and restore files to and from the archive, so they can concentrate on programming while the tool maintains clean, uncluttered, and current working directories with a minimum of developer effort. In the case of multiple developers, the tool should provide a mechanism to maintain file integrity so that only one developer is able to edit a file or portions of that file at any one time.

Version Control System provides both the archives and the right tool for managing them.

Terminology

This chapter uses the following concepts and terms:

archive
The file in which the Version Control System stores all successive revisions of a workfile, including the current (or “tip”) version. An archive contains information about each revision, including when the file was modified, who modified it, what each modification was, and the user ID of any user who has locked it.

archive directory
A directory or set of directories where the Version Control System stores its archive files.

change description
A change description is the text stored with each revision that explains the changes made to it. You enter a change description each time you check in a workfile.

lock
A lock is the mechanism Version Control System uses to prevent multiple users from updating the same revision at the same time. A workfile is writable only if you get it with a lock. Without a lock, the workfile is opened in read-only mode.

revision
A Revision is an instance of a workfile stored in an archive.

initial revision
The oldest revision in an archive.

tip revision
The newest revision in an archive.

workfile
A file that you want to place under version control. Workfiles can be any type of file, including source code files, Clarion .APP files, documentation files, executables, graphic files, etc.

Version Control for all your Workfiles

Version Control System (VCS) lets you save and catalog incremental versions of *all* your critical development files, including (but not limited to) .APP, .DCT, .TPL, .TRF, .CLW, .PRJ, .ICO, .BMP, .EXE, .LIB, and .DLL.

Each revision is identified by date, time, author, and change statistics, plus a user-supplied description of changes since the last revision.

Further, TopSpeed's Version Control System lets you group together all the specific file revisions that comprise milestones (Beta 2, Release 1.5, etc.) in your project's life cycle. This is accomplished with Version Labels that can be applied to one or more files.

You can easily track the evolution of each individual development component, as well as the entire project; and throughout the project, you can instantly revert to any prior revision level, or version.

Team Development Facilities

In addition to intelligent version control, TopSpeed's Version Control System provides superior control mechanisms that make good programming practices simple to implement.

Procedure-level locking
Allows you to share application files among multiple developers. That is, specific procedures may be “owned” by one developer so that no one else can change them. These procedures are available in read-only mode to other developers, for review, code generation, and testing of their own work.

File-level locking
Allows you to share other files, such as dictionaries and libraries, among multiple developers .

In addition, Version Control System provides intelligent file retrieval—retrieving a .PRJ or .APP file, for instance, automatically retrieves its associated files (.DCT and .CLW).

Plus, an enhanced project definition file supports *custom* intelligent file retrieval from multiple archive directories to multiple working directories with default lock settings for each file.

All within the Clarion Development Environment

Version Control System provides this rich set of version control and archive management options directly from the Clarion development environment. You can access all of the common version control procedures from the **Archive** menu within the Clarion development environment.

Version Control Tutorial

In this section, you will learn how to:

- Create an archive.
- Check In revisions.
- Check Out files using procedure locks and file locks.
- Create a project definition file.
- Use the Utilities.

We will use the completed Clarion tutorial application for this tutorial. Since these files are supplied at install in \CLARION5\EXAMPLES\TUTOR you may do this tutorial at any time. The tutorial assumes that no configuration has been done for the Version Control environment. Any variances you experience will likely be the result of site configuration that is not the same as the install defaults, in which case we recommend reviewing the options in the manual and coming to an understanding of how Version Control is set to work for you.

Even if you've already used a version control product before, you should still complete this tutorial or at least read through it. There are many features unique to Clarion and they are highlighted in the tutorial.

The Archive Menu

The Version Control system adds an **Archive** menu to the Clarion environment. This menu is dynamic; it changes based on whether files are open or not. Begin by examining the **Archive** menu when no files are open.

Starting Point:
The Clarion development environment should be open and the Pick dialog closed.

1. **Select** **Archive** (on the menu).

   The following choices on the Archive menu.:

These menu items are documented in the *Using the Version Control System* section.

Configure Archive Directories

The first step is to define the directories where archive files will be placed. For this project we will use a local subdirectory under the working directory.

1. Use Windows Explorer or other tool to create the directory \CLARION5\EXAMPLES\TUTOR\ARCHIVE.
2. **Select** **Archive ➤ Configuration.**
3. Navigate to the *\CLARION5\EXAMPLES\TUTOR\ARCHIVE* directory.
4. **Press** the **Add** button.

5. **Press** the **OK** button to save these configuration settings.

   See *Configuring the Version Control System* for specifics about the many other configuration options of the VCS and set them to suit your needs when using the VCS for live projects.

Create archives

It is easiest to create archives when the files are closed.

> **Note:** **Archive creation may be restricted if your site is using security - talk with your team leader if you are unable to create archives for the tutorial.**

Place the APP under Version Control

1. **Select** **Archive ➤ Check In.**

   The **Select File to Check In** dialog opens.
2. **Navigate** to the *\CLARION5\EXAMPLES\TUTOR* directory.

   The following APP files should display:

This dialog allows multiple selections and two levels of filtering files. Use the **File Type** filter to display files of a specific extension. You can also check the **Show only files under Version Control** box to further limit the list of files by displaying only those files that already have an archive. Do not check this box when creating a new archive.

3. **Select** *tutorial.app* and **press** the **OK** button.

The **Check In List** dialog opens. Of course, there's only one file in the list. The change description is filled in for you (it defaults to "Initial Revision" when creating a new archive).

4. **Mark** the **Delete workfile** button.

The recommended action after Check In is to delete the workfiles. This prevents you from making modifications to the project outside of version control. You might want to leave read-only workfiles, for testing or evaluation. You would use the last option, **Keep writable workfile and lock**, when you reach some milestone while working, and want to continue working, but also want to save the project in its current state in the version control system.

5. **Type** *As provided by TopSpeed* in the **Version Label** entry box.

   Version labels let you tag a revision number with a meaningful text name. This is the preferred method for relating a release (e.g., Version 1.850) with a revision (e.g., 1.273). Version labels are especially powerful as they can be applied to different revisions in different files, linking together a set.

   Using a version label is required to automatically move the templates with the APP file. The **Check In Templates** option is enabled if a version label is added and a template definition file is specified in your configuration.

6. **Press** the **Check In** button to continue.

7. In the **Create Archive** dialog, **highlight** *\CLARION5\EXAMPLES\TUTOR\ARCHIVE* as the archive directory.

8. **Type** a description of the project/set of files.

9. **Press** the **OK** button.

After you create the archive and add a revision of the current state of the APP, you are returned to Clarion.

Place the DCT under Version Control

Now we'll create an archive for the dictionary. We will delete the dictionary workfile also, but won't have to check it out unless we need to modify it because a read-only copy is automatically retrieved when checking out the APP.

1. **Select** **Archive** ➤ **Check In**.

2. In the **Select file to Check In** dialog, **select** *Dictionary (*.DCT)'* in the **File Type** droplist.

3. **Navigate** to the *\CLARION5\EXAMPLES\TUTOR* directory.

4. **Highlight** *tutorial.dct*, and **press** the **OK** button.

One of the reminder messages that assist in learning to use Version Control appears. These reminders are enabled at installation and can be disabled in the local Configuration.

5. **Press** the **OK** button to close the reminder screen.
6. In the **Check In List** dialog, **select** **Delete workfile** for the **After Check In action.**
7. **Press** the **Check In** button.
8. In the **Create Archive** dialog, highlight *\CLARION5\EXAMPLES\TUTOR\ARCHIVE* as the archive directory
9. **Type** a description of the file, and **press** the **OK** button.

   This returns you to the Clarion development environment.

Retrieving Workfiles from Archives

Now we'll check out the application.

1. **Select** **Archive ➤ Check Out** to open the **Select File to Check Out dialog.**

   This dialog is where you select the destination directory, and the file to check out.

2. **Select** *Application (*.APP)* in the **File Type** droplist.
3. **Select** the *tutorial.app*.
4. **Navigate** so the *\CLARION5\EXAMPLES\TUTOR* directory is the **Destination Directory**
5. **Press** the **OK** button.

The **APP File Check Out** dialog allows you to specify which procedures you will be able to modify. Procedures can move back and forth between the **Don't Lock** and **Lock** list boxes by DOUBLE-CLICKING on a procedure name, or by selecting one or more files and pressing the **Lock>>** or **<<Don't Lock** buttons.

Note: Global Properties for an application are treated like a procedure—only one developer can modify them at a time.

6. **DOUBLE-CLICK** on the three update procedures (UpdateCustomer, UpdateOrder, and UpdateProduct).

   This adds the procedures to the **Lock** list.

7. **Check** the **Load Application after Check Out** box.

8. **Press** the **Check Out** button.

   When the file has loaded, notice that most of the procedures are marked [Read-Only]. Only the three procedures that you locked are not marked and those are the only three that can be edited. Go ahead and look around. The [Read-Only] procedures cannot be accessed by many of the pop-up menu options, and the **OK** button is disabled on the **Procedure Properties** dialog. However, you can go into the window and embeds, evaluate and even copy code for use. Global properties are similarly protected. If you press the Make button, the project will compile—you can test your work.

9. **Select** the *UpdateCustomer* procedure and make a modification (arrange the prompts and fields on the UpdateCustomer window).

10. **Make and run** the project again, notice your changes are implemented.

11. **Close** the application.

Archive Menu (with files open)

The Version Control System works to keep you focused on the task at hand, developing your application. When one or more files are open, the VCS menu changes to reflect specific Check In and Check Out options:

The first menu option becomes **Checkpoint** for the active file. This allows you to update the archive with the current state of the file you are editing without closing the file.

The second option becomes **Revert** for the active file. This option overwrites your existing (open) file with the version of the file as you last accessed it in the version control archive. If you had only checked the file out then you will get the file as it was at that point, if you have checkpointed it then you will get the file as of the last checkpoint.

Note: If a file is opened read-only, then Check In and Check Out are simply disabled—no provision for checkpoint or revert is available.

1. **Select Archive ➤ Checkpoint tutorial.app.**

The **Checkpoint** option bypasses the **Select file to Check In** dialog and opens the **Check In List** dialog.

The change description automatically contains the name and revision of the dictionary used by the application as well as a list of the procedures that were modified.

2. In the **Change Description** text box, **type** *Arranged prompts* next to *UpdateCustomer* - .

> **Note: Using Checkpoint forces the selection of Keep writable workfile and lock. This allows you to continue working and retains the procedure locks.**

3. **Press** the **Check In** button to continue.

Adding more procedure locks

You may discover that you need to modify additional procedures that were not locked on Check Out. You can re-Check Out the application and specify additional procedures (if they are not locked by another developer).

1. **Close** the tutorial.app.
2. **Select** **Archive ➤ Check Out.**
3. **Select** *tutorial.app*.
4. **Press** the **Revisions>>** button.

The Revisions button reveals details about the highlighted file's revision history. You may see the history of this file by selecting each revision and reading its description. Notice the version label under revision 1.0. You can check out a file by selecting either the revision number or version label. If you do not select a specific revision, Version Control retrieves the tip (latest) revision.

Note: Checking out a non-tip (earlier) revision is a special case, generally done to evaluate why some old code worked that has stopped working, or to rebuild a system as it exists at a customer site that is reporting problems.

For all files except procedure-locked APP files, if you check out a non-tip (earlier) revision with a lock, you are warned that a branch will be created at Check In. Branching is an advanced feature, and although supported through the Version Control interface, there is no provision for merging parrallel development.

If you check out a non-tip revision of a procedure-locked APP file a fully writable APP file is retrieved (no procedures are [Read-Only]) but no locks are placed, so it *cannot* be checked back in.

5. **Highlight** the tip (latest) revision
6. **Navigate** so the *\CLARION5\EXAMPLES\TUTOR* directory is the **Destination Directory**.
7. **Press** the **OK** button

The **APP File Check Out** dialog now shows three procedures with your user ID as the lock owner (the lock owner in the graphic above is *Tutorial*). This is what other developer see when they check out the .APP, and they would not be able to move those locked procedures into the lock list.

You cannot release locks here either. That is done at Check In when you do not keep a writable file. You may however, add procedures that are not already locked to the lock list and get a new copy of the APP.

Next, we will select a few more procedures:

8. **CLICK** on *CustInvoiceReport*, then **SHIFT-CLICK** on *Main*.

   This selects all the procedures between the two.

9. **Press** the **Lock>>** button.

10. **Clear** the **Overwrite procedures in existing APP file** box.

    The **Overwrite procedures in existing APP file** option allows you to start over with what was last stored in the archive. Leave it unchecked - we don't want to lose our modifications in UpdateCustomer!

11. **Press** the **Check Out** button to retrieve a new APP and load it.

    You are warned that a writable file will be overwritten— this appears whenever you try to overwrite a file.

12. **Press** the **OK** button to confirm that you know what you are doing.

    When the file has loaded, notice that the additional procedures you checked out (locked) are no longer marked [Read-Only].

13. **Close** the application.

Checking out other files

Only procedure-locked APP files allow multiple developers to simultaneously work on the same file (albeit different parts of that file). All other files are locked on a file-by-file basis—you get it writable or read-only.

The dictionary is a good example of a file that can only be modified by one person at a time. To modify a DCT, you have to check it out with a lock.

1. **Select** **Archive ➤ Check Out.**
2. In the **Select file to Check Out** dialog, **select** *Dictionary (*.DCT)* in the **File Type** droplist.
3. **Navigate** so the *\CLARION5\EXAMPLES\TUTOR* directory is the **Destination Directory.**
4. **Highlight** *tutorial.dct*, and **press** the **OK** button.

The **Check Out List** shows the one file to be retrieved.

5. **Check** the **Lock** option to lock the revision
6. **Press** the **Check Out** button.

Reports

Several reports are available to track the progress of development and determine the current status of project components.

1. **Select** **Archive ➤ Reports.**

   The Select file for Reporting dialog has a similar interface to the Select file to Check Out, allowing you to select archive directories and then choose a file in that directory.
2. **Select** the *\CLARION5\EXAMPLES\TUTOR\ARCHIVE* directory.
3. **Select** the *tutorial.app*.
4. **Press** the **OK** button

The Report dialog is the launching point for running any report(s) that you wish for the selected archive. Simply choose a report and press the **Report** button. When you finish viewing the report you will return here where you may select another report.

5. **Select** *List locked procedures*, then **press Report**.
6. **Close** the report preview window when you are finished.
7. **Press** the **Cancel** button to end reporting.

Utilities

Utilities provide a secondary interface to the archives. You can use them to determine the current status of locks as well as change certain attributes of the archives. Modifying archives using Utilities should be considered a last resort when you cannot accomplish the desired result with Check In or Check Out.

The Utilities also operate similarly to Select file to Check Out. Selecting an archive directory displays the archives in that directory, then select an archive to see its status. You can also select individual revisions of a selected archive to review the description of it.

> **Note:** **Everywhere else in the environment the Version Control System displays the workfile name (e.g. tutorial.app) when selecting an archive (Check Out, Reports, etc.) but for best performance in Utilities, the actual archive file names are displayed.**

1. **Select Archive ➤ Utilities**.
2. **Select** the *\CLARION5\EXAMPLES\TUTOR\ARCHIVE* directory.
3. **Select** *tutorial.apv*.

   This is the archive for the Tutorial.APP.

The status section shows that there are locked procedures in tutorial.apv. The two revisions are displayed in a list box,

4. **Select** *revision 1.0* and notice that the **Description** changes

5. **Select** *As provided by TopSpeed* and notice that it is the same as *revision 1.0*

6. **Select** *revision 1.1* again, and **press** the **Procedures** button

All of the procedure locks in the APP are displayed here. A method of unlocking them is provided here. You should read the *Utilities* section for details on using this—misuse could prevent a developer from checking in legitimate changes.

7. **Press** the **Cancel** button.

8. **Select** *tutorial.dcv*.

This is the archive for the Tutorial.DCT.

The procedures button is now disabled; it is only available for application files with procedure locks enabled. The status now shows that a revision is locked, the description shows it is by you. Revisions can be unlocked and even deleted here. You should read the *Advanced Topics* section before undertaking such actions to ensure you understand the implications.

9. **Press** the **Done** button to close the **Utilities** dialog.

Creating Project Definition files

Project definition files allow you to work with several files at one time. This is a text file, with a CFG extension. An interface is provided to maintain these files:

1. **Select** **Archive ➤ Project Definition.**

   First we'll create a new project definition file.

2. **Press** the **Add Workfile** button.

3. The Directory should be \CLARION5\examples\tutor, if not, move to it.
4. **CLICK** on the first APP displayed
5. **SHIFT-CLICK** on the last APP displayed

   This selects all files between the two.

6. **Press** the **OK** button.

   This populates the three tutorial APPs in the workfile list box and returns you to the Add Workfile dialog. You simply continue selecting files until you have the desired set defined. You can use the File Type filter to speed up file selection.

7. **Press** the **Cancel** button to close the **Add Workfile** dialog.

   Project definition files can specify their own archive directories. Entering the directory speeds up retrieval, even when the directory is also specified in the master configuration file.

8. **Press** the **Select Archive Directories** button.

9. **Navigate** to the *\CLARION5\EXAMPLES\TUTOR\ARCHIVE* directory in the directory list box and **press** the **Add >>** button.

10. **Press** the **OK** button.

To open and save your files, use the menu on the Project Definition dialog.

11. **Select File ➤ Save**

    A standard file dialog opens to allow you to provide a name for the file.

12. **Type** *tutorials* in the filename entry box, then **press** the **Save** button.

13. **Press** the **Close** button.

Working with Project Definition files

Project definition files are available for Check In, Check Out, and Reports.

1. **Select Archive ➤ Check In.**
2. **Select** *Project Definition* in the **File Type** droplist.
3. **Navigate** to the *\CLARION5\EXAMPLES\TUTOR* directory.
4. **Select** *tutorials.cfg*, then **press** the OK button.

The Check In List now shows 4 files to process: the three APP files and the CFG file itself. The project definition files are archived to easily allow changes in the file set over time to be correctly reflected at Check Out.

Each file is checked in separately, so you can create new archives for some files (in this case for three of the four files) and not for others. Depending on configuration, unmodified workfiles in a set might not be checked in at all. Previously hidden buttons are now visible—**Check In All** and **Skip**.

Check In All processes all the files in the list once you have set the **After Check In action**.

Skip works in conjunction with **Check In**, allowing you to step through the files in the list and process each one or not.

Checking out and running reports work on the file sets in a similar manner. At check out, you will have to go through the APP File Procedure Locks dialog for every APP file in the CFG file.

Template Definition files

Another purpose of a project definition files is to define the templates and other files that are required to completely recreate an APP file as it was when checked in. A sample file, TEMPLATE.CFG was placed in \CLARION5\TEMPLATE during installation. It contains references to all of the Clarion templates in that directory and the class source files in \CLARION5\LIBSRC. You should modify this file if:

- you have third party templates or classes in use
- you use local redirection files
- you did not install to the default directory (C:\CLARION5)
- there are other external files to be included in your projects

Template definition files can be specified in local configuration and as a part of a project definition file, under the advanced button on each dialog.

Summary

This concludes the tutorial. You may experiment with these files, then delete the contents of \CLARION5\EXAMPLES\TUTOR\ARCHIVE and remove it from your list of archive directories in Archive ➤ Configuration.

Let's review:

- You create an archive by checking in a file or set of files.
- Checking in files as you work creates an archive containing each revision.
- Only procedure-locked APP files allow multiple developers to simultaneously work on the same file (albeit different parts of that file). All other files are locked on a file-by-file basis—you get it writable or read-only.
- If you frequently need to retrieve a certain set of files, especially a subset of an application, you can create a project definition file (.CFG) specifying archive directories unique to that project, as well as multiple destination directories and default locks.
- Utilities provide a secondary interface to the archives. You can use them to determine the current status of locks as well as change certain attributes of the archives.

TopSpeed Version Control System (VCS)

Version Control System lets you create and maintain version control archives and developer working directories with the following choices on the Clarion **Archive** menu:

Check In
Check Out
Project Definition
Utilities
Reports
Configuration

Basics

With Version Control System you create an archive when you *check in* a workfile for the first time. To modify the file, you *check out* a copy of the workfile from the archive. After you finish modifying the workfile, you check it back in as a *revision*, adding a change description and specifying what to do with the workfile.

By checking out a revision with a *lock*, you can modify the file while preventing others from doing so. This mechanism protects your work and allows Version Control System to provide multiple developer support for hand-coded projects (.PRJ and .CLW files) and individual files like dictionaries, documentation files, etc.

When you do not place a lock during check out, the workfile is opened in read-only mode and may not be checked back in. This may be desirable for code review or testing. Even if the read-only attribute is manually removed the Version Control System will not allow a check in of that workfile.

Version Control System was designed specifically for the Clarion developer. For the hand-coder, selecting a project file (.PRJ) to check in or out also acts on all the project's source code files (.CLWs). Checking out an application file (.APP) also checks out a read-only copy of the application's dictionary (.DCT). Version Control System also supports *procedure-level locking* within an .APP file, allowing multiple developers simultaneous, but safe, read/write access to an .APP file.

Version Labels

A version label is a symbolic name for a revision of a workfile. You can assign the same version label to any number of revisions stored in different archives. For example, you could use a version label such as "Friday Build," "Version 1.5," or "Beta 2" to identify a particular version of a system or application.

For projects comprised of many files, a version label lets you identify the particular revision of each file that makes up a complete version of your application. Version labels let you operate (check out or check in) on all the files at the same time by referencing a single name so you don't have to remember which particular file revisions made up a particular version of your software.

Version Control System Projects

You can control, on a file-by-file basis, what to lock and what to leave available to other developers. If you frequently need to retrieve a certain set of files, especially a subset of an application, you can create a project definition file (.CFG) specifying archive directories unique to that project, as well as multiple destination directories and default lock settings.

Reports and Utilities

Six different reports are available to provide access to the information stored in the archive about each file's evolution and current lock status.

Several archive utilities round out the Version Control System offering.

Shared Archives

Team Development manages two or more developers that share development responsibilities on a project. Access to archives is typically achieved by placing them on network drives, while the working directories may be local or on the network - so long as each developer has a unique set of directories.

Code Overwrite

Using VCS, you minimize the risk of overwriting changes made by another developer or having your own changes overwritten.

Locks—Show other users when a file is in use

A lock controls access to revisions by warning other users that a file is in use. When you check out a modifiable copy of a revision, a lock is placed on that revision. Another user cannot modify the revision until you unlock it, either by checking it in, or through the utilities. In this way, you can prevent multiple updates to the same revision. File locks are placed on all files except the Clarion application file (.APP).

Sharing an APP file

The CLARION application file is shared in a team environment by locking individual procedures within it. Another developers may check out the same file, but within the application generator they will have read-only access to procedures that are locked.

User ID

Each developer must have a unique user ID. You can specify how the Version Control System recognizes the user ID with several methods. The default method is by the use of the environment variable—VCSID—which you specify during setup. You can also specify that the user ID be obtained from the network login, or force the developers to login to the version control system. For more information, see *Setting your Version Control System User ID* in the section called *Customizing the Version Control System*.

Components

The primary components of the Version Control System are a set of DLLs that are a part of the Clarion development environment and a 32-bit DDE server. These provide the bulk of the interface and are loaded (and unloaded) with Clarion development environment.

The archive engine is Intersolv's PVCS—the industry standard for software configuration management on a LAN. The storing and retrieving of revisions is performed by PVCS DLLs licensed to TopSpeed.

VCS Administrator
A separate program which provides an easy method to update the configuration for all users. This utility also controls the security features of the VCS and several global (all users) environment settings.

VCS Status
Another utility which shows the status of all locks by developer and by file.

Configuring the Version Control System

The VCS must be configured to operate correctly. There are three configuration files, (see below), and the processing order of the three files is fixed. It is important to know this sequence, because some options are specified in more than one of these files. Settings in the first-read file are overwritten by settings in the second and even the third file. If possible, try to set up all users with the same location for the master configuration file (MASTER.CFG). One or more *team leaders* should be designated, to supervise the maintenance of these configuration files.

Configuration Files

Use VCS Administrator to set the following configuration options for all users at one time (see *Using the VCS Administrator*).

- MASTER.CFG (usually on a network drive) is read first, on every Check In or Check Out to re-initialize the settings.
- C5VCS.CFG (in the \CLARION5\BIN directory) is read second. This file is created for each user/machine, and contains any local configuration settings and user preferences.
- *Project*.CFG (a project definition file) is read last if you Check In or Check Out a project definition file. In addition to containing a list of files to manage together, it can also specify archive directories, template definition, and event triggers.

