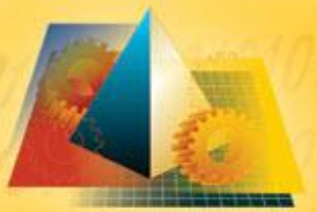


Clarion

Business Math Library

SoftVelocity Incorporated
www.softvelocity.com

Printed in the United States of America (0409)

Contents

Documentation Conventions 7

Typeface Conventions ... 7
Keyboard Conventions ... 7
Usage Conventions and Symbols ... 8
Reference Item Format ... 8
KEYWORD (short description of intended use) ... 9

Overview 11

Getting Started ... 11

About this Document 12

About the Business Math Library 12

Business Math Library Template Support ... 12
Business Math Library Hand-Code Support ... 13
Including the BUSMATH.PR file in a Project ... 13

Finance Library 15

Overview ... 15
Finance Library Components ... 15
Finance Library Conventions ... 16
Parameters ... 16
Rounding ... 16
Sign Conventions ... 17
Procedures ... 18
AMORTIZE (amortize loan for specific number of payments) ... 18
APR (annual percentage rate) ... 20
COMPINT (compound interest) ... 21
CONTINT (continuous compounding interest) ... 22
DAYS360 (days difference based on 360-day year) ... 23
FV (future value) ... 24
IRR (internal rate of return) ... 25
NPV (net present value) ... 28
PERS (periods of annuity) ... 31
PMT (payment of annuity) ... 33
PREPMT (payment of annuity with prepayment) ... 35
PREFV (future value with prepayment) ... 36
PREPERS (periods of annuity with prepayment) ... 37
PRERATE (rate of annuity with prepayment) ... 39
PREPV (present value with prepayment) ... 41
PV (present value) ... 43
RATE (rate of annuity) ... 44
SIMPINT (simple interest) ... 46

Business Statistics Library 47

Overview ... 47
Business Statistics Library Components ... 47
Business Statistics Procedures ... 48
FACTORIAL (factorial of a number) ... 48
FREQUENCY (frequency count of an item in a set) ... 49
LOWERQUARTILE (lower quartile value of a set) ... 50
MEAN (mean of a set) ... 51
MEDIAN (median of a set) ... 52
MIDRANGE (midrange of a set) ... 53
MODE (mode of a set) ... 54

PERCENTILE (pth percentile value of a set) ... 55
RANGEOFSET (range of a set) ... 56
RVALUE (linear regression correlation coefficient) ... 57
SS (sum of squares) ... 58
SSXY (sum of squares for x and y) ... 59
ST1 (student's t for a single mean) ... 60
SDEVIATIONP (standard deviation of a population) ... 61
SDEVIATIONS (standard deviation of a sample) ... 62
SUMM (summation of a set) ... 63
UPPERQUARTILE (upper quartile value of a set) ... 64
VARIANCEP (variance of a population) ... 65
VARIANCES (variance of a sample) ... 66

Business Math Templates 67

Overview ... 67
Template Components ... 67
Class Finance ... 68
Class Statistics ... 69
Registering the Template Classes ... 70
Adding Extension Templates to Your Application ... 71
Embedding Code Templates in Your Application ... 71
Finance Code Templates ... 72
AMORTIZE ... 72
APR ... 73
COMPINT ... 73
CONTINT ... 74
DAYS360 ... 74
FV ... 75
IRR ... 75
NPV ... 76
PERS ... 76
PMT ... 77
PV ... 77
RATE ... 78
SIMPINT ... 78
Business Statistics Code Templates ... 79
FACTORIAL ... 79
FREQUENCY ... 79
LOWERQUARTILE ... 80
MEAN ... 80
MEDIAN ... 81
MIDRANGE ... 81
MODE ... 82
PERCENTILE ... 82
RANGEOFSET ... 83
RVALUE ... 83
SDEVIATION ... 84
SS ... 84
SSxy ... 85
ST1 ... 85
SUMM ... 86
UPPERQUARTILE ... 86
VARIANCE ... 87

Business Math Example Application **89**

Overview 89
Exploring the Example Application 89
Finance 89
Statistics 89

Index: **91**

Documentation Conventions

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

Typeface Conventions

Italics	Indicates what to type at the keyboard, such as *Type This*. It also indicates the text of a window's title bar.
ALL CAPS	Indicates keystrokes to enter at the keyboard, such as ENTER or ESCAPE, and also indicates Clarion Language keywords. For more information on these keywords, see the *Language Reference* PDF.
Boldface	Indicates commands or options from a pull down menu or text in a dialog window.
`COURIER NEW`	Used for diagrams, source code listings, to annotate examples, and for examples of the usage of source statements.

These graphics indicate information that is not immediately evident from the topic explanation.

