R&R Report Writer® Technical Bulletin

Creating User-Defined Functions (UDFs)

Product:	R&R Report Writer®	**Oper Sys:**	DOS®
	ARPEGGIO™ Report Designer		Microsoft® Windows®
Version:	R&R V3.0 and higher		Windows 95
	ARPEGGIO V1.0 and higher		Windows NT®

Summary

User-Defined Functions (UDFs) allow you to define general purpose calculations that can be used in any report.

This technical bulletin provides exercises to help you become comfortable creating and using UDFs so you can begin to build your own UDF library.

NOTE: For R&R Report Writer Xbase versions, UDFs are stored in a file named RR.UDF located in the R&R Report Writer program directory. For R&R Report Writer SQL versions and ARPEGGIO Report Designer, UDFs are stored in a file called RSW.UDF located in the program directory.

Naming UDFs

R&R Report Writer allows you to enter a name up to 20 characters, provided the first four characters are unique. This enables you to abbreviate UDF names to four characters when entering a calculated field expression. For instance, in the following examples we have selected names that clearly identify what each UDF does. However, when you see a long name such as Phone_Format() remember all you need to type is Phon().

You can use upper, lower or mixed case for the UDF name. However, we suggest using mixed case, as in these examples, to distinguish UDFs from predefined functions.

Creating a Simple UDF

Below are three examples of creating simple UDFs. The first, BTrim(), is used to trim spaces. The second, TLen(), returns the length of the data in a database field. TStr() converts a number to a character string.

BTrim()

R&R Report Writer includes a predefined function to remove leading spaces from a character string and another predefined function to remove trailing spaces. The steps below show how to create a simple UDF to combine both functions.

The LTRIM() function trims spaces from the left side of a string and the RTRIM() function trims spaces from the right side. Both the LTRIM() and RTRIM() functions can be applied to the same character string to remove both leading and trailing spaces. For example, to remove leading and trailing spaces from an address you could use the following expression.

```
LTRIM(RTRIM(address))
```

You can simplify this expression by defining a UDF that performs both trim operations. If you name this function BTrim(), since it trims spaces from both sides of a string, you can rewrite the expression as:

```
BTrim(address)
```

The BTrim() UDF is created using the /Field/Calculate/User-Function/Create command.

First, enter the declaration, which tells R&R Report Writer the name of the function, the data types to be used, and the names and order of input values (called arguments). Enter the following declaration for BTrim().

```
BTrim(c_string)
```

This declaration means that the BTrim() function requires a single character argument that is referred to as "string".

After entering the declaration, enter the following formula:

```
LTRIM(RTRIM(string))
```

The argument name is used within the formula to refer to the value passed to the UDF as input. UDFs operate on input arguments, such as "string" in this example, rather than on fields in your report. This characteristic of UDFs makes them suitable for general purpose use with any field or expression.

Defining BTrim() as:

```
LTRIM(RTRIM(address))
```

defeats the purpose of creating a UDF, since it would work only with the ADDRESS field.

In defining BTrim() it makes no difference whether you first apply the RTRIM() function or the LTRIM() function to the string. You will get the same result using either of the following expressions.

```
LTRIM(RTRIM(string)) OR RTRIM(LTRIM(string))
```

You can now use the expression BTrim(char) to remove extra spaces from both ends of a character expression.

NOTE: *char* is used to indicate a character expression and *num* to indicate a numeric expression.

TLen()

The LEN() function applied to a database field returns the field width. Therefore, it is common to use the expression LEN(TRIM(fieldname)) to find the length of the actual data. You can reduce the typing required by creating the UDF TLen(char) to find the *trimmed length* of a specified character expression.

Define the TLen() UDF as:

```
Declaration: TLen(c_string)
Formula:     LEN(TRIM(string))
```

TStr()

TStr() is another UDF to reduces typing. It converts a number to a character string, trimming away any leading spaces.

