

CLARION 5

**Business
Math Library**

COPYRIGHT 1998 by TopSpeed Corporation
All rights reserved.

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

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

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

Trademark Acknowledgments:

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

CONTENTS

1 - INTRODUCTION	7
Documentation Conventions	7
Typeface Conventions	7
Keyboard Conventions	7
Usage Conventions and Symbols	8
Reference Item Format	8
About this Manual	10
About the Business Math Library	10
Installing the Business Math Library	11
Business Math Library Template Support	11
Business Math Library Hand-Code Support	12
2 - FINANCE LIBRARY	15
Overview	15
Finance Library Components	15
Finance Library Conventions	16
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
PREFV (future value with prepayment)	25
IRR (internal rate of return)	26
NPV (net present value)	28
PERS (periods of annuity)	30
PREPERS (periods of annuity with prepayment)	32
PMT (payment of annuity)	34
PREPMT (payment of annuity with prepayment)	35
PV (present value)	36
PREPV (present value with prepayment)	37
RATE (rate of annuity)	38
PRERATE (rate of annuity with prepayment)	39
SIMPINT (simple interest)	40

3 - BUSINESS STATISTICS LIBRARY	41
Overview	41
Business Statistics Library Components	41
Procedures	43
FACTORIAL (factorial of a number)	43
FREQUENCY (frequency count of an item in a set)	44
LOWERQUARTILE (lower quartile value of a set)	45
MEAN (mean of a set)	46
MEDIAN (median of a set)	47
MIDRANGE (midrange of a set)	48
MODE (mode of a set)	49
PERCENTILE (pth percentile value of a set)	50
RANGEOFFSET (range of a set)	51
RVALUE (linear regression correlation coefficient)	52
SS (sum of squares)	53
SSXY (sum of squares for x and y)	54
ST1 (student's t for a single mean)	55
SDEVIATIONP (standard deviation of a population)	56
SDEVIATIONS (standard deviation of a sample)	57
SUMM (summation of a set)	58
UPPERQUARTILE (upper quartile value of a set)	59
VARIANCEP (variance of a population)	60
VARIANCES (variance of a sample)	61
4 - BUSINESS MATH TEMPLATES	63
Overview	63
Template Components	63
Registering the Template Classes	66
Adding Extension Templates to Your Application	67
Embedding Code Templates in Your Application	68
Finance Code Templates	69
AMORTIZE	69
APR	71
COMPINT	72
CONTINT	73
DAYS360	74
FV	75

IRR	76
NPV	77
PERS	78
PMT	79
PV	80
RATE	81
SIMPINT	82
Business Statistics Code Templates	83
FACTORIAL	83
FREQUENCY	84
LOWERQUARTILE	85
MEAN	86
MEDIAN	87
MIDRANGE	88
MODE	89
PERCENTILE	90
RANGEOFFSET	91
rVALUE	92
SDEVIATION	93
SS	94
SSxy	95
ST1	96
SUMM	97
UPPERQUARTILE	98
VARIANCE	99
5 - BUSINESS MATH EXAMPLES	101
Overview	101
Exploring the Example Application	101
INDEX	103

1 - INTRODUCTION

Documentation Conventions

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

Typeface Conventions

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

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



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

Keyboard Conventions

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

Usage Conventions and Symbols

Symbols are used in the syntax diagrams as follows:

<u>Symbol</u>	<u>Meaning</u>
[]	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(<div><div>parameter1</div><div>alternate</div><div>parameter</div><div>list</div></div>	[parameter2])
---------	----------	---	------------------

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:	<div><div>FieldOne = FieldTwo + FieldThree</div><div>FieldThree = KEYWORD(FieldOne,FieldTwo)</div></div> <div><div>!Source code example</div><div>!Comments follow the “!”</div></div>

About this Manual

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

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

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

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

About the Business Math Library

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

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

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

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

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

Installing the Business Math Library

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

Business Math Library Template Support

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

Registering the Template Classes

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

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

Adding Extension Templates to Your Application



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

Ship List DLLs

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

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

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

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

Business Math Library Hand-Code Support

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

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

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

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

The Finance Library file set contains the following files:

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

The Business Statistics Library file set contains the following files:

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

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

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

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

Including the BUSMATH.PR file in a Project

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

2 - FINANCE LIBRARY

Overview

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

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

Finance Library Components

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

The Finance Library components (files) are:

...\LIB\C5FINL.LIB	16-bit Finance objects that link into your executable.
...\BIN\C5FIN.DLL	16-bit Finance procedures.
...\LIB\C5FIN.LIB	16-bit Finance stub file for resolving link references to procedures in C5FIN.DLL.
...\LIB\C5FINxL.LIB	32-bit Finance objects that link into your executable.
...\BIN\C5FINx.DLL	32-bit Finance procedures.
...\LIB\C5FINx.LIB	32-bit Finance stub file for resolving link references to procedures in C5FINx.DLL.

...\TEMPLATE\FINANCE.TPL

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

...\LIBSRC\CWFINPRO.CLW

Prototypes for the Finance procedures.

Finance Library Conventions

Parameters

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

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

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

Rounding

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

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

Sign Conventions

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

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

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

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

Procedures

AMORTIZE (amortize loan for specific number of payments)

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

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

$$\text{PeriodicRate} = \text{AnnualInterestRate} / (\text{PeriodsPerYear} * 100)$$

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

Internal Formulas:

$$\begin{aligned}\text{PRINCIPAL} &= \text{payment} + (\text{balance} * \text{rate}) \\ \text{INTEREST} &= \text{payment} - \text{principal} \\ \text{ENDINGBALANCE} &= \text{balance} + \text{principal}\end{aligned}$$

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 downward
        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(AmortizeQueue)                   !Add all period values to Q
    BeginningBalance = EndingBalance      !Make a new beginning balance
    TotalInterest += Q:Interest           !Add up total interest
END                                       !End loop
```

APR (annual percentage rate)

APR(*rate*,*periods*)

APR

Returns the effective annual interest rate.

 $rate$

A numeric constant or variable containing the *contracted* interest rate.

periods

A numeric constant or variable containing the number of compounding periods per year.

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

Return DataType: **DECIMAL**

Internal Formulas:

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

Example:

[illegible]

COMPINT (compound interest)

COMPINT(*principal*,*rate*,*periods*)

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

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

Applied interest rate may be calculated as follows:

$$\begin{aligned}\text{PeriodicRate} &= \text{AnnualInterestRate} / (\text{PeriodsPerYear} * 100) \\ \text{AppliedRate} &= \text{PeriodicRate} * \text{TotalPeriods}\end{aligned}$$

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.
<i>principal</i>	A numeric constant or variable containing the beginning balance, initial deposit, or loan.
<i>rate</i>	A numeric constant or variable containing the <i>applied interest rate</i> 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:

$$\begin{aligned}\text{PeriodicRate} &= \text{AnnualInterestRate} / (\text{PeriodsPerYear} * 100) \\ \text{AppliedRate} &= \text{PeriodicRate} * \text{TotalPeriods}\end{aligned}$$

Return Data Type: **DECIMAL**

Internal Formulas:

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

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

Example:

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

DAYS360 (days difference based on 360-day year)

DAYS360(*startdate*,*enddate*)

DAYS360

Computes the difference in days, between two given dates.

startdate

A numeric constant or variable containing the beginning date.

enddate

A numeric constant or variable containing the ending date.

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

Return Data Type: **LONG**

Internal Formulas:

DAYS DIFFERENCE = ending date - beginning date

where:

ending date = 360(year) + 30(month) + z

z = 30 if ending date = 31 and beginning date > 29

z = ending date if ending date <> 31 and beginning date < 29

and:

beginning date = 360(year) + 30(month) + z

z = 30 if beginning date = 31

z = beginning date if beginning date <> 31

Example:

DaysDifference = DAYS360(StartDate,EndDate)

FV (future value)

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

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

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

Periodic rate may be calculated as follows:

$$\text{PeriodicRate} = \text{AnnualInterestRate} / (\text{PeriodsPerYear} * 100)$$

Return Data Type: **DECIMAL**

Internal Formulas:

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

where $\text{int}(\text{periods})$ is the integer portion of the *periods* parameter.

Example:

```

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

```


PREFV (future value with prepayment)

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

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

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

Periodic rate may be calculated as follows:

$$\text{PeriodicRate} = \text{AnnualInterestRate} / (\text{PeriodsPerYear} * 100)$$

Return Data Type: **DECIMAL**

Internal Formulas:

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

where $\text{int}(\text{periods})$ 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.
<i>investment</i>	A numeric constant or variable containing the initial cost of the investment (a negative number).
<i>cashflows</i> []	A single dimensioned DECIMAL array containing the amounts of money paid out and received during discrete accounting periods.
<i>periods</i> []	A single dimensioned DECIMAL array containing period numbers identifying the discrete accounting periods in which cash flows occurred.
<i>rate</i>	A numeric constant or variable containing the desired <i>periodic</i> 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:

$$\text{PeriodicRate} = \text{AnnualInterestRate} / (\text{PeriodsPerYear} * 100)$$

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:

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

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

Example:

```

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

InvestmentTransactions  FILE,DRIER('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.
<i>investment</i>	A numeric constant or variable containing the initial cost of the investment (a negative number).
<i>cashflows</i> []	A single dimensioned DECIMAL array containing the amounts of money paid out and received during discrete accounting periods.
<i>periods</i> []	A single dimensioned DECIMAL array containing period numbers identifying the discrete accounting periods in which cash flows occurred.
<i>rate</i>	A numeric constant or variable containing the desired <i>periodic</i> 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:

$$\text{PeriodicRate} = \text{AnnualInterestRate} / (\text{PeriodsPerYear} * 100)$$

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:

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

Example:

```

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

InvestmentTransactions  FILE,DRIER('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.
<i>presentvalue</i>	A numeric constant or variable containing the present value of the investment.
<i>rate</i>	A numeric constant or variable containing the <i>periodic</i> rate of return.
<i>payment</i>	A numeric constant or variable containing the periodic payment.
<i>futurevalue</i>	A numeric constant or variable containing the amount of the desired or targeted future value of the investment.

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

Periodic rate may be calculated as follows:

$$\text{PeriodicRate} = \text{AnnualInterestRate} / (\text{PeriodsPerYear} * 100)$$

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

Return Data Type: DECIMAL

Internal Formulas:

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

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

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

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

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

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

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

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

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

Example:

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

PREPERS (periods of annuity with prepayment)

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

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

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

Periodic rate may be calculated as follows:

$$\text{PeriodicRate} = \text{AnnualInterestRate} / (\text{PeriodsPerYear} * 100)$$

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

Return Data Type: DECIMAL

Internal Formulas:

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

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

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

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

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

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

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

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

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

Example:

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

PMT (payment of annuity)

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

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

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

Periodic rate may be calculated as follows:

$$\text{PeriodicRate} = \text{AnnualInterestRate} / (\text{PeriodsPerYear} * 100)$$

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

Return Data Type: DECIMAL

Internal Formulas:

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

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

Example:

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

PREPMT (payment of annuity with prepayment)

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

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

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

Periodic rate may be calculated as follows:

$$\text{PeriodicRate} = \text{AnnualInterestRate} / (\text{PeriodsPerYear} * 100)$$

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

Return Data Type: DECIMAL

Internal Formulas:

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

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

Example:

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

PV (present value)

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

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

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

Periodic rate may be calculated as follows:

$$\text{PeriodicRate} = \text{AnnualInterestRate} / (\text{PeriodsPerYear} * 100)$$

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:

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

where $\text{frac}(\text{periods})$ is the fractional portion of the *periods* parameter.

Example:

```

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

```

PREPV (present value with prepayment)

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

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

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

Periodic rate may be calculated as follows:

$$\text{PeriodicRate} = \text{AnnualInterestRate} / (\text{PeriodsPerYear} * 100)$$

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

Return Data Type: DECIMAL

Internal Formulas:

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

where $\text{frac}(\text{periods})$ 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.
<i>presentvalue</i>	A numeric constant or variable containing the present value of the investment.
<i>periods</i>	A numeric constant or variable containing the number of periods in which a cash flow occurred.
<i>payment</i>	A numeric constant or variable containing the periodic payment amount.
<i>futurevalue</i>	A numeric constant or variable containing the amount of the desired or targeted future value of the investment.

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

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

Return Data Type: DECIMAL

Internal Formulas:

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

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

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

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

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

Example:

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

PRERATE (rate of annuity with prepayment)

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

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

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

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

Return Data Type: DECIMAL

Internal Formulas:

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

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

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

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

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

Example:

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

SIMPINT (simple interest)

SIMPINT(*principal*,*rate*)

SIMPINT

Returns simple (uncompounded) interest.

principal

A numeric constant or variable containing the beginning balance, initial deposit, or loan.

rate

A numeric constant or variable containing the simple or contract interest rate.

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

Return Data Type: **DECIMAL**

Internal Formulas:

$\text{SIMPLE INTEREST} = \text{principal} * \text{rate}$

Example:

```

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


3 - ***BUSINESS STATISTICS LIBRARY***

Overview

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

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

Business Statistics Library Components

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

The Business Statistics Library components (files) are:

...\LIB\C5STATL.LIB

16-bit Business Statistics objects that link into your executable.

...\BIN\C5STAT.DLL

16-bit Business Statistics procedures.

...\LIB\C5STAT.LIB

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

...\LIB\C5STATxL.LIB

32-bit Business Statistics objects that link into your executable.

...\BIN\C5STATx.DLL

32-bit Business Statistics procedures.

...\LIB\C5STATx.LIB

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

...\TEMPLATE\STATISTC.TPL

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

...\LIBSRC\CWSTATPR.CLW

Prototypes for the Business Statistics procedures.

...\LIBSRC\CWSTATDT.CLW

TYPE definitions for QUEUES passed to Business Statistics procedures.

Procedures

FACTORIAL (factorial of a number)

FACTORIAL(*number*)

FACTORIAL

Computes the factorial of a number.

number

A numeric constant or variable containing a positive integer value.

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

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.
<i>dataset</i>	The label of a QUEUE with its first component field defined as a REAL.
<i>searchvalue</i>	A numeric constant or variable containing the value to search for in the QUEUE.
FREQUENCY counts the number of instances of a particular value (<i>searchvalue</i>) within a numeric set (<i>dataset</i>). The procedure operates on the numeric set defined by all the entries in the first component of the designated QUEUE (<i>dataset</i>).	

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

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

Return DataType: REAL

Example:

```

StatSetX   QUEUE,PRE()
X          REAL
          END

FrequencyOfX  REAL
SearchValue   REAL(1)

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

```

LOWERQUARTILE (lower quartile value of a set)

LOWERQUARTILE(*dataset*)

LOWERQUARTILE

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

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

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

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

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

Return DataType: **REAL**

Example:

```

StatSetX    QUEUE,PRE()
X            REAL
            END

LowerQuartileOfSet    REAL

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

```

MEAN (mean of a set)

MEAN(*dataset*)

MEAN

Returns the mean or average of a set of numbers.

dataset

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

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

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

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

Return DataType: REAL

Example:

```

StatSetX   QUEUE,PRE()
X          REAL
          END

MeanOfSet   REAL

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

```

MEDIAN (median of a set)

MEDIAN(*dataset*)

MEDIAN

Returns the middle value of an ordered numeric data set.

dataset

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

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

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

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

Return DataType: REAL

Example:

```

StatSetX  QUEUE,PRE()
X          REAL
          END

MedianOfSet REAL

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

```

MIDRANGE (midrange of a set)

MIDRANGE(*dataset*)

MIDRANGE Returns the average of the highest and lowest value in a numeric set.

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

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

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

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

Return DataType: REAL

Example:

```

StatSetX      QUEUE,PRE()
X              REAL
              END

MidrangeOfSet  REAL

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

```


MODE (mode of a set)

MODE(*dataset, modeset*)

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

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

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

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

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

Return DataType: REAL

Example:

```

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

```

PERCENTILE (pth percentile value of a set)

PERCENTILE(*dataset*,*percentile*)

PERCENTILE Returns a value that divides an ordered numeric data set into two portions of specified relative size.

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

percentile A numeric constant or variable containing an integer value in the range: $1 \geq p \leq 99$.

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

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

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

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

Return DataType: REAL

Example:

```

StatSetX      QUEUE,PRE()
X              REAL
              END

PercentileOfX  REAL
DividePoint    REAL(90)

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

```

RANGEOFSET (range of a set)

RANGEOFSET(*dataset*)

RANGEOFSET Returns the difference between the largest and smallest values in a numeric set.

