Chapter 10
Using Functions

Introduction

R&R offers over 100 predefined functions you can use in calculated field expressions. A predefined function is a formula that performs a specified operation such as returning the system date or the current page number.

You can also define and save your own user-defined functions (UDFs) that can be used in any report just like predefined functions. For information about UDFs, see the **Using User-Defined Functions** section of this chapter.

Figure 10.1 lists and briefly describes the functions, using the abbreviations listed below to indicate the input and output data type of each expression. Note that the R&R *N* data type accommodates any numeric value.

Abbreviation	***Refers To***
A	A table alias
C	A character expression
D	A date expression
DT	A datetime or time expression
I	An item (numeric, character, date, memo, or logical)
L	A logical expression
N	A numeric expression
T	A time expression

Brackets around the abbreviation for an input argument (for example [A]) indicate that the argument is optional.

Function	*Description*	*Input Data Type*	*Output Type*
ABS	Returns absolute value	N	N
ADDDAYS	Adds number of days to date	D,N	D
ADDMONS	Adds number of months to date	D, N	D
ADDWKS	Adds number of weeks to date	D, N	D
ADDYRS	Adds number of years to date	D, N	D
AGED	Determines aged status of date	D,N,N	L
ASC	Converts character to numeric code	C	N
AT	Searches substring	C, C, [L]	N
BLANKNUM	Returns an empty numeric value	none	N
CAPFIRST	Converts first letter to upper case	C,[C]	C
CASE	Returns result based on value of item	I,I,[I,...]default	I
CDLL	Enables calling of a DLL-based function	C, C, C	C
CDOW	Converts day of date to character	D	C
CHR	Converts numeric code to character	N	C
CLOOKUP	Returns character value from another table	I,C,C,C,[C]	C
CMONTH	Converts month of date to character	D	C
COPY	Returns number of copy currently printing	none	N
CTDT	Converts character value to datetime	C	DT
CTOD	Converts character to date	C	D
CTOS	Converts character interval to seconds	C	DT
CTOT	Converts character value to time	C	DT
DATE	Returns system date	none	D
DAY	Returns day-of-month number	D	N
DAYSBTWN	Calculates number of days between dates	D, D	N
DBF	Returns full path and name of table	[A]	C
DELETED	Identifies deleted record	A	L
DLOOKUP	Returns date value from another table	I,C,C,C,[C]	D
DOW	Calculates day-of-week number	D	N
DQTR	Calculates date of first day of quarter	D	D
DTEADD	Adds specified interval to datetime	I,N,DT	DT
DTEDIFF	Calculates difference between datetime values	I,DT,DT	N
DTEPART	Returns specified interval as an integer	I,DT	N
DTLOOKUP	Returns datetime value from another table	I,C,C,C,[C]	DT
DTOC	Converts date to character string	D, [N]	C
DTTOC	Converts datetime to character string	DT	C
ERROR	Returns true if item has error value	I	L

EXP	Calculates the value of e**n	N	N
FLIP	Swaps words before and after a character	C, C	C

Figure 10.1 Function Descriptions (Continued on Next Page)

Function	*Description*	*Input Data Type*	*Output Type*
FUTURE	Determines whether date is in the future	D, [N]	L
HALF	Returns calendar half-year of date	D	N
HISCOPE	Returns high scope value	none	C
IIF	Selects one of two expressions	I, I, I	I
INLIST	Looks up value in list of values	I, [I,...]	N
INRANGE	Determines if value is within range	I, I, I	L
INT	Discards digits to right of decimal	N	N
ISALPHA	Determines if first character is a letter	C	L
ISBLANK	Determines if value is blank	I	L
ISLOWER	Determines if first character is lower case	C	L
ISUPPER	Determines if first character is upper case	C	L
LEFT	Selects substring starting at left	C, N	C
LEN	Calculates length of string or field	C	N
LIBNAME	Returns full path of current report library	none	C
LLOOKUP	Returns logical value from another table	I,C,C,C,[C]	L
LOG	Returns natural logarithm	N	N
LOSCOPE	Returns low scope value	none	C
LOWER	Converts upper to lower case	C	C
LTRIM	Removes leading blanks	C	C
LUPDATE	Returns date of last table update	A	D
MAX	Returns higher of two values	N, N	N
MIN	Returns lower of two values	N, N	N
MOD	Returns the remainder of division	N, N	N
MONLEN	Returns number of days in month	D	N
MONSBTWN	Calculates months between two dates	D, D	N
MONTH	Returns month-of-year number	D	N
NDOW	Returns date of next specified DOW	D,N	D
NLOOKUP	Returns numeric value from another table	I,C,C,C,[C]	N
OVER	Determines whether date is past specified number of days	D,N	L
PAGENO	Returns current report page number	none	N
PAST	Determines whether date is in the past	D,[N]	L
PDOW	Returns date of previous specified DOW	D,N	D
PREVIOUS	Returns value in previous composite record	I	I
QTR	Calculates calendar quarter of date	D	N
QUERY	Returns current query expression	none	C

RECCOUNT	Returns record count of table	A	N
RECNO	Returns composite record number	[A]	N
REPLICATE	Repeats character expression	C, N	C

Figure 10.1 Function Descriptions (Cont'd)

Function	*Description*	*Input Data Type*	*Output Type*
REPNAME	Returns name of current report	none	C
RIGHT	Selects substring ending at right	C, N	C
RIPARAM	Returns value of runtime parameter	C	C
ROUND	Rounds off number	N, N	N
RRUNIN	Returns 1 or control table record # in Runtime	none	N
RTRIM	Removes trailing blanks	C	C
SCANNING	Returns true if scanning selected table	A	L
SOUNDEX	Returns value of string based on sound	C, [C]	C
SPACE	Produces a string of blank spaces	N	C
SPELLNUM	Spells a number	N	C
SQRT	Calculates square root	N	N
STOC	Converts seconds to character string	DT	C
STR	Converts number to character	N, [N], [N]	C
STRCOUNT	Returns number of occurrences of substring	C,C,[L]	N
STRREP	Replaces one substring with another	C,C,C [L]	C
STRSEARCH	Returns starting position of nth substring	C,C,N,[L]	N
STUFF	Replaces part of string with new string	C, N, N, C	C
SUBDAYS	Subtracts number of days from date	D, N	D
SUBMONS	Subtracts number of months from date	D, N	D
SUBSTR	Selects substring	C, N, [N]	C
SUBWKS	Subtracts number of weeks from date	D, N	D
SUBYRS	Subtracts number of years from date	D, N	D
TIME	Returns system time	none	C
TODATE	Converts datetime to date	DT	D
TOTIME	Converts datetime to time	DT	T
TRANSFORM	Returns formatted character data	C or N, C	C
TRIM	Removes trailing blanks	C	C
TTOC	Converts time to character	DT or T	C
TTOS	Converts a time value to seconds	DT	N
UDFNAME	Returns pathname of UDF library	none	C
UPPER	Converts lower to upper case	C	C
VAL	Converts character string to number	C	N
WDCOUNT	Returns number of words in string	C, [C]	N
WEEK	Calculates week-of-month for date	D	N
WKSBTWN	Calculates weeks between two dates	D, D	N
WORD	Returns *n*th word of a character string	C, N, [C]	C

YEAR	Returns year number of date	D	N
YRSBTWN	Calculates years between two dates	D, D	N

Figure 10.1 Function Descriptions (Cont'd)

Alphabetical List of R&R Functions

ABS

Returns the absolute value of a numeric expression. The result is a positive number.

Syntax: ABS(N), where **N** is any numeric expression.

Example: To calculate the absolute value of the difference between two numeric fields NUM1 and NUM2, create the field:

ABS(NUM1 – NUM2)

If the value of NUM1 is 50 and the value of NUM2 is 100, then the value of ABS(NUM1 – NUM2) is 50.

ADDDAYS

Calculates a date by adding a number of days to a date. Note that you can also add dates by using the + operator.

Syntax: ADDDAYS(D,N) where **D** is a date expression and **N** is a numeric expression representing a number of days.

Example: To calculate the date a payment is due by adding the payment terms (NET) to the date the product was ordered (ORDERDATE), create the field:

ADDDAYS(ORDERDATE,NET)

If the value of NET is 30 and ORDERDATE is 4-1-95, then the value of ADDDAYS(ORDERDATE,NET) is 5-1-95.

ADDMONS

Calculates a date by adding a number of months to date.

Syntax: ADDMONS(D,N) where **D** is a date expression and **N** is a numeric expression representing a number of months.

Example: To calculate the date of the final payment of a loan by adding the number of months (TERMS) to the starting date of the loan (LOANINIT), create the field:

ADDMONS(LOANINIT,TERMS)

If the value of LOANINIT is 01/10/92 and the value of TERMS is 48, then ADDMONS returns 01/10/96.

ADDWKS

Calculates a date by adding a number of weeks to a date.

Syntax: ADDWKS(D,N) where **D** is a date expression and **N** is a numeric expression representing a number of weeks.

Example: To calculate a date for a return visit to the doctor by adding the number of weeks to the current appointment date (CURDATE), create the field:

```
ADDWKS(CURDATE,6)
```

If the value of CURDATE is 01/10/95, then ADDWKS returns a value of 02/21/95.

ADDYRS

Calculates a date by adding a number of years to a date.

Syntax: ADDYRS(D,N) where **D** is a date expression and **N** is a numeric expression representing a number of years.

Example: To calculate a follow-up date for an inoculation by adding the number of years of immunity (IMMUNYRS) to the most recent inoculation date (INOCDATE), create the field:

```
ADDYRS(INOCDATE,IMMUNYRS)
```

If the value of INOCDATE is 04/01/87 and IMMUNYRS is 7, then this calculated field yields 04/01/94.

AGED

Returns logical True if date is aged between specified values.

Syntax: AGED(D,N1,N2) where **D** is a date expression and **N1** and **N2** are numeric expressions representing numbers of days.

Example: To print an overdue notice if a payment is between 30 and 60 days after the order date, use the following expression:

```
IIF(AGED(ORD_DATE,30,60), "Payment is now overdue.
Please remit now to avoid a late penalty.", "")
```

ASC

Converts the first character of a string of characters to its numeric value. This function is the opposite of CHR.

Syntax: ASC(C) where **C** is a character expression.

Example: To convert the first character of a product code (PRODCODE) to its numeric value, create the field:

```
ASC(PRODCODE)
```

If the value of PRODCODE is ABC, then ASC returns the numeric value of the letter A, which is 65 (decimal).

AT

Searches for a substring within a character expression and returns the number of the starting position of the substring within the character expression or 0 if the substring is not found.

Syntax: AT(C1,C2,L) where **C1** is the substring character expression and **C2** is the character expression within which the substring is to be located. The optional logical argument **L** controls the case sensitivity of the function. When **L** is true, the function is case sensitive; when it is false, the function is case insensitive. When this argument is not provided, the function uses the case sensitivity setting in the RRW.SRT file (see Chapter 6, "Setting Defaults," for information about configuring R&R's case sensitivity).