As an example of how this affects behavior, consider the settings when checking out a project definition file. The **master configuration file** is read first, and the VCS is made aware of the archive directories specified there.

```
VCSDIR=x:\archive\project1
```

The **local configuration file** is read next. This may contain archive directories as well. These directories may replace, or append to, the directories specified in the master file.

Let's suppose there was one directory that is appended, as follows:

```
VCSDIR=x:\archive\project1;h:\home\greg\archive\project1
```

Finally, the **project definition file** is read. If it contains archive directories, *they always replace any previously specified.*

```
VCSDIR=u:\project1
```

As soon as you Check Out another application, the master configuration file and the local configuration files are reread, and the archive directories are reset.

```
VCSDIR=x:\archive\project1;h:\home\greg\archive\project1
```

Ideally, ALL users should point to the same MASTER.CFG and as many archive directories and other settings should be made there using the VCS Administrator.

To configure locally, select Archive ➤ Configuration. The configuration dialog allows you to define the archive directories, and specify settings.

Note: Your current User ID is displayed in the title bar of this dialog.

Archive Directories

After configuration, the next step is deciding where to store the archive files. This should typically be a networked drive to allow sharing the files, and also to take advantage of any network backup strategies. There are many schemes for the directory structure(s) to hold the archive files, depending on the environment. We suggest that you read through this section to familiarize yourself with the configurable options, then use the Configuration Guidelines at the end of this section to assist in planning your setup.

We have found the best performance and easiest management when the least number of archive directories are used. Even if you use a sophisticated directory structure on the working machine, where each type of file is placed in separate subdirectories, you should consider storing all of the files that will be archived in one directory per project.

This is because the PVCS engine searches for archives and the placement of the archive files is not nearly as important as the workfiles. Also, you should use the **Up** and **Down** buttons to sort the directories so the frequently used directories appear first in the list.

Use File Manager, Explorer, or any other utility to create the directories you'd like to use. It is not difficult to change these later if you develop a better idea of how you want to use directories.

In the left-hand list box, move through the directories to the archive directories that you want to use and press the **Add** button. If you want to remove an entry from the archive directory list, select it and press **Remove**. Use the **Up** or **Down** buttons to sort the list.

Special Considerations

It is a good idea to enter all directories in the VCS Administrator and update the master configuration. This allows all developers to see the same archives and use the VCS Status utility to determine locks before attempting to check out a file. Directories entered here would be for personal use.

The VCS searches for archives in the directories you list here, in the order that they are listed. Use the **Up** and **Down** buttons to sort the list and minimize search time.

Append to master config VCSDIR
Check this option if you are using a centrally managed MASTER.CFG file and allow developers to add archive directories not listed in the master configuration.

The **Login** button allows you to change your User ID for this session. It is reread from the environment the next time you run VCS. This button is enabled only if login is a valid User ID source (in VCS Administrator). To login as another user, the user name must already be defined in the user list, and if a password is in use by that user it must be entered as well.

Press the **Advanced** button to set more configuration options, described below.

Advanced Configuration

Default APP File Handling
There are two Locking choices. **File locking** allows only one developer at a time to work on the APP. **Procedure locking** allows developers to Check Out copies of the APP and work on any parts that are not already locked. The archives for these methods are not the same and can not be changed, so when the archive for an APP file is created (upon initial Check In), VCS must know the locking method to use.

File Lock
Restricts an APP file to a single developer at a time. This provides the best performance and is the best option in a single developer environment.

Procedure Locks
Allows sharing of an APP file among multiple developers at the same time.

Always Ask
Select to be prompted during Check In for which lock method to use.

Check In of Unchanged Workfiles
By default, the PVCS engine does not create a new revision for unmodified workfiles. You can specify the behavior you want.

Force Check In
Select this option if you want to create a new revision even though the file was not modified. This can be useful at times when you have a release version and want to create a revision with a version label for every file, whether modified or not.

Unlock - Don’t Force Check In

Select this option to allow checking in unmodified files without adding a revision. This is useful to keep the archives from listing unnecessary revisions.

Always Ask
Select this option to allow prompting to determine how to handle unmodified files each time you check one in.

Tip: Procedure-locked APP files will generally appear modified to the PVCS engine and be checked in. The only occasion this is not so is if no one else has checked in the APP since your last check in and you did not modify the APP.

Keep path with Project Definition workfiles
Check this option to store the full path of workfiles in project definition files. This is useful when checking in and out files from multiple directories using a project definition file.

Create Project directories on Check Out without prompting
Check this option to create directories stored in a project definition file on checkout. This provides an easy way of replicating a project on a new machine.

Note: The remaining items can only be set locally for each individual developer.

Project Definition Workfile Path Override Drive
To accommodate users with different drive setups (e.g., one developer works on drive D: while the others use drive C,:) this setting specifies a drive letter which overrides the drive letter stored in project files.

Template Definition
Type the fully qualified path and file name of a CFG file that contains a listing of the templates you want to keep in version control. This file could include templates, class files, etc. If specified, you have the option of moving the files listed herein in and out of archive when checking in and out an APP file. For details on building project definition files, see the *Project Definition Files*. For more details on the use of the automatic archiving and restoring option, see the *Check In* and *Check Out* sections.

Display Reminder Messages
Check this option to receive prompts that help you learn how to use the VCS. This option is enabled at installation, and is quite helpful at first. After you are comfortable with the system, you will probably want to disable the messages.

VCS Internet Server
If you are doing development remotely enter the URL of the VCS Internet Server.

Using the VCS Administrator

The VCS Administrator is a supplementary program that allows you to set many of the configuration items “globally.” These settings affect all users. As explained earlier, a shortcut is not automatically created during installation to prevent *experimental* use that could lock everyone out of their projects.

If you have not already done so, you can create a shortcut now to VCSADMIN.EXE in your CLARION5\BIN directory.

The first time you run the VCS Administrator there are no security checks. Subsequent launches require a password and the privilege to modify the access database. The VCS Administrator alerts you on exit if there are no users with the required privileges and a password.

To use the administrator, you must set up all users to point to the same location for the *master configuration file*.

You can verify this by examining the ISLV.INI file in each developer’s WINDOWS directory:

```
[PRODUCT_LOCATION]
PVCS OEM VM_6.0.00_WIN32=drive:path
```

This drive and path is the location of MASTER.CFG. Workstations that are not set to read this file will not be affected by “global” settings made in the VCS Administrator.

First Time in the Administrator

Because the Administrator offers so much power and flexibility, it is password protected. Login is not required the first time, but you must add at least one user with a password before exiting. Later we’ll examine how users, groups, and privileges work together.

Administrator setup

1. **Start** the VCS Administrator (VCSADMIN.EXE in your CLARION5\BIN directory).
2. **Select** the **Security** tab.
3. **Check** the **Provide Security for VCS Administrator only** box.
4. **Select** the *All Users* group, then **press** the **Insert** button.

5. **Type** in your full name.
6. **Type** in your VCSID.
7. **Type** your password in the **Password** entry box and again in the **Confirm Password** entry box.
8. **Select** the **Privileges Granted** tab
9. **Press** the **Add** button, **highlight** *TeamLeader*, then **press** the **Select** button.

   This establishes you as a Team Leader with all associated rights.

10. **Press** the **Add** button, **highlight** *SuperUser*, then **press** the **Select** button.

    This grants you the rights to modify the database of user access rights.
11. **Press** the **OK** button,

The next time you run the VCS Administrator, you are prompted with a login dialog. Your VCSID should be automatically filled in. Enter your password, and press the **Login** button to continue.

What can you do in the Administrator?

The VCS Administrator provides a method to update the configuration for all users. Both the archive directory list and most of the advanced settings can be managed here. In addition, you can specify global settings which are not configurable by individual developers. These include specifying event triggers on Check In and Check Out, how the user's Ids are obtained, and other fine-tuning options.

Note: The VCS Administrator is a one-way update of the master configuration file. That is to say, your settings in the Administrator are stored in a database and written out to MASTER.CFG on exit. Manual edits of MASTER.CFG are not reflected in the Administrator and are overwritten on the next use.

Archive Directories

By far the most important configuration item is the list of archive directories. Use the **Add** and **Remove** buttons to create the list of directories in which archives are stored. You must decide where on the network to store the archive files. Use the configuration guidelines at the end of this section to assist in planning your setup. Remember to minimize the number of archive directories. Even if the local machine's directory tree is elaborate, you should simplify the archive structure to one directory per project. This technique significantly improves performance.

This archive directory method also means that you cannot list archive directories that could contain the same archive names in them—the first archive is always be found first and the second archive would never be accessed. If you must use duplicate archive names, you should create project definition files to access the duplicate archives. Directories specified in a project definition file always replace any directories specified in the master file.

Configuration

The Configuration dialog in VCS Administrator closely resembles the dialog for setting local configurations on each machine. The same controls are specified in both places to permit both global settings and local overrides.

Default APP File Handling

There are two Locking choices. **File locking** allows only one developer at a time to work on the APP. **Procedure locking** allows developers to Check Out copies of the APP and work on any parts that are not already locked. The archives for these methods are not the same and can not be changed, so when the archive for an APP file is created (upon initial Check In), VCS must know the locking method to use.

File Lock

Restricts an APP file to a single developer at a time. This provides the best performance and is the best option in a single developer environment.

Procedure Locks

Allows sharing of an APP file among multiple developers at the same time.

Always Ask

Select to be prompted during Check In for which lock method to use.

Ignore local APP file handling setting

Allows you to require the global APP file handling settings be used by all developers, regardless of their local configuration. This allows global control by an administrator or team leader without the need to configure each developer's computer.

Check In of Unchanged Workfiles

By default, the PVCS engine does not create a new revision for unmodified workfiles. You can specify the behavior you want.

Force Check In

Select this option if you want to create a new revision even though the file was not modified. This can be useful at times when you have a release version and want to create a revision with a version label for every file, whether modified or not.

Unlock - Don't Force Check In

Select this option to allow checking in unmodified files without adding a revision. This is useful to keep the archives from listing unnecessary revisions.

Always Ask

Select this option to allow prompting to determine how to handle unmodified files each time you check one in.

Ignore local Unmodified File setting

Allows you to require the global **Unmodified File** setting be used by all developers, regardless of their local configuration.

Tip: **Procedure-locked APP files will generally appear modified to the PVCS engine and be checked in. The only occasion this is not so is if no one else has checked in the APP since your last check in and you did not modify the APP.**

Keep path with Project Definition workfiles

Check this option to store the full path of workfiles in project definition files. This is useful when checking in and out files from multiple directories using a project definition file.

Ignore local Keep Path setting

Allows you to require the global **Keep Path** setting be used by all developers, regardless of their local configuration.

Create Project directories on Check Out without prompting

Check this option to create directories stored in a project definition file on check out. This provides an easy way of replicating a project on a new machine.

Ignore local Create Directories setting

Allows you to require the global **Create Directories** setting be used by all developers, regardless of their local configuration.

Global

Directory for Temporary Archive Copies

A temporary archive copy is a backup file that PVCS creates and then deletes to protect archives and its internal data. By specifying a directory for temporary archive copies, you prevent archive data loss from a network crash. For example, if the network crashes while you are checking files out of network archive directories, and it corrupts the archive files, you can retrieve uncorrupted copies from your archive temporary directory.

When you perform actions that modify archives, such as checking files in, assigning version labels, and promoting revisions, PVCS first creates a temporary copy of the archive and performs the action on the copy. It then replaces the original archive with the copy. By default, temporary archives are created in the archive directory with the name PVCS*nnnn*.TMP, where *nnnn* is a hexadecimal number.

Directory for other Temporary Files

Specify a directory for other internal temporary files, perhaps a directory where files are purged immediately.

Developer User ID Source

A user ID source is an operating system, network, or other value that the Version Control System can use to determine a user's

identity. This enables VCS to check if the user is authorized to access archives. VCS attempts to obtain user identification from each source in the list. If it reaches the end of the list without obtaining a valid user identification, it logs any activity as user "UNKNOWN" except for procedure-lock enabled .APP files, where it prevents access and displays an error message.

Double-click on a user ID source to enable or disable it. Use the **Up** or **Down** buttons to specify the order that the Version Control System uses to search for a valid User ID.

Tip: Enabling more options than necessary could lead to users being locked out of items if their user IDs are not the same in all sources. VCS uses the first ID it finds in the specified sources and that ID may not be valid in VCS.

Environment Variable
Version Control obtains the user ID from the VCSID environment variable. This is the default setting. The VCSID is set during setup.

Tip: The VCSID is not a secure source for an user ID. Users can circumvent security by logging in as another user or resetting the value of the VCSID environment variable.

Windows Networking
Version Control obtains the user ID from the Microsoft Wnet API, i.e., the Windows Login name.

Novell NetWare
Version Control obtains the user ID from the user's NetWare login name.

Multiple Networks in Use
Select this source for systems that provide a user identification mechanism, such as UNIX, or for environments in which more than one network is in use.

Login to VCS
This option allows users to login with the Configuration dialog. If this is the only method specified then a login dialog is always presented when Clarion starts. To use the **Login to VCS** option, you must enter users and passwords in an access control database using the Security options in the VCS Administrator.

Case Sensitive
Check this box to make the developers' User IDs case sensitive.

Archive Extensions
Specify the archive file extensions. By default, the Version Control System replaces the last character of the file's suffix with a 'v' (SAMPLE.APP archives to SAMPLE.APV). This default can cause conflicts for files with the same name and

similar suffixes; for example Clarion template file suffixes are .TPL and .TPW—both would become .TPV archives. This would not pose a problem if the filenames were always different, but this may not always be true with some template sets.

Work File Extension

Specify the workfile extension identifying the file type to which the new archive suffix template applies.

Archive File Extension

Enter the archive suffix pattern that Version Control System uses to derive the file name extension. Use any alphanumeric characters or the ? wildcard to replace a single character.

Tip: It's good practice to enter *both* conflicting file extensions and their resolutions here to self-document the conflict.

Event Triggers

An *event trigger* is a mechanism that causes an action to occur in response to a particular Version Control event. When Version Control detects the event, it executes the event trigger command line specified for that event.

Events are points in program processing, such as before and after the VCS checks in files, or before and after the VCS checks out files. An event can trigger an external program. For example, you might run a batch file (.BAT) that copies files to a resource directory or run an executable program (.EXE) to send an e-mail message when a revision is checked in.

To set up event triggers, specify a program on the appropriate line in the Administrator.

This is very useful with multi-APP projects, where LIB dependencies must be kept in mind.

For example, a pre-Check In trigger might call a batch file that copies the LIB, DLL, and EXE files to a common reference directory (use XCOPY with the /d switch to only copy newer files). Then a pre-Check Out trigger might call a batch file that copies the LIB, DLL, and EXE files from this reference directory to the working directory.

Furthermore, the event trigger contains parameters that reference the User ID, the name of the archive and workfile, the revision number, etc. that can be passed to the batch file (or program).

Tip: You can prevent the display of a DOS box by specifying a PIF for the BAT file instead of the BAT file itself.

Event Information

The event trigger obtains specific information about the event when it occurs as shown below. Not all information is available for every event.

	Check In		Check Out	
	PrePut	PostPut	PreGet	PostGet
EventArchive	■	■	■	■
EventChgDesc	❑	❑		
EventDate	■	■	■	■
EventLock	❑	❑	❑	❑
EventName	■	■	■	■
EventRevision	■	■	■	■
EventTime	■	■	■	■
EventUserID	■	■	■	■
EventVersion	❑	❑	❑	❑
EventWorkfile	■	■	■	■

■ This parameter is always available.
❑ This parameter is sometimes available based on its usage.

EventArchive
The name of the archive including the full path.

EventChgDesc
The change description. Limited to 1kb in size

EventDate
The date when the trigger was executed, in packed decimal numeric form. For example, 20 January 1998 appears as 980120.

EventLock
Contains the value YES if a lock was requested and is not defined otherwise.

EventName
The name of the event (PrePut, PostPut, PreGet, etc.)

EventRevision
The revision number being operated on.

EventTime
The time when the trigger was executed, in packed decimal numeric form. For example, 2:35:26PM appears as 143526.

EventUserID
The user ID of the person performing the operation.

EventVersion
The version label.

EventWorkfile
The name of the workfile including the full path.

Passing Event Information

The Version Control System passes event information through the environment or command-line macros. If information is defined in environment variables, the event trigger queries the environment for values it needs.

For example, a batch file might query %EVENTARCHIVE%, while a 32-bit Clarion program would perform a GetEnvironmentVariableA() API call. See the *Special Considerations* section for more information. In the situations where event information cannot be passed using the environment method, use the *command-line macros* method.

Command-Line Macros

An event trigger command line can reference event information macros. These macros have the same name as the event information names listed above, but they must be enclosed with double underscores, such as __EventArchive__, __EventWorkfile__, etc. VCS expands the macro when it executes the event trigger.

The following example illustrates an EventTrigger directive configured to call XCOPY, copying the workfile to a review directory if it is newer than the existing file:

```
XCOPY __EventWorkFile__ x:\project1\review /d
```

When the event occurs and VCS executes the trigger, the command line expands to the following:

```
XCOPY c:\projects\project1\source\project.dct x:\project1\review /d
```

Revision Storage Options

Revision storage options help you control archive size and processing speed. Use these options to control whether the Version Control System does the following:

- Stores deltas or workfile copies. Deltas are the changes between one instance of a workfile and its predecessor. You can choose to store deltas or to store complete copies of the workfile. While storing complete copies makes the archive larger, it is more efficient for performing actions on large binary files.
- Compresses deltas. If you choose to store deltas only, you can compress them to reduce archive size.
- Compresses the work image. The work image is a complete copy of the tip (latest) revision on the trunk and each branch. You can compress the work image to reduce archive size.

While compression saves disk space, it also decreases performance. If you have adequate disk space, you may not want to use compression at all. To select compression options for your particular working environment, consider the following:

- Number of changes to stored workfiles. If you make numerous changes to a small text file, compressing the deltas won't result in significant disk space savings. However, if you make numerous changes to a large text file, you may be able to gain disk savings by compressing the deltas, work image, or both.
- File types. Non-text files, such as Clarion DCT, file locked APPs, and database files, usually produce large deltas, even if changes to the files are minor. For these file types, you can produce significant disk savings by compressing the deltas only. To reduce processing time when performing actions on these file types, you can store complete copies of the workfile instead of generating deltas.

You can use the statistics in the table below to help decide whether you should use compression. Your results might be different.

File Type	Disk Savings	Increase in Check Out Time
10kb text file	54%	25%
50kb text file	50%	40%
50kb executable	25%	50%
250kb executable	23%	25%

Tip: **By default, the Version Control System does not compress archives. When you turn compression on, the VCS compresses revisions in new archives, not existing ones. The default for storing prior revisions is to use deltas.**

To specify revision storage options, add records to the table. Use the workfile extension (APP, DCT, etc.) and select Yes or No to compress the tip and/or the deltas and whether to store deltas or workfile copies.

Security

Provide Security for VCS Administrator only?

This checkbox allows entry of users so you can secure the use of the Administrator without enabling security for all VCS functions. This was turned on above in *First time in the Administrator* for that purpose. To use security for archive access requires you to enter all users into a database, as well as assigning privileges to each user.

For more information on security, see *Advanced Tasks*.

Advanced

Miscellaneous Directives

This text box is provided for experienced PVCS users and complex sites working with technical support. PVCS directives entered here are placed directly in MASTER.CFG without validation or error checking. This immediately affects all Version Control System users even if they are already running Clarion when you apply the changes.

In general, the use of this option is not recommended.

Configuration Strategies

Single developer, non-networked environment

Create a directory for each project. Under the project directory, create subdirectories to group project files by function.

Create an archive directory as a subdirectory of each project directory.

Use the VCSID environment variable for the developer ID source. Set your configuration options from the Archive menu in Clarion. You do not need to use the VCS Administrator.

Use file-locked APP archives.

Single developer, networked environment

Create a directory for each project. Under the project directory, create subdirectories to group project files by function.

On the network, create an archive directory for each project .

Use the appropriate network login for the user ID source. Configure your workstation within Clarion. You do not need to use the VCS Administrator.

Use file-locked APP archives.

Multi-developer, single-project environment

Create a network directory for your project. Under the project directory, create subdirectories to group files by function.

Duplicate this project directory structure on each developer's workstation. Users should keep only the source files that they are currently editing on their workstations. Use event triggers to move copies of .LIB or executable files between the network directories and developer's workstations.

Create an archive directory on the same level as the project directory on the network.

Use a single location for the master configuration file to ensure that all developers use the same directive settings. Use VCS Administrator to configure VCS behavior. You should also set up security to use access lists, protecting archives from unauthorized access.

Use the appropriate network login for the user ID source.

Use procedure-locked APP archives. If some development might be done on the complete file in some applications, leave it to the user to choose which type of locks to use when creating archives.

Multi-developer, multi-project environment

Create a separate network directory for each project. Under each project directory, create subdirectories to group files by function.

For each project a developer is working on, duplicate the network project directory structure on the developer's workstation. Users should keep only the source files that they are currently editing on their workstations. Use event triggers to move copies of .LIB or executable files between the network directories and developer's workstations.

Create an archive subdirectory under each network project directory.

Use a single location for the master configuration file to ensure that all developers use the same directive settings. Use VCS Administrator to configure VCS behavior. You should also setup security to use access lists to protect archives from unauthorized access.

Use the appropriate network login for the user ID source.

As in the single-project environment, use procedure-locked APP archives. If some development might be done on the complete file in some applications, leave it to the user to choose which type of locks to use when creating archives.

Customizing the Version Control System

This section contains information about a required workstation configuration setting to enable full use of the Version Control System and optional customizations that can increase productivity.

Setting your Version Control User ID

If you didn't enter your user ID when you installed the VCS, you will have to do so now—a user ID is required to share APP files. To specify your user ID after installation, edit or add the following line to your AUTOEXEC.BAT:

```
SET VCSID=your_name
```

Replace *your_name* with the VCSID you want. Do not use any space characters. If you make this modification after setup, you will have to reboot your computer before using Version Control.

File Masks

You may store file types (masks) in the C5VCS.INI file to customize the files available for Check In and Check Out. Edit C5VCS.INI in your Clarion5\BIN directory, with entries similar to this:

```
[Workfile Types]
1=Application (*.APP)
2=Project (*.PRJ)
3=Includes (*.INC)
```

You must retain a numeric sequence— the VCS will stop reading this section when there is a break in the numerical sequence. If the INI file or this section name is not found, the VCS uses its built-in defaults.

The file masks available to select workfiles in a project definition are also separately configurable in the INI file. Here is an example:

```
[Config File Types]
1=Application (*.app)
2=Dictionary (*.DCT)
3=CLARION Source (*.CLW)
```

Security

If your location uses the security features of Version Control System, like logins, assigned privileges, etc., you should read the *Security* section in *Advanced Tasks* now and set your security plan into action.

Using VCS Status

The VCS Status utility queries the archive directories listed in the master configuration file and displays any files which have locks. Workstations that are not set to read the same MASTER.CFG file will not see the archives of other users.

When the program is launched, it builds a list of archives, locks, and user IDs. This is displayed in two views—**by File** and **by User**. Each of these shows a tree detailing who has placed file or procedure locks. Use the refresh button to rescan the archives and update the display.

If a Check In or Check Out is in progress then the only information displayed about that archive is the user ID of the developer who has the file in use and a message—Archive in use, try later—displays.

Using the Version Control System

The two basic actions performed in Version Control´—Check In and Check Out—are covered in detail in this section.

When you check out a revision from an archive, the version Control System creates a workfile in the destination directory. Getting a writable copy of a file causes a lock to be placed on the archive, which prevents developers from overwriting each other's changes.

You can check out a specific revision by revision number or version label. If you don't specify one of these, you will get the tip, or most recent, revision.

Once the file is checked out you edit it normally. If it is read-only or contains read-only procedures, it will generate and compile for you to test your work.

Archive Menu

When there are no files open, the following menu options appear under **Archive:**

Check In
Places a copy of a workfile into archive.

Check Out
Retrieves a workfile from archive.

Project Definition
Allows you to create a CFG file that will allow you to move multiple workfiles into and out of archive in one operation.

Utilities
Provide low level tools for maintaining the archives

Reports
Provide information on current lock status and prior changes made.

Configuration
Allows you to specify various parameters controlling the VCS behavior.

Once files have been opened, the following menu options appear under **Archive:**

Checkpoint filename.ext
Performs a save of the file and check it in to archive, leaving it open for continued work.

