RMS Report Writing

Examples for designing custom report in RMS

by RMS Support Center

RMS uses the Report Builder report writing tool to allow users to design customized Reports using RMS data. This banded report writing tool is flexible and powerful, but requires patience and skill.

This document contains many examples of reports that illustrate interesting and powerful ways to access and display RMS data using the report writer. Step through the example one by one in order to get the most out of this document.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foreword</td>
<td>0</td>
</tr>
<tr>
<td><strong>Part I RMS Query Browser</strong></td>
<td></td>
</tr>
<tr>
<td>1 RMS Query Browser</td>
<td>3</td>
</tr>
<tr>
<td>2 Database Info</td>
<td>4</td>
</tr>
<tr>
<td>3 SQL Editor</td>
<td>5</td>
</tr>
<tr>
<td><strong>Part II SQL Primer</strong></td>
<td>8</td>
</tr>
<tr>
<td>1 Select Statement</td>
<td>8</td>
</tr>
<tr>
<td>2 Distinct Clause</td>
<td>11</td>
</tr>
<tr>
<td>3 Where Clause</td>
<td>13</td>
</tr>
<tr>
<td>4 Dates and SQL</td>
<td>15</td>
</tr>
<tr>
<td>5 Order By Clause</td>
<td>21</td>
</tr>
<tr>
<td>6 Case Statement</td>
<td>23</td>
</tr>
<tr>
<td>7 SQL Comparison Operators</td>
<td>23</td>
</tr>
<tr>
<td>8 Oracle Functions</td>
<td>25</td>
</tr>
<tr>
<td><strong>Part III RAP Primer</strong></td>
<td>32</td>
</tr>
<tr>
<td>1 Overview</td>
<td>32</td>
</tr>
<tr>
<td>2 Data Types</td>
<td>33</td>
</tr>
<tr>
<td>3 Operators</td>
<td>33</td>
</tr>
<tr>
<td>4 If Statement</td>
<td>35</td>
</tr>
<tr>
<td>5 Case Statement</td>
<td>37</td>
</tr>
<tr>
<td>6 While Statement</td>
<td>40</td>
</tr>
<tr>
<td>7 Procedures and Functions</td>
<td>41</td>
</tr>
<tr>
<td>8 Gotchas</td>
<td>42</td>
</tr>
<tr>
<td>Index</td>
<td>0</td>
</tr>
</tbody>
</table>
1 RMS Query Browser

1.1 RMS Query Browser

The RMS Query Browser is a custom tool inside of the RMS Reports designer which will allow you to test your standard select queries for functionality. You can also use the Exporting capabilities to save your results sets out to native .xls format or to a .csv file.

Clicking on the table names under the Schema listings will automatically select the table and display its Field Info. Use this information to create powerful select statements which you can use to enhance the quality of your reports.

**Note:** You will not be able to perform any type of updates or alter the database in anyway using this utility. It is strictly used for browsing and exporting data to native excel format.
1.2 Database Info

Clicking on a table name on the Database Info tab will automatically select all of the data from this table and display it in the SQL Results Grid.

In addition you can also click on the Field Info to get more detail about the table you are viewing.
Such as the fields type and data length and whether they allow nulls or not.

<table>
<thead>
<tr>
<th>ID</th>
<th>FIELD_NAME</th>
<th>DATA_LENGTH</th>
<th>ALLOW_NULLS</th>
<th>FIELD_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FL_CCDE</td>
<td>1</td>
<td>N</td>
<td>VARCHAR2</td>
</tr>
<tr>
<td>2</td>
<td>SHOW_RECORD</td>
<td>1</td>
<td>N</td>
<td>VARCHAR2</td>
</tr>
<tr>
<td>3</td>
<td>TITLE</td>
<td>42</td>
<td>Y</td>
<td>VARCHAR2</td>
</tr>
</tbody>
</table>

1.3 **SQL Editor**

The SQL Editor can be used to type in your select statements.