The predefined function STR() converts a number to a character string, but may insert leading spaces. To ensure that there aren't any leading spaces in the resulting character string it is common to use the expression LTRIM(STR(fieldname)). You can replace this with a more compact expression by defining a UDF.

```
Declaration:  TStr(n_number)
Formula:      LTRIM(STR(number))
```

Once defined in this manner, the expression TStr(num) can be used to convert a numeric expression to a character string without leading spaces.

Creating More Complex UDFs

Generating a random number can be useful in forecasting, simulation, and decision-making applications. Random() is a more complex UDF that demonstrates self-referencing calculated fields.

The algorithm used in the Random() UDF is derived from an algorithm described in the book titled *The Art of Computer Programming* by Donald Knuth (Volume 2, page 9). This algorithm has the property that each random number generated is a function of the previous one. In R&R Report Writer we refer to this property as self-reference.

The most common example of self-reference is an expression such as i=i+1. In R&R Report Writer this can be implemented by creating a calculated field named i, defined by the expression i+1. This expression means that each time a new value for i is computed, it is computed by adding 1 to the previous value of i. The value of i in the first composite record will be 1, in the second composite record will be 2, etc.

The following UDF definition generates a sequence of random numbers between 0 and 65535, one number for each composite record.

Declaration:	Random(n_randnum)
Formula:	MOD(3677*randnum+3,65536)

It is important to understand how to use this UDF properly. The simplest usage would be the following calculated field expression, which generates a sequence of random numbers between 0 and 65535. This example also clearly illustrates the self-referencing property since the field named randNumber applies the Random() function to itself.

```
randNumber = Random(randNumber)
```

In some applications you may need to generate a sequence of random numbers within a different range, say between 0 and 1. In R&R Report Writer, this is accomplished by defining the calculated field randNumber as explained above, and then creating another calculated field that scales the number to within the desired range. Define the following calculated field expression to scale randNumber to a fraction greater than or equal to 0 and less than 1.

```
randFract = randNumber/65536
```

The randFract field can be used in other calculations. However, if you want this field to appear on the report, you need to add the desired number of decimal places using the /Field Format command. Then set the number of integer places to zero using the /Field Width command.

To generate a sequence of random digits between 0 and 9 use the following calculated field expression.

```
randDigit = INT((randNumber/65536)*10)
```

To generate a sequence of random numbers between 1 and 10 just add one to the previous expression as in the following calculated field expression.

```
randTen = INT((randNumber/65536)*10)+1
```

To generate a sequence of random numbers between 1 and 100 just multiply the fraction by 100 instead of 10 as in the following calculated field expression.

```
randHundrd = INT((randNumber/65536)*100)+1
```

Remember that the sequence of numbers generated in the Random() function is always the same, because the starting, or *seed*, value in the randNumber field is always 0 (as is true with all calculated and total fields in R&R Report Writer).

For some reports, it may be desirable to generate a different sequence of random numbers each time the report is run. This can be accomplished by modifying the UDF formula to use a variable seed value.

One satisfactory way to do this is to set the seed value to the number of seconds in the current DOS time. This is done in the following modified Random() UDF.

```
Declaration:  Random(n_randnum)
Formula:       MOD(3677*IIF(RECNO()=1,VAL(SUBSTR(TIME(),7,2)),randnum)+3,65536)
```

Note that the number of seconds, VAL(SUBTR(TIME(),7,2)), is returned only for the first composite record, when RECNO() equals 1. This seeds the Random() UDF with a number between 0 and 59, providing 60 different sequences of random numbers.

Other UDF Examples

In_USA() and In_Canada()

A user wrote two functions that determine whether an address is located in the USA or in Canada. These expressions are easily converted to UDFs. Both UDFs take one input argument, a character expression containing a two character state or province code, and return a true or false value.