Example: The following expression returns 11 since 'substring' starts at the 11th character in 'this is a substring':

```
AT('substring','this is a substring')
```

BLANKNUM

Generates an empty numeric value.

Syntax BLANKNUM( ), no arguments. This function will return a blank numeric value only in the following cases: a) if it is the sole function in a calculation; b) if it is used as either the second or third argument to the IIF( ) function. (If used in any other way in a formula, this function will return 0.)

Example: Suppose you have a report that consists of survey questions that require a response of A, B, C, or D in a field named RESPONSE. Create the following field to assign the values 1 through 4 to A through D and to assign any other response a blank value:

```
IIF(RESPONSE="A",1,IIF(RESPONSE="B",2,
IIF(RESPONSE="C",3,IIF(RESPONSE="D",4,BLANKNUM( )))))
```

When you create an average based on this field, R&R will ignore blank values.

CAPFIRST

Converts the first character of each word in a character string to upper case and any other character to lower case. A word is a group of contiguous characters followed by a space, a hyphen, or any user-designated characters.

Syntax: CAPFIRST(C1,C2) where **C1** is a character expression and **C2** is an optional character expression that specifies the word break character(s), the character(s) marking word separations. If **C2** is absent, R&R uses the space and hyphen characters to determine word breaks. If you supply **C2**, R&R uses only **C2** to determine word breaks (see WDCOUNT for an example).

Example: The expression: CAPFIRST("tHiS iS aN exAMple.") returns: This Is An Example.

CASE Returns a result based on the value of an item. You can use CASE instead of a nested IIF expression to return results conditionally when there is more than one condition.

Syntax: CASE(I, value-1, result-1,...,value-n, result-n, default) where all values must be of the same data type as **I** (item); and all results must be of the same data type, which may be different from the data type of the item. If **item** is equal to **value-1**, R&R returns **result-1**; if equal to **value-n**, R&R returns **result-n**, and so on. If **item** is not equal to any listed value, R&R returns default. This function selects only on the basis of equality. Memo fields may be used as results, but not as test values.

Example: To assign different strings based on the value of PRODCODE, create the field:

```
CASE(PRODCODE, "101", "Desk chair", "102",
"Printer stand", "103", "File cabinet", "")
```

This CASE expression is simpler than but equivalent to the following nested IIF expression:

```
IIF(PRODCODE = "101", "Desk chair",
IIF(PRODCODE = "102", "Printer stand",
IIF(PRODCODE = "103", "File cabinet", "")))
```

CDLL Enables calling of a DLL-based function from within an R&R report. You might use CDLL( ) when you want to write a DLL-based function to perform an operation that R&R's functions don't support, such as a trigonometric operation. CDLL( ) also gives you access from R&R to functions that are used by other elements of your application, since DLLs are available to all parts of a Windows application.

Syntax: CDLL(string1,string2,string3) where **string1** is the name of the DLL that contains the function, **string2** is the name of the function, and **string3** is an argument being passed to the DLL function.

Example: For example, the calculated field expression:

```
CDLL("CONVERTS.DLL","MILES_KILO",STR(DISTANCE))
```

uses the STR function to convert the value of DISTANCE into a character string and passes the string value to the MILES_KILO function in CONVERTS.DLL, which converts the distance in miles to kilometers.

CDLL( ) expects a boolean return value from the called DLL function: true to indicate the function executed successfully; false to indicate an error. If the DLL returns a false value, CDLL( ) returns an error string. If the DLL function executes successfully, it should overwrite its input string with the output string to be returned by the R&R CDLL( ) function. R&R passes the input and output strings using an 8000-byte buffer. (For more information about CDLL( ), see the *Developing Applications* manual.)

CDOW

Returns the name of the day of the week of a given date.

Syntax: CDOW(D) where **D** is a date expression.

Example: To print or display the day of the week for a field INITDATE, create the field:

```
CDOW(INITDATE)
```

If the value of the INITDATE field is 12-7-93, then CDOW yields "Tuesday".

CHR

Converts a numeric value to one character based on the character set of the font applied to the field; that is, constitutes a change in data type from numeric to character. You can use CHR to print or display a character for which there is no keyboard representation (for example, the bullet character). This function is the opposite of ASC.

Syntax: CHR(N) where **N** is an integer from 1 to 255.

Example: CHR(212) returns the trademark symbol (™) when the Symbol font is applied to the field.

CLOOKUP

Returns the value of a character field from another table. When you want a single character value from another table, you can create a calculated field using CLOOKUP( ) instead of setting a database relation.

Syntax: CLOOKUP(I,C1,C2,C3,[C4]) where **I** is the field you want to use as the linking field to the lookup table's index key. **C1** is the name of the character field whose value you want to retrieve from the lookup table, **C2** is the lookup table name, **C3** is the name of the lookup index file, and **C4** is an optional tag for a CDX, MDX, or WDX index. The index key must be an exact, full match to the linking field.

Example: To get the value of the COMPANY field from the RRCUST table using CUST_NO as the linking field, create a calculated field whose expression is:

```
CLOOKUP(CUST_NO,"COMPANY","C:\RR\RRCUST","C:\RR\RRCUST")
```

Note that the table and index names must have full path names unless they are in the same directory as the master table. You do not need to supply a file extension for the table name. If the index file has the default extension specified in your R&R configuration, you do not need to include the extension.

CMONTH

Returns the name of the month of a given date.

Syntax: CMONTH(D) where **D** is a date expression.

Example: To print or display the name of the month for a field INITDATE, create the field:

```
CMONTH(INITDATE)
```

If the value in the INITDATE field is 4-1-95, then CMONTH yields April.

COPY

Returns number of the report copy currently being printed.

Syntax: COPY( ), no arguments.

Example: If you are printing 12 copies of a report, you can include the number of each report copy in a Footer line. Create a calculated field that contains the COPY( ) function and insert the field on the Footer line:

```
-------- Copy ## of 12 --------
```

For the third copy, this line produces:

```
-------- Copy 3 of 12 --------
```

CTDT

Converts a date or datetime character expression to a datetime value.

Syntax: CTDT(C) where **C** is a character expression representing a date or datetime. The expression must be entered in the format specified by the Windows International date setting, using spaces, hyphens (-), slashes(/), or periods (.) as separator characters. If the year is less than 100, the 20th century date is assumed. For example, 95 is assumed to be 1995. R&R's date range is 03/01/1600 through 12/31/2400.

Example: CTDT("12/20/95") returns a datetime value of 12/20/95 12:00:00 am (when the Windows date format is set to "Short").

CTOD

Converts a date entered or stored as a character expression to date data type. If applied to an invalid date string, CTOD produces the error value. You can substitute a wildcard of * or @ for the month, day, and/or year. Since wildcard dates are used only for selection purposes (that is, in filters), R&R will display them as asterisks if you insert them on the report layout.

Syntax: CTOD(C) where **C** is a character expression representing a date. The expression must be entered in the format specified by the Windows International date setting, using spaces, hyphens (-), slashes(/), or periods (.) as separator characters. If the year is less than 100, the 20th century date is assumed. For example, 95 is assumed to be 1995. R&R's date range is 03/01/1600 through 12/31/2400.

Example: CTOD("12/20/95") returns a date value of 12/20/95 (when the Windows date format is set to "Short").

Example: Define a field to use in a query. Name the field FILTDATE:

```
CTOD(STR(INVMONTH,2)+"/*/94")
```

If INVMONTH is 7, the value of FILTDATE is then " 7/*/95" and a query can be used to select all records for the month of July, 95.

CTOS

Converts a character expression to a number of seconds using the format specified as the second argument to the function. You can use this function to convert a character field value that represents a time interval (rather than an absolute time) into a number of seconds.

Syntax: CTOS(C,*Keyword*) where **C** is a character expression representing a time interval and **Keyword** specifies how the function will interpret the input value. Valid Keywords are as follows:

Keyword	**Meaning**
hh	Hours
mm	Minutes
ss	Seconds
hh:mm	Hours:Minutes
hh:mm:ss	Hours:Minutes:Seconds
mm:ss	Minutes:Seconds

If no Keyword is specified, the default is hh:mm:ss (hours:minutes:seconds).

Example: Assume that you have a CHARSEC field that contains character values representing a number of seconds. For a CHARSEC value of 30:30, the field CTOS(charsec,mm:ss) would return a numeric value of 1830, since the **mm:ss** keyword tells CTOS to interpret 30:30 as 30 minutes:30 seconds. The field CTOS(charsec,hh:mm:ss) would return a numeric value of 109800, since the hh:mm:ss keyword tells CTOS to interpret 30:30 as 30 hours:30 minutes.

CTOT

Converts a character expression representing a time of day to a time field. You can use this function to convert a character field value that represents an absolute time into an appropriately formatted time field.

Syntax: CTOT(C) where **C** is a character expression representing a time of day. The character expression must be in either 24-hour format or in 12-hour format with am, pm, or international time suffixes.

Example: Assume that you have a CHARTIME field that contains character values representing a time of day. For a CHARTIME value of "0:20", the field CTOT(chartime) would return a time value of 12:20:00am. For a CHARTIME value of "8:20pm" the field CTOT(chartime) would return a value of 8:20:00pm.

DATE

Returns the system date at the time the report was started. This date is maintained by your computer or entered when you start your computer.

Syntax: DATE( ) no arguments.

Example: To print/display the date in a Header line, create a field SYSDATE where SYSDATE is a calculated field that contains the function DATE( ), and then insert the field into a Header line:

--------- Date printed: mm/dd/yy ---------

This line then produces:

--------- Date printed: 4/1/95 ---------

DAY

Returns the number of the day in the month.

Syntax: DAY(D) where **D** is a date expression.

Example: To print or display the day of the month for each invoice date INV_DATE, create the field:

DAY(INV_DATE)

If the value of INV_DATE is 01/16/95, DAY(INV_DATE) returns 16.

DAYSBTWN

Calculates the number of days between two dates. This date difference is derived by subtracting **D2** from **D1**.

Syntax: DAYSBTWN(D1,D2) where **D1** and **D2** are both date expressions.

Example: To calculate the number of days between the date a payment is due (DUEDATE) and the current system date DATE( ), create the field:

DAYSBTWN(DATE( ),DUEDATE)

Sample Data:

DATE( )	DUEDATE	DAYSBTWN(DATE( ),DUEDATE)
4/23/94	4/13/94	10
4/12/94	4/12/94	0
4/13/94	4/23/94	–10

DBF

Returns a character string that contains the complete path and name of the table alias argument. If no table alias argument is entered, the path and name of the master table is returned.

Syntax: DBF(A) where **A** is the alias of any currently attached table.

Example: To retrieve the full path and name for the CUSTOMER table, create the field:

DBF(CUSTOMER)

If the CUSTOMER table is on drive C in the DBASE subdirectory, DBF returns C:\DBASE\CUSTOMER.DBF.

DELETED

Returns the delete status of the record from the specified table. Returns a logical True (T) if a record is marked for deletion.