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

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

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

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

Return DataType: REAL

Example:

```

StatSetX      QUEUE,PRE()
X              REAL
              END

RangeOfSetX    REAL

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

```

RVALUE (linear regression correlation coefficient)

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

RVALUE	Returns the correlation coefficient of the linear relationship between the paired data points in the data set.
<i>dataset</i>	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.
<i>meanX</i>	A numeric constant or variable containing the average of the X values (optional).
<i>meanY</i>	A numeric constant or variable containing the average of the Y values (optional).

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

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

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

Return DataType: **REAL**

Example:

```

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

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

```

SS (sum of squares)

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

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

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

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

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

Return DataType: REAL

Example:

```

StatSetX      QUEUE,PRE()
X              REAL
              END

SumOFSquaresX  REAL

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

```

SSXY (sum of squares for x and y)

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

SSXY	Returns the sum of the products of the differences between each data point and its associated (either X or Y) mean value.
<i>dataset</i>	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.
<i>meanX</i>	A numeric constant or variable containing the average of the X values (optional).
<i>meanY</i>	A numeric constant or variable containing the average of the Y values (optional).

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

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

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

Return DataType: **REAL**

Example:

```

StatSetXY      QUEUE,PRE()
X              REAL
Y              REAL
              END
SumOfSquares   REAL
CODE
  STADAT:Id = STA:Id                      !prime the record buffer
  STADAT:ItemNumber = 1
  SET(STADAT:KeyIdItemNumber,STADAT:KeyIdItemNumber !position file pointer
  LOOP                                              !load the QUEUE
    NEXT(StatisticsData)                        !read next record
    IF ERRORCODE() OR STADAT:Id NOT = STA:Id
      BREAK
    ELSE
      StatSetXY:X = STADAT:X                  !load the QUEUE buffer
      StatSetXY:Y = STADAT:Y                  !load the QUEUE buffer
      ADD(StatSetXY)                          !add to the QUEUE
    END
  END
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.
<i>dataset</i>	The label of a QUEUE with its first component field defined as a REAL.
<i>hypothesizedmean</i>	A numeric constant or variable representing a hypothesized mean value associated with the statistical test.

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

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

Return DataType: **REAL**

Example:

```

StatSetXY      QUEUE,PRE()
X              REAL
              END
TValue         REAL

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

```

SDEVIATIONP (standard deviation of a population)

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

SDEVIATIONP Returns the standard deviation of the entire population of a numeric set.

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

meanofset A numeric constant or variable representing the average of the data set's values (optional).

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

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

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

Return DataType: REAL

Example:

```

StatSetX      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
    NEXT(StatisticsData)        !load the QUEUE
                                !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.
<i>dataset</i>	The label of a QUEUE with its first component field defined as a REAL.
<i>meanofset</i>	A numeric constant or variable representing the average of the data set's values.

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

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

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

Return DataType: REAL

Example:

```

StatSetXY      QUEUE,PRE()
X              REAL
              END
StandardDeviation REAL

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

```

SUMM (summation of a set)

SUMM(*dataset*)

SUMM

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

dataset

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

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

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

Return DataType: **REAL**

Example:

```

StatSetX          QUEUE,PRE()
X                 REAL
                  END
SummationOfSet     REAL

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

```

UPPERQUARTILE (upper quartile value of a set)

UPPERQUARTILE(*dataset*)

UPPERQUARTILE

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

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

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

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

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

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

Return DataType: REAL

Example:

```

StatSetXY      QUEUE,PRE()
X              REAL
              END
UpperQuartileOfSet  REAL

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

```

VARIANCEP (variance of a population)

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

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

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

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

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

Return DataType: **REAL**

Example:

```

StatSetXY          QUEUE,PRE()
X                  REAL
                  END
VarianceOfSet      REAL

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

```

VARIANCES (variance of a sample)

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

VARIANCES

Returns the variance of a sample of a numeric set.

dataset

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

meanofset

A numeric constant or variable representing the average of the data set's values (optional).

VARIANCES computes the variance of a given sample data set.

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

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

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

Return DataType: REAL

Example:

```

StatSetXY      QUEUE,PRE()
X              REAL
              END
VarianceOfSet  REAL

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

```


4 - *BUSINESS MATH TEMPLATES*

Overview

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

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

Template Components

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

Class Finance**#EXTENSION Templates**

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

#CODE Templates

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

Class Statistics**#EXTENSION Templates**

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

#CODE Templates

FACTORIAL Factorial of a Number

FREQUENCY Frequency Count of an Item in a Set

LOWERQUARTILE
Lower Quartile Value of a Set

MEAN Mean of a Set

MEDIAN Median of a Set

MIDRANGE Midrange of a Set

MODE Mode of a Set

PERCENTILE pth Percentile Value of a Set

RANGEOFSET Range of a Set

rVALUE Linear Regression Correlation Coefficient

SS Sum of Squares

SSxy Sum of Squares for X and Y

ST1 Student's t (single mean)

SDEVIATION Standard Deviation of a Set

SUMM Summation of a Set

UPPERQUARTILE Upper Quartile Value of a Set

VARIANCE Variance of a Set

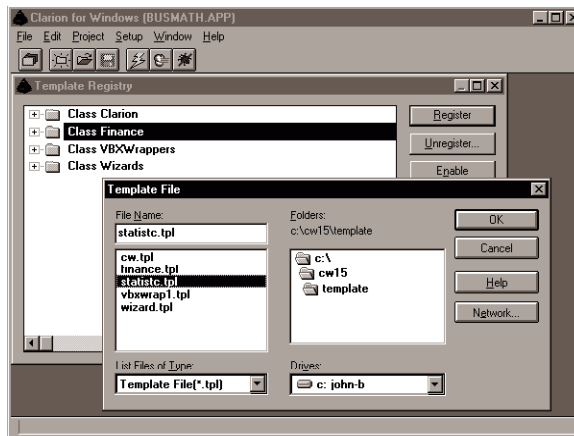
Registering the Template Classes

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

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

❑ *To register the business math template classes:*

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



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

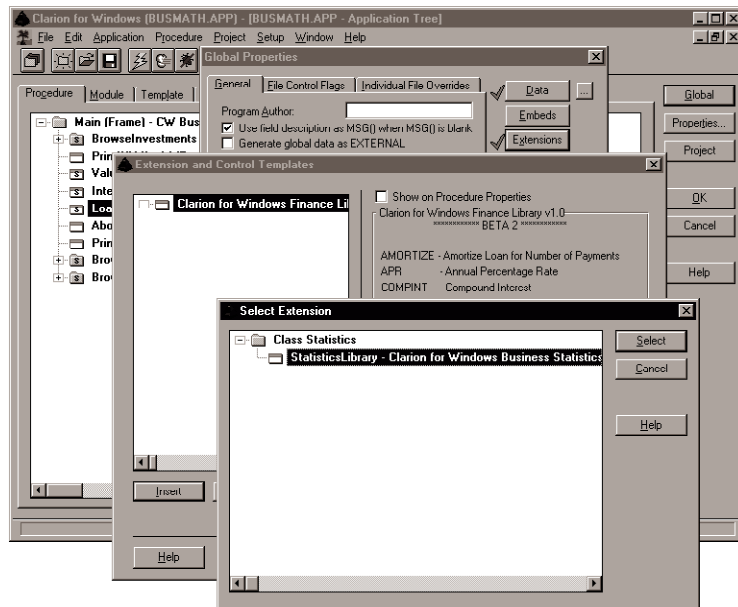
Adding Extension Templates to Your Application



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

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

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



Embedding Code Templates in Your Application

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

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

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

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

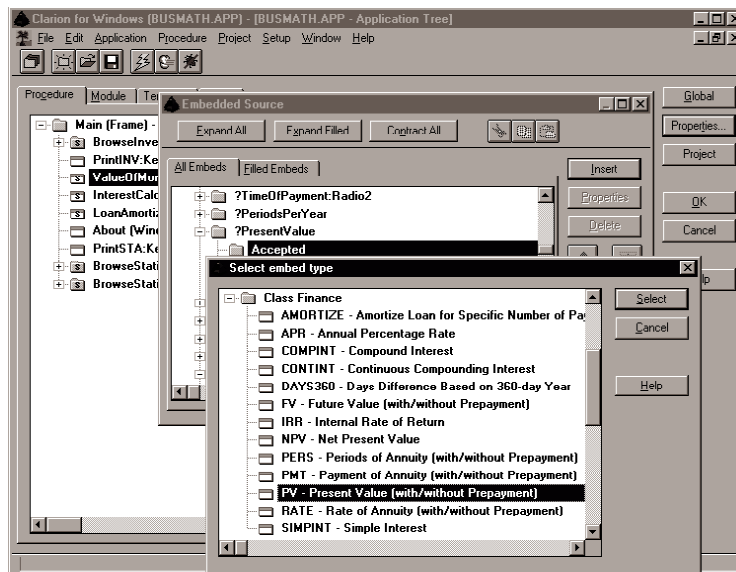
2. Highlight an embed point and press the **Insert** button.