Revert filename.ext

Closes the current file, restore the file as last checked in by you, (this will overwrite the current file but maintain any locks) and reopen the file.

From the Clarion Pick List, the **Check Out** dialog also displays when you select a file which is archived with the workfile deleted. This allows you to use the Pick List in the same manner as you would for applications not under version control.

Reminder messages

There are two reminder messages that assist in learning the Version Control System. They are enabled at installation, and can be disabled under Advanced Configuration. The messages appear in two cases:

- After closing a file, the VCS asks if you want to check in the file that you just closed. If you press the Yes button, the Check In dialog appears.
- After Check Out of a revision, VCS reminds you how to open the file that you just checked out. This message does not appear for APP files if you have enabled the **Load Application after Check Out** option.

Basic Tasks

Check In

To save a copy of your work at the current state, you check in a copy of it. This creates a new revision in the archive. Check in creates an archive file if one is not found in your configured archive directories.

The archive file contains a complete copy of the last revision added to it and binary deltas to create every prior revision. The Version Control System records your user ID and the date and time of the check in, and allows the entry of up to 64kb of programmer's notes. If you checked out the file in write-mode, checking it in removes the locks from the archive so other developers can now check out a writable copy of the file.

You should check in your workfile after you have changed it, to make those changes available for use at another time or by other developers. This could be at any milestone, such as after getting a new procedure to work, before attempting a fundamental change, or at the week-end tidy up. Unless you are passing a potentially troublesome procedure to another developer, it is important to only check in code that compiles successfully—otherwise you may prevent others from testing their changes.

Selecting a file to check in

You begin by selecting a file to check in using a fairly standard dialog with a File Type selector to filter the entries in the Files list box. The title bar indicates the active filter.

You can further limit the file list to **Only show files under version control** which only displays items with a corresponding archive. This is useful when you might have several test versions of APPs in addition to the actual project APPs.

> **Note:** **The Only show files under Version Control option must be unchecked in order to place a file into version control the first time.**

The list box allows multiple file selections, CTRL-CLICK to select multiple files one-by-one or SHIFT-CLICK to select a range of files.

Select the file(s) to check in by selecting the correct drive, directory, and file. The working directory and filters are stored and used as defaults for the next Check In.

File Types and Related Actions

Your selection of a file type to Check In affects various parameters and defaults. In general, Check In adds a revision with the next incremental revision number and performs the **After Check In** trigger action.

The droplist contains the following file type filters:

***.APP**
Displays only the Clarion Application files in that directory,

***.DCT**
Displays only the Clarion Dictionary files in that directory,

***.PRJ**
Displays only Clarion project files for a hand-coded program,

***.CFG**
Displays only the Version Control System's project definition files, and

.
Displays all files

On Check In, if the .APP file or any of its procedures was checked out with a lock, a revision is added to the archive. If you specify **Delete Workfiles** only the APP file is deleted; the generated source files (.CLW, .INC, etc.) and the DCT are not deleted.

The DCT file type checks in a Clarion dictionary. You are reminded at Check In to fully document your changes, as removing files or fields can have dramatic effects on an APP.

For hand-coded programs, use the *.PRJ file type to select the project you want. Version Control System reads the .PRJ file and searches the archives for the .CLW files in the project. On Check In, all writable files are moved back to archives.

A project definition file (*.CFG) allows you to work with groups of files in one operation. On Check In, any files opened with a lock are added to archive. For more information about Project Definition files, see *Advanced Tasks*.

. allows you to select one miscellaneous file for Check In. If you routinely need a set of other files, you should create a project definition file for them..

Check In Options

The Check In List allows you to specify here optional settings for check in behavior.

To Check in files:

1. **Select** a file to check in

   Version Control System builds a list of all files to Check In. Of course, if the selection is an APP or DCT file, or a miscellaneous file (*.*), only one file is archived. The number of files to process displays, along with the first file.

2. **Press** the **Skip** button to bypass a file, if needed

3. For each file, you can **provide** a change description and specify the action you want to perform on the workfiles after check in.

4. **Specify** a version label or revision number, if appropriate.

5. **Press** the **Check In All** button to apply the same settings to all of the files, or step through the files one-at-a-time using the **Check In** button, modifying any settings, as necessary. If there is only one file the **Check In All** and **Skip** buttons are disabled.

Note: If an archive for this workfile was not found in any archive directory, you see the message alerting you that the archive was not found. If you continue the Check In, the Create Archive dialog is displayed.

Change Description

The change description allows you to record changes made since the last revision. This is used in reports and provides a history of additions or modifications you have made. This entry allows 64kb of text, and word-wrap is enabled.

For an APP file, the VCS automatically fills in the change description with the name and version of the DCT used (if the DCT is read-only) as well as the names of the procedures modified. You can simply add your notes next to each procedure name.

Check the **Copy to blank descriptions** box for the one description and it is copied to other files which do not have a change description.

Workfile Management

The After Check In radio buttons specify the action to perform on the workfile after it is checked in to archive. These actions prevent modifications to workfiles outside of version control. This setting is stored and used as the default for the next Check In.

The options are:

Keep read-only workfile

This option allows you to keep a copy of the workfile for review or testing.

Delete workfile
This is useful if you like to keep your working directories clean—only having files there if they are in the process of being edited.

Keep writable workfile and lock
Choose this option if you plan to keep editing the workfile but want to archive your work in progress. This setting allows you to load an APP file after check in is complete.

Revision numbers

The revision number is made up of a major number, a period, and a minor number – for example, 1.14. Either of these numbers can range inclusively from 0 to 65,535.

Generally you should let the Version Control System assign the next revision number for archives. The default action is to increase the minor number by one. When you want to use a new major number, or use a specific revision number, mark the **Specify** (revision number) option and type a revision number. The major number must be the same or greater than that of the last revision. If the major number is not incremented the minor number must be greater than the minor number of the prior revision.

Version Labels

Version labels provide a “friendly” description of a revision. Version labels enable you to operate on all of the files associated with a particular version at the same time by referencing a single name.

We recommend adding a version label on every potential release. Labels like ‘Beta 2 Build L’, etc. are very descriptive and useful to identify candidate builds. This also makes it easy to use the utilities and add a second label to the released revisions.

A floating version label is a version label that stays or “floats” with the tip revision in an archive. This type of label is useful when you want to refer to the latest revision in an archive without knowing its revision number. To create a floating version label, check **Floating Label** box.

Check In Templates

When you are checking in an APP file or a CFG file that contains one or more APP files you can also check in the templates at the same time. To enable this option, a template definition file must be specified in the local configuration or project definition file (or both) and a version label is required.

Creating an Archive

When you check in a file and a corresponding archive file is not found, Version Control System creates a new archive. This dialog lets you describe the workfile and select an archive directory to create the archive file in.

If the workfile is a Clarion application file (.APP) and you have specified *Always Ask* as the default APP file handling, you are prompted for the locking method to use for the APP file. Press **Yes** to create an archive that supports procedure locking and **No** for a file-locked archive.

Access list

If you are using the security features, you will see an **Access List** button on this dialog. Press it to open the brings up a pair of list boxes. On the left are the groups and users defined in the administrator. Highlight and move groups and users that should have access to this archive to the **users/groups with access** list. If you do not create an access list for an archive, any user can access the archive, but only perform the actions for which they have privileges.

Check Out

Checking a file out of archive gives you access to it for review, editing, or printing. When you check out a revision from an archive, the Version Control System creates a workfile in the destination directory. Getting a writable copy of a file places a lock on the archive, which prevents users from overwriting each other's changes, and causes a new revision to be created in the archive when it is checked in again.

You can check out a specific revision by revision number or version label. If you don't specify one of these, you will get the tip, or most recent, revision.

Selecting a File to Check Out

You begin by selecting a file to check out. Select an archive directory and use the File Type selector to filter the entries in the Files list box. The title bar indicates the active filter. Indicate the destination directory by selecting the correct drive and directory. The archive directory, destination directory and file type settings are stored and used as defaults for the next Check Out.

1. **Highlight** the file you want to check out.
2. To view detailed information about versions stored in the archive, **press** the **Revisions** button.

3. By default, the tip (latest) revision is checked out. If a you want a different revision, **press** the **Revisions>>** button to review the list of Revisions and select the one you want. Note that the revisions list box also displays version labels (indented). You can select version labels in the same manner as revision numbers.

> **Tip:** **When you check out a non-tip revision with a lock and check it back into archive, it becomes a branch of that revision. There is not currently any mechanism provided to merge this branch with a later revision, and you will be warned on Check In if a branch would be created.**

For example, one might create a branch when revision 3.1 has already been sent to the customers, but a defect is later found, while you are working on 3.8, which is incomplete. You would check out 3.1, correct the defect, and on check in it becomes revision 3.1.1.1. You will then have to correct the same code in 3.8 because you cannot merge a branch.

File Types and Related Actions

Your selection of a file type to Check Out changes various parameters and defaults. In general Check Out retrieves a copy of the workfile(s) selected and locks the archive as specified.

The droplist contains the following file type filters:

***.APP**
Displays only the Clarion Application files in that directory,

***.DCT**
Displays only the Clarion Dictionary files in that directory,

***.PRJ**
Displays only Clarion project files for a hand-coded program,

***.CFG**
Displays only the Version Control System's project definition files, and

.
Displays all files

When you Check Out a procedure lock-enabled .APP file, the dictionary is required. If it is found in the destination directory or the directory specified by the APP file, then no action is necessary. If the dictionary is not found there, or is a read-only copy, the Version Control System looks in the archive directories for an archive of the dictionary. If an archive is found, a copy of the tip revision is retrieved without a lock (read-only) and automatically placed in the destination directory.

Note: If the dictionary is still unavailable you will be notified and the Check Out will abort.

The DCT file type checks out a Clarion dictionary.

For hand-coded programs, use the *.PRJ file type to select the project you want. The VCS reads the .PRJ file and searches for archives for the .CLW files in the project. The default on Check Out is to retrieve writable files with the archive locked.

The project definition file (*.CFG) allows you to work with groups of files in one operation. Check Out gets the tip revision of the files specified, defaulting to retrieve writable files with the archive locked. See *Advanced Tasks* for more information.

. allows you to select one miscellaneous file for Check Out. If you routinely need a set of other files, you should create a project definition file for them.

APP Files

The **APP File Check Out** dialog lets you easily specify which procedures you may edit in this session.

The Version Control System defaults to retrieving all procedures without a lock—you must select any procedures you want to edit and lock them. You cannot edit any procedures that another developer has already checked out (and therefore locked). You can select multiple procedures with the keyboard (SHIFT + ARROW KEYS) or the mouse (CTRL-CLICK) or just DOUBLE-CLICK to move procedures back and forth from the **Lock** and **Don't Lock** lists. Arrange the procedures in the appropriate column for locked or not then set any desired options before pressing the Check Out button. Lock selections are saved for each developer and used as defaults the next time you check out this application.

If you have checked out this APP and need to lock additional procedures, you can move them to the lock list. Your existing locks remain until you check them in. Check Out preserves the work in progress (the read-write procedures in the existing APP) and refreshes all other procedures in the APP; setting newly locked procedures to read-write. This is an easy way to bring your APP up to date with the work of other developers even if you don't add locks.

If you want to revert to where you started, use the **Overwrite locked procedures in existing APP file** option. This updates every procedure in your copy of the APP file with the latest revision in the archive, replacing your work this session

but leaving your locks in place to allow you to continue working on the procedures you have checked out.

The VCS records the version of Clarion you used when the APP was last checked in. If you are using a different version on Check Out, you are warned and given the chance to abort. Read the release notes for each new version of Clarion for details about APP file upgrades.

Other Files

Miscellaneous files ('*.*') check out the one file that was selected. Checking out Clarion project files ('*.PRJ') or Version Control System project definition ('*.CFG') files, builds a list of all files in the project. You have the opportunity to retrieve any or all of these files.

Note: APP, PRJ, and CFG files are handled correctly even when selected as a miscellaneous file ('*.*').

The **Check Out List** dialog displays the number of files to process with the name of the first file displayed. The number of files to be processed decrements as the Check Out proceeds through the files to be processed. You should verify that the destination directory is correct, decide whether the archive should be locked or not (the **Select Locks** button allows you to select the files to lock).

The **Check Out All** button processes all files. If you need to change the destination of some of the files, use the **Check Out** button to retrieve files one at a time, and change the directories as necessary. If you have to work with a project that requires various archive and/or working directories, or a combination of locks/no locks, you should create a project definition file.

Select Locks

Press this button to display a list which indicates how files should be retrieved in this session. The Version Control System defaults to retrieving all files with a lock. Lock/no lock selections are stored for project definition files and used as defaults for subsequent Check Outs. These settings are superceded by the settings in this list. You can select multiple files for selection with the keyboard (SHIFT +ARROW KEYS) or the mouse (CTRL-CLICK) or double-click on a file to move it from one list to the other. Arrange the files in the appropriate column (for locked or not), then press the **OK** button.

Revisions>>

Press the this button to view detailed information about revisions stored on a file-by-file basis as you step through the file list with the **Check Out** and **Skip** buttons. Note that the revisions list box also contains version labels (indented). You can selected version labels in the same manner as revision numbers.

If you are checking out a non-tip revision that does not have a version label, you will have to step through each file, selecting

the revision you want and pressing the Check Out button. If a version label is present you may select it for the first file and press the **Check Out All** button.

Lock
Check the **Lock** box to lock every file checked out regardless of lock settings in the **Select Locks** list or project definition. You can change this on a file by file basis when retrieving the files using the **Check Out** or **Skip** buttons.

Skip Files not Locked
Check this box to speed up the check out process by skipping retrieval of files that are not flagged to be locked. Remember you probably need all the files that make up a project to compile and test your changes.

Refresh ReadOnly Workfiles
Check this box when you have previously checked out a project and wish to discard your changes and replace your workfiles with the last archive. You must be the same user who locked the archive. If you do not want to continue working, and want to remove locks without checking in, use the Utilities.

Undo a Check Out

If you have placed locks that you need to remove but do not want to Check In the file(s) to remove the locks, follow these steps.

Procedure-locked APP file

1. **Select** **Archive ➤ Utilities.**
2. **Locate** the file by choosing the archive directory it is located in and **highlight** it.
3. **Press** the **Procedures** button
4. **Select** the locked procedures and **press** the **Unlock Procedures** button.

   You can select multiple procedures in this list box.
5. When you have set all the procedures that you want to unlock, **press** the **Apply Changes** button.

> **Note:** **You should delete the workfile so that modifications are not inadvertently made. With the locks removed, the workfile cannot be checked back in.**

All other files

> **Note:** **If the security model is not in use, only the owner of the locks can remove them. Using security, developers granted the BreakLock privilege could remove any locks.**

1. **Select** Archive ➤ Utilities.
2. **Locate** the file by choosing the archive directory it is located in and **highlight** it.
3. **Locate** the revision where the status indicates it is locked (the lock owner is the first line in the description).

   This is usually the tip (latest) revision.
4. **Press** the Unlock button.

> **Note:** **You should delete the workfile so that modifications are not inadvertently made. With the locks removed, the workfile cannot be checked back in.**

Undo a Check In

If you have Checked In a revision that you want to remove, follow these steps. The window of opportunity to prevent propagating a bad revision is short, it ends when there is any activity on the archive—a check in or check out.

Procedure-locked APP file

Because multiple developers may simultaneously have procedure locks in an APP some coordination is required to remove the tip revision. In many cases it may be more efficient to remedy coding errors by simply Checking Out the file again, correct the deficiencies, then Check it In again.

1. **Select** Archive ➤ Utilities.
2. **Locate** the file by choosing the archive directory it is located in and **highlight** it.

   Review the description of the tip revision to ensure this is the revision that you mean to delete. If another revision has been added since the one you wanted to undo deleting it will not affect the new tip, you will have to Check Out the file again, correct the deficiencies, then Check it In again.
3. If there are any locked procedures, you must remove them to continue.

   If security is not in use then each developer will have to release their locks using steps a and b below. Otherwise have a team leader (or anyone with the BreakLock privilege) remove the locks

4. **Select** the procedures to unlock and **press** the **Unlock Procedures** button. You may select multiple procedures in this list box.
5. When you have set all the procedures that you want to unlock, **press** the **Apply Changes** button.
6. **Press** the **Delete Revision** button. If you are certain this is the correct revision, **press** the **OK** button on the confirmation dialog, otherwise Cancel.
7. Have each developer **move** their working APP file to a safe location.
8. Have each developer **check out** the APP again. The previous procedure locks are automatically listed in the Lock on Check Out list box.

   It is important that they do not add or remove procedures from this list.
9. Once each developer has re-established their locks, they can **copy** the saved APP file back to the working directory, overwriting the one checked out in step 8.

All other files

1. **Select** **Archive** ➤ **Utilities.**
2. **Locate** the file by choosing the archive directory it is located in and **highlight** it.

   Review the description of the tip revision to ensure this is the revision that you mean to delete.
3. If the revision is locked, **press** the **Unlock** button
4. **Press** the **Delete Revision** button. If you are certain this is the correct revision, **press** the **OK** button on the confirmation dialog, otherwise Cancel.
5. If the revision was locked, and you still have a writable copy of the workfile, **press** the **Lock** button now so you will be able to Check In your work when it is satisfactory.

Reports

Archive reports summarize information about archives and their revisions. It provides the team with information about projects which is useful to monitor the development process, review file histories, and identify lock owners.

Reporting is easy in the Version Control System. Simply select a file to report in the same manner as you select a file to Check Out, and choose a report type. After previewing the report, you may save it to disk or print it. Closing the preview dialog returns you to the Report dialog, where you can choose another report type on the same archive. For Clarion projects and

project definition archives that contain multiple files, the **Report All** button is available. This creates a single report for all of the files in the project.

Types of Reports

There are five reports you can generate for any file archive, and one that is only available for APP files:

Full
Contains both the archive summary and the revision summary.

No Revision Information
Contains the archive summary only, including:

- Archive and workfile names
- Date and time of archive creation
- User ID of the archive owner
- Last revision number
- Number of revisions in the archive
- Revision numbers of locked revisions (if any)
- User ID of the lock owners (if any)
- Archive attributes
- Version labels and associated revisions (if any)
- Workfile description

Revision Information Only
Contains the revision summary only, including:

- Revision number and change description
- Date and time of check-in and modification
- Number of added, deleted, and moved lines
- User ID of each revision author

List Locked Revisions
Lists each locked revision and the locker's User ID.

List Version Labels
Lists each version label and the revision number to which it is assigned.

List Locked Procedures
Available for APP files only, this report lists each locked procedure, the lock owner's User ID, and the revision number for which it was locked.

Advanced Tasks

Project Definition Files

One of the most powerful features of the Version Control System is the project definition file. This allows you to move multiple files in and out of an archive in a single operation. The files may be stored in more than one archive directory and each workfile can has its own working directory. Lock selections are not specified in the project definition dialogs, but are retained for each user after Check Out. Procedure-lock enabled APP files can be placed in a project definition—the **APP File Check Out** dialog is presented for each one on check out.

The project definition file is a text file that contains workfile names (and optionally, their working directories), and can include archive directory names, event triggers, and a template definition file. It is archived also, in the first archive directory listed.

Maintaining Project Definition files

You should create a new file in the main working directory of your project. When working with an existing file, you will have to check it out with a lock in order to modify it.

To create a Project Definition File:

1. **Select** **Archive** ➤ **Project Definition.**
2. **Add** workfiles to the Project Definition File (see *Adding and Removing Workfiles*).
3. Optionally, **specify** Archive Directories (see *Archive Directories*).
4. Optionally, **specify** any Advanced options (see *Advanced*).
5. Optionally, **specify** any Event Triggers (see *Event Triggers*).
6. Optionally, **specify** any Template Definition Files to include in the Project Definition (see *Template Definition Files*).
7. **Press** the **Close** button.

   If you close a modified Project Definition File, you are prompted to save.
8. **Press** the **Yes** button , specify a filename, then **press** the **Save** button.

The Project Definition dialog has its own **File** menu, supporting standard file actions (e.g., Save).

Adding and Removing Workfiles

To add workfiles, press the **Add Workfile** button. This opens a standard file dialog, with various File Type selectors to assist file location. Multi-select is enabled in the file list box. Pressing the OK button adds the selected files to the project definition and returns you to the file select dialog. Press Cancel when you are finished selecting files.

If you check the **Keep path with Project Definition workfiles** box in the Advanced Configuration dialog, the full path (including drive letter) is stored with each workfile. This is useful when checking in and out files from multiple directories using a project definition file. If developers use different drives but the same directory structure, set the **Workfile Drive** in the Advanced Configuration for any developer who is using a different drive. On Check Out, the drive in the workfile path is be replaced with their specified drive.

However, if the same path is not available for all developers, you should clear the **Keep path with Project Definition workfiles** checkbox so that only the workfile name is stored. You can still use multiple working directories, but you will have to step through the Check Out process one file at a time to set the correct directory for each file.

After you have added all the files, you can to move it up and down in the list or remove it from the list. If you are not able to use the **Keep path with Project Definition workfiles** option, arranging the files by destination directory can save time during Check Out.

Archive Directories

You can optionally specify archive directories for a project. If directories are not specified here, then VCS looks for archives in the directories listed in the master configuration file. The archive directories listed in a project definition file ***replace*** the directories listed in configuration for actions on the project and its files. This allows you to use the same file name for different files in various projects and keep them in different archives.

For example, suppose that in \ACCT you have a file, GL.APP. The archive file is in \ARCHIVE\ACCT. If you modify the application for a customer, a different copy of GL.APP might be in \ACCT\SMITHCO. However, because the Version Control engine searches for archives in the archive directories listed, and uses the first one found, checking in \ACCT\SMITHCO\GL.APP could result in its being stored in the wrong archive. Using project definition files for each of these, and specifying unique archive directories, ensures that the revisions are stored in the intended location.

Add archive directories by pressing the **Select Archive Directories** button. This dialog allows adding and sorting archive directories in the same manner as the configuration dialog. Highlight the directory in the left-hand list box and

press the **Add** button. If you want to remove an entry from the archive directory list, highlight it and press the **Remove** button.

> **Tip:** **The Version Control engine searches for archives in the order that the archive directories are listed. Use the Up and Down buttons to minimize search time.**

Advanced

Press the **Advanced** button to specify event triggers and a template definition file for this project. Entries made in each of these items supercede the settings stored in other configuration files.

To recap, event triggers allow you to specify a command line which executes when the prompt indicates (e.g., Prior to Check In) and can also be configured globally in the master configuration file using the VCS Administrator.

Template definition files are a special case of a project definition file and can also be specified by the user in Archive ➤ Configuration ➤ Advanced.

Event Triggers

Enter the name of the file you want to run when the files in this project definition file are checked in or out. The events occur for each individual file. See *Event Triggers*' in *Using the VCS Administrator* for more details about event triggers.

Template Definition Files

The template definition file is simply a project definition file itself that is processed last. An example file—TEMPLATE.CFG— is created during setup and is placed in the \CLARION5\TEMPLATE\ directory.

A Template Definition File is a list all of the files for a project that are needed to recreate any prior version exactly as it was when archived, and to place them under version control also. The example file lists all of the template files as well as the source files in the \CLARION5\LIBSRC directory. You may wish to add other files, such as third-party templates, local redirection files, etc.

To be able to automatically move these files with an APP file you must specify the full path and name of the template definition file in the local configuration. To archive these files with a project, specify a file here. This can be a different file and overrides the local configuration.

> **Note:** **The VCS only processes a template definition file when at least one APP file is a part of the project, and a version label is specified at Check In or Check Out.**

Utilities

Use the utilities to lock or unlock a specific revision, to delete revision(s), and maintain version labels for revisions within an archive.

Select an archive directory to display a list of archives, then select an archive to display its status, revisions, and revision information. View any of the revisions within the selected archive by selecting it in the **Revisions** list box. The status area displays the lock status of the selected archive and selected revision and, for an .APP file, if there are locked procedures.

Revision Maintenance

At times, you may want to manually set the status of locks on a revision. Perhaps a file was checked out without a lock, the read-only attribute removed, and you want to save the modifications made (a lock is required to check in a revision). Or a developer might leave the project with several procedures locked, and no one can check them out.

The **Lock** button locks the selected revision in the selected archive. It is enabled only if the selected revision is not locked. Typically, you should lock the tip (latest) revision to create a new tip. Similarly, the **Unlock** button unlocks the selected revision and is enabled only if that revision is locked.