Syntax: DELETED(A) where **A** is the alias of any currently attached table.

Example: In a multi-table report that uses the tables ORDERS, CUSTOMER, ITEMS, and PRICES, create the following calculated field expression to determine the delete status of the records in CUSTOMER:

DELETED(CUSTOMER)

If the first two records of CUSTOMER have been marked for deletion and the following two records have not, R&R would report the following:

Name	Deleted?
Britten Jr., Roland	T
Adams, Abner N.	T
Keylour Jr., Harrison	F
Stern, Twila	F

DLOOKUP

Returns the value of a date field from another table. To retrieve the value of a date field from another table, you can create a calculated field using DLOOKUP( ) instead of setting a database relation.

Syntax: DLOOKUP(I,C1,C2,C3,[C4]) where **I** is the field you want to use as the linking field to the lookup table's index key, **C1** is the name of the date field whose value you want to retrieve from the lookup table, **C2** is the lookup table name, **C3** is the name of the lookup index file, and **C4** is an optional tag for a CDX, MDX, or WDX index. The index file's key must be an exact, full match to the linking field.

Example: To retrieve the value of the DATE field from the RRORDERS table using CUST_NO as the linking field, create a calculated field whose expression is:

```
DLOOKUP(CUST_NO,"DATE","C:\RR\RRORDERS","C:\RR\RRORDERS")
```

Note that the table and index names must have full path names unless they are in the same directory as the master table. You do not need to supply a file extension for the table name. If the index file has the default extension specified in your R&R configuration, you do not need to include the extension.

DOW

Converts the day-of-week of a given date expression to a number from 1 to 7. Sunday is day 1, Monday is day 2, etc.

Syntax: DOW(D) where **D** is a date expression.

Example: To convert the day of the week for a field INVDATE to its numeric form, create the field:

```
DOW(INVDATE)
```

Sample data:

INVDATE	*Day #*
01/11/92	2
01/13/93	4
01/15/93	6

DQTR

Returns the date of the first day of the calendar quarter for a given date. (See also the QTR function.)

Syntax: DQTR(D) where **D** is a date expression.

Example: To calculate the first day of the quarter for the date in a date field INITDATE, create the following field:

```
DQTR(INITDATE)
```

If the value in the INITDATE field is 5-4-95, then DQTR(INITDATE) yields 4-1-95, the first day of the second quarter.

DTEADD

Adds a specified interval to a datetime.

Syntax: DTEADD(I,N,DT) where **I** specifies the interval to be added, **N** is how many of the specified intervals to add, and **DT** is the datetime expression being added to.

The following table lists the time period that corresponds to each valid interval value. Note that for this function, y (Day of Year) and w (Weekday) are equivalent to d (Day).

Interval	*Time Period*
yyyy	Year
q	Quarter
m	Month
y	Day of Year
d	Day
w	Weekday
ww	Week
h	Hour
n	Minute
s	Second

Example: To add 5 weeks to the value of the datetime field DTEND, create a calculated field with the following expression:

DTEADD(ww,5,DTEND)

DTEDIFF

Calculates the difference between two datetime values.

Syntax: DTEDIFF(I,DT1,DT2) where **I** is the interval used to calculate the difference between **DT1** and **DT2**. Valid interval values are the same as those for the DTEADD function.

Example: To calculate the difference in days between the values of the datetime fields DTSTART and DTEND, create a calculated field DAYDIF with the following expression:

DTEDIFF(d,DTSTART,DTEND)

This expression yields the following sample results:

DTSTART	DTEND	DAYDIF
01/10/95	03/14/95	63
04/13/95	06/20/95	68

DTEPART

Returns as an integer the specified part of a given datetime expression. Valid interval values are the same as those for the DTEADD function.

Syntax: DTEPART(I,DT) where **I** is the interval and **DT** is the datetime expression.

Example: If the date portion of a datetime field named DTEND is 04/06/95, the expression DTEPART(ww,DTEND) produces a value of 14, indicating the 14th week of the year.

DTLOOKUP

Returns the value of a datetime field from another table. When you want a single datetime value from another table, you can create a calculated field using DTLOOKUP( )instead of setting a database relation.

Syntax: DTLOOKUP(I,C1,C2,C3,[C4]) where **I** is the field you want to use as the linking field to the lookup table's index key. **C1** is the name of the datetime field whose value you want to retrieve from the lookup table, **C2** is the lookup table name, **C3** is the name of the lookup index file, and **C4** is an optional tag for a CDX, MDX, or WDX index. The index key must be an exact, full match to the linking field.

Example: To get the value of the datetime field DTEND from the JOBS table using JOB_NO as the linking field, create a calculated field whose expression is:

```
DTLOOKUP(JOB_NO,"DTEND","C:\RR\JOBS","C:\RR\JOBS")
```

Note that the table and index names must have full path names unless they are in the same directory as the master table. You do not need to supply a file extension for the table name. If the index file has the default extension specified in your R&R configuration, you do not need to include the extension.

DTOC

Converts a date expression to a character expression in the format specified by the Windows International date setting. When the optional numeric argument is 1, DTOC returns a character string in the format yyyymmdd, which is useful for sorting and indexing by date.

Syntax: DTOC(D,N) where **D** is a date expression and **N** is an optional numeric expression.

Example: The expression DTOC(DATE) yields the following sample results when the Windows date format is "Short" and Day and Month leading zeros are specified:

DATE	Character Conversion
01/10/95	"01/10/95"
01/13/95	"01/13/95"

The expression DTOC(DATE,1) yields the following sample results:

DATE	Character Conversion
01/10/95	"19950110"
01/13/95	"19950113"

By using 1 as the numeric argument, you can use a simple DTOC expression like DTOC(DATE,1) to replace expressions like the following:

```
STR(YEAR(DATE))+STR(MONTH(DATE))+STR(DAY(DATE))
```

DTTOC

Converts a datetime expression to a character string.

Syntax: DTTOC(DT) where **DT** is a datetime expression.

Example: To convert the date and time values of a datetime field named DTEND to character strings, create a calculated field with the expression DTTOC(DTEND).

ERROR

Returns logical true if the input item has the error value or logical false if not. R&R represents an error value as asterisks. This function allows you to define and use your own error value. For example, you may want to print "You have an error" instead of a string of asterisks.

Syntax: ERROR(I) where **I** can be any data type.

Example: To print 0 instead of * as the error value for the numeric field AMOUNT, create the following field:

```
IIF(ERROR(AMOUNT),0,AMOUNT)
```

Use this calculated field instead of AMOUNT.

EXP

Calculates the value of e**n where **e** is the base of natural logarithms.

Syntax: EXP(N) where **N** is a numeric expression.

Example: To determine the value of e**2.000, use the following field:

EXP(2.000)

The result is 7.3891

FLIP

Exchanges the data before a specified character with the data following the character. Commonly used to flip last name with first name based on a comma between the two parts of the name. See examples.

Syntax: FLIP(C1,C2) where **C1** and **C2** are character expressions. **C1** is the character expression to flip and **C2** is the "flip character," often a comma.

When **C2** is a comma, FLIP eliminates the leading and trailing spaces for the part following the comma and puts a space between the flipped parts. See examples.

C2 may also include an asterisk before or after the flip character. The asterisk causes R&R to return only part of **C1**. Placing an asterisk after the flip character returns only the data after the flip character. Placing an asterisk before the flip character returns only the data before the flip character. See examples.

Example: To flip the data in a FULLNAME field from last-name, first-name order to first-name last-name order:

FLIP(FULLNAME,',')

Sample data:

FULLNAME	FLIP(FULLNAME,',')
Britten Jr., Roland	Roland Britten Jr.
Jefferson, Elvira A.	Elvira A. Jefferson
Adams, Abner N.	Abner N. Adams
Gladdin III, John	John Gladdin III

Example: To return only the first name of FULLNAME:

FLIP(FULLNAME, ',*')

Sample data:

FULLNAME	FLIP(FULLNAME,',*')

Britten Jr., Roland	Roland
Jefferson, Elvira A.	Elvira A.

Example: To return only the last name of FULLNAME:

FLIP(FULLNAME,'*,')

Sample data:

FULLNAME	FLIP(FULLNAME,'*,')
Britten Jr., Roland	Britten Jr.
Jefferson, Elvira A.	Jefferson

FUTURE

Returns logical True if supplied date is in the future and logical False if not. If the optional argument "1" is included, the current date is considered to be in the future.

Syntax: FUTURE(D,[1]) where **D** is a date expression and 1 is an optional numeric argument that causes the current date to be considered as in the future.

Example: To determine whether the value in a field named DUE_DATE is in the future, use the following expression:

FUTURE(DUE_DATE)

HALF

Returns the calendar half-year of the specified date. If date is in the range January 1 to June 30, this function returns 1; if in the range July 1 to December 31, it returns 2.

Syntax: HALF(D) where **D** is a date expression.

HISCOPE

Returns the scope "ending value" as a character string. If high scope is end-of-file, in other words if the ending value is blank, HISCOPE returns a null string.

Syntax: HISCOPE( ) no arguments

Example: To print the starting and ending dates included in a report in the report Header, use HISCOPE and LOSCOPE. First, create two calculated fields, one for each value. Then, insert the fields in a Header line.

Profit Margins from mm/dd/yy to mm/dd/yy

IIF

Evaluates a condition and returns one of two values, depending on whether the condition is true or false.

Syntax: IIF(condition,true-value,false-value) where **condition** can be any data type. (See the Notes below for an explanation of how non-

logical expressions are evaluated as True or False.) The values returned, **true-value** and **false-value**, must be the same data type, but need not be the same data type as **condition**. **True-value** and **false-value** can be memo fields. By using a memo field as the **true-value** and empty quotation marks ("") as the **false-value**, you can use IIF to return an empty memo field without having an empty field in your text memo file.

If **condition** involves comparison of character expressions, the result that IIF returns may depend on how R&R is configured to handle case sensitivity (see Chapter 6, "Setting Defaults," for information about configuring R&R's case sensitivity).

Note: If **condition** is a logical expression, IIF returns **true-value** if the expression is TRUE and **false-value** if the expression is FALSE.

Example: To print the title of each name in a mailing-labels report based on the gender of the addressee, create the field:

```
IIF(SEX = 'F','Ms. ', 'Mr. ')
```

Note: If **condition** is a character expression that evaluates to NULL or empty (""), IIF returns **false-value**. If the expression is not empty, IIF returns **true-value**.

Example: To print "Missing" if the character field TITLE is empty and "Okay" if it contains information:

```
IIF(TITLE,"Okay","Missing")
```

Note: If **condition** is a numeric expression, IIF returns **false-value** if the expression evaluates to NULL or 0 and **true-value** if the expression evaluates to non-zero.

Example: To print memo field THANKS if DUE is 0 or memo field SENDCASH, if not:

```
IIF(DUE,SENDCASH,THANKS)
```

Note: If **condition** is a date expression that evaluates to empty (for example, a date field that contains no date within it), IIF returns **false-value**; otherwise, IIF returns **true-value**.