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

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

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

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



Finance Code Templates

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

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

AMORTIZE

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

Prompts for AMORTIZE

Description:
AMORTIZE shows precisely which portions of a loan payment (Payment) constitute the interest portion (Interest) and the portion (Principal) which is applied towards repayment of the loan.
The computed amounts are based upon a loan balance (Current Balance), a periodic interest rate (Rate), and one or more number of payments (Number of Payments). The remaining balance (Ending Balance) is also calculated.

Input Parameters:

Current Balance:

Rate (Periodic):

Payment:

Number of Payments:

Return Values:

Principal:

Interest:

Ending Balance:

☐ Display Return Values

Notes:
* Payment, Principal, and Interest are negative numbers.
** Rate is the "per period rate".

OK Cancel Help

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:

$$\text{PeriodicRate} = \text{AnnualInterestRate} / (\text{PeriodsPerYear} * 100)$$

$$\text{ActualRate} = \text{PeriodicRate} * \text{TotalPeriods}$$

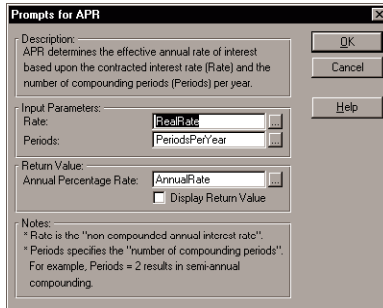
Payment	The label of a variable containing the amount of the desired payment (a negative number).
Number of Payments	The label of a variable containing the amount of the number of payments to amortize.
Principal	The label of a DECIMAL variable to receive the portion of the payment(s) applied to pay back the loan (a negative number).
Interest	The label of a DECIMAL variable to receive the portion of the payment(s) applied towards loan interest (a negative number).
Ending Balance	The label of a DECIMAL variable to receive the remaining loan balance.
Display Return Values	Checking this box generates a DISPLAY statement for each of the return values.

Example code generated by the AMORTIZE Code template:

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

APR

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



The image shows a dialog box titled "Prompts for APR". It contains the following sections:

- Description:** APR determines the effective annual rate of interest based upon the contracted interest rate (Rate) and the number of compounding periods (Periods) per year.
- Input Parameters:**
 - Rate:** A text input field containing "RealRate".
 - Periods:** A text input field containing "PeriodsPerYear".
- Return Value:**
 - Annual Percentage Rate:** A text input field containing "AnnualRate".
 - Display Return Value:** An unchecked checkbox.
- Notes:**
 - * Rate is the "non compounded annual interest rate".
 - * Periods specifies the "number of compounding periods".
 - For example, Periods = 2 results in semi-annual compounding.

On the right side of the dialog box are three buttons: "OK", "Cancel", and "Help".

The template prompts for the following parameters:

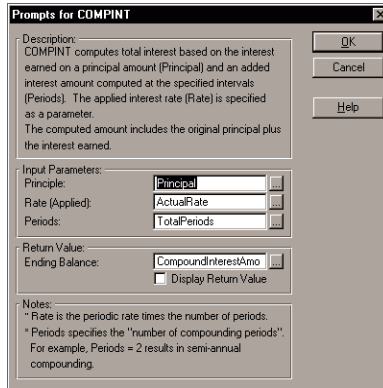
- | | |
|-------------------------------|--|
| Rate | The label of a variable containing the amount of the <i>contracted</i> interest rate. |
| Periods | The label of a variable containing the amount of the number of compounding periods per year. |
| Annual Percentage Rate | The label of a DECIMAL variable to receive the calculated annual percentage rate. |
| Display Return Value | Checking this box generates a DISPLAY statement for the return value. |

Example code generated by the APR Code template:

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

COMPINT

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



Prompts for COMPINT

Description:
COMPINT computes total interest based on the interest earned on a principal amount (Principal) and an added interest amount computed at the specified intervals (Periods). The applied interest rate (Rate) is specified as a parameter.
The computed amount includes the original principal plus the interest earned.

Input Parameters:

Principal:

Rate (Applied):

Periods:

Return Value:

Ending Balance:

☐ Display Return Value

Notes:
* Rate is the periodic rate times the number of periods.
* Periods specifies the "number of compounding periods".
For example, Periods = 2 results in semi-annual compounding.

OK
Cancel
Help

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:

$$\text{PeriodicRate} = \text{AnnualInterestRate} / (\text{PeriodsPerYear} * 100)$$

$$\text{AppliedRate} = \text{PeriodicRate} * \text{TotalPeriods}$$

Periods

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

Ending Balance

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

Display Return Value

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

Example code generated by the COMPINT Code template:

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


CONTINT

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

Prompts for CONTINT

Description:
CONTINT computes interest based on a principle amount (Principle) and an interest amount added continuously computed using the applied interest rate (Rate). The computed amount includes the original principal plus the interest earned.

Input Parameters:
Principle:
Rate (Applied):

Return Value:
Ending Balance:
☐ Display Return Value

Notes:
* Rate is the periodic rate times the number of periods.

OK Cancel Help

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:

$$\text{PeriodicRate} = \text{AnnualInterestRate} / (\text{PeriodsPerYear} * 100)$$

$$\text{ActualRate} = \text{PeriodicRate} * \text{TotalPeriods}$$

Ending Balance

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

Display Return Value

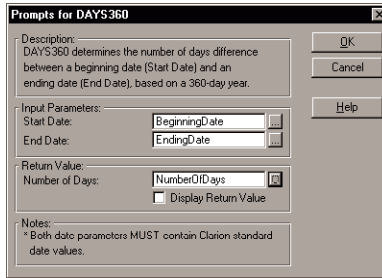
Checking this box generates a DISPLAY statement for the return value.

Example code generated by the CONTINT Code template:

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

DAYS360

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

A screenshot of a dialog box titled "Prompts for DAYS360". The dialog box contains several sections: "Description:" with text explaining the procedure; "Input Parameters:" with fields for "Start Date:" (containing "BeginningDate") and "End Date:" (containing "EndingDate"); "Return Value:" with a field for "Number of Days:" (containing "NumberOfDays") and a checkbox for "Display Return Value"; and a "Notes:" section at the bottom stating that both date parameters must contain Clarion standard date values. On the right side of the dialog box are buttons for "OK", "Cancel", and "Help".

The template prompts for the following parameters:

Start Date

The label of a variable containing the beginning date.

End Date

The label of a variable containing the ending date.

Number of Days

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

Display Return Value

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

Example code generated by the DAYS360 Code template:

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

FV

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

Prompts for FV

Description:
FV determines the future value of an amount (Present Value) based upon the total number of periods (Periods), the periodic interest rate (Rate), and a payment amount (Payment).

Input Parameters:

Present Value:

Periods:

Rate (Periodic):

Payment (May be EMPTY):

Return Value:

Future Value:

☐ Display Return Value

Time of Payments:

☐ Beginning of Periods

Notes:
* If the Payment prompt is empty then 0 is assumed.
* If the Payment prompt is NOT empty and the "Beginning of Periods" box is checked ON then the calculations take into account the added interest earned on each period's payment using the PREFV function.

Buttons: OK, Cancel, Help

The template prompts for the following parameters:

- Present Value** The label of a variable containing the present value of the investment.
- Periods** The label of a variable containing the number of periods in which a cash flow occurred.
- Rate (Periodic)** The label of a variable containing the *periodic* interest rate. Periodic rate may be calculated as:

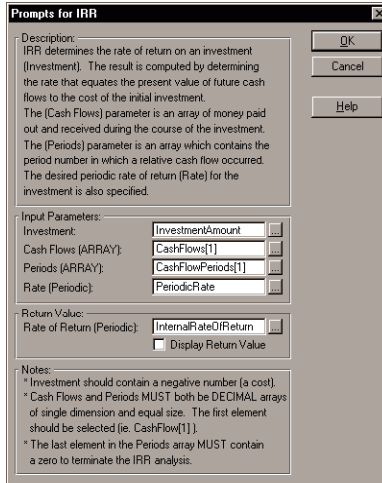
$$\text{PeriodicRate} = \text{AnnualInterestRate} / (\text{PeriodsPerYear} * 100)$$
- Payment** The label of a variable containing the periodic payment amount (optional).
- Future Value** The label of a DECIMAL variable to receive the calculated future value.
- Display Return Value** Checking this box generates a DISPLAY statement for the return value.
- Beginning of Periods** Checking this box generates a call to PREFV. PREFV assumes payments are made at the beginning of the compounding period and calculates interest accordingly.