The **Delete Revision** button permanently deletes the selected revision in the selected archive. It is enabled only if the selected revision is not locked.

Note: There is no recovery available once a revision is deleted.

Procedure Locks

Lock/Unlock of a revision is disabled for procedure-locked .APP files, as this is a lower-level function. Press the **Procedures** button to display locked procedures within the selected APP file and the User ID of the developer with the lock. This button is only available when you select the tip (latest) revision of an APP file. If the security features are not in use, only the person who placed the locks can remove them. If security is in use then the person who placed the locks as well as users with the BreakLock privilege (included in TeamLeader and SuperUser) may remove the locks.

Tip: If you remove ANY procedure locks, the APP that the developer has CANNOT be checked in and any changes made to that or other locked procedures are forfeited.

You may select multiple procedures here, using CTRL-CLICK OR SHIFT-CLICK. The **Unlock** button processes the selected procedures and displays *<Lock will be removed>* for procedures that you have the right to remove. Press the **Apply Changes** button to implement the unlock status to the archive. The **Cancel**

button exits the APP File Procedure Locks dialog box without modifying the lock status in the archive.

Access List

If you are using the security features, you will see a button for the **Access List** on this dialog. This opens a pair of list boxes. On the left are the groups and users defined in **VCS Administrator**. Highlight and move groups and users that should have access to this archive to the **Users/Groups with Access** list box. If you do not create an access list for this archive, any user can access the archive, but only perform actions for which they have privileges.

Version Labels

We recommend using version labels. The utilities allows you to manage them. Select an archive file to work with, then select a revision, or version label. You may add a version label to a selected revision, or use an existing version label.

To edit or delete a version label, you must select a specific version label.

Adding a version label to a project definition file will result in a dialog offering to add this version label to all files that make up the project definition.

If you selected a specific revision of the project definition file the version label is applied to the same revision number in all the project files. This is generally only a good idea for revision 1.0 After the first revision, the revision numbers will likely not be the same for every file unless one developer has been checking them all out and in together. To modify this behavior, check the **Apply Version Label to Trunk Tip** or **Floating Version Label** box. If you choose to apply it to the tip, the version label appears under the highest revision for the selected major number of the definition file. If you specify a floating version label, it appear under the highest revision for the major number you add it to, and will continue to float to each successive revision until the major number changes. For example, if you added a floating version label to revision 1.12, it would float up to 1.84, but not to 2.17.

You can also add a second version label to an existing one; this applies it to all files at the same point in development. For example, if your “Beta 2, Build L” passes quality assurance testing and is released as “Version 2.1, “ you can add the version label “Release 2.1” to the “Beta 2, Build L” label.

Security

Security and Team Development

Security establishes a framework designed to protect data. The structure provides developers with different levels of expertise and responsibilities the access privileges they need to do their jobs, while preventing actions outside of their abilities or areas of responsibility.

The Version Control System provides two methods of managing security:

Access lists
An *access list* is a list of users and groups stored ***in*** an archive that specifies who is authorized to perform Version Control operations on that archive. To access an archive containing an access list, a User ID or Groups to which that user belongs must be in that list. If an archive does not contain an access list, all users have unlimited access to it.

Access privileges
An *access privilege* is a special word you assign to users and groups to control whether they can use particular Version Control features. If you do not assign privileges to a developer, that developer will have unlimited use of VCS features in any archive to which the developer has file access.

Security Strategies

Using access lists is the simplest form of security. You create a database of users and groups, but need not assign privileges for most users. For archives you want to protect, you create an access list that specifies who can work with the archive.

Another method is to assign privileges only to developers who must be restricted. Developers without assigned privileges are free to access archives, but a developer with the GetTip privilege, or any Non-Team Member, is not be able to place locks on archives.

Most commonly, all users and groups are entered into the database and the privilege assignment will be among the composite privileges: Team Leader, Team Member, and Non-Team member. Using access lists in addition to this provides the highest security possible.

Setting up Security in VCS

To use either or both of these security features, you must add users to an access control database. The Version Control System uses this database to determine which archives each user can access and which features they can use. The access control database is maintained using the VCS Administrator.

To use security, all users must be configured to use the same location for the master configuration file. You can verify this by examining the ISLV.INI file in each developer's WINDOWS directory:

```
[PRODUCT_LOCATION]
PVCS OEM VM_6.0.00_WIN32=drive:path
```

This drive and path is the location of MASTER.CFG. Workstations that are not set to read use this file are affected by settings made in the VCS Administrator.

The VCS Administrator is not added to the Clarion group during installation to prevent experimental use that could lock everyone out of their projects. You can create a shortcut to VCSADMIN.EXE in your \CLARION5\BIN directory to use the VCS Administrator.

The first time you run the Administrator there are no security checks. Subsequent launches require a password and the privilege to modify the access database. The Administrator alerts you on exit if there are no users with the required privilege and a password.

Tip: **Take the time to write out a security plan of users, projects, and application dependencies for the developers of at least one project, before launching the Administrator. Without a security plan you could lock everyone out of their projects.**

Privileges

Privileges are predefined words that you assign to a user to permit the use of one or more Version Control features. You assign privileges according to the security level of a developer's working environment. You can assign liberal privileges that enable users to perform almost any action, or limit their privileges to only those that are necessary to accomplish their assigned duties. Version Control supports the following privilege types:

Base Privileges

Base privileges represent the access rights to use certain features. For example, the AddVersion privilege enables users to assign version labels.

Base Privilege Name	Associated Action/Task
GetTip, or GetNonTip	Check out a revision
LockTip, or LockNonTip	Check out with a lock
InitArchive	Check in and create new archive
ChangeAccessList	Define access list members
PutBranch, or PutTrunk	Check in a revision
StartBranch	Check in revision, create branch
AddVersion	Check in, assign a new version label
ViewArchive	View an archive report
ModifyVersion	Change/Remove a version label
DeleteVersion	Remove a version label
Unlock	Unlock a lock you placed
BreakLock	Unlock another user's lock
DeleteRevNonTip, or DeleteRevTip	Delete a revision
ChangeAccessList	Change access list members
ChangeOwner	Change the archive owner

Composite Privileges

Composite privileges are composed of two or more base privileges. Composite privileges include the SuperUser, Unlimited, TeamLeader, TeamMember, and NonTeamMember privileges, which provide users with a wide range of rights. These privileges are described below:

SuperUser

Grants unlimited access to any archive, whether or not the user is on the access list.

Unlimited

Grants all base privileges. Users with this privilege can perform any operation on any archive if at least one of the following is true:

The access list for the archive contains the user's ID,

The access list for the archive contains a group, to which the user belongs,

The access list for the archive is empty.

Other VCS composite privileges:

Base Privilege	Team Leader	Team Mbr	Non Team Member
GetTip, or GetNonTip	•	•	•
LockTip, or LockNonTip	•	•	
InitArchive	•		
ChangeAccessList	•		
PutBranch, or PutTrunk	•	•	
StartBranch	•		
AddVersion	•	•	
ViewArchive	•	•	•
ModifyVersion	•		
DeleteVersion	•		
Unlock	•	•	
BreakLock	•		
DeleteRevNonTip, or DeleteRevTip	•		
ChangeAccessList	•		
ChangeOwner	•		

How Privileges are Applied

When assigning privileges, keep the following in mind:

If you assign no privileges to a user, they have the equivalent of an *Unlimited* privilege level.

When using Access Lists: If a user is a member of one or more groups in an archive's access list, that user has the combination of all the privileges of all those groups.

When using Assigned Privileges: Users cannot perform any operation that violates their assigned privileges, even if they belong to groups that have greater privileges.

The SuperUser privilege grants all privileges. This privelege must be assigned or implied for any developer who is required to maintain the access control database; **VCS Administrator** requires at least one developer be assigned this privilege.

Managing Users and Groups

VCS Administrator allows you to define users and groups of users in an access control database. The purpose in defining individual groups and users is to grant them privileges. They are displayed in a tree format, listing each group and its members. To add a group, highlight *Version Control Systems*, and press the **Insert** button. To add Users, highlight any other item and pressing the **Insert** Button.

Users

To add a developer, highlight the *All Users* group and press the Insert button. Supply the user's full name, VCSID, and password if used. The password must be entered twice to confirm it. On the **Privileges Granted** tab, press the **Add** or **Remove** buttons to grant or rescind privileges. On the **Groups Belonged To** tab, press the **Add** or **Remove** buttons to add or remove a user from groups.

Note: The All Users group is required for system use and cannot be removed.

Groups

To add Groups, highlight the topmost entry (Version Control Systems) and press the **Insert** button. Supply the group name (with no spaces). On the **Privileges Granted** tab, assign privileges to the group. Groups cannot have the *SuperUser* or *ViewAccessDB* privileges—those privileges are only allowed for users. All members of the group automatically inherit the rights assigned to the group. The **Group Members** tab allows you to add and remove users from the group.

Logins

You can specify how the Version Control System recognizes the User ID. See *Developer User ID Source* in *Using the VCS Administrator* for more information. Once you have entered users into the database, you can include Login to VCS as a source .

To force users to login, enable only **Login to VCS**. This requires a User ID (and password, if necessary) to launch the Version Control System. This also enables the Login button on the Configuration dialog, allowing a user to change their User ID for the current session.

You can enable multiple methods, in which case **Login to VCS** should be last. A login dialog is only be displayed if a VCSID is not found by another source.

Problem Solving / Troubleshooting

General

Some Archive Directories are not displayed

If you do not see all of your archive directories in the selection list of Check In, Check Out, Reports, and Utilities, it means that one or more of the directories listed in the Options dialog was not found. PVCS stops reading the list when it reaches a directory that does not exist, so that directory, and the others listed after it, are not used. Simply remove or correct the offending directory to continue to use the others. If these are network directories, and you are often not connected to the network, consider putting the network directories in the MASTER.CFG file and only local ones in the Configuration dialog, and turn on the Append to master config VCSDIR option.

Navigating directories under Novell NetWare

If you do not see [.] in the directories list box of Check Out, Configuration, etc., you will need to configure your workstation to display the parent directories (the double dots). Enable the following setting in the NET.CFG file or on the property tab of the Novell Client for Windows 95:

```
Show Dots = ON
```

Check Out

File Is Locked by Another User

If you try to check out and lock a workfile when another user already has the workfile locked, you see the following error message:

```
The archive for the workfile file_name is already locked by the user
user_name.
```

The user who has the file locked must check it in before you can check it out with a lock. You should either ask that user to check in the file, or check out a read-only copy of it without a lock. If you check out a file without a lock, you can only view or print it; you can’t edit it.

If the user who locked the file is unavailable, ask your system administrator to log in as that user and release the lock.

You Don’t Get the Revision You Expect

If you get a different revision of a file than you expect, you may have more than one archive with the same name in different archive directories. PVCS checks out files from the first archive found, searching the archive directories in the order listed in the Version Control System Configuration dialog.

Remember, the archive directories listed in a project definition file ***supercede*** the directories listed in both the master configuration file (MASTER.CFG), and in the local configuration file (C5VCS.CFG), for actions on the project and its files. This allows you to use the same file name for different files in various projects and keep them in different archives.

Check In

If two or more developers Check In a new procedure (one that was not in the revision they checked out) that has the same name, only defined procedures are checked in. That is to say, in the event of a name conflict, *ToDo* procedures are skipped. The developer is alerted if any of the new procedures are *ToDo* and the Check In will abort. Delete or rename the duplicate procedure. It is a good practice to not reference procedures until they are prototyped, unless you are planning to code it soon.

Recovery from a Network Crash

Follow the steps below to recover from a network crash that occurred while the Version Control System was updating an archive:

1. Use **Archive ➤ Reports** to generate Full information on the original archive. This report indicates whether or not the archive was corrupted.
2. Do one of the following:

   If the archive is not corrupted, delete any temporary files in both the archive directory and the archive temporary directory, and retry the operation that you were performing during the network crash.

   If the archive is corrupted, look in the archive directory for a temporary file named **PVCSnnnn.TMP**. Perform step 1 on this file to determine if it is corrupted. Do one of the following:

   If the temporary file in the archive directory was not corrupted, go to step 3.

   If the temporary file in the archive directory was corrupted, look in the *archive temporary directory* for a temporary file named **PVCSnnnn.TMP**. This directory would either be the destination directory or the directory specified as ‘Directory for Temporary Archive Copies’ on the Global tab of the VCS Administrator. Perform step 1 on that file to determine if it is corrupted.

   If the temporary file in the archive temporary directory is not corrupted, go on to step 3. If it is corrupted, restore the archive from your network backup system.
3. Copy the temporary archive to the archive directory and rename it with the original archive file name.

Version Manager Users

InterSolv's PVCS provides the engine for all of the Version Control System's archive storage and retrieval. There is no functional difference between an archive created with PVCS Version Manager and Version Control System—you can use Version Manager interchangeably to interface with developers on other platforms.

There are a few differences since Version Manager is designed to work with a 3GL programmer's files, and the VCS is designed to work with Clarion.

APP files - The Clarion Version Control System supports procedure locking in Clarion .APP files, allowing multiple developers to simultaneously access them. While this information is stored in the archive, the non-Clarion PVCS Version Manger is not aware of it. Therefore, you will corrupt your .APP file archive and be unable to Check In any outstanding procedures if you check in revisions to the .APP file using PVCS Version Manager.

Branches - The Clarion Version Control System supports branching, but not merging. Therefore, when you get a revision that is older than the tip revision, then put it back in the archive, it becomes a branch, which is the same behavior as the (non-Clarion) PVCS Version Manager under these circumstances.

VCSID - If you didn't enter a VCSID when you installed PVCS your changes are be stored in the archive as 'PVCSUSER.' If you share archives with others, it will be difficult to track who made each change.

Tip: You will not be allowed to Check Out procedure-lock enabled APP files without a VCSID.

Configuration Files - The Clarion Version Control System reads various configuration files, and (non-Clarion) PVCS directives in one file may override those in another. Here is the sequence files are read:

1. MASTER.CFG is read first, on every Check In or Check Out to re-initialize the settings. Let's say it has the NOMULTILOCK directive.
2. TEAMLINK.CFG (in the Windows directory) is then read. This file is created for each user/machine, and can contain any standard directives. It probably won't have many directives in it, and NOMULTILOCK is still in force.
3. If you create a project configuration file, then it is read last. Any directive placed in here would add to or alter the PVCS environment; e.g., MULTILOCK would enable multiple locks for that set of files being Checked Out.

As soon as you Check Out another application, 1 and 2 above are reread and multiple locks are disallowed again.

8 - DATA MODELLER

Overview

Data Modeller is a graphical, POINT and CLICK database dictionary editor for Clarion.

Data Modeller draws your dictionary design on a graphical (visual) Design Pad. The Design Pad displays your files in boxes containing field and key names. You can optionally display field types, pictures, key structures and prompts as well. You can create relationships between files simply by dragging and dropping. Data Modeller displays the resulting relationships as lines drawn between the boxes.

Dictionary, file, key, field and relationship attributes are immediately accessible with the right mouse button—simply RIGHT-CLICK on the item you want to modify and Data Modeller provides the appropriate popup menu choices for the selected item.

Tip: The right mouse button is the key to a successful relationship with Data Modeller!

Editing data dictionaries with Data Modeller is very fast and very easy, plus you can print the graphical representation of the dictionary—up to 10 x 10 pages large! Of course you can print a detailed text report of the dictionary too.

Data Modeller has comprehensive built-in error checking that ensures your dictionaries are accurately designed with the minimum of field name and type inconsistencies. Data Modeller analyzes existing dictionaries too.

Data Modeller provides a wealth of time-saving utilities:

Field Pool
Stores entire field definitions (independent of any files) for easy reuse. For example, you may use a customer number field in several files in your dictionary. Add the field to the field pool, then simply select it from the field pool for each additional file that uses it. This guarantees that a particular field is defined the same way every time. In addition, Data Modeller optionally propagates changes from the field pool to each instance of the field in the dictionary.

Key Pool
Stores entire key definitions for easy reuse. Add the key to the key pool, then simply select it from the pool for each file that uses it.

Type Pool
Stores field type definitions for easy reuse. For example, you may frequently create dates using a LONG, with picture @D6 and the

Initial Value set to TODAY(). Store all these attributes as a Field Type. The next time you create a date field, select the attributes from the Type Pool.

Favorite Pictures
Instead of typing in the picture every time you create a field, POINT and CLICK to select a picture from the favorite pictures list.

Data Type Defaults
Create defaults for data types, complete with size and picture. For example, you may only use BYTE fields for check boxes—simply check the True or False attribute in the **Data Type Defaults** dialog. Whenever you create a new field, the default attributes are set.

Prompts Manager
Perform global search and replace on prompts, adjust the position of the prompt colons, and generate prompt hot keys. Use the Prompts Manager to strip out undesired underscores, including those that may have remained from very old Clarion applications.

Display Subsets
If you are working with a very large dictionary you can focus on specific areas by defining and viewing subsets of the files.

SQL Script Generator
Data Modeller automatically generates custom SQL scripts for creating and maintaining the SQL tables defined within your data dictionary.

Technical Support

Primary support for Data Modeller is provided by TopSpeed. However, you may also contact Pea Brain Software directly at:

Pea Brain Software cc
PO Box 51694
Wierdapark
VERWOERDBURG 0149
SOUTH AFRICA

or by FAX at: +27-12-663-6395
or by email at: peabrain@global.co.za

In order to provide you with quick efficient replies we ask that you be as clear and concise as possible in your queries. Always include the version number of your Data Modeller (for example, version 2.0, release 001). To see the version number, choose **Help ➤ About** from the menu.

If Data Modeller fails with a specific dictionary, please send both the Clarion dictionary file and the .TXD file that are causing the problem.

Design Pad

The Design Pad contains the graphical representation of the active dictionary. When displaying a dictionary, it looks something like this:

Data Modeller displays database files as boxes containing field and key names. It displays relationships between the files as lines drawn between the boxes.

RIGHT-CLICK on the Design Pad to access the Design Pad popup menus. A different popup menu appears for each discrete area in the Design Pad.

Let's look at each of the menus in turn.

File Popup Menu

RIGHT-CLICK on a file name. The resulting popup menu contains the following options.

File to Populate
Toggles the File to Populate mode. When the mode is on, the background color of the file name is red, and you can add new fields to the file simply by CLICKING on the field names in the Field Pool.

File Properties
Opens the **File Properties** dialog. This is the same as the Clarion File Properties dialog.

Summary
Opens the **File Statistics** dialog which displays information about the file including the number of keys and fields as well as the record length.

Delete File
Delete the file *and* its relationships from the dictionary. Data Modeller will prompt first for confirmation.

Copy File
Opens the **Copy File** dialog which lets you create a new file with an identical layout.

Tools (sub-menu)
Provides quicker access to the tools that are more fully described later, in the section regarding the Data Modeller menus:

Edit Comments
An oversize comment field, allowing both short file descriptions and extensive programmers' notes.

Browse / Convert File . . .
Displays the contents of the selected file.
Please refer to the much fuller description of this function that can be found, under *Tools*, in the section titled *The Data Modeller Menus.*

Create Convert Program . . .
Allows you to convert the contents of a modified file to the new layout.
Please refer to the much fuller description of this function that can be found, under *Tools*, in the section titled *The Data Modeller Menus.*

Create Test Data . . .
Enables you to generate test data for the selected file in the Design Pad.
Please refer to the much fuller description of this function that can be found, under *Tools*, in the section titled *The Data Modeller Menus.*

DM View Browser
An add-on product from Pea Brain Software (the creators of Clarion Data Modeller). Available now, it will generate SQL-like queries for Clarion and for most well-known DBMS files. Contact information for Pea Brain Software can be found at the front of this manual.

Create New Empty File
Opens the **File Properties** dialog to create a new (empty) file. Similar to the Create File command in the Dictionary menu. In contrast to the related command Create File and Populate.

Delete File from disk
Deletes the file and its associated relations from the dictionary.

Print Structure
Prints a report with the details of this file only. All printing functions in an application, including Print Preview, can be suppressed at the developer's option, through the **User Preferences** dialog.

Export Structure As Source
Use this dialog to generate the Clarion source code declaration for the selected file.

Export File	The name of the .CLW file to be generated. Type the full file name, including the extension.
Append File	Check to append this file declaration to an existing file. If this option is not checked, any existing file will be overwritten.
Export Comments:	Include the short file description. (see Edit Comments)

Export Long Descriptions:
Include the full text of any developer notes.
(see Edit Comments)

Keys (sub-menu)

Create New Key
Opens the **Key Properties** dialog which is the same as the corresponding Clarion dialog.

Add Selected Key
Adds the selected key in the Key Pool to the file. This option is disabled until you select a key.

Fields (sub-menu)

Create New Field
Opens the **Field Properties** dialog which is the same as the corresponding Clarion dialog. By selecting the **Multiple** option, initial development can go much faster, since on OK the user is returned to the top instead of exiting to the Design Pad.

Add Selected Field
Adds the selected field in the Field Pool to the file. This option is disabled until you select a field.

Subsets (sub-menu)

Hide File Only
Hides this file only. See also: Define Subsets.

Hide File and Related Files (Setup)
Hides this file and all related files.

Hide All But... and Related Files (Setup)
Hides all the files in the dictionary, except for this file and its related files.

Hide All But... and Related Files (First Level)
Hides all the files in the dictionary, except for this file and its related files.

Cancel Subset Display
Un-hides all the files in the dictionary.

Key Popup Menu

Key Properties
Opens the **Key Properties** dialog, which looks and acts the same as the corresponding Clarion dialog.

Edit Comments
Description fields for the key.

Delete Key
Deletes the selected key *and* all relationships depending on this key.

Field Popup Menu

Field Properties
Opens the **Field Properties** dialog. This dialog is similar to its equivalent in Clarion. It allows full access to, and editing of, the field attributes. There are three additional prompts:

Choose from Type Pool CLICK to select the field attributes from the Type Pool.

Add To Type Pool
CLICK to create a new Field Type. Opens the **Type Pool Name** dialog which asks for the name of the new type.

(Select Favorite Picture)
CLICK to choose one of your favorite pictures from the **Favorite Pictures** dialog.

Summary
Opens the **Field Summary** dialog which displays the primary attributes of the field (Type, Length, Picture, Prompt and any warning messages). You can also change the picture and prompt.

Copy Field
Opens a pre-loaded **Field Properties** dialog which lets you create a new field with identical attributes and a different name.

Delete Field
Deletes the field from the file. Note that this can affect both keys and relations that include the deleted field. If the key contained more than just this field, no further action is taken. If the key contained only this field, both the key, and any relations associated with the key are removed.

Edit Comments
Access to the two file description fields.

Show only files where used
Displays a subset of the dictionary, revealing the files where this field is used. Select Cancel Subset Display from the View Menu, to return the Design Pad to normal.

Relationship Popup Menu

RelationProperties
Opens the **Relation Properties** dialog for the selected relationship. This dialog is identical to its equivalent in Clarion, with the exception that in Data Modeller, you can name the relationships.

Tip: Use meaningful relationship names to make your data dictionary self-documenting.

You can display the relation names on the title bar of the Design Pad. Choose **View ➤ Relation Names** to toggle the display of relation names. Set the default in the **User Preferences** dialog in the **Other Settings** tab. You can set the color of the relation names in the **Color Settings** tab.

Relation Name
Type a name for the relation. For Example, 'Lookup Product' would be a meaningful name for a one-way relationship used for lookups. This name is stored in the dictionary as a 'user setting', so you won't lose it when you transfer the dictionary between Data Modeller and Clarion.

Tip: In Data Modeller, you can create relationships by dragging and dropping. See Creating a Two-way Relationship and Creating a One-way Relationship.

Delete Relation
Deletes the selected relationship.

Print Relations
Prints a report showing all the relations for the dictionary.

Goto Primary File
Moves the mouse cursor to the primary file in the relationship.

Goto Secondary File
Moves the mouse cursor to the secondary file in the relationship.

Select fields for View Browser
Only activated for the Relation View Browser (see below).

Relation View Browser
An add-on product (with DM View Browser) from the creators of Clarion Data Modeller. Available with wizards to help you generate *virtual relations*, and also queries for Clarion and other databases via ODBC. Contact information for this product can be found at the front of this manual.

Open Area Popup Menu

Dictionary Properties
Opens the **Dictionary Properties** dialog.

Create File
Opens the **File Properties** dialog to create a new (empty) file with a blank key and a blank field to act as placeholders for the respective popup menus. When you enter a Key or a Field, the respective placeholder disappears.

Create File and Populate
Creates a new file and automatically fills it with selected fields from the Field Pool.