The original expressions have been enhanced to work regardless of upper- or lower-case input. In addition, the abbreviation QC has been added as an alternative for the Canadian Province of Quebec (PQ).

```
Declaration:  InUSA(c_state)
Formula:      UPPER(state) $ "AL,AK,AZ,AR,CA,CO,CT,DE,DC,
              FL,GA,HI,ID,IL,IN,IA,KS,KY,LA,ME,MD,MA,MI,
              MN,MS,MO,MT,NE,NV,NH,NJ,NM,NY,NC,ND,OH,OK,
              OR,PA,RI,SC,SD,TN,TX,UT,VT,VA,WA,WV,WI,WY"

Declaration:  In_Canada(c_province)
Formula:      UPPER(province) $ "AB,BC,LB,MB,NB,NF,NT,NS,
              ON,PE,PQ,QC,SK,YK"
```

The expressions In_USA(char) and In_Canada(char) now can be used to determine whether an address is located in the USA or in Canada.

NMonth()

This is an expression used to convert a month number between 1 and 12 to the name of the corresponding month. This UDF is simple to construct and is defined as follows.

```
Declaration:  NMonth(n_month)
Formula:      CMONTH(CTOD(STR(month)+"/1/98"))
```

The expression NMonth(num) now can be used to convert a month number to the corresponding month name.

Phone_Format(), Soc_Sec_Format(), and Zip_Format()

These UDFs can be used to format telephone numbers, social security numbers, and ZIP codes that are stored as a sequence of digits in a character field.

NOTE: The Phone_Format() and Zip_Format() UDFs use the TLen() UDF defined earlier in this technical bulletin.

The Phone_Format() UDF accepts either a seven-digit or ten-digit number and adds the appropriate punctuation. This UDF has a long name; however, you can abbreviate function names in a calculated field expression to their first four characters. For example, you can abbreviate the expression Phone_Format(PHONE) to Phon(PHONE).

```
Declaration:  Phone_Format(c_string)
Formula:      TRANSFORM(string,IIF(TLen(string)=7,
              "@R XXX-XXXX","@R (XXX) XXX-XXXX"))
```

The Soc_Sec_Format() UDF accepts a nine digit character string and adds the punctuation for US Social Security numbers.

```
Declaration:  Soc_Sec_Format(c_string)
Formula:      TRANSFORM(string,"@R XXX-XX-XXXX")
```

The Zip_Format() UDF accepts either a five digit or nine digit character string containing a US ZIP Code, or a six character string containing a Canadian Postal Code. The CASE() function is used to select among the three possibilities. The input string is returned unchanged if its length is not five, six, or nine.

```
Declaration:  Zip_Format(c_string)
Formula:      TRANSFORM(string,CASE(TLen(string),5,"@R XXXXX",
              6,"@R XXX XXX",9,"@R XXXXX-XXX",""))
```

Dear()

The following UDF presents an expression for creating personalized form letter greetings from incomplete data. It includes multiple input arguments and uses two internal UDFs.

Internal UDFs are user-defined functions that are used only by other UDFs. You would never refer to these UDFs directly in a calculated field expression. They are useful because they enable you to break up a long formula into a few shorter formulas. The Dear() UDF illustrates this technique.

You must first create the two internal UDFs. They test for the presence of a first and last name, respectively. The names begin with an underscore so that they are sorted last, out of the way, in the function menus.

```
Declaration:  _fname(c_first)
Formula:      first#' ' .and. RIGHT(WORD(first,1),1)#' '
Declaration:  _lname(c_title,c_last)
Formula:      title#"" .and. last#""
```

Now create the Dear() UDF, which refers to the two internal UDFs as:

```
Declaration:  Dear(c_title,c_first,c_last,c_generic)
Formula:      IIF(_lname(title,last),title-' '+last,IIF(_fname(first),WORD(first,1),
              generic))
```

The Dear() UDF formula examines a title, first name, and last name and creates an appropriate greeting based on the data available. A generic greeting, as in "Dear Friend", is returned when there is no first or last name.

PadLeft(), PadCenter(), and PadRight()

These three expressions pad the data in a database field with hard spaces. The difference between regular spaces and hard spaces is that hard spaces can be underlined. These UDFs allows you to create reports that look like preprinted forms.