Example code generated by the FV Code template:

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

IRR

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



Prompts for IRR

Description:
 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 (Cash Flows) 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 relative cash flow occurred.
 The desired periodic rate of return (Rate) for the investment is also specified.

Input Parameters:

Investment:

Cash Flows (ARRAY):

Periods (ARRAY):

Rate (Periodic):

Return Value:

Rate of Return (Periodic):

☐ Display Return Value

Notes:
 * Investment should contain a negative number (a cost).
 * Cash Flows and Periods MUST both be DECIMAL arrays of single dimension and equal size. The first element should be selected (ie. CashFlow[1]).
 * The last element in the Periods array MUST contain a zero to terminate the IRR analysis.

Buttons: OK, Cancel, Help

The template prompts for the following parameters:

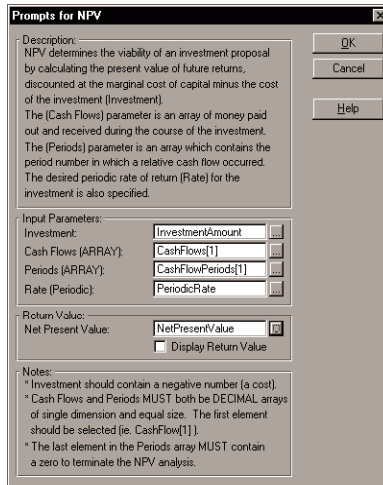
Investment	The label of a variable containing the initial cost of the investment (a negative number).
Cash Flows	The label of a single dimensioned DECIMAL array containing values in the amount of monies paid out and received during discrete accounting periods.
Periods	The label of a single dimensioned DECIMAL array containing period numbers identifying the discrete accounting period in which a corresponding cash flow occurred. <i>The last element in this array MUST contain a zero to terminate the IRR analysis.</i>
Rate (Periodic)	The label of a variable containing the desired <i>periodic</i> rate of return.
Rate of Return	The label of a DECIMAL variable to receive the calculated internal rate of return.
Display Return Value	Checking this box generates a DISPLAY statement for the return value.

Example code generated by the IRR Code template:

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

NPV

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



Prompts for NPV

Description:
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 (Cash Flows) 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 relative cash flow occurred.
The desired periodic rate of return (*Rate*) for the investment is also specified.

Input Parameters:

Investment:

Cash Flows (ARRAY):

Periods (ARRAY):

Rate (Periodic):

Return Value:

Net Present Value:

☐ Display Return Value

Notes:

- * Investment should contain a negative number (a cost).
- * Cash Flows and Periods MUST both be DECIMAL arrays of single dimension and equal size. The first element should be selected (ie. CashFlow[1]).
- * The last element in the Periods array MUST contain a zero to terminate the NPV analysis.

Buttons: OK, Cancel, Help

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. <i>The last element in this array MUST contain a zero to terminate the IRR analysis.</i>
Rate (Periodic)	The label of a variable containing the desired <i>periodic</i> rate of return.
Net Present Value	The label of a DECIMAL variable to receive the calculated net present value.
Display Return Value	Checking this box generates a DISPLAY statement for the return value.

Example code generated by the NPV Code template:

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

PERS

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

The template prompts for the following parameters:

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

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

$$\text{PeriodicRate} = \text{AnnualInterestRate} / (\text{PeriodsPerYear} * 100)$$

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

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

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

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

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

Example code generated by the PERS Code template:

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

PMT

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

The image shows a 'Prompts for PMT' dialog box. It contains a description of the PMT procedure, input parameters for Present Value, Periods, Rate (Periodic), and Future Value, a return value for Payment Amount, a checkbox for 'Display Return Value', a checkbox for 'Beginning of Periods', and a notes section explaining the 'Beginning of Periods' option.

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:

$$\text{PeriodicRate} = \text{AnnualInterestRate} / (\text{PeriodsPerYear} * 100)$$

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

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

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

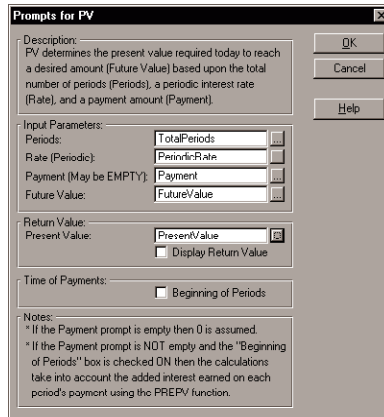
Beginning of Periods Checking this box generates a call to PREPMT. PREPMT assumes payments are made at the beginning of the compounding period and calculates interest accordingly.

Example code generated by the PMT Code template:

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

PV

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



Prompts for PV

Description:
PV determines the present value required today to reach a desired amount (Future Value) based upon the total number of periods (Periods), a periodic interest rate (Rate), and a payment amount (Payment).

Input Parameters:

Periods: TotalPeriods

Rate (Periodic): PeriodicRate

Payment (May be EMPTY): Payment

Future Value: FutureValue

Return Value:
Present Value: PresentValue
☐ Display Return Value

Time of Payments:
☐ Beginning of Periods

Notes:
* If the Payment prompt is empty then 0 is assumed.
* If the Payment prompt is NOT empty and the "Beginning of Periods" box is checked ON then the calculations take into account the added interest earned on each period's payment using the PREPV function.

OK Cancel Help

The template prompts for the following parameters:

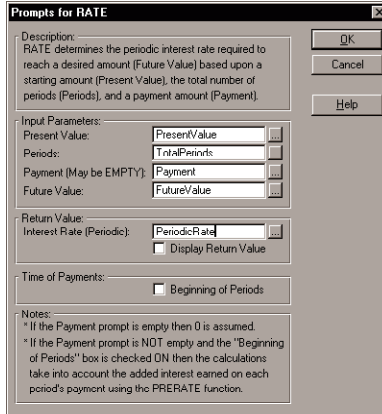
- | | |
|-----------------------------|--|
| Periods | The label of a variable containing the number of periods in which a payment is made. |
| Rate (Periodic) | The label of a variable containing the <i>periodic</i> rate of return. Periodic rate may be calculated as: |
| | $\text{PeriodicRate} = \text{AnnualInterestRate} / (\text{PeriodsPerYear} * 100)$ |
| Payment | The label of a variable containing the periodic payment amount (optional). |
| Future Value | The label of a variable containing the desired or targeted future value of the investment. |
| Present Value | The label of a DECIMAL variable to receive the calculated present value. |
| Display Return Value | Checking this box generates a DISPLAY statement for the return value. |
| Beginning of Periods | Checking this box generates a call to PREPV. PREPV assumes payments are made at the beginning of the compounding period and calculates interest accordingly. |

Example code generated by the PV Code template:

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


RATE

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



The image shows a dialog box titled "Prompts for RATE". It contains several sections:

- Description:** RATE determines the periodic interest rate required to reach a desired amount (Future Value) based upon a starting amount (Present Value), the total number of periods (Periods), and a payment amount (Payment).
- Input Parameters:**
 - Present Value: [Text box]
 - Periods: [Text box]
 - Payment (May be EMPTY): [Text box]
 - Future Value: [Text box]
- Return Value:**
 - Interest Rate (Periodic): [Text box]
 - ☐ Display Return Value
- Time of Payments:**
 - ☐ Beginning of Periods
- Notes:**
 - * If the Payment prompt is empty then 0 is assumed.
 - * If the Payment prompt is NOT empty and the "Beginning of Periods" box is checked ON then the calculations take into account the added interest earned on each period's payment using the PRERATE function.

 On the right side of the dialog box are three buttons: OK, Cancel, and Help.

The template prompts for the following parameters:

Present Value	The label of a variable containing the present value of the investment.
Periods	The label of a variable containing the number of periods in which a payment is made.
Payment	The label of a variable containing the periodic payment amount (optional).
Future Value	The label of a variable containing the desired or targeted future value of the investment.
Rate (Periodic)	The label of a DECIMAL variable to receive the calculated periodic rate.
Display Return Value	Checking this box generates a DISPLAY statement for the return value.
Beginning of Periods	Checking this box generates a call to PRERATE. PRERATE assumes payments are made at the beginning of the compounding period and calculates interest accordingly.