> **Tip:** You must switch tagging on in the Field Pool (by RIGHT-CLICKING anywhere in that box), and then CLICK the fields that you want in the file before you select this option.

Import Structure From File
See the section on the File menu, for more about this command.

Aliases
Opens the Alias Editor. You can set up aliases for any file in the dictionary. See *Aliases* for more information.

Select Subsets
Another way to select particular parts of the database, if those subsets have been defined (see below). See also, in this manual, Select Subset in the section on the View menu.

Locate Files
Opens the **Locate Files** dialog. This is most useful when you have a dictionary that is very large, with many files. Simply highlight the file you want to locate then CLICK on OK (or DOUBLE-CLICK on the file name). The Design Pad places the mouse cursor on the selected file.

> **Tip:** The fastest way to work with Data Modeller, is in 100% zoom mode, and to locate files by DOUBLE-CLICKING in the Locate File dialog. If you have a large dictionary, 'scale to fit' is not efficient as it can take a long time to redraw all the files. See also Define Subsets.

Define Display Subsets
Opens the **Define Subsets** dialog, which lets you define, save or open a subset display in the Design Pad. See *Display Subsets* for more information.

Cancel Subset Display

Cancels the current subset display and un-hides all hidden files in the Design Pad.

Re-Position Files

Rearranges the Design Pad, so that the files are displayed from left to right with the larger files on the left, the smaller files on the right.

Note: Re-Position Files is not reversible!

Re-size View

Centers the dictionary display on the Design Pad.

Data Pools

Field Pool

The Field Pool lists all the fields that have been defined in your dictionary. This makes it easy, quick and accurate to create a new file based on existing fields, and it also helps avoid common bugs in your applications.

Locating Fields

CLICK on a field name in the Field Pool. All instances of the field in the dictionary are immediately highlighted.

If there is more than one instance of the field, they are displayed in different colors. The first instance that directly corresponds with the version selected in the Field Pool is displayed in one color. All other instances are displayed in a second color. You can choose the colors in the **User Preferences** dialog.

Fields that have the same name

By default the Field Pool lists all the fields in your dictionary, displaying each version of fields that have the same name. These are color coded so that identical fields are tagged with a blue flag icon, while fields that have the same name but different attributes are tagged with a red flag icon.

Fields that have different attributes are always displayed. In the **Advanced Settings** tab of the **User Preferences** dialog, you can choose whether or not to display those fields that have identical attributes.

Changing a field's attributes

You can choose in the **Advanced Settings** tab of the **User Preferences** dialog whether to cascade the changes automatically to other fields that have the same name and *identical* attributes. Fields with the same name but *different* attributes are not affected.

Field Pool Popup Menu

Set Filter

Opens the **Pool Filter** dialog which lets you display a subset of the fields in the Field Pool. This is particularly useful if you have a large dictionary with many fields defined.

For example, CLICK on the A tab to list only those fields whose names begin with 'A' (or 'a'). CLICK on the All tab to cancel the subset display.

Show Only Files Where Used
Displays a subset of the dictionary, revealing just the files where this field is used. Select Cancel Subset Display from the View Menu, to return the Design Pad to normal.

Create Field
Opens the Field Properties dialog to create a new field.

Quick Add Opens the Quick Fields dialog, which lets you create a number of fields rapidly. The Quick Fields feature was inspired by its equivalent in the Clarion Quick Start utility. However, in Data Modeller you can choose the data types for your fields.

Field Name	Type a field name.
Data Type	Select a data type from the drop-down list.
Size	Type the size required. (For numeric data types this field is disabled.)
Use Data Type defaults	Check to automatically apply attributes from the data type pool.
OK	Save the new field to the quick list.
Edit	Edit the selected field.
Delete	Delete the selected field

Field Properties
Opens the **Field Properties** dialog to change the attributes of the selected field. Note that if you have multiple fields with the same name, any changes that you make may be carried through to the other fields that have the same attributes. You can control this feature in the **User Preferences** dialog.

Delete Field
Deletes the field from the Field Pool. The field is also deleted from all files, keys and relations.

Copy Field
Opens the Field Properties dialog. The attributes of the selected field are copied into the dialog. You must type a new name for the field.

Tag
Toggles tagging mode in the Field Pool. With tagging on, you can highlight multiple fields. CLICK on the Add Selected Field option in the File Name Popup Menu to add all the selected fields to a file. CLICK on the Create and Populate File option in the Open Area Popup Menu, to create a new file with all the selected fields automatically populated.

Untag All
Quickly releases multiple tags (checks).

Import From...
Lets you merge dictionaries. You can import all fields from the selected dictionary. Only the first occurrence of each field is imported. Data Modeller does not check whether field names already exist so the import may result in duplicate field names in the Field Pool.

Key Pool
Opens the Key Pool *and* closes the Field Pool.

Key Pool

The Key Pool lists all the keys that have been defined in your dictionary.

Locating Keys

CLICK on a key name in the Key Pool. Every occurrence of that key throughout the dictionary is highlighted on the Design Pad. The highlight color can be set in the **User Preferences** dialog.

Creating a Key

In the Design Pad, RIGHT-CLICK on the File Name of the file for which you wish to create a key. From the popup menu, select Keys -> Create New Key.

Using an Existing Key

You can add an existing key to a file. Highlight the key in the Key Pool. Now RIGHT-CLICK on an existing key in the file. Select Keys->Add Selected Key. The key is added to the file. Note that the file must have have a key, before another can be added this way.

Tip: **If the key fields don't exist in the file, they are automatically added to the file as well.**

Key Pool Popup Menu

Opens the **Key Properties** dialog for the selected key. The **Key Properties** dialog is similar to its equivalent in Clarion. The only difference is in the Fields tab, where you choose the key fields directly from a list box.

Note: DOUBLE-CLICK **on a field in the Key Fields listbox to change the sort order.**

Delete Key
Deletes the selected key. The key is also deleted from its file. If the key is involved in a relation, the relation is also deleted.

Field Pool
Opens the Field Pool and closes the Key Pool.

Type Pool

The Type Pool lets you store a number of pre-configured Field Types. For exampl e, you may always use date fields that are of type LONG, with the picture @D6. Instead of having to configure those attributes every time you create a new date field, you can store them as a Field Type, and then simply select this type from the Type Pool when you create another similar field.

To add a type to the Type Pool, create your field complete with attributes in the **Field Properties** dialog. Now CLICK on **Add To Type Pool**. The field's attributes are stored in the Type Pool.

The next time you wish to create a new field of the same type, simply type the field's name in the **Field Properties** dialog, then CLICK on **Choose From Type Pool**. Talk about time saving!

All field attributes are saved and restored except the Report and Window controls, and the Must be in file validity check.

Type Pool Popup Menu

This menu lets you manipulate Field Types that you have created from the **Field Properties** dialog.

Edit Properties
Opens the **Type Properties** dialog where you can change the attributes of the selected Field Type. The dialog is essentially the same as the **Field Properties** dialog, with the exception of the Type Name.

Type Name Enter a meaningful name for the Field Type. The name appears in the Type Pool for selection when you are creating a new field.

Delete Type
Deletes the selected Field Type from the Type Pool.

The Data Modeller Menus

File Menu

The File menu lets you open, close, import and export dictionary files.

Import Current DCT
Imports the dictionary currently loaded in Clarion.

Export to Current DCT
Exports your changes back to the dictionary currently loaded in Clarion.

New Dictionary
Create a new dictionary. You will be asked whether you wish to save the current dictionary.

Open TXD
Opens a selected .TXD file for editing.

Close Dictionary
Close the current dictionary. You will be asked whether the dictionary should be saved.

Export TXD Saves the dictionary to a .TXD file. Note that this option does not update the currently loaded dictionary (if any) in Clarion. In order to so, select the File, Export to Current DCT option.

Export As
Creates a .TXD file with the name you specify.

Export to SQL Script Opens the Export Script Type dialog, which allows to define the type of SQL script you wish to create. See *SQL Script Generation* below.

Delete TXD File
Deletes the .TXD file of your choice from your hard drive.

Printer Setup Calls the Windows Print Setup dialog.

[Recent Files]
The names of the four most recently edited dictionaries are listed in the File menu. To open one of these dictionaries, CLICK on the name or type Alt-# where # is the number of the dictionary in the list.

Exit
Exit Data Modeller.

Edit Menu

The Edit menu provides the following choices:

Aliases
Opens the **Alias Editor**. See the section named **File Aliases** later in this chapter.

Dictionary Menu

The Dictionary menu provides the following choices:

Properties
Opens the **Dictionary Properties** dialog.

Create File
Opens the **File Properties** dialog to create a new (empty) file with a blank key and a blank field to act as placeholders for the respective popup menus. When you enter a Key or a Field, the respective placeholder disappears.

Create File and Populate
Creates a new file and automatically fills it with selected fields from the Field Pool.

Tip: **You must switch tagging on in the Field Pool, and CLICK on the fields that you want in the file before you select this option.**

View Menu

The View menu lets you change the way the files are displayed in the Design Pad.

Structures
Toggles the display of keys, fields, and relations, (as a group), with the files.

Fields
Toggles the inclusion of field names with each file.

Pictures
Toggles the inclusion of field pictures next to the field names.

Types
Toggles the illustration of data types next to the field names.

Prefixes
Toggles the inclusion of file prefixes with the field names.

Key Structures
Toggles the visible breakdown of fields making up each key.

Relations
Toggles the display of relation lines between files.

Virtual Relations
Toggles the display of Virtual Relations between files.

Relation Names
Toggles the display of relation names. Note that this is a temporary switch. Set the default mode in User Preferences.
Switch Relation Names OFF to speed up the repainting of the Design Pad window. You can switch them back on just before you print the dictionary.

Select Subset . . .
Opens an empty list until subsets have been defined.

Define Display Subsets . . .
If your dictionary has many files, it can become difficult to work with. It is sometimes handy to be able to view only the files that you are concerned with at any particular time. Data Modeller allows this by means of Display Subsets. You can define and save subsets using this dialog. Select files for a new subset with a DOUBLE-CLICK. Blue indicates the file is visible on the Design Pad, red indicates the file is not displayed. DOUBLE-CLICK on the file names to toggle the display flag.

Save As
Enter a name for the defined subset. This can be used to identify the subset if you wish to invoke it again at a later time.

Load
Reload a defined subset. Choose the subset from the drop-down list box. CLICK on OK to invoke the subset view.

Un/Hide
CLICK to hide or un-hide the highlighted file in the File Name list box.

Tip: **You can also create a subset very quickly using options on the File Popup Menu. For example, you can focus on a specific file and its related files simply by clicking on Hide All But... and Related Files.**

Cancel Subset Display
Cancels the subset display (if in effect) and causes the Design Pad to show all files in the dictionary.

Prompts Mode
Toggles the View setting that controls whether the field prompts are displayed in place of the field names. This may be beneficial if you've used obscure names for your fields.
See also: Prompts Manager

Aliases
Opens the Alias Editor. You can set up aliases for any file in the dictionary. See **File Aliases** later in this chapter, for more information.

See Comments
Toggles the display of file descriptions and comments on the Design Pad.

See Do Not Populate
Toggles “N” (for Not), next to the file name.

Locate Files
Opens the **Locate Files** dialog. This is most useful when you have a dictionary that is very large, with many files. Simply highlight the file you want to locate then CLICK on OK (or DOUBLE-CLICK on the file name). The Design Pad places the mouse cursor on the selected file.

Re-Position Files
Rearranges the Design Pad, so that the files are displayed from left to right with the larger files on the left, the smaller files on the right.

Note: Position Files is not reversible!

Re-size View
Centers the dictionary display on the Design Pad.

Zoom Menu

The Zoom menu lets you display as much of the dictionary as you want.

You can also zoom by clicking on the Design Pad with the left mouse button, if you Activate Left Mouse Button Zoom in the User Preferences dialog.

Zoom +
Reveals more file, field name, and relation details, by showiing less of the big picture.

Zoom -
Reveals more of the schema, at the expense of some structure detail.

Scale to Fit
Re-scales the Design Pad so that all the files in the dictionary are visible.

25%
Scales the Design Pad so that the files are displayed at 25%.

50%
Scales the Design Pad so that the files are displayed at 50%.

75%
Scales the Design Pad so that the files are displayed at 75%.

100%
Scales the Design Pad so that the files are displayed at 100%.

User Define
Opens the **Zoom Factor** dialog which lets you specify the factor by which the Design Pad is scaled.

Tools Menu

The Tools menu contains options that allow you to analyze your dictionary, control the Design Pad and edit your own user-defined data types.

Prompts Manager
Use the **Prompts Manager** dialog to organize and control the prompts that have been set up in your dictionary.

Apply to Selected File only
Allows the settings for prompts tobe made file-by-file, or for the entire dictionary.

Remove Colons
Select to remove all colons from all prompts throughout your dictionary.

Add Colons
Select to add colons to all prompts throughout your dictionary.

Spaces between text and colon
The number of spaces that will be generated between the last character of the prompt and the colon.

The Microsoft Windows interface guidelines call for prompts to have a colon at the immediate right hand side of the text like this:

My Prompt:

Some developers, however, prefer their prompts to have a space before the colon like this:

My Prompt :

With Data Modeller, either style is easy to generate!

Hot Keys
Select to generate or delete hot keys for your prompts. The algorithm used is as follows: if possible the first letter of the prompt is used. If that has already been used, then the first letter of the second word (if it exists) is used. If that letter has already been used, then the second letter of the first word is used, and so on. If there are no unique letters left, the first letter found with the least duplicates is used.

Create Hot Keys
Select to generate hot keys in all prompts.

Remove Hot Keys
Select to remove all existing hot keys from all prompts.

Remove All Underscores
Select to remove underscores from all the prompts in the dictionary. (Those of us who knew (and loved!) Clarion Professional Developer will appreciate this Prompt Manager utility. In ancient times programmers did not have the space-character option, and so they used underscores to make labels and prompts more readable, as in FIRST_NAME_:. When you create a dictionary by importing a CPD file definition, therefore, you'll see that "legacy" prompts are infected with underscores. Use this procedure to get rid of them.)

Search and Replace
Select to perform search and replace on all prompts.

Search For The string to search for.

Replace With	The string with which to replace the search string. If you leave this blank, it will have the effect of deleting the search string.
Case Sensitive	Check this box to make the search case sensitive.

Change Justification

New Justification	The choices are Left, Right, Center, and Decimal. You may also choose "None".

Properties Manager . . .

Change many of the principal properties and options for one, for a group, or for *all* files or fields. Double-click on a field or a file to "tag" that item and add it to the batch for editing. Less frequently-changed attributes will still have to be changed via the **Edit Properties** dialog in Clarion.

Volumetrics Calculation

A handy tool for rough extrapolation of your capacity expectations for a specified file, based on the ratio of filesize to records in the existing file. Permits early estimation of performance penalties associated with different layouts.

Browse / Convert File . . .

Displays the contents of the selected file. The Data Modeller File Browser is similar to Clarion's. Using either the menus or a right-click popup, list or **Print** the records in a specified file; **Order** the records by any key, or leave unsorted (i.e., "record order"); **Filter** the records by any value of any field; Change the field formatting in a specified file, even while the file is in use. Especially useful is the **Header** option, which allows you to view and change the **label**, **picture**, and data **type** of any column (a.k.a. *field*) attributes.
In addition, the powerful Data Modeller browser has the following features:

You can display and edit the contents of ARRAY fields. (+ - Keys to scroll). BLOBs can also be displayed using the appropriate viewer.

You can **Search** for and **Replace** any field in a file.

Create Convert Program . . .

Opens the **File Convert** dialog, which allows you to convert the contents of a modified file to the new layout.
Data Modeller generates a source file which may be compiled to create a data conversion program. Data Modeller does a better job

of data conversion than Clarion in that complex data types such as arrays are automatically handled correctly.

Source File Name:
The name of the existing data file that you wish to convert.

Source Dictionary:
The name of the dictionary that contains the original file layout corresponding to the existing data file that you wish to convert. Note that Data Modeller automatically creates a backup copy of your dictionary as it is loaded into Data Modeller. This backup copy is given the extension .BXD. You can select the Source Structure from the backup file or from any other dictionary.

Source Structure:
The name of the Source Structure containing the original layout of the data file.

Target File Name:
The name of the file to which you would like convert the original data.

Target Dictionary:
The name of the dictionary that contains the new file layout to which you want to convert the existing data file.

Target Structure:
The name of the Source Structure containing the new layout of the data file.

Program Name:
Type the name that will be used for the generated source file.

Rename the new file to its original name...
Toggle ON to rename the new data file to the original's name after the data has been copied into the new file. Use this switch when you wish to convert a data file to a new structure.

Create Test Data . . .
This dialog enables you to generate test data for the selected file in the Design Pad.

Disk File Name:
Type (or select) the name of the physical data file into which the test data will be placed. The file extension is optional.

Dictionary:
The current dictionary loaded in Data Modeller will be used. This cannot be changed.

Structure:
Type (or select) the file structure of the data file into which the test data will be placed. Make sure that the file structure and filename are matched - you will overwrite data if the wrong disk target is selected.

Data Profile:
Click to open the **Data Types** dialog.

Required no of Records:
Type the number of records you wish to generate. Data Modeller will generate up to 200,000 records per file.

Create Data:
Click to generate the test data. If Data Modeller detects records in the file before the generation starts it will prompt you to clear the file or append records to the file.

Set File Export Sequence . . .
In the file selected for export, click and drag fields into the preferred order.

Analyse Import Error . . .
When errors are detected as Data Modeller attempts to import a dictionary ("*The dictionary that you have tried to load has duplicate Ident numbers and cannot be imported*"), enter the given line number in this dialog to display the "duplicate Ident numbers". You can then edit the source dictionary, in the source application. See: **Duplicate Ident Numbers** at the end of this manual.

Print Menu

Print File Reference
Prints a detailed text report of your dictionary, one page per file.

Print Relations
Prints a report of all the relations in your dictionary.

Print Field Details
Prints a report of all the fields in your dictionary.

Print Design
Prints a comprehensive report, listing all files, fields, keys and relations in your dictionary.

Number of pages wide
Specify the width of the design (in pages). The height is adjusted automatically. This setting lets you make a readable representation of very large dictionaries. The largest allowable width is 10 pages.

Do not print background shading
This setting is ON by default. Check it OFF if you have a color printer—the dictionary is then printed in full color. If you have a non-color printer we recommend leaving the setting ON both to save ink and because shading usually does not work well on black and white printers.

Analysis Report

Prints a report of the discrepancies found in your dictionary. For example, fields in different files that have the same name but different attributes are listed. This report can be used to help avoid bugs in your applications.

Setup Menu

Preferences

Opens the **User Preferences** dialog where you can customize the look and feel of Data Modeller. See *User Preferences* below.

Set Working Directory

If you don't set one, Data Modeller will require you to when you first import a dictionary (.TXD file). If later in the session you bring in a dictionary by another name, Data Modeller will ask you to confirm the current work directory, or specify another one.

Data Type Defaults

Opens the **Data Type Defaults** dialog. This lets you set the default characteristics for fields of each data type. For example you may know that most of your STRING fields are length 80 and picture @s20. DECIMALs are usually currency fields with length 8,2 and a picture of @N$12.2. The defaults you set here are automatically applied to new fields created in Data Modeller. See also: *Quick Add*.

Type

Select a default from a choice of sixteen Clarion data types.

Field Length

Type the length of the field. (If the data type is numeric, this field is disabled.)

Picture

A good choice for a default might be the currency picture, @N~R~11.2.

Case

Set to Uppercase, "Word Capitalize" (i.e., capitalize the first letter of all words), or Normal.

Typemode

Select "Insert" or "Overwrite" (Typeover).

Help Menu

Contents
Opens the Windows Help application and displays a list of main topics.

Search for Help On
Opens the Search dialog in the Windows Help application, allowing you to search for help topics containing a specific keyword.

How to Use Help
Opens the Windows Help application and displays instructions for using the Help system.

About
Displays the program name, version, registration, and copyright information.

User Preferences

Select **Preferences** in the **Setup** menu. This dialog lets you change settings that control the appearance and behavior of Data Modeller.

Color Settings

This tab controls the Design Pad display.

Locate Fields

Selected Sets the Design Pad color for the current selected field. When you select a field name in the Field Pool, all instances of the name in files in the Design Pad are highlighted in the color chosen.

Duplicate Sets the highlight color for duplicate fields. See also: Field Pool.

Choose two well-contrasted colors so that you can easily see the selected field, and quickly recognize any duplicates.

Relation Lines

CLICK on the buttons to set colors for the One to Many, Many to One, Lookups, Alias lines, and Virtual Relations in the Design Pad.

File Appearance

CLICK on the buttons to select the colors for the respective file components displayed in the Design Pad.

Changing the **Pad Font** is particularly useful if you have a high resolution monitor and a very busy dictionary! There are fonts provided with MS-Word, for example, which are clearly readable even when the writing on your screen is very small.

Background

You can only change the Design Pad background color when no dictionary is loaded.

Other Settings

This TAB controls a variety of Data Modeller preferences and settings.

Design Pad

Check **Show Relation Names** to actually label the relation lines on the Design Pad. This sets the default for all dictionaries loaded into Data Modeller. To temporarily toggle the display of relation names, choose **View ➤ Relation Names** in the View Menu.

> **Tip:** **Switch Relation Names OFF to speed up the repainting of the Design Pad window. You can switch them back on just before you print the dictionary.**

Check **Show Field and Key Pools on the right-hand side** to move the list box.

Check **Highlight all occurrences...** to reveal all instances of a field across the database, when you CLICK on the field name either in the Design Pad or the Field Pool.

Other self-evident options are to **Create Hot Keys Automatically,** to **Untag fields after populating** (fields to a file), and to **Add Colons to Field Prompts.**

Printing

Check **Preview report before printing** to preview reports before they are sent to the printer.

Check **Allow printer setup on reports** to switch on the Printer Setup icon in the Report Preview window.

Check **Include long descriptions** if you wish to see the long descriptions in reports that list field attributes.

Check **Include short descriptions** if you wish to see the short descriptions in reports that list field attributes.

Subset Creation and Related Files

Check **Cascade related file display to all levels** to control the creation of Display Subsets. If this check box is ON, when you choose **Hide File and Related Files**, all related files, no matter how distantly, are hidden. For example if file A is related to file B which is related to file C, all three files are hidden. If the check box is OFF then only files A and B are hidden.

Importing Dictionaries

Switch on or off any system prompting of your import options. Choose whether or not to keep the source dictionary file after the import.

Tips

Check **Show Tip of the Day** to see useful hints whenever you load a new dictionary.

Check **Disable Tooltips** to prevent tooltips from displaying.

Application

Check **Always keep Data Modeller on top of other applications,** if you prefer it that way. We don't recommend this setting when Data Modeller is maximized!

Advanced Settings

This TAB controls a variety of more advanced Data Modeller preferences and settings.

Duplicated Fields

Check **Cascade Field Property Edits** to have changes made to the first occurrence of a field, automatically cascade to the other identical instances of the same field. When this option is ON, the **Cascade Edits** tab becomes visible.

> **Note: Changes are never cascaded to fields that have the same name but different attributes.**

CLICK on **Show duplicated Fields only once . . .** to display only one instance of duplicate field names in the Field Pool. If this option is OFF, then the Field Pool displays only those duplicate fields whose field attributes are not identical.

CLICK on **Keep field in Field Pool when deleting from file** to keep the field with its attributes in the Field Pool even though it has been deleted from a file.

Relation Lines

Type the minimum horizontal distance that a relation line should be drawn at right angles to the edge of the file box in the Design Pad:

Source Coding

Check the **Load Clipboard With Selected Field or Key** option to automatically copy the name to the clipboard whenever you CLICK on a field. Then you can switch back to Clarion and simply paste the name into place.

Dictionary Settings

Check **Restore View and Zoom...** to restore the appearance of your dictionary when you reload it. Otherwise the dictionary is scaled to fit.

Check **Load dictionary in 100% zoom mode** to load the dictionary at 100% zoom mode. See also the Zoom Menu. This setting overrides the Restore View setting. Otherwise the dictionary is scaled to fit.

Relation symbols

Check **Swap relation symbols** to swap the arrow and ball symbols that appear at either end of the relation lines.

Cascade Edits

The **Cascade Edits** tab is only visible if you have checked **Cascade Field Property Edits** in the **Advanced Settings** tab.

Check the properties that you want to be updated when you change a duplicate field.