Example: To return logical T if DATE contains data or logical F if not:

```
IIF(DATE,.T.,.F.)
```

Note: If **condition** is a memo field, IIF returns **true-value** if the memo field contains characters, even if the characters are only spaces and tabs. Otherwise, IIF returns **false-value**.

Example: To return the string "See Comment" if COMMENT contains characters and the string "No Comment" if COMMENT does not contain characters:

```
IIF(COMMENT, "See Comment", "No Comment")
```

INLIST

Compares a value with each item in a list of values to determine if the value is included in the list. R&R compares the first value with each value in the list. If it doesn't find a match, INLIST returns 0. If it does find a match, it returns a number corresponding to the position of the value in the list (that is, 1 for the first value, 2 for the second value, and so on).

Syntax: INLIST(value,list–value–1,...,list–value–n) where all arguments must be the same data type: character, numeric, date, or logical expressions, and where there must be at least one **list–value**. If the values are character expressions, matching is not sensitive to upper or lower case, unless you have configured R&R for case sensitivity (see Chapter 6, "Setting Defaults," for information about configuring R&R's case sensitivity).

Example: To test each record in the ITEMS table to determine if its product code field (PRODNO) is one of three items in a list, create the field:

```
INLIST(PRODNO,'901','902','903')
```

Sample data:

PRODNO	INLIST(PRODNO...)
904	0
903	3
901	1
911	0
902	2

INRANGE

Determines if a value is within a specified range of values. Returns logical True if the value is equal to or greater than the lowest value and equal to or less than the highest value. Returns False if not.

Syntax: INRANGE(value,low–value,high–value) where all arguments must be the same data type; character, numeric, date, or logical expressions. If the values are character expressions, matching is not sensitive to upper or lower case, unless you have configured R&R for case sensitivity (see Chapter 6, "Setting Defaults," for information about configuring R&R's case sensitivity).

Example: To determine if the price of each item in a PRICES table is within the range of 4.95 and 6.95, create the field:

INRANGE(LISTPRICE,4.95,6.95)

Sample data:

PRODNO	LISTPRICE	INRANGE(LISTPRICE...)
903	12.95	F
901	6.95	T
902	5.00	T
904	4.95	T

INT

Converts a numeric expression to an integer by discarding all digits to the right of the decimal point.

Syntax: INT(N) where **N** is a numeric expression.

Example: INT(NUMBERS)

Sample data:

NUMBERS	INT(NUMBERS)
14.345	14
45.543	45
62.987	62

ISALPHA

Determines if the first character of a character expression is any upper or lower case letter, including non-English letters. Returns logical True if yes; False if not.

Syntax: ISALPHA(C) where **C** is a character expression.

Example: To determine in which records the character field PARTNO begins with a letter, create the field:

ISALPHA(PARTNO)

Sample data:

PARTNO	ISALPHA(PARTNO)
abcd	T
1bc2	F
z101	T

ISBLANK

Determines whether an expression contains the empty value. Returns logical True if expression contains empty value; otherwise returns logical False.

Syntax: ISBLANK(I) where **I** can be any data type except logical.

Example: To display or print an error in the report if the CUST_NO field contains an empty value, create the field:

```
IIF(ISBLANK(CUST_NO),"Error",STR(CUST_NO))
```

ISLOWER

Determines if the first character of a character expression is a lower case letter, including non-English letters. Returns logical True if yes; False if not.

Syntax: ISLOWER(C) where **C** is a character expression.

Example: To determine in which records the character field PARTNO begins with a lower case letter, create the field:

```
ISLOWER(PARTNO)
```

Sample data:

PARTNO	ISLOWER(PARTNO)
ABCDE	F
abcde	T

ISUPPER

Determines if the first character of a character expression is an upper case letter, including non-English letters. Returns logical True if yes; False if not.

Syntax: ISUPPER(C) where **C** is a character expression.

Example: To determine in which records the first character of character field PARTNO is in upper case, create the field:

```
ISUPPER(PARTNO)
```

Sample data:

PARTNO	ISUPPER(PARTNO)
AbCdE	T
aBCde	F

LEFT

Selects a specified number of characters starting from the left-most character of a character expression.

Syntax: LEFT(C,N) where **C** is a character expression and **N** is an integer numeric expression representing the number of characters to select.

Example: To create a field PARTNO to contain the first 4 characters of the field PARTID, use LEFT as follows:

```
LEFT(PARTID,4)
```

Sample data:

PARTID	PARTNO
abcdyy	abcd
abefxx	abef

LEN

Calculates the length of a character expression. Trailing spaces, if any, are counted. The length of a null string is zero.

Syntax: LEN(C) where **C** is a character expression.

Example: To produce a list of authors and titles with a dot leader (...) between them, use LEN with REPLICATE and RTRIM:

```
AUTHOR–REPLICATE(".",60–LEN
(AUTHOR–RTRIM(TITLE)))+RTRIM(TITLE)
```

Sample report (using a monospaced font):

```
Barr.............Handbook of Artificial Intelligence
Nierenberg.........How to Read a Person Like a Book
Wiener.................................Cybernetics
```

LIBNAME

Returns the full path and name of the current report library file.

Syntax: LIBNAME( ), no arguments.

Example: To include the complete path and name of the report library file, create a calculated field consisting only of LIBNAME( ). Then insert the field in an appropriate area of the report.

LLOOKUP

Returns the value of a logical field from another table. To retrieve the value of a logical field from another table, you can create a calculated field using LLOOKUP( ) instead of setting a database relation.

Syntax: LLOOKUP(I,C1,C2,C3,[C4]) where **I** is the field you want to use as the linking field to the lookup table's index key, **C1** is the name of the logical field whose value you want to retrieve from the lookup table, **C2** is the lookup table name, **C3** is the name of the lookup index file, and **C4** is an optional tag for a CDX, MDX, or WDX index. The index file's key must be an exact, full match to the linking field.

Example: To find out whether a customer has an approved credit line, you can get the value of the CREDIT field from the RRCUST table using CUST_NO as the linking field. Create a calculated field whose expression is:

```
LLOOKUP(CUST_NO,"CREDIT","C:\RR\RRCUST","C:\RR\RRCUST")
```

Note that the table and index names must have full path names unless they are in the same directory as the master table. You do not need to supply a file extension for the table name. If the index file has the default extension specified in your R&R configuration, you do not need to include the extension.

LOG

Calculates the natural logarithm of a given number. If applied to a negative number or zero, LOG produces the error value.

Syntax: LOG(N) where **N** is a numeric expression.

Example: To determine the natural log of 7.3891, create the field LOG(7.3891), which returns a value of 2.0000. To determine the natural log of 8 times 12, create the field LOG(8*12), which returns a value of 4.5643

LOSCOPE

Returns the scope "starting value" as a character string. If low scope is beginning-of-file, in other words if the starting value is blank, LOSCOPE returns a null string.

Syntax: LOSCOPE( ) no arguments.

Example: See HISCOPE.

LOWER

Converts upper case letters to lower case letters; opposite of UPPER.

Syntax: LOWER(C) where **C** is a character expression.

Example: To convert the upper case letters in a character field PRODCODE, create the field:

```
LOWER(PRODCODE)
```

Sample data:

PRODCODE	LOWER(PRODCODE)
ABCD	abcd
ABCE	abce

LTRIM

Removes the leading spaces from a character expression. Use this function to remove leading spaces that result from using the STR function, for example.

Syntax: LTRIM(C) where **C** is a character expression.

Example: To left justify the page number field (PAGE) in a document index, create the calculated field:

LTRIM(STR(PAGE,5))

Sample data:

Topic	Page
Word truncation	12
Word-wrap	8

LUPDATE

Returns the date that the specified table was last updated. If the table alias is omitted, the master table date is returned.

Syntax: LUPDATE(A) where **A** is the alias of any currently attached table.

Example: To include the date the table with the alias PAYMNTS was last updated in a report of payments received so far this month, create a calculated field with the expression:

LUPDATE(PAYMNTS)

You might then insert this field in the Title or Page Header band, introduced by the text field "Last update on:"

MAX

Determines the higher value of two numeric expressions.

Syntax: MAX(N1,N2) where **N1** and **N2** are numeric expressions.

Example: To prevent the discounted price from going below $10, create the field:

MAX(DISCPRICE,10)

MIN

Determines the lower value of two numeric expressions.

Syntax: MIN(N1,N2) where **N1** and **N2** are numeric expressions.

Example: To prevent discounts of greater than 50%, create the field:

MIN(DISCRATE,50)

MOD

Calculates the remainder of a division. Returns the number representing the remainder of one numeric expression divided by another. If used with zero as the denominator, MOD produces the error value.

Syntax: MOD(N1,N2) where **N1** and **N2** are numeric expressions. **N1** is the dividend, and **N2** is the divisor.

Example: To convert data in a field SIZE that is expressed in inches to data expressed in yards and inches, create the field:

MOD(SIZE,36)

Sample data:

Data	Yards	Inches
SIZE	INT(SIZE/36)	MOD(SIZE,36)
38	1	2
72	2	0
7	0	7
119	3	11

MONLEN

Returns the number of days in the month for the specified date.

Syntax: MONLEN(D) where **D** is a date expression.

Example: The expression MONLEN({9/01/1994}) returns 30, the number of days in the month of September.

MONSBTWN

Calculates the number of full months between two dates. This date difference is derived by subtracting **D2** from **D1**.

Syntax: MONSBTWN(D1,D2) where **D1** and **D2** are date expressions.

Example: To calculate the number of months between the starting date of a contract (STARTDATE) and the ending date (ENDDATE), create the field:

MONSBTWN(ENDDATE,STARTDATE)

Sample data:

ENDDATE	STARTDATE	MONSBTWN(ENDDATE,STARTDATE)
03/14/94	12/10/93	3
03/02/94	12/10/93	2
12/10/93	03/10/94	–3

MONTH

Returns the number of the month of a date expression.

Syntax: MONTH(D) where **D** is a date expression

Example: To print the month number of dates in the date field STARTDATE, create the field:

MONTH(STARTDATE)

Sample data:

STARTDATE	MONTH(STARTDATE)
12/10/94	12
04/23/94	4

NDOW

Returns the next date of the day-of-week number that is supplied as its second argument.

Syntax: NDOW(D,N) where **D** is a date expression and **N** is a numeric expression specifying day-of-week (with 1 being Sunday, 2 Monday, etc.).

Example: To determine the date of the next Saturday following STARTDATE, use an expression like the following:

```
NDOW(STARTDATE,7)
```

NLOOKUP

Returns the value of a numeric field from another table. To retrieve the value of a date field from another table, you can create a calculated field using NLOOKUP( ) instead of setting a database relation.

Syntax: NLOOKUP(I,C1,C2,C3,C4) where **I** is the field you want to use as the linking field to the lookup table's index key, **C1** is the name of the numeric field whose value you want to retrieve from the lookup table, **C2** is the lookup table name, **C3** is the name of the lookup index file, and **C4** is an optional tag for a CDX, MDX, or WDX index. The index file's key must be an exact, full match to the linking field.