Example code generated by the RATE Code template:

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

SIMPINT

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

Prompts for SIMPINT

Description:
SIMPINT determines an "interest amount earned" based solely on a given amount (Principal) and the simple interest rate (Rate).

Input Parameters:

Principal:

Rate:

Return Value:

Simple Interest Amount:

☐ Display Return Value

Notes:
* The amount returned ONLY reflects interest earned.

OK
Cancel
Help

The template prompts for the following parameters:

Principal

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

Rate (Applied)

The label of a variable containing the simple or contract interest rate.

Simple Interest Amount

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

Display Return Value

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

Example code generated by the SIMPINT Code template:

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

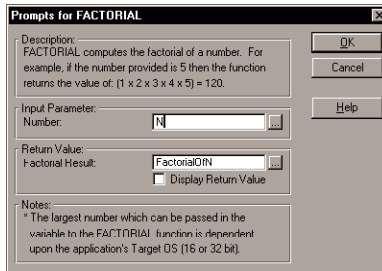
Business Statistics Code Templates

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

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

FACTORIAL

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



The template prompts for the following parameters:

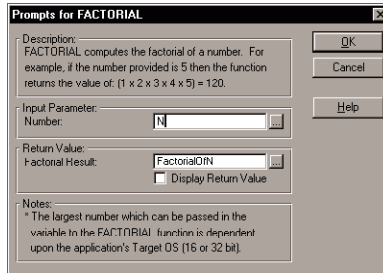
- | | |
|-----------------------------|--|
| Number | The label of a variable containing the number to factorialize. |
| Factorial Result | The label of a REAL variable to receive the calculated factorial amount. |
| Display Return Value | Checking this box generates a DISPLAY statement for the return value. |

Example code generated by the FACTORIAL Code template:

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

FREQUENCY

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



The template prompts for the following parameters:

- | | |
|-----------------------------|---|
| Data Set | The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to search. |
| Search Value | The label of a variable containing the particular value to count. |
| Frequency Count | The label of a REAL variable to receive the number of instances counted. |
| Display Return Value | Checking this box generates a DISPLAY statement for the return value. |

Example code generated by the FREQUENCY Code template:

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

LOWERQUARTILE

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

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

Prompts for LOWERQUARTILE

Description: LOWERQUARTILE computes a value (Lower Quartile Value) such that at most 25% of the set's values are less than the amount, and at most 75% of the set's values are greater than the amount. For example, given the data set: [1,2,3,4,5,6,7,8] the lower quartile value is 2.5.

Input Parameter:
Data Set: StatSetX

Return Value:
Lower Quartile Value: LowerQuartileOfSet
☐ Display Return Value

Notes:
* Data Set must be a QUEUE with the first component defined as a REAL variable.
* The function operates on the data set defined by the (first component) numeric values contained in all QUEUE entries.
* In general, the LOWERQUARTILE function is only meaningful when the number of elements in the Data Set is large (ie. greater than 20).

Buttons: OK, Cancel, Help

The template prompts for the following parameters:

Data Set

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

Lower Quartile Value

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

Display Return Value

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

Example code generated by the LOWERQUARTILE Code template:

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

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

MEAN

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

Prompts for MEAN

Description:
MEAN computes the arithmetic average (Mean Value) of a set. The arithmetic average is the summation 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 $(1+2+3+4+5) / 5 = 3$.

Input Parameter:
Data Set: StatSetX

Return Value:
Mean Value: MeanValue
☐ Display Return Value

Notes:
* Data Set must be a QUEUE with the first component defined as a REAL variable.
* The function operates on the data set defined by the (first component) numeric values contained in all QUALIFY entries.

OK
Cancel
Help

The template prompts for the following parameters:

- | | |
|-----------------------------|--|
| Data Set | The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to analyze. |
| Mean Value | The label of a REAL variable to receive the calculated value. |
| Display Return Value | Checking this box generates a DISPLAY statement for the return value. |

Example code generated by the MEAN Code template:

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

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

MEDIAN

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

Prompts for MEDIAN

Description:
MEDIAN finds 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 $(2 + 4) / 2 = 3$.

Input Parameter:
Data Set:

Return Value:
Median Value:
☐ Display Return Value

Notes:
* Data Set must be a QUEUE with the first component defined as a REAL variable.
* The function operates on the data set defined by the (first component) numeric values contained in all QUEUE entries.

Buttons: OK, Cancel, Help

The template prompts for the following parameters:

- | | |
|-----------------------------|--|
| Data Set | The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to analyze. |
| Median Value | The label of a REAL variable to receive the calculated value. |
| Display Return Value | Checking this box generates a DISPLAY statement for the return value. |

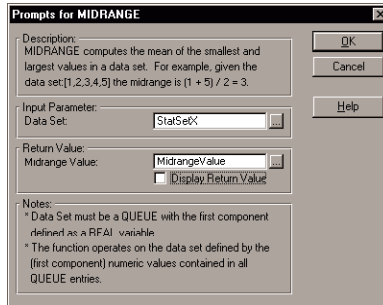
Example code generated by the MEDIAN Code template:

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

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

MIDRANGE

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

A dialog box titled "Prompts for MIDRANGE" with a close button (X) in the top right corner. It contains several sections: "Description:" with a text area explaining the procedure; "Input Parameter:" with a label "Data Set:" and a text field containing "StatSetX" and a browse button (...); "Return Value:" with a label "Midrange Value:" and a text field containing "MidrangeValue" and a browse button (...), followed by a checkbox labeled "Display Return Value"; and "Notes:" with a list of two bullet points. On the right side of the dialog are three buttons: "OK", "Cancel", and "Help".

Prompts for MIDRANGE

Description:
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 $(1 + 5) / 2 = 3$.

Input Parameter:
Data Set: StatSetX

Return Value:
Midrange Value: MidrangeValue
☐ Display Return Value

Notes:
* Data Set must be a QUEUE with the first component defined as a REAL variable.
* The function operates on the data set defined by the (first component) numeric values contained in all QUEUE entries.

OK Cancel Help

The template prompts for the following parameters:

- | | |
|-----------------------------|--|
| Data Set | The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to analyze. |
| Midrange Value | The label of a REAL variable to receive the calculated value. |
| Display Return Value | Checking this box generates a DISPLAY statement for the return value. |

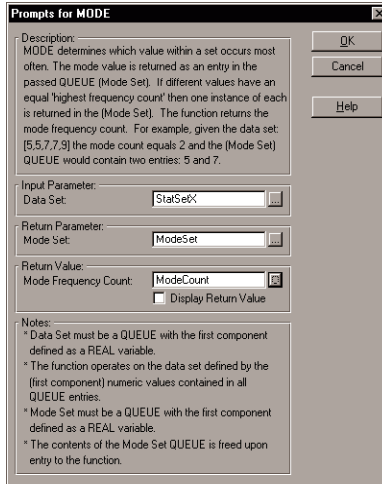
Example code generated by the MIDRANGE Code template:

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

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

MODE

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



Prompts for MODE

Description:
MODE determines which value within a set occurs most often. The mode value is returned as an entry in the passed QUEUE (Mode Set). If different values have an equal "highest frequency count" then one instance of each is returned in the (Mode Set). The function returns the mode frequency count. For example, given the data set [5,5,7,7,9] the mode count equals 2 and the (Mode Set) QUEUE would contain two entries: 5 and 7.

Input Parameter:
Data Set: StatSetX

Return Parameter:
Mode Set: ModeSet

Return Value:
Mode Frequency Count: ModeCount
☐ Display Return Value

Notes:
* Data Set must be a QUEUE with the first component defined as a REAL variable.
* The function operates on the data set defined by the (first component) numeric values contained in all QUEUE entries.
* Mode Set must be a QUEUE with the first component defined as a REAL variable.
* The contents of the Mode Set QUEUE is freed upon entry to the function.

Buttons: OK, Cancel, Help

The template prompts for the following parameters:

- | | |
|-----------------------------|---|
| Data Set | The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to analyze. |
| Mode Set | The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE receives the value or values that occur most often in the data set. |
| Mode Frequency Count | The label of a REAL variable to receive the calculated value. |
| Display Return Value | Checking this box generates a DISPLAY statement for the return value. |

Example code generated by the MODE Code template:

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

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

PERCENTILE

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

Prompts for PERCENTILE

Description:
PERCENTILE computes a value (Percentile 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.