Pressing F5 will execute your statements and display the results in the SQL Results Grid.
<table>
<thead>
<tr>
<th>CONTRACT_ID</th>
<th>PROJ_DS</th>
</tr>
</thead>
</table>
| J3000433   | Work will include general design and construction projects to include but not limited to:
| J3000329   | FY03 MILCON KNMD023014 Installation of Flightline Security Fencing and Gates. |
| J3000483   |                                                                           |
| J3000484   | Replace two deteriorated automatic transfer switches (ATS) on the E1 and E2 circuit. |
| J3000089   | This is a test project to test the RMS/CMS interface.                   |
| J3000111   |                                                                           |
| J3000112   |                                                                           |
| 97C0058    | Replace existing roofs on the 9th and 10th floors of Wings A, B, and C with new to... |
| J3000865   |                                                                           |
| J3000863   |                                                                           |
| J3000864   |                                                                           |
| PW99C006   | Palau Compact Road Project, Babeldao Island, Republic of Palau           |
| 99C0012    | FY96 MCA P/N 42470, Kolekole Connector Road, Kolekole-Tinmel Connector and... |
| J3000353   | Basic Contract for Indefinite Delivery Indefinite Quantity (IDIQ) Construction and Se... |
| DAC83-01-T-0001 |                                                        |
| J3000889   |                                                                           |

Records: 345  Columns: 2  0%
RMS Report Writing

Example Report
For Training Purposes

Example
2 SQL Primer

2.1 Select Statement

The select statement is used to retrieve data from the database. The format is:

```
select columns from tables;
```

Let's get a list of contract id's from the contract table.

```
select contract_id from contract
```

SQL is not case sensitive. Keywords can be place in caps, but that is not a requirement. Case is important when we get to the actual data but only for the data. In other words, if I query looking for "dc000904", then "DC000904" and "Dc000904" will not be returned.

You can also alias a column using the AS keyword, or you can leave it out. If your new column name includes a space, you need to enclose the alias in quotes.
If I wanted to select all of the columns then I would do a select * pronounced "Select Star" or "Select All". It would look like this:

```sql
select * from contract
```
You can also do math on number columns. Math in SQL follows the normal order of precedence. Multiplication (*) and Division (/) before addition (+) and Subtraction (-). Operators of the same priority are evaluated left to right. Use parentheses to change the order of evaluation.

```sql
SELECT contract_id, act_tot, 2*act_tot+10 "My Custom Number" FROM contract
```
NOTE: A NULL value is a column value that has not been assigned or has been set to NULL. It is not a blank, space or a zero. It is undefined. Because a NULL is undefined, there is no such thing as NULL math. As a result the following will equal NULL:
- NULL + 4 = NULL
- NULL * 3 = NULL
Since NULL is undefined, all math using a NULL returns a NULL.

2.2 Distinct Clause

Many times there are multiple rows with the same value, and we want to return only one copy of the row. If the boss wants a list of the offices currently containing contracts, we can query that from the contract table.

```sql
SELECT office_symbol FROM contract
```
We have 346 contracts, and we got 346 rows back. But, notice that some contracts have the same office_symbol. What we want is a list of the distinct office_symbols (one row for each office_symbol). SQL provides the DISTINCT clause for this result.

```
1  select Distinct(office_symbol) from contract
```
The Distinct Clause removes the duplicate rows.

### 2.3 Where Clause

The WHERE clause also limits the number of rows in the results set. The WHERE clause is a logical comparison and returns the row if the WHERE clause is true and excludes the row if the clause is false.

**TRUE**

```
1 select fmt_contract_delivery from contract
2 where contract_id = 'J3000211'
```
Notice that the column used in the WHERE clause is not one of my selected columns. It can be, but there is no requirement for it to be selected. Also, note that I capitalized the argument. The field is a varchar2 or a character string. Character strings are enclosed in single quotes. Although capitalization does not matter in the SQL command syntax, it does matter with data. The information that is stored in the database might be capitalized or not. If I use my WHERE clause regardless of case I could get back a result set that is not what I wanted.