Example: To retrieve the value of the PRICE field from the RRPRICES table using PRODUCT_NO as the linking field, create a calculated field whose expression is:

```
NLOOKUP(PRODUCT_NO,"PRICE","C:\RR\RRPRICES","C:\RR\PRICES")
```

Note that the table and index names must have full path names unless they are in the same directory as the master table. You do not need to supply a file extension for the table name. If the index file has the default extension specified in your R&R configuration, you do not need to include the extension.

OVER

Returns logical True if more than a specified number of days have passed since the supplied date.

Syntax: OVER(D,N) where **D** is a date expression and **N** is a numeric expression specifying number of days.

Example: To print a message indicating that a payment is past due, use an expression like the following:

```
IIF(OVER(DUE_DATE,90),"Payment is now overdue 90 days.
A 10 percent penalty will be added if not paid within 10 days."," ")
```

As a result, the overdue notice will print only for those values of DUE_DATE that are more than 90 days in the past.

PAGENO

Returns current report page number.

Syntax: PAGENO( ) no arguments.

Example: To print/display a page number in a Footer line of a report, create a field PAGENUM to contain PAGENO( ) and use it in a Footer line:

```
----------   Page: 99   ----------
```

This line then produces:

```
----------   Page: 1   ----------
```

PAST

Returns logical True if supplied date is in the past and logical False if not. If the optional argument "1" is included, the current date is considered to be in the past.

Syntax: PAST(D,[1]) where **D** is a date expression and **1** is an optional numeric argument that causes the current date to be considered as in the past.

Example: To determine whether the value in a field named DUE_DATE is in the past, use the following expression:

```
PAST(DUE_DATE)
```

PDOW

Returns the previous date of the day-of-week number that is supplied as its second argument.

Syntax: PDOW(D,N) where **D** is a date expression and **N** is a numeric expression specifying day-of-week (with 1 being Sunday, 2 Monday, etc.).

Example: To determine the date of the Monday immediately preceding STARTDATE, use an expression like the following:

```
PDOW(STARTDATE,2)
```

If the value of STARTDATE is 6/02/94, this expression will produce a value of 05/30/94, the date of the Monday preceding 6/02/94.

PREVIOUS

Returns the value of the specified field in the previous composite record. (Note that records that do not satisfy the filter currently in force are not accessible with this function.) This function is useful for suppressing repeating data in fields other than group fields and for performing calculations with fields in different composite records.

Syntax: PREVIOUS(field name) where **field name** is the name of any field in the report.

Example: If your orders table contains fields for customer number (CUSTNO), order number (ORDERNO), and order date (DATE), you can use the PREVIOUS function to calculate the number of days since any customer's previous order, assuming that orders are sorted by date within customer.

The expression for a DAYSINCE field might be:

```
IIF(PREVIOUS(CUSTNO)=CUSTNO,DATE–PREV(DATE),0)
```

Translated, this expression means that if the customer number is the same as the customer number in the previous composite record (in other words if R&R is reading a record for a second or subsequent order by the same customer), return a value that is the current order date minus the order date in the previous composite record. Otherwise, return 0.

QTR

Calculates the number of the calendar quarter of a date expression. (See the DQTR function.)

Syntax: QTR(D) where **D** is a date expression. 1/1 to 3/31 is Quarter 1; 4/1 to 6/30 is Quarter 2; 7/1 to 9/30 is Quarter 3; 10/1 to 12/31 is Quarter 4.

Example: To calculate the quarter number of each date in the STARTDATE field, create the field:

QTR(STARTDATE)

Sample data:

STARTDATE	QTR(STARTDATE)
12/10/92	4
04/23/93	2
09/18/93	3

QUERY

Returns the current query expression.

Syntax: QUERY( ) no arguments.

Example: To print or display the current query, create a calculated field whose expression is QUERY( ) and insert the field. R&R will print the query text using a word-wrapped, left-aligned format with a 50-character width. If no query is specified for the report, this function returns "Include all records."

Note that this function provides text only for the current query definition; you cannot use QUERY with any other function or with operators.

RECCOUNT

Returns the number of records in the specified table.

Syntax: RECCOUNT(A) where **A** is the alias of one of the tables used in the report. Alias is a required field.

Example: In a report that uses the ITEMS table, create the calculated field:

RECCOUNT(ITEMS)

Then use the field in a Header line to report on the number of records in the table:

The ITEMS table has 40 records

where the number 40 is the result of the calculated field expression above.

RECNO

Without an argument, RECNO( ) identifies the current composite record number. Composite records are numbered sequentially (starting with 1) after R&R does a sort and/or a query and reads all related records. RECNO(A) returns the record number from the table with the specified alias.

Syntax: RECNO(A) where **A** is the alias of one of the tables used in the report. Alias is optional.

Example: After setting a lookup relation from ORDERS to CUST and a Scan relation from ORDERS to ITEMS, create the calculated fields:

Record1 = RECNO( )
Record2 = RECNO(CUST)
Record3 = RECNO(ITEMS)

Sample data:

Customer	Item	Record1	Record2	Record3
Britten	Nut	1	1	1
Britten	Bolt	2	1	2
Adams	Nut	3	3	1
Adams	Bolt	4	3	2
Adams	Wrench	5	3	3

REPLICATE

Repeats a character expression a specified number of times.

Syntax: REPLICATE(C,N) where **C** is the character expression that is to be copied and **N** is the numeric expression representing the number of copies. Maximum number of characters is 254.

Example: To print a line of asterisks after each record, create the field REPLICATE('*',50) and include it in a Record line after a line of fields, such as in the following report layout:

```
Customer Name           Order Date  Amount
<xxxxxxxxxxxxxxxxxxxxx mm/dd/yy  ($999.99)
<xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Sample report output:

```
Customer Name           Order Date  Amount
Britten, Jr., Roland    01/10/94   $216.00
****************************************
Gladdin III, John       01/10/94    $96.00
****************************************
Adams, Abner N.         01/10/94   $144.00
```

REPNAME

Returns as a character string the current report name.

Syntax: REPNAME( ), no arguments.

Example: To print the report name as a title, create a calculated field consisting only of REPNAME( ). Then place the field in the Title band of the report.

RIGHT

Selects the right-most specified number of characters of a character expression.

Syntax: RIGHT(C,N) where **C** is a character expression and **N** is a numeric expression that defines the number of characters to select.

Example: To create a PARTNO field to contain the last 5 characters of the 8-character PARTID field, use RIGHT as follows:

```
RIGHT(PARTID,5)
```

Sample data:

PARTID	PARTNO
ababyyyy	byyyy
ababzzzz	bzzzz
abab	abab

RIPARAM

In reports run with Runtime, returns as a character string the value of a specified character field in the runtime control table or text control file. Returns asterisks for reports run interactively.

Syntax: RIPARAM(C), where **C** is a character string that is the name of a character field in the runtime control table or file.

Example: To include the user-supplied title stored in the TITLE field in the runtime control table, insert a field with the following expression on the Title line of your report:

```
RIPARAM("TITLE")
```

ROUND

Rounds a numeric expression to a specified number of decimal places.

Syntax: ROUND(N1,N2) where **N1** is the numeric expression to be rounded and **N2** is the number of decimal places. Note that **N2** may be signed (e.g., ROUND(260,–2) = 300.)

Example: To round numbers in a numeric field NUMBERS to 0 decimal places, create the field:

```
ROUND(NUMBERS,0)
```

Sample data:

NUMBERS	ROUND(NUMBERS,0)
23453.3455	23453 (rounds down)
98789.7656	98790 (rounds up)

RRUNIN

With R&R Runtime, returns 1 if a report is run using a text control file; if using a control table, returns the record number of the report in that control table.

Syntax: RRUNIN( ), no arguments.

Example: If you develop a runtime report that uses RIPARAM to incorporate user-supplied text in the title, you can use IIF and RRUNIN to specify an alternate title when you run the report interactively. To print the text “Report Title” instead of the user-specified text supplied by RIPARAM, create the field:

```
IIF(RRUNIN( ),RIPARAM(TITLE),"Report Title")
```

RTRIM Removes the trailing spaces of a character expression. Since R&R automatically trims trailing spaces off all character fields when printing, this function is normally used only to concatenate character fields or to find the length of the actual data in a database field.

Syntax: RTRIM(C) where **C** is a character expression.

Example: In the CUSTOMER table, to print the state next to the city with only a comma and one space between the two fields, create the following field:

```
RTRIM(CITY)+',  '+STATE
```

R&R prints the field as follows:

```
Detroit, MI
Columbia, MO
Portland, OR
```

SCANNING Returns a true value if the specified table is being scanned; otherwise returns false. This function is useful for sorting, grouping, and conditionally printing data in multiple-scan reports (see Chapter 17, "Creating Multiple-Scan Reports").

Syntax: SCANNING(A) where **A** is the alias of any currently attached table.

Example: If you have all of your charges information in one table whose alias is CHARGES and all of your payments information in another table whose alias is PAYMNTS, you can use the SCANNING function to create a field that will contain either a charge date (CHGDATE) or a payment date (PAYDATE), depending on which table is being scanned. The expression for this field, which might be called SORTDATE, will be:

```
IIF(SCANNING(CHARGES),CHGDATE,PAYDATE)
```

When you insert this field on a report, the charge date will print when the charges table is being scanned, and the pay date will print when the payments table is being scanned. You can also use the field as a sort field, to sort charge and payment amounts together by date.

SOUNDEX Returns a four-character string that represents the way the input expression sounds. If the character expression is longer than one word, R&R looks only at the first word in the expression. This function allows you to locate records based on how they sound,

providing that the first letter is the same (e.g., Myers and Meyers, but not Copper and Kopper).

The conventions SOUNDEX uses to represent the sound of a word are as follows:

1. The first letter of the word is retained.
2. All occurrences of a, e, h, i, o, u, w, and y in positions other than the first position are dropped.
3. The remaining letters after the first letter are assigned the following numbers:

b, f, p, v	1
c, g, j, k, q, s, x, z	2
d, t	3
l	4
m, n	5
r	6

4. Repeated, adjacent occurrences of letters that have been assigned the same number are omitted.

For example, following these conventions, SOUNDEX represents both “Lauren” and “Loren” as L650.

Syntax: SOUNDEX(C1,C2) where **C1** is a character expression and **C2** is an optional character expression that represents the word break character(s), the character(s) marking word separations. If **C2** is absent, R&R uses the space and hyphen characters to determine word breaks. If you supply **C2**, R&R uses only **C2** to determine word breaks (see WDCOUNT for an example).

Example: In a report based on a customer table containing names that could have been misspelled due to poor voice transmission, you can use this function to find a customer’s record even though the name may have been misspelled.

To select all records where the customer’s last name sounds like Meyers, create two calculated fields — one with the expression SOUNDEX("Meyers"), the other with the expression SOUNDEX(LASTNAME).

Then define a query that selects all records where the values of these calculated fields are equal. The query will select records with LASTNAME values of Meyers, Myers, and Miers.

SPACE Produces a character expression made up of a specified number of spaces (blanks).

Syntax: SPACE(N) where **N** is a numeric expression representing the number of spaces desired.

Example: To create an underlined blank space on a report, create a calculated field whose expression is SPACE(99). Insert this field on your report layout and use Format ⇒ Font to apply the underscored style. Then use Format ⇒ Field to specify the desired width of the field. As long as the field is left-aligned, centered, or right-aligned, your report will contain an underline of the specified width.

SPELLNUM Spells the integer part of a numeric expression. The first character of the resulting character string is in upper case; all other characters are in lower case. If the number is negative, R&R returns a string that begins with the word "Minus". If the integer part of the number is 0 or if the field is blank, R&R returns the string "Zero". If the string exceeds the maximum

string length of 254 characters, R&R returns a string of asterisks.

Syntax: SPELLNUM(N) where **N** is a numeric expression.

Example: To convert numbers in field NUMBERS into their English equivalents, create the field:

SPELLNUM(NUMBERS)

Sample data:

NUMBERS	SPELLNUM(NUMBERS)
0.00	Zero
0.99	Zero
25.01	Twenty-five
–5.99	Minus five
Blank	Zero

SQRT

Calculates the square root of a non-negative numeric expression. If applied to a negative number, SQRT produces the error value.

Syntax: SQRT(N) where **N** is a numeric expression.

Example: To print the square root of the contents of the numeric field NUMBERS, create the field:

SQRT(NUMBERS)

Sample data:

NUMBERS	SQRT(NUMBERS)
9	3
64	8
1024	32

STOC

Converts a numeric value representing a number of seconds to a character string.

Syntax: STOC(N, *Keyword*) where **N** is a numeric expression representing a number of seconds and **Keyword** specifies the output format.

Valid Keywords are as follows:

Keyword	**Meaning**
ss	Seconds
mm	Minutes
hh	Hours
mm:ss	Minutes:Seconds
hh:mm	Hours:Minutes
hh:mm:ss	Hours:Minutes:Seconds

If no Keyword is specified, the default is hh:mm:ss (hours:minutes:seconds).

Example: Assume that you have a field SECFIELD that contains numeric values representing a number of seconds. You can use STOC to create a calculated field CHARSEC to convert the numeric values to character strings formatted according to the specified Keyword.

Sample data:

CHARSEC Expression	SECFIELD Value	Output
STOC(secfield,ss)	109832	109832
STOC(secfield,mm)	109832	1830
STOC(secfield,mm:ss)	109832	1830:32
STOC(secfield,hh)	109832	30
STOC(secfield,hh:mm:ss)	109832	30:30:32
STOC(secfield,ss)	5	5
STOC(secfield,mm:ss)	5	0:05
STOC(secfield,hh:mm:ss)	5	0:00:05

STR

Converts a numeric expression to a character expression. This function is the opposite of VAL; that is, it constitutes a change in data type from numeric to character.

Syntax: STR(N1,N2,N3) where all arguments are numeric expressions. **N1** is a value to be converted and is required. **N2** is an optional argument representing the length of the string in number of characters. **N3** is an optional argument representing the number of decimal places.

If the value will not fit in the specified length, the STR function returns a string of asterisks instead.

If you omit the argument **N2**, STR uses 10 as the default length. If you omit the argument **N3**, STR uses 0 as the default number of decimal places.

If the value has more decimal places than specified in **N3**, STR rounds off the number to **N3** decimal places.

Example: To convert a product code field (PRODCODE) from a numeric to a character expression, create the field:

STR(PRODCODE)

Sample data:

PRODCODE	STR(PRODCODE)
12345	" 12345"
9912567	" 9912567"

(default length of 10 is used)

STRCOUNT

Returns the number of times a substring occurs within a character expression.

Syntax: STRCOUNT(C1,C2,L) where **C1** and **C2** are character expressions and **C1** is a character substring of **C2**. The optional logical expression **L** controls the case sensitivity of the function. When **L** is true, the function is case sensitive; when it is false, the function is case insensitive. Without the third argument, the function uses the case sensitivity setting in the RRW.SRT file (see Chapter 6, "Setting Defaults," for information about configuring R&R's case sensitivity).

Example: You might have a ten-character field (RESPONSE) that contains a string of the letters Y and N in either upper or lower case representing Yes and No responses to a questionnaire. To count the number of Yes responses, create a calculated field whose expression is STRCOUNT("Y",RESPONSE).

STRREP

Replaces each occurrence of a character substring with another and returns the full character string.

Syntax: STRREP(C1,C2,C3,L) where **C1** is the full character expression, **C2** is the substring to be replaced, and **C3** is the replacement substring. The optional logical expression **L** controls the case sensitivity of the function. When **L** is true, the function is case sensitive; when it is false, the function is case insensitive. When this argument is not provided, the function uses the case sensitivity setting in the RRW.SRT file (see Chapter 6, "Setting Defaults," for information about configuring R&R's case sensitivity).

Example: To replace every occurrence of the text "CDS" with "Concentric Data Systems" in a character field named COMMENT, create a field with the following expression:

```
STRREP(COMMENT,"CDS","Concentric Data Systems")
```

STRSEARCH

Returns the beginning position of the *n*th occurrence of a character substring.

Syntax: STRSEARCH(C1,C2,N,L) where **C1** is the substring, **C2** is the full character string, and **N** is the number of the occurrence. If **N** is

negative, STRSEARCH begins its search for the substring's position at the end of the character string. (Note that the position of the first character in a string is 1, not 0.)

The optional logical expression **L** controls the case sensitivity of the function. When **L** is true, the function is case sensitive; when it is false, the function is case insensitive. When this argument is not provided, the function uses the case sensitivity setting in the RRW.SRT file (see Chapter 6, "Setting Defaults," for information about configuring R&R's case sensitivity).

Example: To extract the nickname from a BOXER_NAME character field, create a field with the following expression:

```
SUBSTR(BOXER_NAME,STRSEARCH("(",BOXER_NAME,1)+1,
STRSEARCH(")",BOXER_NAME,1)–STRSEARCH("(",BOXER_NAME,1)–1)
```

For a value of BOXER_NAME such as "Billy (The Bruiser) Johnson," this field will return : The Bruiser.

STUFF

Replaces any part of a character expression.

Syntax: STUFF(C1,N1,N2,C2) where **C1** and **C2** are character expressions and **N1** and **N2** are numeric expressions. **C1** is the expression to be changed. **N1** is the starting position of the replacement. **N2** is the number of characters of **C1** to be removed. **C2** is the replacement expression. **C2** need not be the same length as **N2**.

Example: To change four characters of PRODID to BB, starting with the third character, create the field:

```
STUFF(PRODID,3,4,'BB')
```

Sample data:

PRODID	STUFF(PRODID,3,4,'BB')
JTddddT	JTBBT
JTdefgT	JTBBT

SUBDAYS

Calculates a date by subtracting a number of days from a date. Note that you can also subtract dates by using the – operator.

Syntax: SUBDAYS(D,N) where **D** is a date expression and **N** is a numeric expression representing number of days to be subtracted.

Example: To back-schedule the beginning date of a project based on the closing date (FINALDATE) and number of days (20) the project will take, create the field:

SUBDAYS(FINALDATE,20)

If FINALDATE is 05/21/94, then this function yields 05/01/94 as the starting date for the project.

SUBMONS

Calculates a date by subtracting a number of months from a date.

Syntax: SUBMONS(D,N) where **D** is a date expression and **N** is a numeric expression representing number of months to be subtracted.

Example: To back-schedule the beginning date of a project based on the closing date (FINALDATE) and number of months (24) the project will take, create the field:

SUBMONS(FINALDATE,24)

If FINALDATE is 01/10/96, then this function yields 01/10/94 as the starting date for the project.

SUBSTR

Extracts a substring from a character expression.

Syntax: SUBSTR(C,N1,N2) where **C** is the character expression from which you want to extract a substring. **N1** and **N2** are numeric expressions. **N1** is the starting position for the extraction. **N2** is the length of the extraction in number of characters and is an optional argument. If you omit **N2**, the substring extends to the end of **C**.

Example: Suppose you want to extract the 4 characters representing product code from an 8-character product number field (PRODID) and that these 4 characters begin with the third character in the product number.

To do this, create the field:

SUBSTR(PRODID,3,4)

Sample data:

PRODID	SUBSTR(PRODID,3,4)
JT3A2378	3A23
JT4B6895	4B68

SUBWKS

Subtracts a number of weeks from a date.

Syntax: SUBWKS(D,N) where **D** is a date expression and **N** is a numeric expression representing number of weeks.

Example: To back-schedule the beginning date of a project based on the closing date (FINALDATE) and number of weeks (10) the project will take, create the field:

SUBWKS(FINALDATE,10)

If FINALDATE is 4/1/93, then this function yields 1/22/93 as the starting date for the project.

SUBYRS

Subtracts a number of years from a date.

Syntax: SUBYRS(D,N) where **D** is a date expression and **N** is a numeric expression representing number of years.

Example: Stock options are 100% vested four years from the date of grant. Records in the table contain the 100% vested date, DATEVEST. SUBYRS(DATEVEST,4) gives the date the option was granted.

TIME

Returns the character representation of the system time at the time the report was started. The time is in 24 hour format, hh:mm:ss.

Syntax: TIME( ), no arguments.

Example: To print the system time in a Header line, create the field SYSTIME to contain the TIME function:

------ TIME: < xxxxxx ------

yields the following:

------ TIME: 03:05:51 ------

TODATE

Converts a datetime expression to a date.

Syntax: TODATE(DT), where **DT** is a datetime expression.

Example: You can use this function to extract just the date portion of a datetime expression. For example, if one of the values of DTEND is 04/06/95 03:20:45, the expression TODATE(DTEND) would return 04/06/95 (omitting the time portion of the datetime value).

TOTIME

Converts a datetime expression to a time.

Syntax: TOTIME(DT), where **DT** is a datetime expression.

Example: You can use this function to extract just the time portion of a datetime expression. For example, if one of the values of DTEND is 04/06/95 03:20:45, the expression TODATE(DTEND) would return 03:20:45 (omitting the date portion of the datetime value).

TRANSFORM

Returns formatted character data that results from applying a picture to a numeric or character expression.

Syntax: TRANSFORM(C1 or N,C2) where the first argument is either **C1**, a character expression, or **N**, a numeric expression. **C2** is a character expression that contains the format to be applied to the first argument. The format expression has two parts, a function and a template. At least one must be present. If both are present, the function is first.

Function — A function begins with an @ character, followed by one or more character symbols, and ends with a space character (' '). For example, '@C '. Figure 10.2 lists the functions available.

Template — A template comprises a string of special characters, each of which represents an output character position. A template can also contain characters that are to be output literally. Figure 10.3 lists the special characters that can be used to form a template.

Use	*To*	*Sample Output*
C	Output CR (credit) after a positive number; e.g., TRANSFORM(123,"@c ")	" 123.00 CR"
X	Output DB (debit) after a negative number; e.g., TRANSFORM(–123,"@x ")	" 123.00 DB"
(	Enclose Negative numbers in paren-theses; e.g., TRANSFORM(–123,"@(")	" (123.00)"
B	Left justify numeric data; e.g., TRANSFORM(123456,"@b ")	"123456.00 "
Z	Output a zero value as spaces (blanks); e.g., TRANSFORM(0,"@z ")	" "
!	Convert lower case letters to upper case; e.g., TRANSFORM("Cole","@! ")	"COLE"
R	Avoid replacing input characters with literal characters from the template; applies to character expressions only e.g., TRANSFORM("ABC", "@R X.X.X.") versus TRANSFORM("ABC","X.X.X.")	"A.B.C" "A.C"

Figure 10.2 TRANSFORM Function

Notes:

1. R&R ignores invalid function characters. For example, "@c3 " is treated the same as "@c ".

2. When R&R encounters mutually exclusive characters such as "@(x ", the last function specified overrides any preceding function; i.e., "@(x " behaves as "@x ".

Template Character	***Sample Action and Example***	***Output***
Numeric Data		
9 or #	Substitutes next digit of number, TRANSFORM(123,'99999')	" 123"
$ or *	Defines “fill character”; used in place of leading blanks, TRANSFORM(12345,"$9,999,999") TRANSFORM(12345,"*9,999,999")	"$$$$12,345" "****12,345"
.	Represents position of decimal point, TRANSFORM(12345,"999,999.99")	" 12,345.00"
,	Represents position of comma (or fill character), TRANSFORM(12345,"9,999,999")	" 12,345"
Character Data		
X or x	Substitutes next character of data, TRANSFORM("Anderson","x x x xx")	"A d r on"
!	Substitutes upper case equivalent of next character of data, TRANSFORM("Coleman","!!!!")	"COLE"

Figure 10.3 TRANSFORM Template Characters

Notes:

1. For numeric data, when a function is given without template characters, R&R normally outputs ten integer and two decimal places.
2. For character data, R&R treats 9, #, $, *, . and , literally when they appear in the template. If the format argument contains "@R ", literals do not replace characters from the input argument.

   For example:
   TRANSFORM("Henderson",'xxxxx.xx')
   produces "Hende.so", but
   TRANSFORM("Henderson",'@R xxxxx.xx')
   produces "Hende.rs".

TRIM Removes trailing spaces in a character expression. This function is identical to RTRIM.

Syntax: TRIM(C) where **C** is a character expression.

Example: See RTRIM

TTOC Converts a time expression or the time portion of a datetime expression to a character string.

Syntax: TTOC(DT) where **DT** is a datetime or time expression.

Example: To extract the time portion of the datetime expression DTEND as a character string, create a calculated field with the expression TTOC(DTEND).

TTOS Converts a time expression or the time portion of a datetime expression to a number of seconds. You can then use the numeric output to perform certain interval calculations (for example, the difference between two time values).

Syntax: TTOS(DT) where **DT** is a datetime or time expression. If **DT** is a datetime expression, the date portion is ignored.

Example: To convert the time portion of the datetime expression DTSTART to a numeric value representing a number of seconds, create a calculated field with the expression TTOS(DTSTART).

Sample data:

DTSTART Value	TTOS(DTSTART) Output
05/05/95 12:30:00 am	1800
05/05/95 9:00:00 am	32400
05/05/95 1:00:00 pm	46800
05/05/95 8:26:00 pm	73560

UDFNAME Returns the path and name of the UDF library file.

Syntax: UDFNAME( ), no arguments.

Example: To print the path and name of the R&R UDF library, create a calculated field consisting only of UDFNAME( ). Then place the field in an appropriate band of the report.

UPPER Converts lower case letters of a character expression to upper case letters.

Syntax: UPPER(C) where **C** is the character expression you wish to convert to upper case.

Example: To print the CITY field from the CUSTOMERS table in upper case letters, create the field:

UPPER(CITY)

Sample data:

CITY	UPPER(CITY)
Columbia	COLUMBIA
Jefferson City	JEFFERSON CITY
Kansas City	KANSAS CITY

VAL

Converts character representation of numbers into numeric values (i.e., this is a change in data type from character to numeric). This function is the opposite of STR.

Syntax: VAL(C) where **C** is a character expression containing numbers. VAL ignores leading spaces. Also, VAL converts only digits, one decimal point, and a leading minus sign and stops converting a character value when it encounters a non-digit, an embedded minus sign, or a second decimal point.

Example: VAL(CHARACTERS)

Sample data:

CHARACTERS	VAL(CHARACTERS)
346-34-7132	346
(617)376-1234	0
–123	–123
–123–456	–123
123abc	123
123.123.123	123.123

WDCOUNT

Returns the number of words in a character expression.

Syntax: WDCOUNT(C1,C2) where **C1** is the character expression whose words are to be counted. **C2** is an optional character expression that represents the word break character(s), the character(s) marking word separations. If **C2** is absent, R&R uses the space and hyphen characters to determine word breaks. If you supply **C2**, R&R uses only **C2** to determine word breaks.

Examples: WDCOUNT("this is a character-string") yields 5.

WDCOUNT("this/character/separates/words words","/") yields 4 since R&R uses "/" to break words and ignores the space character.

WEEK

Returns a number representing the week-of-the-month for a date expression. Week numbers, which start at 1, are determined by the calendar date. Weeks begin on Sunday and end on Saturday.

Syntax: WEEK(D) where **D** is a date expression.

Example: To print the week number of the date field DATE from the ORDERS table, create the field:

WEEK(DATE)

Sample data:

DATE	WEEK(DATE)
06/28/95	5
07/07/95	2
07/28/95	5

WKSBTWN

Calculates the number of full calendar weeks between two dates. This date difference is derived by subtracting **D2** from **D1**.

Syntax: WKSBTWN(D1,D2) where **D1** and **D2** are date expressions. If the number of days is less than 7, the result is 0.

Example: To calculate the weeks between the starting date of a contract (STARTDATE) and its completion date (FINALDATE), create the field:

WKSBTWN(FINALDATE,STARTDATE)

Sample data:

FINALDATE	STARTDATE	WKSBTWN
04/14/95	04/10/95	0
04/23/95	04/07/95	2
04/28/95	04/07/95	3

WORD

Returns a word based on its place in a character expression. For example, you can use this function to print/display the first or last word in the expression.

Syntax: WORD(C1,N,C2) where **C1** is a character expression (string), **N** is a numeric expression that represents the number or place of the word in the expression, and **C2** is an optional character expression that represents the word break character(s), the character(s) marking word separations. If **N** is a positive number, R&R

processes the expression starting at the beginning of the string. If **N** is a negative number, R&R processes the expression starting at the end of the string. For example, when **N** is 1, R&R returns the first word of the expression; when **N** is –1, R&R returns the last word of the expression. When **N** is 0 or greater than the number of words in the character expression, R&R returns an empty character field.

If **C2** is absent, R&R uses the space and hyphen characters to determine word breaks. If you supply **C2**, R&R uses only **C2** to determine word breaks (see WDCOUNT for an example).

Examples: To print/display the last word in a character field DESCRIPTN, create the expression:

WORD(DESCRIPTN, –1)

Sample data:

DESCRIPTN	WORD(DESCRIPTN, –1)
PC Cable Connector	Connector
PC 3 1/2" Diskettes	Diskettes

YEAR

Returns the year part of a date expression. The result is a four-digit numeric value.

Syntax: YEAR(D) where **D** is a date expression.

Example: To calculate the year number of a date field BIRTHDATE, create the field:

YEAR(BIRTHDATE)

Sample data:

BIRTHDATE	YEAR(BIRTHDATE)
12/10/67	1967
11/19/71	1971

YRSBTWN

Calculates the number of full years between two date expressions. This date difference is derived by subtracting **D2** from **D1**.

Syntax: YRSBTWN(D1,D2) where **D1** and **D2** are date expressions.

Example: To calculate the age of all employees in a personnel table based on date of birth field (BIRTHDATE) and the current date DATE( ), create the field:

YRSBTWN(DATE( ),BIRTHDATE)

Sample data:

DATE( )	BIRTHDATE	YRSBTWN(DATE( ),BIRTHDATE)
03/31/95	12/10/65	29
03/31/95	04/11/60	34
04/01/95	07/23/57	37
04/01/95	12/05/51	43

Using User-Defined Functions

This section explains how to create, edit, and use User-Defined Functions (UDFs) in R&R. You create UDFs to process field values according to an expression or formula that you define. The ability to define UDFs provides two main advantages:

- You can use the UDFs that you develop in any R&R report.
- Since you define the expression and specify the arguments, you can create UDFs to perform complex operations beyond those available through the use of predefined functions.

After you have created a UDF, it is saved in a file called RR.UDF, which is created in the R&R program directory or in the link directory defined during installation. When naming a UDF, you can use lower or mixed case to distinguish its name from the names of predefined R&R functions. You can then use the UDF in calculated field expressions in any report just as you would use a predefined function, by typing its name or selecting it from the Function list box.

Creating a User-Defined Function

A user-defined function definition has two parts: a *declaration* and an *expression*.

The declaration serves three purposes:

- It names the UDF;
- It identifies argument names and data types to be used in the UDF's expression;
- It provides a template for the actual UDF reference.

The expression defines the calculations and processing to be performed on the UDF arguments. Unlike a calculated field expression, a UDF expression uses the argument names as specified in the declaration rather than actual field names.

Procedure

To begin creating a user-defined function, select Calculations ⇒ User Function. If no UDFs have yet been defined for the current report, the New User Function dialog appears. If one or more UDFs have already been defined, select New in the User Functions dialog to display the New User Function dialog.

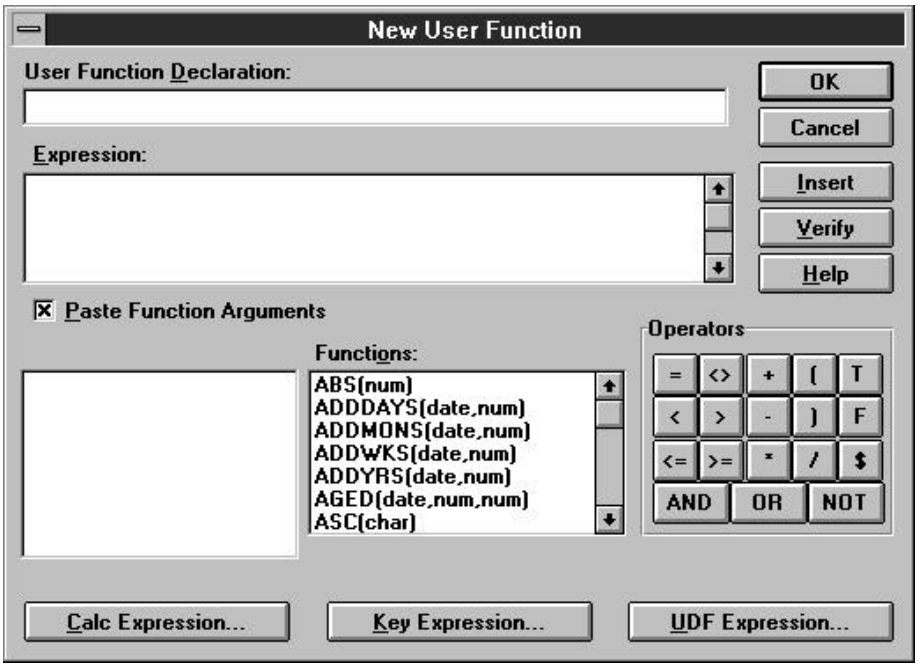

Figure 10.4 New User Function Dialog Box

Briefly, the steps for creating a UDF are as follows:

1. In the User Function Declaration edit box, enter a declaration using the following syntax:

   udfname (ArgType_ArgName1, ArgType_ArgName2, …)

2. In the Expression edit box, define the function expression by typing all or part of the expression directly in the edit box or by selecting the appropriate buttons to insert fields, operators, and/or functions into the expression.
3. Select Verify to determine whether the expression syntax is correct.
4. Select OK.

Declarations

A UDF declaration, which can be up to 256 characters long, consists of the following parts:

- ❑ UDF name in upper, lower, or mixed case

- In parentheses, the data type and name of each argument to the UDF

The syntax for the declaration is:

udfname(Argtype_ArgName1,Argtype_ArgName2, ...)

For example, the declaration for a sum function that adds two numbers might be SUM(N_A,N_B). Translated, this declaration means that the name of the UDF is SUM. It takes two numeric arguments (represented by N_), the first of which will be referred to as A and the second as B in the UDF expression.

To take a more realistic example, the declaration for a function called ISWEEKEND, which determines whether a given date falls on a weekend, might be ISWEEKEND(D_ANYDATE). This declaration means that the ISWEEKEND function takes one date argument, which will be referred to in the UDF expression as ANYDATE.

When you use a UDF in a calculated field expression, the UDF reference must conform to the structure defined by the UDF declaration. For example, if the UDF declaration specifies one date argument, when you include the UDF reference in a calculated field, you specify one date field as an argument to the UDF.

UDF Names

UDF names can be up to 20 characters long; the first 4 characters must be unique (that is, no two UDF names can have the same first 4 characters). When you use a UDF in a calculated field expression, you need to include only the first 4 characters of the UDF name.

UDF Arguments

UDF arguments represent expressions that the function will operate on. In the declaration, each argument consists of a letter representing the argument data type, an underscore character (_), and a name.

Use one of the following letters to indicate argument data type:

- C for a character argument
- D for a date argument
- L for a logical argument
- M for a memo argument
- N for a numeric argument (fixed or floating point)

An argument name, which follows the data type indicator and underscore character, must be unique in the first 9 characters (for example, a

declaration such as UDF(C_ARG,N_ARG) is not valid, since both arguments have the same name) and must start with a letter. The name can contain only letters, numbers, and the underscore character.

Argument names indicate how an argument will be referred to in the UDF expression that defines the function's operation. For example, if the declaration for ISWEEKEND is ISWEEKEND(D_ANYDATE), the expression must use the name ANYDATE to refer to the date variable. If the declaration is ISWEEKEND(D_A), the expression must use A to refer to the date variable.

Actual values are "passed" to a UDF through the intermediary of argument names. When you use the UDF in a calculated field expression, you substitute actual expressions or field names for the argument names in the declaration. For example, when you use the ISWEEKEND(D_ANYDATE) function, you can substitute a field name or another expression for the argument name as follows: ISWEEKEND(NEWYEARS) or ISWEEKEND({01/01/94}). The value of the field or expression is then used in the UDF's calculations wherever ANYDATE appears.

To take another example, a function that calculates the difference between two date values might have the following declaration:

```
DDIF(D_DATE1,D_DATE2)
```

When you use this UDF in a calculated field expression, a function such as DATE( ), a field such as BIRTHDATE, or an expression such as CTOD("10/17/62") can be substituted for either argument. This substitution supplies values for the function's calculations.

UDF Expression

A UDF's expression defines the function's operation. The expression has the same structure and uses the same functions and operators as a calculated field expression, but argument names instead of field names represent actual values that will be supplied when the UDF is used in an expression.

For example, the expression for ISWEEKEND is:

```
(DOW(ANYDATE)=7).OR.(DOW(ANYDATE)=1)
```

This expression uses the predefined DOW( ) (day of week) function, a relational operator (=), and a logical operator (OR) to operate on a date expression represented by the argument name ANYDATE. The DOW( ) function returns a number from 1 to 7 that represents the day of the week of

the ANYDATE expression. The = operator tests to see if this number is 7 (Saturday) or 1 (Sunday). The OR operator means that if the number of the day is either 7 or 1, the expression is true; that is, the date falls on a weekend.

When you include this UDF in a calculated field, the field expression that replaces the ANYDATE argument will supply a date value for the expression to process.

Note that the argument type prefixes (C_, D_, L_, M_, N_) are *not* included in the UDF expression.

Inserting Predefined Functions

When you select a predefined function and insert it in a UDF expression, in most cases you must supply one or more arguments (fields or operators) to that function. You can use the "Paste Function Arguments" setting as an aid to providing function arguments. When "Paste Function Arguments" is On (the default), R&R inserts the arguments for a function into the expression. The first argument is highlighted; any item you insert or type as an argument (a field or operator, for example) replaces the highlighted argument name.

For example, suppose that you have selected the SUBSTR function and inserted it into your UDF expression. With "Paste Function Arguments" on, the function is inserted as follows:

```
SUBSTR(char,num,opt num)
```

When you enter or insert the first argument, it replaces "char" in the expression. You can then supply the other function arguments as needed.

When "Paste Function Arguments" is Off, R&R inserts function names without supplying argument names. Instead, the function name is inserted followed by open/close parentheses — for example, SUBSTR( ).

Modifying a User-Defined Function

To edit a previously created UDF, select Calculations ⇒ User Function. Select the UDF you want to edit from the list box; then select Edit to display the Edit User Function dialog box. The Edit User Function dialog is the same as the New User Function dialog.

To modify the existing expression, position the edit cursor as necessary to insert or delete elements. To delete a portion of the expression, drag the mouse over that portion and press Delete. Following the procedures described in the section in this chapter on creating a UDF, revise the

expression as necessary. When you have completed your changes, select OK.

If the UDF you are changing will change the data type of another UDF or a calculated field expression in the current report, R&R will display a list of UDFs and calculated fields that will become unusable as a result of your change. The items on this list will become impossible for R&R to evaluate as a result of your change. At this point, you can select Cancel to cancel the change or OK to make the change. Selecting OK will cause the affected UDFs and fields to be flagged with question marks in field lists. If any flagged UDFs or fields are used in your report, you will have to correct their expressions before you can run the report.

Because UDFs may be used in multiple reports, you should edit them carefully. R&R cannot notify you of the effect your change may have on other reports.

Deleting a User-Defined Function

To delete a UDF, select Calculations ⇒ User Function. Then select the UDF you want to delete and select Delete.

You cannot delete any UDF that is used in expressions for calculated linking fields in the current report. If the UDF you are deleting is used in another UDF expression or a non-linking calculated field, R&R displays a list of UDFs and/or calculated fields that will be affected by your deletion. The items on this list will be deleted if you delete the UDF. At this point, you can select Cancel to cancel the deletion or OK to delete the UDF. Selecting OK will cause the affected UDFs and fields to be deleted.

Because UDFs may be used in multiple reports, you should be careful about deleting them. R&R cannot notify you of the effect your deletion may have on other reports.

Sample User-Defined Functions

The following sections present examples of user-defined functions and explain the procedures for creating them.

Creating a Salutation

Another example illustrates how a UDF simplifies the process of creating form letters. Suppose you have a number of tables that contain fields for titles (Mr. or Ms.), first names, and last names. Any of these fields can be blank, leading to form letter salutations like “Dear Smith” or “Dear Mr.

John." To generate an acceptable salutation no matter which of these fields is empty, you can create a UDF called SALUTATION.

You want SALUTATION to compensate for missing title and name data by generating salutations like "Dear Mr. Smith:" when there is data in both the title and last name field; salutations like "Dear John," when the first name field contains data but either the title or last name field is empty; and salutations like "Dear R&R User:" when only the last name field contains data.

If MM is the title argument, FN is the first name argument, and LN is the last name argument, the declaration of SALUTATION is:

```
SALUTATION(C_MM,C_FN,C_LN)
```

The expression is:

```
IIF(MM<>"".AND.LN<>"","Dear "+MM–" "+LN–":",
IIF(FN<>"","Dear "+FN–",","Dear R&R User:"))
```

Translated, this expression means that if the title and last name both contain text, output a string composed of "Dear "+MM–" "+LN–":". Otherwise, if the first name contains text, output a string composed of "Dear "+FN–",". Otherwise, output "Dear R&R User:"

Once this UDF has been defined, it can be used with any table containing title, first name, and last name fields. If your report uses a table with fields called MRMS (title), FIRSTNAME, and LASTNAME, you can create a calculated field called GREETING that invokes the SALUTATION function. The expression for GREETING will be:

```
SALUTATION(MRMS,FIRSTNAME,LASTNAME)
```

By supplying these values to the SALUTATION expression, the GREETING field will produce output like that shown in Figure 10.5.

MRMS	FIRST	LAST	GREETING
Ms.	Judy	Smith	Dear Ms. Smith:
Mr.		Miller	Dear Mr. Miller:
Mr.	John		Dear John,
	Lee	Jones	Dear Lee,
		Cook	Dear R&R User:

Figure 10.5 Sample Output Using SALUTATION

Calculating Appreciated Value

Another example illustrates the creation of a numeric UDF that calculates the current value of appreciable assets. Since appreciated value depends on original value, interest rate, and the time that has elapsed since the asset was purchased, the declaration for an appreciated value (AV) function is as follows:

```
AV(N_OV,N_INT,D_DP)
```

OV represents original value, INT interest rate, and DP purchase date. If yearly interest is converted to daily interest, the expression is:

```
OV*(1+(INT/365))^DAYSBTWN(DATE( ),DP)
```

Translated, this expression means that appreciated value will be calculated by taking the original value and multiplying it by 1 plus the interest rate divided by 365 (days in a year) raised to a power that is the number of days since the asset was purchased. This number of days is supplied by the DAYSBTWN( ) function, which calculates the number of days between the current date and the date of purchase.

When you use this function in a calculated field, you replace the OV, INT, and DP arguments with actual expressions. For example, if you have an asset table that includes a COST field containing purchase price and a PURDATE field containing purchase date, you compute appreciated value at 8% interest using the following expression:

```
AV(COST,.08,PURDATE)
```

Two field names and a constant are substituted for the arguments in AV's declaration. This substitution supplies values to the expression that calculates appreciated value. To calculate appreciated value at a different interest rate, all you have to do is change the constant.