The original expressions have been enhanced to allow the desired field width to be specified as an option. If you specify 0 (zero) as the width, the database field width is used. However, it is recommended that you specify the desired width.

Notice the use of the LEN() function and the TLen() UDF to distinguish between the field width and the length of the data in these three UDF definitions.

```
Declaration:  PadLeft(c_string,n_width)
Formula:      SPACE(LEN(string)-TLen(string)+ IIF(width,width-
              LEN(string),0))+TRIM(string)

Declaration:  PadCenter(c_string,n_width)
Formula:      SPACE(((IIF(width,width, LEN(string)))-TLen(string))/2)+
              TRIM(string)+ SPACE(((IIF(width,width, LEN(string)))-
              TLen(string))/2)

Declaration:  PadRight(c_string,n_width)
Formula:      TRIM(string)+ SPACE(LEN(string)-TLen(string)+
              IIF(width,width-LEN(string),0))
```

Ordinal()

This expression is used to convert numbers from cardinal to ordinal. For example, the number 1 is converted to "1st". Use the TStr() UDF, defined above, to enhance the UDF as:

```
Declaration:  Ordinal(n_number)
Formula:      TStr(number)+ CASE(RIGHT(STR(number),2),
              "11","th","12","th","13","th",
              CASE(RIGHT(STR(number),1), "1","st","2","nd","3","rd","th"))
```

You can often accomplish the same result with several different expressions. Sometimes one expression is easier to understand, while another is more compact. The Ordinal() UDF illustrates this point.

Using modulo arithmetic and the MOD() function you can identify numbers ending with 11, 12, 13, 1, 2, and 3 without converting the number to a string.

Modulo arithmetic is extremely useful. The expression MOD(number,100) simply returns the remainder when "number" is divided by 100. If the remainder is 11, 12, or 13, then the number was something like 11, 212, or 6513 etc. Likewise, the expression MOD(number,10) is used to find numbers that end with 1, 2, or 3.

The following expression is comparable to the one above.

```
Declaration:  Ordinal(n_number)
Formula:      TStr(number)+ IIF(INLIST(MOD(number,100),11,12,13),"th",
              CASE(MOD(number,10),1,"st",2,"nd",3,"rd","th"))
```

Also, by using the INLIST() function, we can combine numbers ending with 11, 12, and 13 into a single case. This change enabled us to replace the first CASE() function with the IIF() function since there are now only two cases: numbers that end with 11, 12, and 13, and those that don't.

Whichever formula you use, you can use the expression Ordinal(char) to convert a numeric expression to an ordinal character string.

Title_Sort()

Here is the UDF definition of an expression to rearrange book and movie titles so they sort alphabetically.

```
Declaration:  Title_Sort(c_title)
Formula:      IIF(INLIST(WORD(title,1),"A","An","The"),
              RTRIM(RIGHT(title,LEN(title)-AT(" ",title)))+
              WORD(title,1),title)
```

This UDF works by moving the articles "A", "An", and "The" from the beginning of the title to the end.

Check_Amount()

This is an expression to spell the dollar amount on a check. The SPELLNUM() function is used to spell the integer portion and then a properly formatted fraction for any leftover cents is added. Here is the UDF definition.

```
Declaration:  Check_Amount(n_amount)
Formula:      SPELLNUM(amount)+ IIF(amount-INT(amount)," and "+
              STR((amount-INT(amount))*100,2,0)+"/100","")
```

If there is a fraction (for example, 45 cents) the text "and 45/100" will be appended to the whole dollar amount. If the full amount were 123.45, the complete result from applying the Check_Amount() function would be “One hundred twenty-three and 45/100”

Dot_Leader_Tab()

This expression to adds a dot leader tab to a character expression. The dot leader tab function uses the following definition:

```
Declaration:  Dot_Leader_Tab(c_text)
Formula:      text-REPLICATE('.',254)
```