**FALSE**

```
1 select fmt_contract_delivery from contract
2 where contract_id = 'j300G111'
```
SQL Logical Operators

An SQL query can only have one WHERE clause; however, that clause can contain multiple comparisons. Each comparison returns a TRUE, FALSE or NULL. You evaluate these TRUE/FALSE results using AND and OR to end up with the single TRUE or FALSE for the entire WHERE clause. The AND operator (called conjunction) returns TRUE if both comparisons are TRUE and returns FALSE if either comparison is FALSE.

\[
\text{WHERE } \text{act}_\text{tot} < 100000 \quad \text{AND } \text{contract}_\text{id} = 'J3000111' \quad \text{--TRUE}
\]

Since both comparisons are TRUE, the WHERE clause is TRUE. The logical OR operator (called a disjunction) returns TRUE if either comparison is TRUE, otherwise returns FALSE.

\[
\text{WHERE } \text{act}_\text{tot} < 100000 \quad \text{OR } \text{contract}_\text{id} = 'J3000111' \quad \text{--FALSE}
\]

The WHERE clause returns TRUE.

2.4 Dates and SQL

Dates are stored in the database as large numbers. The actual size of the data number is dependent on the operating system supporting the database. When a date is requested, it is returned in a human readable form.

When date values are compared in the WHERE clause, the format of the date must match the format that the database is using or the comparison will fail. Alternately, if you are using another format, then you must tell the database how your dates is formatted. The default date format for RMS is MM-
DD-YYYY. If you are using an RMS tool this format is set for you each time a new session is created. If you are using your own tool such as TOAD or some other SQL tool you will need to set your session variable using the "ALTER SESSION SET NLS_DATE_FORMAT = 'MM-DD-YYYY'" statement.

To get the current date, you select from a function called SYSDATE. SYSDATE returns the current date from the server operating system supporting the database.

```sql
select sysdate from dual
```

The dual table is a pseudo-table that allows you to execute functions that require selecting from a table.

Lastly, because a date is stored in the database as a number, you can perform date math.

```sql
select sysdate today,
sydate -1 yesterday,
sydate +1 tomorrow from dual
```
Date Functions
Oracle also has some specific functions geared toward the manipulation and formatting of the dates.

add_months(d,n)
The add_months function gives you the same day, n number of months away. The n can be positive or negative.

```
1 SELECT
2 SYSDATE,
3      ADD_MONTHS(SYSDATE,-1),
4      ADD_MONTHS(SYSDATE,2),
5      ADD_MONTHS(SYSDATE,-3)
6 FROM    DUAL
```

last_day(d)
The last_date function returns the last day of the month of the date d. If you want to find the first day of the next month simply add one to the last_day results.
The `next_day` function returns the date of the `day_of_week` after date `d`. `day_of_week` can be the full name or abbreviation. Below, we get the date for next Monday, next Friday, and the first Tuesday of next month.
The `to_char` function will change a date to characters in the format defined in the format field. If you do not define a format, the date will be returned in the default format set for the database. Formatting commands are enclosed in single quotes and are case sensitive and can include any valid data format element.

**Date Formatting Elements**

<table>
<thead>
<tr>
<th>SYSDATE</th>
<th>Next Mon</th>
<th>Next Fri</th>
<th>First Tue</th>
</tr>
</thead>
</table>
D  Day of the week as in 1 thru 7.
DD  Day of the Month as in 1 thru 31.
DDD  Day of the Year as in 1 thru 366.
DY  Day of the Week abbreviated Mon thru Sun
DAY  Day of the Week Monday, Tuesday...
W  Week of the Month as in 1 thru 5
WW  Week of the Year as in 1 thru 53.
MON  Month abbreviated Jan thru Dec.
MONTH  Month Spelled out. January thru December.
YY  Year, Last two digits as in 06 and 07
YYYY  Year, four digit as in 2006 and 2007
YEAR  Year spelled out.
HH  Hour in 12 hour clock. 1 thru 12.
HH24  Hour in 24 hour clock 1 thru 24.
MI  Minutes as in 1 thru 59
SS  Seconds as in 1 thru 59
SSSSS  Seconds of the day as in 1 thru 86399
AM, PM  Meridian Indicator
A.M., P.M.  Meridian Indicator with periods.