Database Drivers

Use this tab to select database drivers that Data Modeller might not otherwise be aware of, for example, the SQL Server or Oracle drivers. Data Modeller uses database drivers for validity-checking of the data dictionary.

If you add a new driver to Clarion, you must register the driver in Data Modeller as well. To do this, simply click on the Refresh button. **All new drivers will automatically be registered.**

File Aliases

An alias creates a second reference for a file without duplicating the file on disk. Using an alias can help in complex relationships. See your Clarion documentation for more information.

Alias Editor

The Alias Editor lets you create or delete aliases for files. Simply RIGHT-CLICK on an open area of the Design Pad, then choose **Aliases** from the popup menu (or choose **Edit ➤ Aliases**).

> **Tip:** **You can add an alias for a file only if the file is already on the Dictionary list.**

Aliased files appear with a different shadow color. You can set up the alias color in the **User Preferences** window by choosing **Setup ➤ Preferences** in the Main Menu.

Create
Opens the **Alias Properties** dialog which provides the following prompts.

Alias Name
Type a data file "name", as you wish to refer to it in your code. The name must be a valid Clarion label.

Alias to File
Choose a file from the drop down list. This is the original file that the alias "references." The drop down list shows only the files previously defined.

Prefix
By default, Clarion will use the first three letters of the Name for the prefix. Optionally specify up to 14 characters to use as a Prefix.

Properties
Opens the **Alias Properties** dialog for the selected alias.

Delete
Deletes the highlighted file alias.

File Relationships

One-way Relationships

One-way relationships are so called because you can't 'see' them from both files. Only one file needs a key which contains the linking field(s). (You can also create two-way relationships.)

Referential Integrity constraints cannot be defined for a one-way relationship.

What's the point of a one-way relationship? In order to have Clarion automatically generate code to perform lookups or validations, you must have a relationship defined between the files. But you may not want the overhead of maintaining a key in each of the two files.

A two-way hypothesis can almost always be manufactured - but many times the relation is deliberately kept simpler. Perhaps for security, but usually just to keep the logic manageable wherever possible. Remember that any design has to have some defined constraints, even with fast and cheap processing. Remember too that we are talking just about one relation that exists between two files. "One-way" doesn't tell you anything about either file - it describes just one simple relation in the midst of perhaps many complex interactions that each of these files has with the rest of the database.

If it can be declared then that in a given relationshsip between two files, a specified file will always be a provider of data, never a seeker of data, then the provider file does not need to know how to get to the other file. (It simply must wait to be found, instead). So the provider file **does not need a key** for this relationship. The seeking file does need to have some handle, some way of sorting and ordering, even *indexing* the records that it finds in the provider file. The only handle that works is a field whose internals are understood both by the seeker and by the provider - namely, a field that is a part of both their records. Only shared fields can ever be *keys*.

The seeker file needs that key - the provider does not. The provider needs only to have that shared field, the field that makes up the key, in order to be sorted and indexed by the Seeker key.

The benefit of a one-way relationship? A little less overhead means slightly faster throughput - but then think of an airline reservation system, for instance, using simple, highly inflexible, "hardwired" lookups, and reusing those pathways a million times an hour. However, the more important benefit for most of us, of simplified relations (no referential integrity to worry about), is the probable earlier delivery of the application, and easier troubleshooting.

CLICK on OK to complete the relationship definition.

Two-way Relationships

Two-way relationships are so called because you can 'see' them from both files. Both files must have a key which contains the linking field(s). (For a simple lookup, this might be an unnecessary overhead - you can also create one-way relationships.)

Referential Integrity constraints can be defined for a two-way relationship.

Relationships can be ONE:MANY. For example, one Invoice record can be related to many Item records. One Customer to many Invoices.

MANY:ONE relationships are exactly the same as ONE:MANY - the viewpoint is merely shifted from the one file to the other.

ONE:ONE relationships are just special cases of ONE:MANY.

MANY:MANY cannot be handled directly by relational database theory - you get around it by creating an extra intermediate file and declaring two ONE:MANY relationships.

In Data Modeller, relationships are represented by lines drawn between the files in the Design Pad. You create a relationship by clicking on one key and dragging to another key. For example, consider the following files:

One invoice can have many items. So you want to create a ONE:MANY between the Invoice file and the Item file. (From the Item file's point of view, of course, the relationship is a MANY:ONE.)

If you want referential integrity to be coded into the application, you must specify the 'On delete' and 'On update' restrictions. The 'On delete' restriction defines the action that must be taken when a record on the ONE side is deleted. For example - what if an Invoice record is deleted? In this familiar example we might say that if line item records exist, no user is allowed to delete the Invoice. The restriction, then, would be set to RESTRICT:

However, we want to allow the user to change the invoice number in the Invoice record. When he does that, though, the changed number must automatically be updated to all the line item records. So we choose CASCADE for 'On Update':

Note that there are no other choices: NO ACTION which merely means that there is **no referential integrity** implemented between the two files and CLEAR which means that the value of the linking field is blanked. For example set the 'On Update' restriction to CLEAR. When the user changes the InvoiceNumber field in the Invoice file, the InvoiceNumber field in all the related Item records would simply be set to zero.

SQL Script Generation

This dialog controls the creation of the SQL script file. Choose **File ➤ Export to SQL Script** to open this dialog.

Output File Name
The name of the SQL script file to be created. Data Modeller automatically appends the suffix .SQL to the name.

Database Name
The name of the database to be created.

Create Database
Check to include SQL statements to create the database.

Device Name
The name of the Device where the database will be stored.

Database Size
The size of the database in Megabytes.

Create Tables
Check to generate CREATE TABLE statements.

Create Indexes
Check to generate CREATE INDEX statements for keys that are not involved in relationships.

Create Primary Keys
Check to generate CREATE PRIMARY KEY statements for keys that are referenced in the ONE side of ONE:MANY relationships.

Create Foreign Keys
Check to generate CREATE FOREIGN KEY statements for keys that are referenced in the MANY side of ONE:MANY relationships.

Create Rules
Check to create sets of triggers to implement validity checks specified in your dictionary.

Drop rules for all fields
Check to generate DROP TRIGGER statements on all fields in the dictionary.

Drop rules to be created
Check to generate DROP TRIGGER statements on existing triggers before generating CREATE TRIGGER statements.

Do not drop rules
Check to prevent DROP TRIGGER statements from being generated.

Create Triggers
Check to generate error codes to be raised by triggers

Raise Error Codes For Triggers
The error codes to be raised by the database triggers. For more information, refer to your database manual.

Large Dictionaries

Very large dictionaries can be slow and cumbersome to edit. Try the following to streamline processing of large dictionaries.

- In **User Preferences**, set Load Dictionary in 100% Zoom Mode ON. This will speed up the re-display of the dictionary.
- Reduce the number of visible files by creating a subset display. Now you can focus in on the files you're working with.
- Find files using the **Locate Files** dialog (RIGHT-CLICK in an open area of the Design Pad).
- Print the dictionary. Print the graphical representation of either the subsets or the entire dictionary. (choose **Print ➤ Print Design**).
- Limit the number of fields displayed in the Field Pool by using the Field Pool Filter (RIGHT-CLICK in the Field Pool, then choose **Set Filter**).

Duplicate Idents

Duplicate Ident numbers are sometimes found in corrupted Clarion dictionaries. The Ident is an identification number that is used by Clarion to synchronize fields declared in a dictionary with controls in an application. The corruption has been found to occur with earlier versions of Clarion, particularly when files have been imported from one dictionary to another.

Data Modeller halts with an error message if you try to load a dictionary with duplicate idents. The error message includes the name of the offending field name. If you create a .TXD file for the dictionary (from the dictionary editor in Clarion), and try to re-import the .TXD file, Clarion stops with an error message which includes the line number where the error was detected. Between the two error messages you should be able to identify the location of the offending ident.

One way of fixing the problem is to manually edit the .TXD file, replacing the duplicate ident with another (large) number that you know has not been used elsewhere in the dictionary. You can then create a new dictionary (with a different name) in Clarion by importing the corrected .TXD file. If the dictionary has been used in an application, you must change the dictionary in the application, re-synchronizing the application and dictionary. See the Clarion documentation for more details. Note that you cannot simply re-import the .TXD file into a new dictionary of the same name, as this may cause synchronization problems that could corrupt your application file.

PART III

WISE FOR CLARION GETTING STARTED GUIDE

Copyright 1994, 1995, 1996, 1997, 1998, 1999 Wise Solutions, Inc. All rights reserved.

Original documentation produced by JagTek® and printed in the United States of America.
Specifications subject to change.

This software and documentation are copyrighted materials. Making unauthorized copies is prohibited by law. No part of the software or documentation may be reproduced, transmitted, transcribed, stored in a retrieval system or translated into any human or computer language without prior written permission of Wise Solutions, Inc.

Notice

Unless otherwise provided by written agreement with Wise Solutions Inc., this publication is provided “as is” without warranty of any kind, expressed or implied, including, but not limited to, the implied warranties of merchantability of fitness for a particular purpose. Some states do not allow disclaimer of expressed or implied warranties in certain transactions, so this statement may not apply to you.

While reasonable efforts have been made to assure the accuracy of this document, in no event will Wise Solutions be liable for direct, indirect, special, incidental, or consequential damages resulting from any defect in this publication or the associated software. Wise Solutions Inc. reserves the right to change this document at any time without obligation to notify anyone.

Trademarks

Windows, Windows 95, and Windows NT are registered trademarks of Microsoft Corporation.

Other brand or product names are trademarks or registered trademarks of their respective holders.

Wise Solutions, Inc. 5880 N. Canton Center Road, Suite 450 Canton, MI 48187-5038 Phone: (734) 981-4970 Fax: (734) 981-9746

05/99

9 - WISE INTRODUCTION AND SETUP

Introduction

With the Wise for Clarion Installation System, you can point-and-click your way to an installation script quickly. You can perform this without any programming knowledge or skill, using the Installation Expert. Common installation functions are added from menus, not by writing code.

Although it's easy to use and learn, it includes features like editing the registry, updating INI files, createing custom dialogs, and even has multi-language support.

NOTE: Wise for Clarion is a major sub-set of Wise's InstallMaker.

You can build your installation either totally within the Wise for Clarion environment or by using the *Generate Wise Installation Script* Template from within the Clarion environment and then process the installation script with Wise for Clarion.

A description about how to use the *Generate Wise Installation Script Template* with Wise for Clarion is found at the end of this chapter.

Tip: Before reviewing the following material see *Distributing Applications* **in the User's Guide and Help➤Contents➤ How Do I ...?➤Deploying Your Application.**

Wise for Clarion Functions

Wise for Clarion is a major sub-set of Wise's InstallMaker. The following highlights the differences between the two products.

Features	**Wise For Clarion**	**Wise** InstallMaker
Installation Expert	X	X
Application Watch Wizard	X	X
Script Editor		X
Expert Property Pages		
Files	X	X
Smart Patch	X	X
Visual Basic		X
Visual Fox Pro		X
ODBC		X
BDE (Borland Data Base Engine)		X
Install Fonts	X	X
Uninstall	X	X
Icons	X	X
Registry	X	X
INI Files	X	X
Autoexec.bat	X	X
Config.sys		X
Add to Path	X	X
Configuration Checks	X	X
Installation Name	X	X
Screen	X	X
Dialogs	X	X
Billboards	X	X
Languages	X	X
Media	X	X

NOTE: Wise for Clarion cannot be installed under Windows 3.1 or Windows NT 3.51 To create 16-bit installation executables that can be run under these operating systems, you will need to upgrade to the Wise InstallMaker.

About Wise Solutions, Inc.

The story of how Wise Solutions became a leader in the creation of installation tools is one of entrepreneurial spirit, hard work, and continual attention to customers' needs.

In 1994, the first versions of the Wise Installation System were distributed as shareware via CompuServe® and various BBS's around the world. By late 1995 they released version 3.0, which established Wise Solutions as the "simple-yet-powerful" installation utility for software developers.

The Clarion Enterprise Edition is proud to bundle a special version of Wise for Clarion for all Enterprise customers.

Ease of Use

One of the great strengths of Wise for Clarion is its ease of use. This allows people without programming experience to easily create fully functional installations.

An easy to use interface

Wise for Clarion includes a rich set of tools that easily supports complex customized installation, while also enabling developers to create simple installations that only take minutes to build. The software's new & improved *Installation Expert* guides you through each critical step in the proper sequence. You just select the type of installation you want to build, specify the source files, choose the desired installation options and the software does the rest! Built in testing tools allow you to instantly run your installation to make sure it is working correctly.

System Requirements

Hardware

You can run Wise For Clarion on any system that meets the minimum system requirements for Microsoft Windows 95™, Windows 98™, or Windows NT. These are our recommended minimums:

- Pentium 90 or equivalent
- 12 Megabytes of Hard Drive space available
- SVGA video - 800x600x256 colors
- On Windows 95/98, 16 Megabytes of RAM.
- On Windows NT, 32 Megabytes of RAM.
- A minimum of 12 to 35 Megabytes free hard disk space, depending on the Setup options you select.

Operating System

Version 7.x of Wise for Clarion is a 32-bit application and can only be installed on a PC that is running one of the following operating systems:

Windows 95 or **Windows 98**

Windows NT 4.0 or better

NOTE: Wise for Clarion cannot be installed under Windows 3.1 or Windows NT 3.51 To create 16-bit installation executables that can be run under these operating systems, you will need to upgrade to the Wise InstallMaker.

Installing the Wise Software

Use this procedure to run Wise for Clarion's setup program. Make sure your PC satisfies the system requirements on the previous page.

Running the Setup Program

1. Insert the Clarion 5 Enterprise Edition CD into your CD-ROM drive.

   The Setup program should launch automatically. If this does not happen, follow steps a - e below:

   a. Click on the Windows Start menu and select *Run….*

   b. In the Run dialog box, type the drive letter for the CD-ROM followed by a colon (for example *D:*).

   c. Click the **Browse** button. In the Browse dialog box, click on the file named Setup.exe.

   d. Click the **Open** button.

   e. In the Run dialog box, click **OK.**

2. Click on the **Wise Solutions Installation System** option**.**

3. When the License Agreement window appears, read the terms and click on the **I Agree** button to accept the agreement.

   NOTE: You must accept this license agreement, otherwise the setup process is cancelled.

4. When the Welcome dialog appears, follow the instructions and click **Next>** to continue.

5. Follow the on-screen directions to complete the rest of this installation.

   - In each case, after entering the required information in a dialog, click the **Next>** button to proceed. When you reach the last dialog, click the Finish> button to complete the installation or click <Back to change any of the information you had entered earlier.

Launching the Software

When you have finished installing the software, you can run it immediately.

From the Windows Start Menu, select the **Programs** folder.

1. Click on the **Wise Installer** folder within the Clarion 5 Enterprise Edition directory or in the folder you installed Wise for Clarion into.

2. Click on the **Wise for Clarion** icon.

3. The program opens with the Tip of the Day displayed. When you close this dialog, the Installation Expert view is displayed.

Registering Your Software

It is very important that you register your software. By registering, you are automatically qualified to receive notifications about future product upgrades. If you did not register your software on-line during the setup process, you may do so as follows:

On-line Registration

You can register your software on-line from within your Wise application. Just open the **Help** menu and choose **Web Connection ➤ Register On-line.** Follow the on screen directions to complete the registration.

Fax / E-Mail Registration

If you are unable to register on-line, you may also do so by filling out the registration card included with your software package and mail it to Wise Solutions.

If you used the Registration Wizard to create a REGISTER.TXT file, you can register by attaching this file to an e-mail and sending it to **register@wisesolutions.com** or fax a printed copy of this file to Wise at **(734) 456-2456 (For calls outside of the U.S., prefix the phone number with the appropriate international access digits).**

Getting Help

Available Support Methods

Web Page

Visit Wise's Knowledge base on the web! Many frequently asked questions, tips and tricks can be found on the web site. Just point your browser to:

http://www.wisesolutions.com/support.htm

Getting Free Program Updates Over the Web

TopSpeed offers version updates to our registered users over the internet. Updates are intermediate product releases which solve minor problems or include enhancements to the existing product. These free updates are available until the next new numbered version is released.

Downloading an Upgrade from the TopSpeed Web Site

You can download the latest Wise for Clarion product upgrade directly from the web. Just point your web browser to the following URL:

http://www.topspeed.com

Follow the directions on screen to download and install the upgrade.

Using Wise for Clarion Documentation & Online Help

Your Wise for Clarion software includes a full HTML-based online help system that provides instant access to topical information.

Wise Online Reference Manual

This PDF document contains information about the entire Wise product suite and is located in the WiseForClarion sub-directory within the Clarion5 directory.

NOTE: Wise for Clarion is a major sub-set of Wise's InstallMaker. The Wise supplied documentation (PDF) does not reflect the options available for Wise for Clarion.

The Wise Online Reference Manual (PDF) is a very comprehensive reference guide but does not present the features and options specific for Wise for Clarion. The Wise PDF should only be used as a secondary reference guide. However; if you would like to reference it, it is located at C:\Clarion5\WiseForClarion\WiseRef2.PDF

NOTE: The Wise On-line Help is a very comprehensive reference and help tool but it does not present the features and options specific to Wise for Clarion. The Wise On-line Help should only be used as a secondary help reference.

Overview of Wise for Clarion

Wise for Clarion is an easy to use, flexible, customizeable tool for creating "application installation" diskettes

You can use the I*nstallation Expert* to totally build your installation. This tool provides an easy-to-follow interface that lets you build new installations step by step. After you use it the first time and see how easy it is, you'll build your future installations in minutes. *See Using the Installation Expert within this chapter*.

You can also use the *Run And Watch Application Wizard* to automatically create your installation scripts. If you're uncertain about which DLLs or OCXs are used by an application that you want to include in an installation, use this Wizard to create an installation script for the files used by any EXE file. This script can then be used as input to the Wise for Clarion *Installation Expert*. *See Run and Watch Application Wizard within this chapter.*

You can also use the Wise for Clarion template from within Clarion to build an installation script that will contain all the elements recognized by the Clarion project system. This script can then be used as input to the Wise for Clarion *Installation Expert*. *See Wise Installation Script template within this chapter*

Want to customize the Wise for Clarion environment? You can customize the steps shown in the *Installation Expert* and you can even add or modify the Dialogs that are displayed during installation. *See Customizing the Environment within this chapter*

Building Installs – Overview

The process of creating a "basic install" is now easier than ever, thanks to the new Installation Expert interface. The Installation Expert guides you through the necessary programming elements in five simple steps.

Specify which files need to be copied to the target computer during the installation. Wise for Clarion provides an easy to use Windows Explorer interface, allowing you to add individual files or folders to the destination file set.

Indicate the additions that are to be made to the destination computer as a result of the installation. These additions include a new program manager group, icon assignments, changes to the Windows registry, updates to system files, or a new INI file.

Allows the installation author to define the system requirements that his/her application need in order to run. Depending on how the options are set, the setup program will either return a warning or abort the installation if the system does not meet the specified requirements.

Enter the title of the software and choose a default installation directory. The title is automatically added to the background screens during setup. Select which dialogs you want to appear during the setup process, providing the user with a variety of installation options.

NOTE: Language Support is available in this step. See Language Support within the Wizard Appearance section of this document.

Choose the media, final format, and destination for the installation file. A **single file installation** bundles all of the files into a single executable. A **floppy based installation** creates a series of files that will fit onto standard floppy disks.

FINISHING UP

When you have finished the Installation Expert, you are ready to compile, test, and run your installation script. After successful testing, choose your media option and make your final installation build. *Easy!*

The next section guides you through the details of each installation step and then shows you how to compile, test, and run your script.

Using the Installation Expert

This section guides you through the process of using the Installation Expert feature in your Wise software to create a new install script from scratch.

> **NOTE: You can also automate the creation of your install script by using either the Run Application and Watch Wizard, or by using the Generate Wise Installation Script Template. (both described later in this chapter.)**

The Installation Expert provides the simplest way to build a complete functional setup script for distributing your software. Its unique graphical interface prompts you for all the critical elements in 5 simple steps. After you have used it once, it will quickly become a natural starting point for all of your future installation projects.

Getting Prepared

Although the Installation Expert automates the process of producing an install script, you still need to provide some basic information to the program. You should be prepared to answer the following questions:

- Do I have all the of program files on the same computer as the Wise Installation Software?
- In which directory will the files be stored on the destination computer?
- Will program icons be placed in the Start Menu or in a Program Group?
- What values or keys need to be added to the Windows' Registry?
- Will changes need to be made to the system files?
- Will new file associations be required to link certain file types with this application?
- Will this setup program be distributed on floppy diskettes?

Opening the Installation Expert

When you open your Wise for Clarion software for the first time, the default view is the Installation Expert screen.

If necessary, click on the **Installation Expert** tab in the lower left corner of the window to open this view.

- Open the **File** menu and select **New** to start a new installation session.

Saving the Installation File

Periodically throughout the following process, you should save the changes to your installation script's file.

1. From the **File** menu, select **Save**.

   The first time you save the file, the **Save As** dialog box opens.

2. If desired, specify a different folder in which to save the installation file.

3. In the **File Name** field, enter a name for this installation script.

   The .wse file extension is applied to the new file.

4. Click the **Save** button.

Navigating the Step Buttons

At the top of the Wise Installation window (just below the Menu bar) are 5 step-buttons, each with one or more commands. These buttons guide you through each distinct process involved in building your installation. Notice that to the right of Step 5, there is a downward pointing arrow button ( ). Clicking on this button removes the commands from the step buttons and reduces their height considerably. (The arrow now looks like this). This

feature increases the viewable area in the main window. You can still access the step commands from the list along the left edge of the window.

In order to keep your screen consistent with this documentation, restore your step buttons to the default setting. To do this, click on the button so it changes back to .

Files and Components

Here's an opportunity to see just how easy your Wise software is to operate. Follow these steps using your own program files to create a new install script.

The first task is to tell the software which program files and components are to be copied to the destination computer during the installation. This process assigns the program files to the appropriate folders.

Assigning Files to a Destination Folder

1. CLICK on the "Step 1" arrow in the upper left corner of the screen and **choose** the **Files** option.
   - The **Select Files** screen opens.

Note: The top section of the Select Files screen shows the files on your computer. The lower section represents the destination computer, where the files will be copied during the installation.

2. In the lower left window, **select** a destination folder where you want to assign some files. All files must be assigned to either the **Application** folder, a Windows folder, or a subdirectory you create.

- The **Application** folder represents the default installation folder for the program, even though you haven't formally named this folder yet. This is where the executables, readme files, and other non-system files are typically assigned.
- System level files, such as fonts and certain .dll's, should be assigned to the appropriate Windows folder. The main system folders are already created for you, though you can add a new folder if needed.
- To create a new destination folder, CLICK the **New Folder** button and **type** a folder name in the dialog box. The new folder is created as a subdirectory of the selected folder.

3. From the upper left section of the Select Files screen, **choose** the files that are to be installed to the destination folder you just selected.

The left window shows all the drives and folders on your system in a familiar Windows Explorer interface. To locate a particular folder, click the [+] sign next to its drive letter or parent directory. Repeat the process until the desired folder is visible, then click on that folder to highlight it.

The right window lists the files in the currently selected folder.

4. Assign the appropriate files to the destination folder.
   - To assign all the files in a single folder, **highlight** the folder in the upper left window and CLICK the **Add Contents** button.

NOTE: The folder itself is not added to the destination path, only its files and subfolders.

   - To assign individual files to the destination folder, **highlight** the file(s) in upper right window and click the **Add File** button.
   - To select multiple files, hold down the CTRL key on your keyboard while clicking on each file.
5. Repeat steps 2-4 (as required) to assign all of your program files to the proper destination folder.
   - To review file assignments, **select** a destination folder. All of the files assigned to that location are listed in the lower right window.

REMOVING FILES FROM THE DESTINATION

To remove unwanted files from the destination folder, **highlight** them in the lower right window and click the **Delete File** button.

File Details

Each file assigned to the destination computer has attributes that classify the file type and determine whether the file should replace existing files during the installation. Although this step is completely optional, you may find it beneficial for certain file types.

To view a file's attributes, **select** the file from the destination computer window and click the **Details** button.

6. When you have finished assigning files, CLICK the **Next>** button to proceed.

- The **Components** screen opens.

NOTE: The Components feature is completely optional. If you do not wish to take advantage of this feature, click the Next> button and proceed to Step 2 – System Additions.

Selecting Components

The Components feature is an option that allows you to "compartmentalize" your program into individual components. During the installation process, users will have the option to decide which components they want to install.