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 "!"
```

Overview

Welcome to the **Clarion Business Math Library** for Clarion 6. It contains common financial and statistical functions used in business applications. You can call these functions from your Clarion programs, from either embedded source, or by using the Code Templates.

Clarion has always had a special affinity for business applications--after all, the language was designed from the ground up for business developers. As a development environment, the one feature that's often drawn the most attention in this area is our Binary Coded Decimal math support. By automatically using integer math for currency calculations, we've insured an accuracy that other tools, which require floating point conversions, can never achieve. We've always focused on tools and methods for helping the business developer achieve higher productivity. It stands to reason that the first module to extend the language for Clarion should be business oriented

You'll save time by calling a single function that you'd otherwise have to laboriously hand code, operation by operation. How much time--you do the math!

Getting Started

The installation has registered two template classes: FINANCE.TPL and STATISTC.TPL. Each of these contains an extension template that enables its associated code templates.

If you want financial functions in an application, add the Global Extension Template for the Finance Template Class and use any of the Finance functions in your application (either though source or a code template). If you want business statistics functions in an application, add the Global Extension Template for the Statistics Template Class and use any of the Statistics functions in your application.

See *Adding Extension Templates to Your Application* and *Embedding Code Templates in Your Application* for details.

The *Example Application* demonstrates some of the many uses of these functions.

About this Document

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 and functions 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 functions which 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 that make the business math functions very 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 functions and procedures in the Business Math Library.

Business Math Library Template Support

Complete Code template support for all of the Business Math procedures and functions is provided. The Code templates self-document the use of the library operations. Also, global Extension Templates automate the process of adding required MAP and Project System entries to an application. Once the appropriate extension template is added, all the code templates are available for use in any embed point.

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

By registering the Business Math Library's Template files, the associated templates are available to all of your applications.

Business Math Library Hand-Code Support

While the #EXTENSION templates automatically handle project system and prototype entries for your Application Generator projects, you must handle these entries for hand-coded Clarion programs. First, include appropriate procedure and function 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 and functions are in the CWFINPRO.CLW file. The prototypes for the Business Statistics Library functions 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
```

If you are using Statistics functions also include the data definition file after the MAP...END:

```
INCLUDE('CWSTATDT.CLW')    ! include Business Statistics Library Defines
```

The Finance Library file set contains the following files:

C70FIN.LIB	Standalone (Deploy with C70FIN.DLL)
C70FINL.LIB	Local Link

The Business Statistics Library file set contains the following files:

C70STAT.LIB	Standalone (Deploy with C70STAT.DLL)
C70STATL.LIB	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.

Finance Library

Overview

This section 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 and function in the Finance Library.

The Finance Library contains procedures and functions which 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 functions into your application or project.

The Finance Library components (files) are:

...\LIB\C70FINL.LIB
Finance objects that link into your executable.

...\BIN\C70FIN.DLL
Finance procedures and functions.

...\LIB\C70FIN.LIB
Finance stub file for resolving link references to functions in C70FIN.DLL.

...\TEMPLATE\WIN\FINANCE.TPL
Finance Library templates. The templates self-document the use of the library functions and procedures. The templates must be registered before they can be used in your application.

...\LIBSRC\WIN\CWFINPRO.CLW
Prototypes for the Finance procedures and functions.

...\LIBSRC\WIN\BUSMATH.PR
Automatically includes the correct business math libraries in your project. In a hand-coded project, insert it as a *Project to include*. In an .APP, the Global Extension Templates automatically include this file in your project.. The BUSMATH.PR project includes the appropriate Business Math Libraries based on the parent project's Target OS and Run-Time Library settings.

Finance Library Conventions

Parameters

Even though most of the Finance function and procedure parameters are declared as data types other than DECIMAL (in 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 function *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 functions and 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 function.
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 functions, the amortization procedure, and the cash flow analysis functions 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 functions and 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).
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).
interest	The label of a DECIMAL variable to receive the portion of the payment(s) applied towards loan interest (a negative number).
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** function 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:

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

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** function 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:

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

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 function, 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:

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

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 function 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:

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

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)

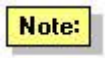
PERS(*presentvalue,rate,payment,futurevalue*)

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 function, which takes into account the added interest earned on each period's payment.

Periodic rate may be calculated as follows:

PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)



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:

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

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

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

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

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

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

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

The PERS function 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 function, 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 function, 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:

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

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
```

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 function, 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:

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

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
```

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 function, 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:

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

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

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

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

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

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

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

The PREPERS function 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
```

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 function, which calculates interest accordingly.

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:

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

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

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

The PRERATE function 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
```

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** function, which calculates interest accordingly.

Periodic rate may be calculated as follows:

PeriodicRate = AnnualInterestRate / (PeriodsPerYear * 100)

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:

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

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
```

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** function, 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
```

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 function, which takes into account the added interest earned on each period's payment.