Example 1

```sql
select TO_CHAR(SYSDATE) "SYSDATE",
       TO_CHAR(SYSDATE, 'MON-DD-YYYY') "Date 1",
       TO_CHAR(SYSDATE, 'MON:DD:YYYY') "Date 2",
       TO_CHAR(SYSDATE, 'DAY, MONTH DD, YEAR') "Date 3"
from dual
```

<table>
<thead>
<tr>
<th>SYSDATE</th>
<th>Date 1</th>
<th>Date 2</th>
<th>Date 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>08-20-2007</td>
<td>AUG-20-2007</td>
<td>AUG-20-2007</td>
<td>MONDAY, AUGUST 20, TWO THOUSAND SEVEN</td>
</tr>
</tbody>
</table>

Example 2

```sql
select TO_CHAR(SYSDATE) "SYSDATE",
       TO_CHAR(SYSDATE, 'HH:MI:SS AM') "Time 1",
       TO_CHAR(SYSDATE, 'HH:MI:SS AM') "Time 2",
       TO_CHAR(SYSDATE, 'HH:MI:SS AM') "Time 3"
from dual
```
to_date(text, format)
The to_date function takes text and uses the formatting codes to convert the text into a date data type. The format is telling the database that the text is in the that format. Remember that the database stores a date as a number, so it must understand what the text is representing as part of that date. The format codes listed in the to_char table are the same for the to_date.

<table>
<thead>
<tr>
<th>SYSDATE</th>
<th>DATE 1</th>
<th>DATE 2</th>
<th>DATE 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007-09-08 09:00 AM</td>
<td>2007-09-08 10:00 AM</td>
<td>2007-09-08 11:00 AM</td>
<td></td>
</tr>
</tbody>
</table>

2.5 Order By Clause

The only way to insure that the rows are ordered the way you want them is have the query specify the sorting using the ORDER BY clause. Also, sorting is always performed last.

```sql
select
    SYSDATE,
    to_date(to_char(SYSDATE, 'MM/DD/YYYY'), 'MM/DD/YYYY') "DATE 1",
    to_date('08/20/2007', 'MM/DD/YYYY') "DATE 2",
    to_date('08/20/2007', 'MM/DD/YYYY') "DATE 3"
from dual
```

```sql
select
    distinct office_symbol
from
    contract
 order by
    office_symbol
```
You can also order by multiple columns.

```sql
select contract_id, office_symbol
from contract
order by office_symbol, contract_id
```
2.6 Case Statement

The case statement is used to change values. It uses the syntax:

```
CASE selection WHEN x THEN y WHEN q THEN r ELSE z END
```

```
SELECT contract_id, MOD_NO, agencycode
CASE agencycode
    WHEN 'C' THEN 'Corps of Engineers'
    WHEN 'E' THEN 'Environmental Protection Agency'
    WHEN 'I' THEN 'Contractor'
    WHEN 'P' THEN 'Potential Responsible Party'
    WHEN 'S' THEN 'State'
    WHEN 'T' THEN 'All other'
    WHEN 'U' THEN 'User'
    ELSE 'Unknown'
END AS Agency Key
FROM pso_log
```

2.7 SQL Comparison Operators

A comparison operator evaluates two values and returns a TRUE, FALSE, or NULL. Comparison operators are used in the WHERE clause to limit the number of returned rows.
Equals  =  WHERE first_name = 'DORINDA'

Not Equals  !=  WHERE state != 'CA'
<=>  WHERE state <> 'CA'
^=  WHERE state ^= 'CA'

Less Than  <  WHERE pay < min_wage
Greater Than  >  WHERE pay > my_pay
Less Than or Equal  <=  WHERE pay <= 2000
Greater Than or Equal  >=  WHERE pay >= 100000

There are special comparison operator that are used with multiple values.

**BETWEEN... AND** Validates that a value is between the first and second values, inclusive

**IN (...)** Validates that a value is contained in the list of values.

**LIKE** Like matches a character pattern. There are two special characters used to match characters. The percent % is zero or more characters wildcard. The underscore _ is a single character wildcard. If you are looking to match one of the special characters then you will need to use the escape character \ so that the database treats it as a literal and not as a wildcard character.

**NOT** The NOT operator simply negates the operator following.

**EXIST** The EXIST operator returns TRUE if a subquery returns at least one row. Likewise, NOT EXIST returns TRUE if the subquery does not return at least one row.

**IS NULL** Returns TRUE if the value is NULL. IS NOT NULL returns TRUE if the value is not NULL.
2.8 **Oracle Functions**

**Built-in SQL Functions**
Basically a function takes one or more inputs and returns one value as a result. All programming languages support functions; and the Oracle database uses a language called PL/SQL to supply most of the standard functions.