Input Parameters:
Data Set: StatSetX
Percentile (pth%): PercentileOfSet

Return Value:
Percentile Value: Percentile
☐ Display Return Value

Notes:
* Data Set must be a QUEUE with the first component defined as a REAL variable.
* The function operates on the data set defined by the (first component) numeric values contained in all QUEUE entries.
* The Percentile variable MUST contain an integer value in the range: 1 >= p <= 99.
* In general, the PERCENTILE function is only meaningful when the number of elements in the Data Set is large (ie. greater than 20).

Buttons: OK, Cancel, Help

The template prompts for the following parameters:

Data Set	The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to analyze.
Percentile	The label of a variable containing the division point expressed as a percentage.
Percentile Value	The label of a REAL variable to receive the calculated value.
Display Return Value	Checking this box generates a DISPLAY statement for the return value.

Example code generated by the PERCENTILE Code template:

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

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

RANGEOFSET

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

The dialog box titled "Prompts for RANGEOFSET" contains the following information:

- Description:** RANGEOFSET computes the difference between the largest and smallest values in the set. For example, given the data set: [1,2,3,4,5], the range is $(5 - 1) = 4$.
- Input Parameter:** Data Set: StatSetX
- Return Value:** Range Value: RangeOfSet
- ☐ Display Return Value
- Notes:**
 - * Data Set must be a QUEUE with the first component defined as a REAL variable.
 - * The function operates on the data set defined by the (first component) numeric values contained in all QUEUE entries.

Buttons: OK, Cancel, Help

The template prompts for the following parameters:

- | | |
|-----------------------------|--|
| Data Set | The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to analyze. |
| Range Value | The label of a REAL variable to receive the calculated value. |
| Display Return Value | Checking this box generates a DISPLAY statement for the return value. |

Example code generated by the RANGEOFSET Code template:

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

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

rVALUE

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

Prompts for rVALUE

Description:
rVALUE computes the Correlation Coefficient of the linear relationship between the paired data points in the data set.

Input Parameters:
Data Set (Paired): StatSetXY
(Optional Parameters)
Mean of X Data:
Mean of Y Data:

Return Value:
Correlation Coefficient: CorrelationCoefficient
☐ Display Return Value

Notes:
* Data Set must be a QUEUE with the first two (X and Y) components defined as REAL variables.
* The function operates on the data sets defined by the paired X and Y (first and second component) numeric values contained in all QUEUE entries.
* The Optional Parameters help to optimize the function. If not provided, either (or both) mean value(s) will be automatically computed by the rVALUE function.

The template prompts for the following parameters:

- | | |
|--------------------------------|--|
| Data Set | The label of a QUEUE whose first two component fields are REAL. The first component of the QUEUE comprises the set of X values to analyze. The second component of the QUEUE comprises the set of Y values to analyze. The two components are treated as x-y coordinate pairs. |
| Mean of X Data | The label of a variable containing the average of the X values (optional). |
| Mean of Y Data | The label of a variable containing the average of the Y values (optional). |
| Correlation Coefficient | The label of a REAL variable to receive the calculated value. |
| Display Return Value | Checking this box generates a DISPLAY statement for the return value. |

Example code generated by the rVALUE Code template:

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

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

SDEVIATION

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

Prompts for SDEVIATION

Description:
SDEVIATION computes the standard deviation of a given data set. The appropriate function is called based upon the data set's type (Type). The 'Population Data Set' radio button should be checked only if the data set contains all of a population's values.

Type:
☐ Sample Data Set
☒ Population Data Set

Input Parameters:
 Data Set:
 (Optional Parameter)
 Mean of Data Set:

Return Value:
 Standard Deviation:
☐ Display Return Value

Notes:
 * Data Set must be a QUEUE with the first component defined as a REAL variable.
 * The function operates on the data set defined by the (first component) numeric values contained in all QUEUE entries.
 * The Optional Parameter helps to optimize the function. If not provided, the mean value will be automatically computed by the function.

OK Cancel Help

The template prompts for the following parameters:

Type	Choose 'Sample' or 'Population.' 'Sample' is the default.
Data Set	The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to analyze.
Mean of Data Set	The label of a variable containing the average of the set's values (optional).
Standard Deviation	The label of a REAL variable to receive the calculated value.
Display Return Value	

Example code generated by the SDEVIATION Code template:

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

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

SS

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

Prompts for SS

Description:
SS computes the summation of the squared differences between each data set value and the data set's mean. The computation is a significant factor in several statistical formulas.

Input Parameters:
Data Set:
(Optional Parameter)
Mean of Data Set:

Return Value:
Sum of Squares Value:
☐ Display Return Value

Notes:
* Data Set must be a QUEUE with the first component defined as a REAL variable.
* The function operates on the data set defined by the (first component) numeric values contained in all QUEUE entries.
* The Optional Parameter helps to optimize the function. If not provided, the mean value will be automatically computed by the SS function.

OK Cancel Help

The template prompts for the following parameters:

- | | |
|-----------------------------|--|
| Data Set | The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to analyze. |
| Mean of Data Set | The label of a variable containing the average of the set's values (optional). |
| Sum of Squares Value | The label of a REAL variable to receive the calculated value. |
| Display Return Value | Checking this box generates a DISPLAY statement for the return value. |

Example code generated by the SS Code template:

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

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

SSxy

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

Prompts for SSxy

Description:
SSxy computes the summation of the products of the differences between each data point and its associated (either X or Y) mean value. The computation is a significant factor in linear regression formulas.

Input Parameters:
Data Set (Paired): StatSetXY
(Optional Parameters)
Mean of X Data:
Mean of Y Data:

Return Value:
Sum of Squares for X and Y: SumOfSquaresForXandY
☐ Display Return Value

Notes:
* Data Set must be a QUEUE with the first two (X and Y) components defined as REAL variables.
* The function operates on the data sets defined by the paired X and Y (first and second component) numeric values contained in all QUEUE entries.
* The Optional Parameters help to optimize the function. If not provided, either (or both) mean value(s) will be automatically computed by the SSxy function.

OK Cancel Help

The template prompts for the following parameters:

Data Set

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

Mean of X Data

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

Mean of Y Data

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

Sum of Squares for X and Y

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

Display Return Value

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

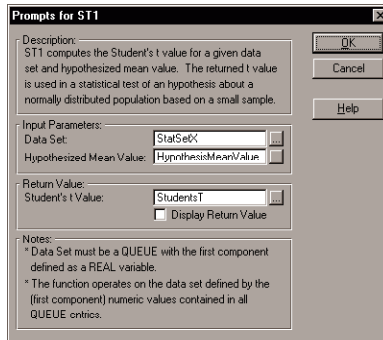
Example code generated by the SSxy Code template:

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

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

ST1

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



The template prompts for the following parameters:

- | | |
|--------------------------------|--|
| Data Set | The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to analyze. |
| Hypothesized Mean Value | The label of a REAL variable containing a hypothesized mean value associated with the statistical test. |
| Student's t Value | The label of a REAL variable to receive the calculated value. |
| Display Return Value | Checking this box generates a DISPLAY statement for the return value. |

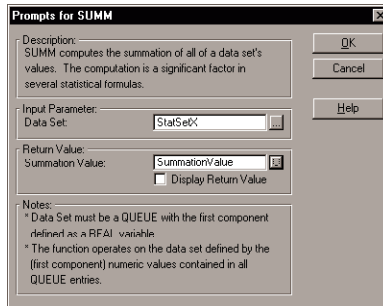
Example code generated by the ST1 Code template:

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

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

SUMM

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



The template prompts for the following parameters:

- | | |
|-----------------------------|--|
| Data Set | The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to analyze. |
| Summation Value | The label of a REAL variable to receive the calculated value. |
| Display Return Value | Checking this box generates a DISPLAY statement for the return value. |

Example code generated by the SUMM Code template:

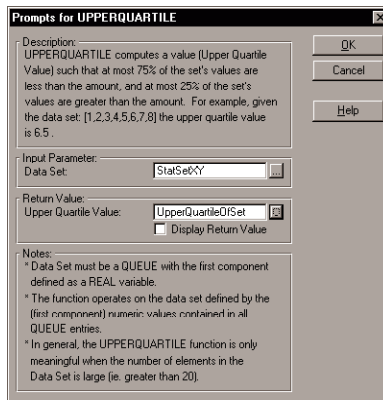
```
SummationValue = SUMM(StatSetX)  
DISPLAY(?SummationValue)
```

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

UPPERQUARTILE

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

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



The template prompts for the following parameters:

Data Set

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

Upper Quartile Value

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

Display Return Value

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

Example code generated by the UPPERQUARTILE Code template:

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

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

VARIANCE

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

Prompts for VARIANCE

Description:
VARIANCE computes the variance of a given data set. The appropriate function is called based upon the set's type (Type). The 'Population Data Set' radio button should be checked only if the data set contains all of a population's values.