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:

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

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

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

The RATE function 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
```

Business Statistics Library

Overview

This section 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 function in the Business Statistics Library.

The Business Statistics Library contains functions 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 functions into your application or project.

The Business Statistics Library components (files) are:

...\LIB\C70STATxL.LIB
Business Statistics objects that link into your executable.

...\BIN\C70STATx.DLL
Business Statistics functions.

...\LIB\C70STATx.LIB
Business Statistics stub file for resolving link references to functions in C70STAT.DLL.

...\TEMPLATE\WIN\STATISTIC.TPL
Business Statistics Library templates. The templates self-document the use of the library functions. The templates must be registered before they can be used in your application.

...\LIBSRC\CWSTATPR.CLW
Prototypes for the Business Statistics functions.

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

...\LIBSRC\BUSMATH.PR
Automatically includes the correct business math libraries in your project. In a hand-coded project, insert it as a *Project to include*. In an .APP, the Global Extension Templates automatically include this file in your project. The BUSMATH.PR project includes the appropriate Business Math Libraries based on the parent project's Target OS and Run-Time Library settings.

Business Statistics 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** function implements the standard factorial formula. For example, if the *number* provided is 5 then the function 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 function 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 function would return a frequency count of 2.

NOTE:

The passed data set does not have to be sorted. The function 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 function 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 function is only meaningful when the number of elements in the data set is large (ie. greater than 20).

NOTE
The passed data set does not have to be sorted. The function 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 function 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 function 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 function 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.

The passed data set does not have to be sorted. The function 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 function 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 function 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 function returns the number of occurrences (mode count).

The function 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 function. 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 function 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 function 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 function is only meaningful when the number of elements in the data set is large (ie. greater than 20).

The passed data set does not have to be sorted. The function 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 function 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.

The passed data set does not have to be sorted. The function 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 function 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 function. If not provided, the value(s) are computed internally by the function. The passed data set does not have to be sorted. The function 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 function 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 function. If not provided, the mean value is computed internally by the function.

The passed data set does not have to be sorted. The function 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 function 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 function. If not provided, the value(s) will be computed internally by the function. The passed data set does not have to be sorted. The function 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 function 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.

The passed data set does not have to be sorted. The function 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' function call should be used only if the data set contains all of a population's values. The function 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 function. If not provided, the value is computed internally by the function.

The passed data set does not have to be sorted. The function 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' function call should be used only if the data set contains less than all of a population's values. The function 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 function. If not provided, the value is computed internally by the function.

The passed data set does not have to be sorted. The function 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 function 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 function 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 function 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 function is only meaningful when the number of elements in the data set is large (ie. greater than 20).

NOTE:
The passed data set does not have to be sorted. The function 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' function call should be used only if the data set contains all of a population's values. The function 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 function. If not provided, the value is computed internally by the function.

NOTE:
The passed data set does not have to be sorted. The function 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' function call should be used only if the data set contains less than all of a population's values. Both functions 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 function. If not provided, the value is computed internally by the function.

NOTE
The passed data set does not have to be sorted. The function 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
```

Business Math Templates

Overview

This section 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 functions into your application by:

- Adding the appropriate procedure and function 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 function.
- Providing a self-documenting method of calling each function 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 Template

FinanceLibrary	Enables access to the associated Code templates from any embed point throughout the application. It also adds the appropriate procedure and function 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 Template

StatisticsLibrary	Enables access to the associated Code templates from any embed point throughout the application. It also adds the appropriate procedure and function 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 **Tools > Edit Template Registry** from the Clarion Environment menu bar.
2. In the *Template Registry* dialog, press the **Register** button.
3. In the *Template File* dialog, select the FINANCE.TPL file in the ..\TEMPLATE\WIN 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\WIN subdirectory, then press **OK.**
6. In the *Template Registry* dialog, press the **Close** button.

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 and function 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** icon 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 (e.g., 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 (e.g., 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.

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.

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 function. 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 function. 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 function. 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 function. 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 (future value) function or the PREFV function. Both functions 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 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 function. 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 function. 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 function or the PREPERS function. Both functions 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 function or the PREPMT function. Both functions 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 (present value) function or the PREPV function. Both functions 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 function or the PRERATE function. Both functions 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 function. 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.

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 function. FACTORIAL implements the standard factorial formula. For example, if the number provided is 5 then the function 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 function. 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 function. 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 function is only meaningful when the number of elements in the data set is large (i.e., 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)
```

The passed data set does not have to be sorted. The function copies the passed set. The passed data set is unchanged.

MEAN

This Code template generates code to call the MEAN function. 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)
```

The passed data set does not have to be sorted. The function copies the passed set. The passed data set is unchanged.

MEDIAN

This Code template generates code to call the MEDIAN function. 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)
```

The passed data set does not have to be sorted. The function copies the passed set. The passed data set is unchanged.

MIDRANGE

This Code template generates code to call the MIDRANGE function. 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)
```

The passed data set does not have to be sorted. The function copies the passed set. The passed data set is unchanged.