For example, if your program consists of a collection of utilities, each with its own executable file, you could create a component for each utility and assign the appropriate executable to it. During the installation, users are able to individually select the components they want installed. Only the selected executables are copied to their system.

This can also be used to make larger files (like spell checkers and dictionaries) optional components in order to save disk space.

NOTE: Do not create subdirectories for your components. The software takes care of this automatically. Only create components if you want to give the user control over installing certain items in your program.

NOTE: If you originally included these files as part of the main application, you will need to delete them from the Always Installed **component and manually add them to the appropriate component folder. See next section, Creating Components, for details.**

Creating Components

1. CLICK on the *Step 1* arrow in the upper left corner of the screen and **choose** the **Components** option.
2. In the Components screen, click the **Add** button.

   The Component Details dialog box opens.

3. **Type** the component name in the **Component Name** field.
4. If you want this option to be installed by default (i.e. the user must manually unselect it) then **check** the **Install this component by default** **check** box.
5. Repeat steps 2-4 for each component you want created.
6. Return to the Select Files screen by clicking the **<Back** button in the lower right corner.
   - Note that the component(s) you created have been added to the Destination Computer section.

   - The main program files have been added to a component named “Always Installed”.
7. Add the files required by each component to the appropriate folder.
   - If you originally included these files as part of the main application, you will need to delete them from the **Always Installed** component and manually add them to the appropriate component folder.

Uninstall

In this step, you determine whether the installation supports uninstall and set the options for the uninstall EXE.

1. CLICK on the *Step 1* arrow in the upper left corner of the screen and **choose** the **Uninstall** option.

- **Do not add support for unistall. Check** the **Do not add support for uninstall** radio button to disable uninstall. **Check** the **Support uninstall** radio button to enable uninstall.
- **Display uninstaller background window.** Displays a gradient window similar to the one displayed during installation.
- **Top and Bottom Color.** Sets the gradient colors for the background window.
- **Uninstaller Font.** Selects the font used in the uninstaller. **Click** the **Browse** button to choose a different font. **Enter** the **size** of the font in the **Point Size** field.
- **Character Set.** The character set used by the uninstaller. (Use zero unless your uninstaller is Japanese, in this case use 128.)
- **Commands.** The uninstaller automatically reads commands from the INSTALL.LOG file created during installation; you can add addtional commands here. **Click** the **Add** button to add a new command. **Click** the **Delete** button to delete a command previously entered.

When you **click** the **Add** button, a drop-down menu appears. This allows you to add one of the following commands:

Delete Files. **Select** the **directory** containing the files to be deleted using the directory tree, then enter the filename in **Filename** field. You can use a wildcard to specify the files to be deleted from this directory.If the directory is not part of the displayed directory structure, **select** the **directory** it should be found in, and **click** the **New Folder** button to add it.

Note: **It is not necessary to use this command to delete files your installation created. Use it to delete additional files, such as those created by your program upon first running.**

Delete Registry Keys. **Choose** the **registry keys** to be deleted during uninstallation using the registry browser. If the key you wish to delete is not displayed, **select** the **key** it should be added to, then **click** the **New Key** button to add it.

Note: **It is not necessary to use this command to delete registry keys your installation created. Use it to delete other registry entries, such as those created during the execution of your program.**

Execute Program. **Choose** the **directory** containing the program that will execute, as part of your uninstall process, from the directory tree in the left pane of the **Execute Program** window. **Choose** the program to be executed from the right hand pane, or **enter** its name in the **Program Filename** field. Only programs that are a part of your installation can be executed.

Note: **For best results, make sure that the program you execute has little or no user interface. The uninstall process should feel like a single program to the user.**

- **Delete files during uninstall.** Mark this checkbox to have files that are in use during the uninstall process deleted.

Assigning Program Groups & Icons

In this step, you specify the additions that will be made to the destination computer, such as new program groups, registry entries, INI settings, and new file associations.

Assigning Icons

1. Click on the *Step 2* arrow in the upper left corner of the screen and choose the **Icons** option.
   - The Icons screen displays below.
2. In the **Default Group Name** field, enter the name of the new program group where you want the program icons to be stored.

3. Click the **Add** button at the bottom of the screen.
   - The *Select File from Installation* dialog box opens.

   - The right side of this window shows the files you selected for installation back in *Step 1*.

4. From the right side of this window, select a file that you want represented as an icon in the program group, and click **OK**.
   - The file is added to the list in the *Icons* window.
   - You can remove a file from this list by **highlighting** it and CLICKing on the **Delete** button. This removes the icon assignment only, not the file. To remove all of the files, CLICK the **Reset** button.
   - You can specify which icon to assign to a file and where to place the icon on the destination computer by CLICKing on the **Details** button and entering this information in the **Icon Details** dialog box.

NOTE: You must assign an icon to your program's executable file, and it is common practice to assign icons to the readme file, help file, and any other information or utility to which the user should have direct access.

5. Repeat step 4 for each file that you want represented with an icon in the program group.
6. When you have finished adding files, click the **Next>** button to proceed to the next section.
   - The *Edit Registry* window is displayed.

Making Registry Entries

1. Click on the **Step 2** arrow at the top of the screen.
   - The **Registry** option should be highlighted. If not, click on the word "Registry" to make this active.
   - The *Edit Registry* screen displays below.

NOTE: The top section of the Edit Registry screen shows the registry keys on your computer. The lower section represents the destination computer's registry.

2. From the upper left section of the Edit Registry screen, **choose** the registry key containing the values that you want to copy to the destination computer's registry.

NOTE: You do not need to specify a destination registry because the keys and values are automatically copied to the appropriate destination.

- To create a new registry key on the destination computer, CLICK the **New Key** button and enter the key information in the Registry Key Settings dialog box.

3. Assign the new registry values to the destination key.
   - To add all the values in an entire key, **highlight** the key in the upper left window and click the **Add Keys** button.
   - To add individual values, **highlight** those file(s) in the upper right window and click the **Add Values** button. To select multiple values, hold down the CTRL key on your keyboard while clicking on each one.
4. Repeat steps 2-4 until you have designated all of the registry settings to the destination computer.
   - You can remove a value from this list by highlighting it in the lower right window and clicking on the **Delete Value** button. To remove an entire Key, select the key in the lower left window and click on the **Delete Key** button.
   - To remove all of the Key settings, click the **Reset** button.
5. When you have finished adding key entries, click the **Next>** button to proceed to the next section.

Creating Registry Entries from Scratch

You can create a new registry key on the destination computer and assign values.

1. Click the **New Key** button. The Registry Key Settings dialog box opens.

2. Specify the key name, value name, data value and data type the provided fields.
3. Click **OK** to create the new key setting.

Adding INI Entries

1. CLICK on the **Step 2** arrow in the top of the screen.
   - The **INI Files** option should be highlighted. If not, **CLICK** on the word "INI Files" to make this active.
   - The INI Files screen displays below.

2. Perform the appropriate action for each INI file you want to create:

   To Create a New Folder on the Destination System:

   1. **Select** the folder in which you want to create the new folder.
   2. CLICK the **New Folder** button. The *Create New Folder* dialog box opens.
   3. **Type** a name for the new folder in the **Folder Name** field.
   4. CLICK **OK** to add the folder.

   To Create a New INI File:

   1. **Select** the folder in which you want to create the INI file.
   2. CLICK the **New File** button. The Edit INI File Settings dialog box opens.

3. In the **File** field, **type** the name for the new INI file.
4. In the **INI File Contents** field, **type** any initial settings that you want to appear in this file.
5. CLICK the **OK** button to create the file.

- The new INI file is listed in the right window of the INI Files screen.

To Append Information to an Existing INI File:

You may want to make changes to one or more of the existing system files on the destination system, such as the WIN.INI and SYSTEM.INI file. This can be done in the following way:

1. **Select** the folder where the system file is located.
2. CLICK the **New File** button and enter the name of the system file to which you want to append (e.g.WIN.INI).
3. In the *INI File Contents* field, **type** the information that you want to merge into the system file.
4. CLICK **OK** to save these settings.
   - When the installation is run, these settings will be merged into the existing system file.
   - To edit or view a file's contents, **select** it from the right window and CLICK the **Details** button. Make the desired changes and CLICK **OK.**
   - To remove an INI file, **select** it from the right window and CLICK the **Delete** button. To remove all INI entries, CLICK the **Reset** button.
5. When you are done making INI entries, CLICK the **Next>** button.

Autoexec.bat

The following dialog enables commands to be added AUTOEXEC.BAT file during installation. The user will be prompted to restart the system after installation.

1. **Add Directory to Path.** This section establishes how the PATH variable should be set to add a directory from your installation to it.
   - **Directory to add to Path.** Enter the pathname of the directory to be added.
   - **Location of new directory**. Place the directory at the beginning or at the end of the PATH variable.
   - **Path selection**. Choose to edit just the first PATH statement in AUTOEXEC.BAT, or to all of them.
2. **Commands to add to Autoexec.bat.** You can enter one or more commands to be added to the AUTOEXEC.BAT file. **Click** *Add* to add a new command (you'll be prompted to enter a new line). **Click** *Delete* to remove a selected line from the list. **Click** *Details* to display the Add Command to AUTOEXEC.BAT dialog (see next topic) allowing you to specify exactly how the file should be edited.

Add Commands to Autoexec.bat

This dialog lets you specify how a line should be added to the AUTOEXEC.BAT file, either at a specific line number or by searching for a specific piece of text.

Text to Insert. Enter the command line you want to add to the AUTOEXEC.BAT file. If your command line refers to a program file, use a full pathname built with variable substitution (such as %MAINDIR% to specify the application directory) so your program will run regardless of the directory the user installed it in. (The PATH variable may not be set when your command is executed, so always use a full pathname).

Line Number. The line at which the new line should be inserted. (Specify zero to append the command to the end of the file.) The Search for Existing Text panel in this dialog overrides any line number specified here; the line number applies only when the text is not found or when you do not specify any text.

Search for Text. Enter the text you wish to search for here. The installation will scan AUTOEXEC.BAT looking for a line that begins with, ends with, or contains the text (depending on the setting of the *Match Criteria* field). If more than one line in the file matches, only the first will be selected

Comment Text. Enter the text to insert at the beginning of the line when it has been found. Inserting “REM ” (without the quote marks and with the trailing space) will cause the line to be commented out and ignored during start-up. This is useful when you are replacing an existing command with a new command and wish to leave the existing command in place but inactive. In this case, you should always set the Insert Action drop-down to cause your new command to be inserted before the existing line so a subsequent installation will find and edit the active command, not the commented-out line.

Insert Action. Select the action to be taken when a line containing the specified text is found. You can either insert your new command before the existing line, replace the existing line with your new command, or insert your new line after the existing one.

Match Criteria. Choose whether the line must begin with, contain, or end with the text entered in the *Search for Text* field.

7. **Ignore white space.** If this checkbox is marked, the search operation ignores spaces and tab characters.
8. **Case-sensitive.** If this checkbox is marked, the search operation distinguishes between uppercase and lowercase text.
9. **Make Backup File.** If this checkbox is marked, the installation makes a copy of the AUTOEXEC.BAT file before editing it. The backup is retained after installation so that the user can undo any changes made to AUTOEXEC.BAT if they cause problems.

User System Checks

Configuration Checks

In this step, you specify a set of conditions and/or recommended settings that must exist on the destination computer in order for this program to run.

1. CLICK on the *Step 3* arrow in the top of the screen and choose the **Configuration Check** option.
   - The *Check Configuration* screen displays below, showing a list of the components which you can set minimum requirements.

2. **Select** the component that you wish to edit and CLICK on the **Details** button (or DOUBLE-CLICK on the component).
   - The *Configuration* dialog box opens.
3. In the **Minimum Level** field, **select** the minimum setting that must exist on the destination computer.
   - For hardware settings, such as screen resolution, color depth, and memory, any setting above (or higher) than the minimum you specify is acceptable.

4. **Choose** the appropriate reaction level for this component test.
   - **Don't Perform Check** bypasses this test. The system will not check this component or issue any warnings.
   - **Recommended** generates a warning message if the destination does not meet the Minimum Level requirement, however the installation will continue.
   - **Required** components causes the installation to abort if the destination system does not meet the Minimum Level requirement.
5. CLICK **OK** to save your components settings.
6. Repeat steps 2-5 for each component you wish to activate.

Wizard Appearance

Entering the Software Title & Default Program Directory

NOTE: Language Support is available in this step, although not listed as an available function on the above dialog. See Language Support within this section.

In this step, you specify the name of your application so it can be built into the background screens during the installation. You also choose the dialogs that appear during the installation to prompt the user for information.

1. Click on the *Step 4* arrow in the top of the screen.
   - The **Installation Name** option should be highlighted. If not, CLICK on the word "Installation Name" to make this active.
   - The *Installation Name* screen is displayed.
2. In the **Setup Title** field, **type** the name of the product that is being installed.
3. In the **Default Directory** field, **type** the directory that you want the software to use as the recommended installation path. Do not enter a drive specification, drive *C:* is assumed, and it cannot be changed here. At runtime a user can select any available drive.

NOTE: The user will have the option of installing to a different directory unless you specify the Destination Directory option in the Dialogs section of setup.

- If you wish to specify a multilevel path, separate each directory level with the "\" character. For example: *CAT\DELUXE TOOLS* would create the *DELUXE TOOLS* subdirectory under the *CAT* directory.
- If you want the directory created under the *\PROGRAM FILES* directory on the specified drive, **check** the **Place default directory under Program Files** check box. For example, if you specify *TEST* as the default directory and check this option, the default installation path would be *C:\PROGRAM FILES\TEST*.
- It is standard practice in Windows 95, 98, and NT applications to be installed under the Program Files directory.

4. CLICK the **Next>** button to continue to the next section.
   - The Dialogs Screen opens.

Selecting Dialogs and Installation Options

This section allows you to select the individual dialog boxes that will appear during the installation. The dialogs that you choose dictate the type of installation options the user will have available to them.

1. CLICK on the *Step 4* arrow in the top of the screen.
   - The **Dialogs** option should be highlighted. If not, CLICK on the word "Dialogs" to make this active.
   - The Dialogs screen displays as follows:

NOTE: Although the message text in the above dialog box indicates you can press the Edit button to launch the Custom Dialog Editor, this feature is not available in Wise for Clarion. This feature enables Dialog customization for the specific install you are processing.

You can edit Dialogs with Wise for Clarion, but it is performed by editing the Dialog Templates - See Editing Dialog Templates within this section. By editing the Dialog Templates, your changes will affect all subsequent installations not the current installation session. You must create a new installation session to use the modified dialogs.

2. **Choose** the dialog boxes that you want to appear by clicking on the corresponding check box.
   - A default set of installation dialogs is automatically selected. You can restore the default set at any time by clicking on the **Reset** button.
   - A brief description of each dialog is provided below.

Default Dialogs

Welcome Dialog
Welcomes the user to the program, recommends exiting all other applications before proceeding, and restates copyright law protecting the program.

ReadMe Dialog
This dialog displays the contents of an existing ASCII text file, such as a readme file. You must specify the path to the text file in the Settings dialog at the bottom of the screen. Click the Browse button to locate the file.

Branding / Serializing Dialog
This dialog provides a place to enter the name of the registered user and/or the name of the company that owns the software.

Destination Directory Dialog
This dialog shows the default installation directory and provides the user the opportunity to change the destination folder and drive.

Backup Replaced Files Dialog
This dialog box provides the user with the option to save original copies of any files that are replaced or updated during the installation. When the software is uninstalled, these files are returned to their original state.

Select Components Dialog

> **Note:** **This option is only accessible if you created a component back in Step 1.**

This dialog box lists the components you created so users can choose the ones they want to install.

Select Icon Group Name Dialog
This dialog box shows the default program manager group that is created and provides the user with the opportunity to change it.

Start Installation Dialog
This dialog box tells the user that the setup program is now ready to install the program files. The user must click the Next> button to start the process, or they can click <Back to edit the information they entered earlier.

Finished Dialog
This dialog box indicates that the application has successfully installed.

Editing Dialog Templates

The Dialog Templates Editor lets you change the appearance of any dialog. Changing a template changes all future installations. You can change or add static items (text, graphics, and other on-screen items that the user can't modify), text editing fields, buttons (including checkboxes and radio buttons as well as push buttons), combo box and list controls, and functions for aligning or spacing dialog items to give your application a professional look.

1. Select **Edit>Dialog Templates**

The Selecting Dialog to Edit dialog now appears.

2. Select a Dialog to Edit and Press OK.

Using the Custom Dialog Editor, you can customize the installation dialogs to meet your requirements.

3. When you are finished choosing the dialog boxes, click the **Next>** button at the bottom of the Custom Dialog Editor screen to continue.

NOTE: By editing the Dialog Templates, your changes will affect all subsequent installations not the current installation session. You must create a new installation session to use the modified dialogs.

Screen

The Screen Dialog sets the background color, gradient, and font for the installation.

- **Background Gradient.** Choose whether to display a full-screen gradient, a three-quarters screen gradient, or no gradient behind the installation dialogs.
- **Title Bar.** If marked, this checkbox causes the installation's title to be displayed at the top of the screen in the title bar.
- **Hide Program Manager.** Mark this checkbox to hide the Program Manager during installation (Windows 3.x only).

NOTE: Although you can check or un-check the Hide Program Manger checkbox, this function is disabled for Wise for Clarion. Wise for Clarion cannot be installed under Windows 3.1 or Windows NT 3.51 To create 16-bit installation executables that can be run under these operating systems, you will need to upgrade to the Wise InstallMaker.

- **No Backgound Gradient.** Mark this checkbox to display no gradient behind the installation dialogs.
- **Top & Bottom Color.** Uses standard Windows color selectors to choose the top and bottom colors for the background gradient. The installation automatically generates a smooth transition between the two colors.
- **Screen Preview**. Shows a mock-up of how your installation screen will look.
- **Bold/Light Fonts.** Choose whether to always display bold or light fonts.

- **Message Box Fonts.** Choose the font to be used. If no font is specified, a standard sans serif font will be used.
- **Point Size.** Select the point size for the text displayed in the installation dialogs. If no size is specified, the standard Windows text size is used.
- **Character Set.** Enter the number of the character set to be used. The default, zero, should be used unless your installation is in Japanese, in this case you should use 128 and make sure you have specified MS Gothic for the font.

When you are finished, click the **Next>** button at the bottom of the Screen dialog to continue.

Billboards

The Billboard dialog lets us select a series of graphics to be displayed during installation. You can use these graphics to inform end users of software features, to remind them to register their new product, or for marketing purposes. Graphics should be 256-color BMP files.

Note: If you display multiple 256-color images on a 256-color display, the graphics may not appear with the correct color unless the same palette has been used for all images..

You can add a new image by **pressing** the **Add** button, or you can delete a graphic by **selecting** it and **pressing** the **Delete** button. Use the **Move Up** and buttons to reorder the images, moving the selected item up or down in the list. The **Details** button displays the Billboard Settings dialog (you must have added an image to the list for this button to be enabled).

Billboard Settings

This dialog sets the options for the billboard images.

- **Pathname.** Enter the path to the image files you wish to display. Images are compiled into the installation EXE file.
- **X Position, Y Position.** Indicates the location on the screen at which the specified image will be positioned. Coordinates should be specified for a 640 X 480 screen.
- **Erase Number.** Determines how may of the graphics currently being displayed should be erased before the new image is displayed. To display only one image at a time, set this field to 1. The oldest image is removed first.
- **Build Effect.** Sets the transition effect used to display the image. You can use this drop-down menu to cause the graphics to fade in, or to move the screen from any edge of the screen.
- **Transparent.** Mark this checkbox to have the pure blue (R=0, G=0, B=255) parts of your image become transparent.
- **Center Horizontal.** Mark this checkbox to display your image centered horizontally.
- **Place at Right.** Mark this checkbox to display your image at the right edge of the screen. (Due to rounding errors inherent in the Windows Metafile format, this option may be slightly inaccurate when used with custom graphics that have been scaled to the screen.)

- **Scale to Screen.** Mark this checkbox to scale the image to take up the same proportion of the screen regardless of the screen resolution. (It is displayed at actual size in 640 X 480 mode.)
- **Hide Progress Bar.** Mark this checkbox to hid the progress indicator while this graphic is being displayed. (A series of graphics can be used as a progress indicator instead of the Progress Bar.)
- **Center Vertical.** Mark this checkbox to center the image vertically.
- **Place at Bottom.** Mark this chekcbox to place the image at the bottom of the screen. (Due to rounding errors inherent in the Windows Metafile format, this option may be slightly inaccurate when used with custom graphics that have been scaled to the screen.)
- **Tile Background.** Mark this checkbox to repeat the graphics edge-to edge to fill the entire screen.
- **Erase All.** Mark this checkbox to remove all previous graphics from the screen before displaying the new one.
- **Timed Display.** Mark this checkbox to automatically display a series of graphics at evenly-timed intervals. This is marked by default and should remain marked for the billboard feature to work correctly.
- **Local Graphics.** Normally, graphics are copied to the installation EXE. Enter the pathname here (typically beginning with the %INST% variable substitution to specify the directory containing the installation EXE) to indicate that the graphic to be displayed is in a separate file. This can be useful to allow you to change graphics without having to rebuild the EXE.

Multiple Language Support

Note: **Only leave the Always Prompt checkbox un-marked when you are not supporting multiple languages with your install. If the Always Prompt checkbox is not marked when using multiple languages, the installation will not present the user language choices.**

If the Always Prompt checkbox is not marked, the installation seems to support multiple language support when running in test mode. However, when actually running the installation the dialog for language choices will not display.

While on step 4, Wizard Appearance, selecting Languages for the left hand list box will enable you to customize the language preferences for your installation.

- **Language List.** Click the **Add** button to add a new language to the installation. The Wise for Clarion Installation supports U.S. English (the default), French, German, and Italian. Click the **Delete** button to remove the selected language from the list. Use the **Move Up** and **Move Down** buttons to rearrange the list (the user is asked to select a language from the list at the start of installation).
- **Default Language.** Choose the default language for the installation.

- **Japanese Font Name and Point Size. Check Japanese Font Name and Point Size. Choose** the **font** to be used for a Japanese installation and its point size.
- **Copy Default.** Copies messages from the default language to all others to provide a starting point for translating the message.
- **Always Prompt.** If this checkbox is marked, the installation will always ask the user what language he prefers even if only one language is available.

Choosing Media and Global Options

In this step, you enter the final pieces of information into the Installation Expert – the media options. This is where you specify whether you want the setup program to be created as a single EXE file, or whether you want it "chopped up" into smaller pieces so it can be distributed on floppy diskettes.

Media

1. CLICK on the *Step 5* arrow in the upper right corner of the screen.
   - The **Media** option should be highlighted. If not **CLICK** on the word "Media" to make this active.
   - The Media screen displays as shown below.
2. Select either **Single File Installation** or **Floppy Based Installation** by clicking on the corresponding radio button.

Single File Installation
This option bundles all of the program files into a single executable file (i.e.,

SETUP.EXE). This is a good choice for installations that will be distributed over a network, BBS, or e-mail. Since this file may be larger than the capacity of a floppy diskette, you should not use this option for programs that will be distributed on floppies.

Floppy-Based Installation
Choose this option if the program will be distributed on floppy diskettes. This will automatically control the file size and create multiple files if necessary to ensure that each setup component will fit on a floppy disk.

3. If you chose Floppy Based Installation, you must also choose the size of the disk onto which the program will be copied. Options include:
   - **3 ½" High Density Floppy**
   - **5 ¼" High Density Floppy**
   - **Custom Size**

     In the Custom Size field, **type** the capacity of the disk you are using, in kilobytes. (1.0MB = 1000KB)

Choosing Global Options

This dialog sets options to be used by the Compiler and the Watch Application Wizard.

1. **CLICK** on the *Step 5* arrow in the upper right corner of the screen.
   - **CLICK** on "Global Options" to make this active.
   - The *Global Options* screen displays as shown below.

- **Prompt to Save**. Mark this checkbox to have the Script Editor prompt you to save your installation script each time you build a new installation EXE. If this checkbox is not marked, the script is always saved automatically before compiling.

NOTE: Wise for Clarion does not support this feature. Checking or un-checking this box will have no effect.

- **Prompt for Changed Compiler Variables.** Mark this checkbox to have the Wise for Clarion System prompt you to specify new values for compiler variables when you build the installation. Compiler variables are set in the Compiler Variables dialog.

NOTE: Wise for Clarion does not support this feature. Checking or un-checking this box has no effect.

- **Show Toolbar Tips.** If this checkbox is marked, tooltips are available for the buttons in the Script Editor toolbar.