Type:
☒ Sample Data Set
☐ Population Data Set

Input Parameters:
 Data Set: StatSetX
 (Optional Parameter):
 Mean of Data Set:

Return Value:
 Variance of Set: VarianceOfSample
☐ Display Return Value

Notes:
 * Data Set must be a QUEUE with the first component defined as a REAL variable.
 * The function operates on the data set defined by the (first component) numeric values contained in all QUEUE entries.
 * The Optional Parameter helps to optimize the function. If not provided, the mean value will be automatically computed by the function.

The template prompts for the following parameters:

- | | |
|-----------------------------|--|
| Type | Choose 'Sample' or 'Population.' 'Sample' is the default. |
| Data Set | The label of a QUEUE whose first component field is a REAL. This first component of the QUEUE comprises the numeric data set to analyze. |
| Mean of Data Set | The label of a variable containing the average of the set's values (optional). |
| Variance of Set | The label of a REAL variable to receive the calculated value. |
| Display Return Value | Checking this box generates a DISPLAY statement for the return value. |

Example code generated by the VARIANCE Code template:

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

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

5 - BUSINESS MATH EXAMPLES

Overview

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

Exploring the Example Application

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

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

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

Finance

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

Interest Calculations

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

Loan Amortization

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

Cash Flow Analysis (BrowseInvestments)

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

Statistics**Linear Regression Analysis** (BrowseStatisticsPairedDataItems)

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

Data Set Analysis (BrowseStatisticsSingleDataItems)

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

Classroom Data Example

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

INDEX

Symbols

#CODE Templates	64, 65
#EXTENSION Templates	64, 65

A

Adding Extension Templates	11, 67
amortization	10, 15
AMORTIZE	18, 69
annuity	30, 34, 38
with prepayment	32, 35, 39
APR	20, 71
average	46, 86

B

Base 10 math	16
BCD math	16
Binary Coded Decimal math	16
Business Math Functions	
calling	63
Business Math Library	
hand-code support	12
implementing	10
installing	11
overview	10
using	11
Business Math Templates	63
access	63
Business Statistics Code Templates	83
FACTORIAL	83
FREQUENCY	84
LOWERQUARTILE	85
MEAN	86
MEDIAN	87
MIDRANGE	88
MODE	89
PERCENTILE	90
RANGE OF SET	91
rVALUE	92
SDEVIATION	93
SS	94
SSxy	95
ST1	96
SUMM	97
UPPERQUARTILE	98

VARIANCE	99
Business Statistics Library	
components	41
overview	41
BUSMATH.PR	12

C

cash flow analysis	10, 15
Class Finance	64
Class Statistics	65
CLFIN16.LIB	15
CLFIN32.LIB	15
CLSTAT16.LIB	41
Code Templates	11, 15, 41
access	11, 64, 65, 67
Business Statistics	83
embedding	68
Finance	69
Statistics	83
COMPINT	21, 72
compound interest	21, 72
CONTINT	22, 73
continuously compounded interest	22, 73
Conventions	7, 8
correlation coefficient	52, 92
count	44, 49, 89
CWFIN16.DLL	15
CWFIN16.LIB	15
CWFIN32.DLL	15
CWFIN32.LIB	15
CWFINPRO.CLW	12, 16
CWSTAT16.DLL	41
CWSTAT16.LIB	41
CWSTAT32.DLL	41
CWSTAT32.LIB	41
CWSTATDT.CLW	42
CWSTATPR.CLW	12, 42

D

date	74
date calculations	23, 74
DAYS360	23, 74
Documentation Conventions	7
documentation format	8

E

ellipsis (...) button	69, 83
Embed point	11, 63, 64, 65, 67
Embedded Source dialog	68
Embedding Code Templates	68
Extension Templates	
adding to your application	11, 67
Extensions and Control Templates dialog	67

F

FACTORIAL	43, 83
Finance Class	64
Finance Code Templates	69
AMORTIZE	69
APR	71
COMPINT	72
CONTINT	73
DAYS360	74
FV	75
IRR	76
NPV	77
PERS	78
PMT	79
PV	80
RATE	81
SIMPINT	82
Finance Functions	
APR	20
COMPINT	21
CONTINT	22
DAYS360	23
FV	24
IRR	26
NPV	28
PERS	30
PREFV	25
PREPERS	32
PREPMT	35
PREPV	37
PRERATE	39
PV	36
SIMPINT	40
Finance Library	
conventions	16
data types	16
overview	15
parameters	16
return values	16
Sign Conventions	17
Finance Procedures	

AMORTIZE	18
Finance Template Class	11
FINANCE.TPL	11, 16, 66
FinanceLibrary	64
FREQUENCY	44, 84
future value	
24, 30, 32, 34, 35, 36, 37, 38, 75, 78, 79, 80, 81	
with prepayment	25
FV	24, 75

G

Global Extensions	11, 67
Global Properties dialog	67

H

Hand-Code Support	12
hypothesized mean	55, 96

I

INCLUDE	12
Installing	11
interest	18, 69
interest calculations	10, 15
internal rate of return	26, 76
IRR	26, 76

K

KEYWORD	
Syntax Diagram	9

L

linear regression	10, 52
LOWERQUARTILE	45, 85, 90, 98

M

MAP	11, 12, 63, 64, 65, 67
MEAN	10, 46, 86
MEDIAN	47, 87
median	10, 45, 85, 98
MIDRANGE	48, 88
MODE	10, 49, 89

N

negative	17
net present value	28, 77
NPV	28, 77

P

payment of annuity	34
with prepayment	35
PERCENTILE	50, 85, 90, 98
periods of annuity	30
with prepayment	32
PERS	30, 78
PMT	34, 79
Population	93, 99
standard deviation	56
variance	60
positive	17
PREFV	25
PREPERS	32
PREPMT	35
PREPV	37
PRERATE	39
present value	28, 36, 77, 80
with prepayment	37
principal	18, 69
project	11, 12, 63, 64, 65, 67
Project Editor	12
project libraries	12
prototypes	11, 12, 15, 41, 63, 64, 65, 67
pth%	90
PV	36, 80

R

RANGEOFSET	51, 91
RATE	38, 81
rate of annuity	38
with prepayment	39
Registering the Template Classes	66
return values	
finance functions	16
Rounding	16
RVALUE	52
rVALUE	92

S

Sample	93, 99
standard deviation	57
variance	61
SDEVATION	93
SDEVATIONNP	56
SDEVIATIONS	57
Select Embed Type dialog	68
Select Extension dialog	67
Sign Conventions	17

SIMPINT	40, 82
simple interest	40, 82
SS	53, 94
SSXY	54
SSxy	95
ST1	55, 96
standard deviation	10, 93
population	56
sample	57
STATISTC.TPL	11, 42, 66
Statistics Class	65
Statistics Functions	
FACTORIAL	43
FREQUENCY	44
LOWERQUARTILE	45
MEAN	46
MEDIAN	47
MIDRANGE	48
MODE	49
PERCENTILE	50
RANGEOFSET	51
RVALUE	52
SDEVATIONNP	56
SDEVIATIONS	57
SS	53
SSXY	54
ST1	55
SUMM	58
UPPERQUARTILE	59
VARIANCEP	60
VARIANCES	61
Statistics Template Class	11
StatisticsLibrary	65
Student's t	55, 96
sum of the squared differences	94
sum of squares	53, 94
coordinate pairs	54
sum of the products	54, 95
SUMM	58, 97
summation	58, 97
syntax diagram	8

T

Template Class	63
registering	66
Template File dialog	66
Template Registry dialog	66
Templates	
Business Math	63
time value of money	10, 15

U

UPPERQUARTILE 59, 85, 90, 98

V

VARIANCE 99

variance 10

 population 60

 sample 61

VARIANCEP 60

VARIANCES 61