MODE

This Code template generates code to call the MODE function. 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)
```

The passed data set does not have to be sorted. The function copies the passed set. The passed data set is unchanged.

PERCENTILE

This Code template generates code to call the PERCENTILE function. 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)
```

The passed data set does not have to be sorted. The function copies the passed set. The passed data set is unchanged.

RANGEOFSET

This Code template generates code to call the RANGEOFSET function. 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)
```

The passed data set does not have to be sorted. The function copies the passed set. The passed data set is unchanged.

RVALUE

This Code template generates code to call the RVALUE function. 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)
```

The passed data set does not have to be sorted. The function copies the passed set. The passed data set is unchanged.

SDEVIATION

This Code template generates code to call the SDEVIATIONP function or the SDEVIATIONS function. 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 Checking this box generates a DISPLAY statement for the return value.

Example code generated by the SDEVIATION Code template:

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

The passed data set does not have to be sorted. The function copies the passed set. The passed data set is unchanged.

SS

This Code template generates code to call the SS (sum of squares) function. 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)
```

The passed data set does not have to be sorted. The function copies the passed set. The passed data set is unchanged.

SSxy

This Code template generates code to call the SSXY function. 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)
```

The passed data set does not have to be sorted. The function copies the passed set. The passed data set is unchanged.

ST1

This Code template generates code to call the ST1 function. 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)
```

The passed data set does not have to be sorted. The function copies the passed set. The passed data set is unchanged.

SUMM

This Code template generates code to call the SUMM function. 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)
```

The passed data set does not have to be sorted. The function copies the passed set. The passed data set is unchanged.

UPPERQUARTILE

This Code template generates code to call the UPPERQUARTILE function. 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 function 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)
```

The passed data set does not have to be sorted. The function copies the passed set. The passed data set is unchanged.

VARIANCE

This Code template generates code to call the VARIANCEP function or the VARIANCES function. 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)
```

The passed data set does not have to be sorted. The function copies the passed set. The passed data set is unchanged.

Business Math Example Application

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 functions. 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 ..\Examples\BUSMATH. Files comprising the example are:

BUSMATH.APP	Application File
BUSMATH.DCT	Dictionary File
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 functions:

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 function.

Cash Flow Analysis (BrowseInvestments)
This procedure 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 NPV (Net Present Value) and IRR (Internal Rate of Return) functions.

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 functions 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 functions.

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 functions.

Index:

About the Business Math Library 12
Adding Extension Templates to Your Application 71
AMORTIZE Code Template 72
APR 20
APR Code Template 73
Business Math Library. 20, 21, 22, 23, 24, 25, 28, 31, 33, 35, 36, 37, 39, 41, 43, 44, 46
Business Math Library Hand-Code Support 13
Business Math Library Template Support 12
Business Math Templates--Overview 67
Business Statistics Code Templates 79
Business Statistics Library Components 47
Business Statistics Overview 47
COMPINT 21
COMPINT Code Template 73
CONTINT 22
CONTINT Code Template 74
DAYS360 23
DAYS360 Code Template 74
Embedding Code Templates in Your Application 71
Example Application 89
FACTORIAL 48
FACTORIAL Code Template 79
Finance Code Templates 72
Finance Library Components 15
Finance Library Conventions 16
Finance Library Overview 15
FREQUENCY 49
FREQUENCY Code Template 79
FV 24
FV Code Template 75
IRR 25
IRR Code Template 75
LOWERQUARTILE 50
LOWERQUARTILE Code Template 80
MEAN 51
MEAN Code Template 80
MEDIAN 52
MEDIAN Code Template 81
MIDRANGE 53
MIDRANGE Code Template 81
MODE 54
MODE Code Template 82
NPV 28
NPV Code Template 76
PERCENTILE 55
PERCENTILE Code Template 82
PERS 31
PERS Code Template 76
PMT 33
PMT Code Template 77
PREFV 36
PREPERS 37
PREPMT 35
PREPV 41
PRERATE 39
PV 43
PV Code Template 77
RANGEOFSET 56
RANGEOFSET Code Template 83
RATE 44
RATE Code Template 78
Registering the Template Classes 70
RVALUE 57
rVALUE Code Template 83
SDEVIATION Code Template 84
SDEVIATIONP 61
SDEVIATIONS 62
SIMPINT 46
SIMPINT Code Template 78
SS 58
SS Code Template 84
SSXY 59
SSxy Code Template 85
ST1 60
ST1 Code Template 85
SUMM 63
SUMM Code Template 86
Template Components 67
UPPERQUARTILE 64
UPPERQUARTILE Code Template 86
VARIANCE Code Template 87
VARIANCEP 65
VARIANCES 66