NOTE: Wise for Clarion does not support this feature. Checking or un-checking this box has no effect.

- **Show Status Bar Tips.** If this checkbox is marked, status bar tips are available for the buttons in the Script Editor toolbar.

NOTE: Wise for Clarion does not support this feature. Checking or un-checking this box has no effect.

- **Append New Items.** If this checkbox is marked, new script actions are inserted after the currently selected action, rather than before, when an action is double-clicked (or *Add Action* is chosen from the *Installation menu*).

NOTE: Wise for Clarion does not support this feature. Checking or un-checking this box has no effect.

- **Color Selection.** This drop-down menu allows you to choose the color for the various types of script actions recognized by the Script Editor.

NOTE: Wise for Clarion does not support this feature. Checking or un-checking this box has no effect.

- **Suppress Version Error.** Mark this checkbox to allow the use of version checking when installing of copying files that do not have version resources. When a file is detected without a version resource, version checking is suppressed and no error is generated.
- **Background Processing.** Mark this checkbox to allow other applications to run during the compilation process. (This slows the compile process by about 50%.) If this checkbox is unmarked, the computer ignores user input during compilation.
- **Smart Create.** Mark this checkbox to automatically rebuild the installation EXE whenever any file incorporated in the installation has been changed. This option will slightly increase the time required to perform the Run or Distribute commands.
- **Fast Create.** Mark this checkbox to speed the compilation process by copying the compressed version of a file from the previous version of the installation EXE to the new one. If the size or date of a file has changed, it will be re-compressed.
- **Run in Manual Mode.** Mark this checkbox to have the installation EXE prompt you for the location of all the directories to be used for installations (Window, System, etc.) whenever the installation is run from within the Wise for Clarion Installation .
- **System DLLs to exclude from Watch Application.** Enter the names of DLLs that should not be included in dependency checking when using the Watch Application Wizard, one on each line.

2. When you are finished selecting the Global Options, you should save your file.

NOTE: Although you can press the Next button and proceed directly to the Distribute function, you should save your work and perform Compile, Test and the Run functions prior to performing the Distribute function.

Compiling & Testing Your Installation

After you have selected all of the options for your installation, you are ready to compile and test the installation script to make sure it is functioning properly.

Compiling

The compiling process builds an executable file with the same name as your Wise script file.

1. CLICK the **Compile** button at the bottom of the Installation Expert window.

- If all the required program files are present, the compiling process begins. While the file is compiling, the status is shown on the screen.

- If any component of the installation is missing or incomplete, an error message displays. Follow the instructions to correct the problem.

Testing Your Installation Script

The testing process runs your installation program, but it does not copy any files to the destination computer or perform any system updates. This allows you to visually review the installation program so you can make sure it is complete.

You can only test your installation after it has been successfully compiled (i.e. compiled with no error messages).

TO TEST YOUR INSTALLATION SCRIPT:

1. CLICK the **Test** button at the bottom of the Installation Expert window.

- The installation program begins.

2. CLICK the **Next>** button to advance through each screen of the setup program.

> **NOTE: Remember that files are not copied to the destination computer during testing. The test installation only appears to copy the files — no actual changes are made to the destination system.**

Running Your Installation Script

After you have tested the installation for cosmetic errors, you can the check the integrity of the program itself by running the installation. Unlike the Test process, this will run the full installation, meaning that files are copied to the destination computer, program groups and icon assignments are created, and system edits are implemented.

You can only run your installation after it has been successfully compiled (i.e. compiled with no error messages.)

1. CLICK the **Run** button at the bottom of the Installation Expert window.
   - The installation program begins.
2. CLICK the **Next>** button to advance through each screen of the setup program. When the installation is done, CLICK the **Finish** button.
3. **Minimize** the Wise Installation window.

4. Run the program from its program group or the folder in which it was installed.
   - Verify that all elements of the program have been created and installed to your satisfaction.

Distributing Your Installation Program

This is the "end of the line" in the installation-building process. When you are certain that your installation is ready for distribution, use this process to convert the Wise installation file into a standard install file, such as *SETUP.EXE*.

1. CLICK the **Distribute** button at the bottom of the Installation Expert window.

   - The *Distribute Wizard* dialog opens.

2. Select the desired settings for your installation file.

   - **Choose** the destination disk for the setup file(s).
   - **Choose** *Floppy disks*, and **press** the **Next** button.

- To assign a label to the destination disk, **type** the label in the **Disk Label** field.
- **Type** the name you want to assign to the executable file. (The default is "SETUP").
- If you wish to leave some blank space on the first disk (for a readme file or similar addition), **type** the amount of space (in kilobytes) that you will need in the **Reserve** field.
- From the **Floppy Disk Size** drop list, **choose** the storage capacity of the floppy disk(s) to which you are copying the files. These values are in Kilobytes. (i.e. 1.4MB = 1400KB). To create a single executable file, select "**0**" in this field.
- If you need to erase the target disk before creating the installation file, **check** the **Erase Disk Before File Copy** check box.

3. **PRESS** the **Finish** button to create the installation file.

- If you are copying files to a floppy, you are prompted to insert the first disk. After inserting the disk, CLICK **OK**.
- The installation file is copied to the specified drive location.

- Follow the on-screen direction, entering additional disks as required.
- When the installation has been successfully copied, a confirmation message appears.

4. CLICK **OK**. The installation is now on the destination disk and can be distributed to others or reproduced.

NOTE: You should always test the master installation disks before distributing them to make sure the disk isn't damaged. It is also a good practice to scan the master disk for viruses prior to duplicating them.

Application and Watch Wizard

This Wizard creates installation scripts by monitoring your system as you execute an application. It automatically determines which DLLs and OCXs are used by the application and adds them to the installation.

NOTE: Before beginning, quit all running applications except Wise for Clarion.

- **Press** the *Wizards* tab on the lower left hand corner of the Installation Expert Window.

- **Press** the button with the camera icon on it.

In this dialog, enter the full pathname of the EXE file to be run (or use the **Browse** button to choose it using an Open dialog). Enter any command line options for the EXE in the Command Options field. Then **click** the Run Application button.

As the program runs, Wise monitors your system to see which DLLs and OCXs are opened. You should exercise all features of the application to ensure that even DLLs needed for obscure or infrequently-needed functions are noted and included in your application.

NOTE: While running your application do not print. This will invoke system files specific to your printer, and others that are already on every Windows system.

After using all the features of your application, exit it and **click t**he **Finish** button in the Watch Application dialog. The Wizard will generate an installation script which can be used as a starting point for a complete installation just open this script file from the Installation Expert main menu.

Customizing the Environment

With the Wise for Clarion system you can tailor the Installation Expert as well as the Dialogs presented during the installation of your application.

- From the Wise for Clarion main menu, **select** *Edit*, and you'll be presented with:

Customizing the Dialog Templates

By **selecting** the *Dialog Templates* menu item and following the instructions, you can edit the templates which are used to build dialogs, including those displayed by the Wise for Clarion Installation System.

Changes you make to these dialog templates will be used for all future installations you build.

NOTE: It is strongly recommended that you back up the Templates directory before making changes to the dialog templates. These are located in the Dialogs folder, which is in the same folder as the Wise for Clarion system.

Customizing the Installation Steps

Using this dialog, you can create multiple views of the installation process to be used by various types of installations, or by different users in your development team. Users can then select the appropriate view from the Show drop-down menu within the Installation Expert.

Wise Installation Script Template

How to Use

See Generate Wise Installation Script Template in the Application Handbook manual.

You can also create an installation script from within the Clarion Environment. Just use the Generate Wise Installation Script Global template. Once you have created the installation script using this template you can use it as a starting point for a complete installation by just opening it from the Installation Expert main menu.

To use the Generate Wise Installation Script Template:

1. Open your application from within the Clarion environment.
2. **Press** the **Global** button from the **Application Tree** dialog.
3. **Press** the **Extensions** button from the **Global Properties** dialog.
4. **Press** the **Insert** button from the **Extension and Control Templates** dialog.
5. S**elect** *Wise - Generate Wise Installation Script* template.

When you **Make** and **Run** your application a Wise installation script will be generated with a file extension of .wse.

INDEX

Symbols

#CODE Templates 72, 73
#EXTENSION Templates 72, 73
.CFG 180
.TXD 243

A

Access List 207
Add Selected Key 240
Add to target 149
Add To Type Pool 234, 241
Adding Extension Templates 19, 75
Alias Editor 258
Alias Name 258
Aliases 236, 246, 258
amortization 18, 23
AMORTIZE 26, 77
annuity 38, 42, 46
  with prepayment 40, 43, 47
Append File 232
Application and Watch Wizard. *See* Wise for Clarion
APR 28, 79
archive 159
Archive Directories 183, 188
archive directory 159
Autoexec.bat. *See* Wise for Clarion
average 54, 94

B

Base 10 math 24
Base Privileges 222
BCD math 24
Billboards. *See* Wise for Clarion
Binary Coded Decimal math 24
Browse / Convert File 232, 249
Building Installs. *See* Wise for Clarion
Business Math Functions
  calling 71
Business Math Library
  hand-code support 20
  implementing 18
  installing 19
  overview 18
  using 19
Business Math Templates 71
  access 71
Business Statistics Code Templates 91
  FACTORIAL 91
  FREQUENCY 92
  LOWERQUARTILE 93
  MEAN 94
  MEDIAN 95
  MIDRANGE 96
  MODE 97
  PERCENTILE 98
  RANGEOFSET 99
  rVALUE 100
  SDEVIATION 101
  SS 102
  SSxy 103
  ST1 104
  SUMM 105
  UPPERQUARTILE 106
  VARIANCE 107
Business Statistics Library
  components 49
  overview 49
BUSMATH.PR 20

C

C5log.txt 126
C5VCS.CFG 182
Cancel Subset Display 233
Cascade Edits 257
Cascade Field Property 256, 257
Case Sensitive 249
cash flow analysis 18, 23
change description 160
Change Field 239
Change Justification 249
Changing a field's attributes 238
Check In 203
check in 179
Check In Options 204
Check In Templates 206
Check Out 207
check out 179
Class Finance 72
Class Statistics 73
CLFIN16.LIB 23

CLFIN32.LIB 23
CLSTAT16.LIB 49
Code Templates 19, 23, 49
  access 19, 72, 73, 75
  Business Statistics 91
  embedding 76
  Finance 77
  Statistics 91
Color Settings 254
Compiling Your Install. *See* Wise for Clarion
COMPINT 29, 80
Composite Privileges 222
compound interest 29, 80
Configuration Checks. *See* Wise for Clarion
Configuration Files 182
Configuration Strategies 197
CONTINT 30, 81
continuously compounded interest 30, 81
Conventions 15, 16
Convert Program 232, 249
Copy
  Field 239
  File 231, 234
Copy to target 149
correlation coefficient 60, 100
Corrupted dictionaries 264
count 52, 57, 97
Create
  Alias 258
  Database 261
  Dictionary 243
  Field 231, 239
  File 236, 244
  Key 240
create archive 203
Create Indexes 261
Create New Empty File 232
Create Rules 261
Create Tables 261
Creating an Archive 207
Creating an Installation Diskette. *See* Wise for Clarion
Customizing the Environment. *See* Wise for Clarion
CWFIN16.DLL 23
CWFIN16.LIB 23
CWFIN32.DLL 23
CWFIN32.LIB 23
CWFINPRO.CLW 20, 24
CWSTAT16.DLL 49
CWSTAT16.LIB 49
CWSTAT32.DLL 49
CWSTAT32.LIB 49
CWSTATDT.CLW 50
CWSTATPR.CLW 20, 50

D

Data Modeler 229
Data Type Defaults 252
Data Type Pool. *See* Type Pool
Database Name 261
Database Size 261
date 82
date calculations 31, 82
DAYS360 ..... 31, 82
Default data types 252
Define Subsets 236
Delete
  Alias 258
  Field 234, 239
  Field type from Type Pool 242
  Files 231
  Key 234, 241
  Relationship 231, 234, 235, 241
Delete File from disk 232
Delete from target 150
Deploying Your Applicaiton. *See* Wise for Clarion
Design Pad 229, 231
  Configuration 233, 236, 245, 247
  Configuration (color) 254
  Configuration (filter) 245
  Configuration (font) 254
  Configuration (pool placement) 255
  Configuration (relation lines) 256
  Popup menus 231
  Scale 247
  Subsets 233
Design Pad - preferences 255
Device Name 261
Dialogs, Default. *See* Wise for Clarion
Dialogs, Editing. *See* Wise for Clarion
Dialogs, Selecting. *See* Wise for Clarion
Dialogs, Template. *See* Wise for Clarion
Dictionary
  Create new 243
Dictionary Analysis 252
Dictionary Properties 236, 244
Dictionary Settings 257
Display Subsets 245
Distributing Your Application. *See* Wise for Clarion
Documentation Conventions 15
documentation format 16
Don't Force Check In 184
Drop rules 261
Duplicate Fields 256
Duplicate Ident numbers 264

E

Edit Comments 231
ellipsis (...) button 77, 91
Embed point 19, 71, 72, 73, 75
Embedded Source dialog 76
Embedding Code Templates 76
Error checking 252
Event Information 194
event trigger 192
Export
  Dictionary (DCT) 243
  Dictionary (TXD) 243
  SQL Script 243
Export Clarion Source Code 232
Export Comments 232
Export File 232
Export Long Descriptions 233
Extension Templates
  adding to your application 19, 75
Extensions and Control Templates dialog 75

F

FACTORIAL 51, 91
Field
  Copy 239
  Delete 239
  Locate 238
  prompts 247
  Quick Reference 234
  Same name 238
Field Pool 229, 238, 241
  Color codes 238
  Filter 238
  Popup menu 238
  Tagging 239
Field Popup Menu 234
Field Properties 234
File
  Copy 231, 234
  Delete 231
File Aliases 236, 246, 258
File Convert 249
File Export Seq 251
File locking 189
File Menu 243
File Popup Menu 231
File Properties 231
File Relationships 259
File-level locking 161
Files
  Reposition 237, 246
Files and Components. *See* Wise for Clarion
Filter
  Field Pool 238
Finance Class 72
Finance Code Templates 77
  AMORTIZE 77
  APR 79
  COMPINT 80
  CONTINT 81
  DAYS360 82
  FV 83
  IRR 84
  NPV 85
  PERS 86
  PMT 87
  PV 88
  RATE 89
  SIMPINT 90
Finance Functions
  APR 28
  COMPINT 29
  CONTINT 30
  DAYS360 31
  FV 32
  IRR 34
  NPV 36
  PERS 38
  PREFV 33
  PREPERS 40
  PREPMT 43
  PREPV 45
  PRERATE 47
  PV 44
  SIMPINT 48
Finance Library
  conventions 24
  data types 24
  overview 23
  parameters 24
  return values 24
  Sign Conventions 25
Finance Procedures
  AMORTIZE 26
Finance Template Class 19
FINANCE.TPL 19, 24, 74
FinanceLibrary 72
Floppy Install. *See* Wise for Clarion
Force Check In 184
FREQUENCY 52, 92

Functions Supported. *See* Wise for Clarion
future value
32, 38, 40, 42, 43, 44, 45, 46, 83, 86, 87, 88, 89
with prepayment ... 33
FV ... 32, 83

G

Getting Help. *See* Wise for Clarion
Global Extensions ... 19, 75
Global Options. *See* Wise for Clarion
Global Properties dialog ... 75

H

Hand-Code Support ... 20
Help. *See* Wise for Clarion
Help menu ... 253
Hide All But... and Related Files ... 233
Hide File and Related Files ... 233
Hide File Only ... 233
Hot Keys
Generate ... 248
Remove ... 248
hypothesized mean ... 63, 104

I

Icons. *See* Wise for Clarion
Ident numbers
Duplicate ... 264
Ignore Difference ... 150
Import
Dictionary (DCT) ... 243
Import Error ... 251
Import From... ... 240
Importing Dictionaries ... 256
INCLUDE ... 20
INI Entries. *See* Wise for Clarion
initial revision ... 160
Install, CD. *See* Wise for Clarion
Install, Compiling. *See* Wise for Clarion
Installing ... 19
interest ... 26, 77
interest calculations ... 18, 23
internal rate of return ... 34, 84
IRR ... 34, 84
ISLV.INI ... 186

K

Key
Create ... 233
Delete ... 241
Locate ... 240
Key Pool ... 229, 240
Popup menu ... 241
Using ... 233
Key Popup Menu ... 233
Key Properties ... 233
KEYWORD
Syntax Diagram ... 17

L

Language Support. *See* Wise for Clarion
Large dictionaries ... 263
linear regression ... 18, 60
local configuration file ... 182
Locate
Fields ... 238
Files ... 236, 246
Keys ... 240
lock ... 160, 179
LOWERQUARTILE ... 53, 93, 98, 106

M

Managing large dictionaries ... 263
Managing Users and Groups ... 224
MANY:MANY ... 260
MANY:ONE ... 260
MAP ... 19, 20, 71, 72, 73, 75
master configuration file ... 182
MASTER.CFG ... 182
MEAN ... 18, 54, 94
Media Options. *See* Wise for Clarion
MEDIAN ... 55, 95
median ... 18, 53, 93, 106
Merge dictionaries ... 240
Merge into target ... 150
MIDRANGE ... 56, 96
MODE ... 18, 57, 97
Modifying
Dictionary ... 236, 244
Fields ... 234
Files ... 231
Keys ... 233
Relationships ... 235
MRU List ... 243
multiple developers ... 161

N

negative ... 25
net present value ... 36, 85
NPV ... 36, 85

O

One-way Relationships ... 259
ONE:MANY ... 260
ONE:ONE ... 260
Open Area Popup Menu ... 236

P

payment of annuity ... 42
  with prepayment ... 43
PERCENTILE ... 58, 93, 98, 106
periods of annuity ... 38
  with prepayment ... 40
PERS ... 38, 86
PMT ... 42, 87
Pool
  Field ... 229
  Key ... 229
  Type ... 229
Populate File ... 236, 244
  Multiple fields ... 239
Population ... 101, 107
  standard deviation ... 64
  variance ... 68
Popup Menu
  Field ... 234
  Field Pool ... 238
  File ... 231
  Key ... 233
  Key Pool ... 241
  Open area ... 236
  Relationship ... 235
  Type Pool ... 242
positive ... 25
Preferences ... 252, 254
PREFV ... 33
PREPERS ... 40
PREPMT ... 43
PREPV ... 45
PRERATE ... 47
present value ... 36, 44, 85, 88
  with prepayment ... 45
principal ... 26, 77
Print
  Dictionary analysis report ... 252
  Dictionary text ... 251
  Fields ... 251
  File definition ... 232
  Relationships ... 235, 251
Print Menu ... 251
Print Preview ... 255, 256
Print Setup ... 243
Printing ... 255
Privileges ... 222
Procedure locking ... 189
procedure-level locking ... 161, 179
project ... 19, 20, 71, 72, 73, 75
project configuration file ... 180
Project Definition Files ... 215
Project Editor ... 20
project libraries ... 20
Project.CFG ... 182
Prompts Manager ... 247
Properties Manager ... 249
prototypes ... 19, 20, 23, 49, 71, 72, 73, 75
pth% ... 98
PV ... 44, 88

Q

Quick Add ... 239
Quick Fields dialog ... 239
Quick Reference ... 234

R

RANGEOFSET ... 59, 99
RATE ... 46, 89
rate of annuity ... 46
  with prepayment ... 47
Re-Position Files ... 237, 246
Recently edited dictionaries ... 243
Record length ... 231
Referential Integrity constraints ... 259, 260
Registering the Template Classes ... 74
Relation Names ... 255
Relation Properties ... 235
Relationship Popup Menu ... 235
Relationships ... 259
  Color ... 254
  Delete ... 231, 241
  One-way ... 259
  Two-way ... 260
Remove All Underscores ... 248
Replace ... 249
Replace target ... 150
Restore View and Zoom ... 257

return values
finance functions ... 24
revision ... 160, 179
Revision numbers ... 206
Revision Storage Options ... 195
Revisions ... 210
revisions ... 159
Rounding ... 24
Rules ... 261
RVALUE ... 60
rVALUE ... 100

S

Sample ... 101, 107
standard deviation ... 65
variance ... 69
Scale to Fit ... 236, 247
SDEVIATION ... 101
SDEVIATIONP ... 64
SDEVIATIONS ... 65
Search and Replace ... 248
Security and Team Development ... 220
Select Embed Type dialog ... 76
Select Extension dialog ... 75
Select Locks ... 210
Setting up Security in VCS ... 221
Setup menu ... 252
Show Only Files Where Used ... 239
Sign Conventions ... 25
SIMPINT ... 48, 90
simple interest ... 48, 90
SQL Script ... 243, 261
SS ... 61, 102
SSXY ... 62
SSxy ... 103
ST1 ... 63, 104
standard deviation ... 18, 101
population ... 64
sample ... 65
STATISTC.TPL ... 19, 50, 74
Statistics Class ... 73
Statistics Functions
FACTORIAL ... 51
FREQUENCY ... 52
LOWERQUARTILE ... 53
MEAN ... 54
MEDIAN ... 55
MIDRANGE ... 56
MODE ... 57
PERCENTILE ... 58
RANGEOFSET ... 59
RVALUE ... 60
SDEVIATIONP ... 64
SDEVIATIONS ... 65
SS ... 61
SSXY ... 62
ST1 ... 63
SUMM ... 66
UPPERQUARTILE ... 67
VARIANCEP ... 68
VARIANCES ... 69
Statistics Template Class ... 19
StatisticsLibrary ... 73
Student's t ... 63, 104
Subsets ... 233
sum of the squared differences ... 102
sum of squares ... 61, 102
coordinate pairs ... 62
sum of the products ... 62, 103
SUMM ... 66, 105
summation ... 66, 105
SuperUser ... 222
syntax diagram ... 16
System Requirements. *See* Wise for Clarion

T

Tag ... 239
Team Developer
configuration ... 180
Technical Support ... 230
Template Class ... 71
registering ... 74
Template File dialog ... 74
Template Registry dialog ... 74
Templates
Business Math ... 71
Test Data ... 250
time value of money ... 18, 23
Tip of the Day ... 256
tip revision ... 160
Tips ... 256
Tools ... 231
Tools menu ... 247
Tooltips ... 256
TopSpeed Version Control System ... 179
Triggers ... 262
Troubleshooting ... 225
TXD ... 243
Type Pool ... 229, 234, 239, 241
Popup menu ... 242
Type Properties ... 242

U

Underscores
  Remove from prompts 248
Undo a Check In 212
Undo a Check Out 211
Untag All 240
UPPERQUARTILE 67, 93, 98, 106
User Preferences 252, 254

V

VARIANCE 107
variance 18
  population 68
  sample 69
VARIANCEP 68
VARIANCES 69
VCS 179
VCS Administrator 181, 186
VCS Internet Server 185
VCS Reports 213
VCS Status 181, 200
VCS Utilities 218
VCSADMIN.EXE 186
VCSID 181, 191
version control 159
version control archives 179
Version Control System 179
Version Labels 179, 206
View menu 245
Visual Design Pad 229, 231
Volumetrics Calculation 249

W

Wise for Clarion 267
  Application and Watch Wizard 318
  Autoexec.bat 294
  Billboards 305
  Building Installs **275**
  Compiling Your Install 314
  Configuration Checks 297
  Customizing the Environment 320
  Dialogs, Default 301
  Dialogs, Editing 302
  Dialogs, Selecting 300
  Dialogs, Template 302
  Distributing Your Install 316
  Files and Components 281
  Floppy Install 310
  Functions Supported 268
  Getting Help 273
  Global Options 311
  Help 273
  Icons 288
  INI Entries 292
  Install, CD 310
  Install, Compiling 314
  Install, Distributing 316
  Install, Floppy Based 310
  Install, Running 315
  Install, Single File 310
  Install, Testing 314
  Installation 271
  Installation Expert, Overview **278**
  Installation Options 300
  Introduction 267
  Language Support 308
  Media Options 310
  Multi-Language Support 308
  Overview **275**
  Registering the Software 272
  Registry Entries 290
  Running Your Install 315
  Screen 304
  Screen, Background Gradient 304
  Screen, Character Set 305
  Screen, Message Box Fonts 305
  Software Title, Entering 299
  System Requirements 270
  Testing Your Install 314
  Uninstall 286
  User System Checks 297
  Wise Installation Script Template 321
  Wizard Appearance 299
workfile 160
Workfile Management 205
workfiles 159

Z

Zoom menu 247
Zoom Percentage 247