**Character or Text Functions**

**upper(string)**

```sql
select upper(proj_ds) from contract where proj_ds is not null
```

**lower(string)**

```sql
select lower(proj_ds) from contract where proj_ds is not null
```
initcap(string)

```sql
1 select 
2   initcap(proj_ds)
3 from 
4   contract 
5 where 
6   proj.ds is not null
```

concat(string1,string2)

```sql
1 select 
2   concat(My Contract ID is ',contract_id) "Data 1", 
3   concat(contract_id,contract_id) "Data 2", 
4   contract_id || ' - ' || contract_id "Data 3"
5 from 
6   contract 
7 where 
8   proj_ds is not null
```
substr(string1, start, count)

```
1 2 3 4 5 6 7
select contract_id,
    substr(contract_id, 1, 2) Prefix
from contract
where proj_ds is not null
```
ltrim(s1,s2) / rtrim(s1, s2)
Sometimes you need to remove characters from the beginning and or end of a string. Normally you are removing spaces, but you may need to remove other characters. Ltrim removes any character in s2 from the front of s1. Think of s2 as a list of characters rather than a word. Rtrim does the same thing except it removes the character from the end of s1. The string s2 defaults to a space. If the characters in s2 are not in s1, then s1 is returned unchanged.

trim(s2 from s1)
The trim function incorporates both ltrim and rtrim in one command. You can set trim to remove the leading, trailing or both. The default is both. The string s2 defaults to a space. When using trim, you can only define one character to trim. With rtrim or ltrim, you are not restricted.

```sql
select
    TRIM('abcedefgh', 'abc') TRIM,  -- leading and trailing
    TRIM('abcedefgh', 'c') TRIM,   -- only leading
    TRIM('abcedefgh', 'g') TRIM,   -- only trailing
    TRIM('abcedefgh', 'abg') TRIM  -- both
from
dual
```

Number Functions

round(n,d)
The round function rounds a number \( n \) to the specified decimal \( d \). The decimal \( d \) can be positive, negative or zero.

```sql
select
    ROUND(1234.345, 2),
    ROUND(1234.315, 0),
    ROUND(1234.315, -2)
from
    dual
```

### Example Results

<table>
<thead>
<tr>
<th>( \text{ROUND(1234.345,2)} )</th>
<th>( \text{ROUND(1234.315,0)} )</th>
<th>( \text{ROUND(1234.315,-2)} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234.35</td>
<td>1234</td>
<td>1200</td>
</tr>
</tbody>
</table>

**trunc(n,d)**

The trunc or truncate function simply drops the digits without rounding. The decimal \( d \) can again be positive, negative or zero. If the number truncated is five or higher, it is still dropped without rounding the next digit up.

```sql
select
    TRUNC(1234.345, 2),
    TRUNC(1234.315, 0),
    TRUNC(1234.315, -2)
from
    dual
```
<table>
<thead>
<tr>
<th>TRUNC(1234.345, 2)</th>
<th>TRUNC(1234.345, 0)</th>
<th>TRUNC(1234.345, 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234.34</td>
<td>1234</td>
<td>1200</td>
</tr>
</tbody>
</table>
Example
3 RAP Primer

3.1 Overview

RAP is a subset of Delphi’s Object Pascal and stands for Report Application Pascal (RAP). Most language elements in Delphi are recognized by the RAP compiler and can be found in the Code Toolbox. The following is a list of elements contained in the Toolbox.

**Data Types:**
- Boolean
- Currency
- Double
- Extended
- Integer
- Single
- Char
- String
- Date
- DateTime
- Time
- Color
- Variant

**Statements:**
- Case statements
- If–then statements
- If–then–else statements
- For loops
- While loops

**Operators:**
- Assignment (=)
- Boolean (and, not, or, xor)
- Class (as, is)
- Math (-, +, *, /, div, mod)
- Relational (<, <=, <>, =, >, >=)
- String (+)
- Unary (-, +)

**Currently Unsupported Elements:**
- Class declarations
- Arrays
- Record types
- Set types

**Value and Variable parameters**
Most parameters are either value parameters (default) or variable (var) parameters. Value parameters are passed by value, while variable parameters are passed by reference. To see what this means, consider the following functions.
3.2 Data Types

Here are the list of Data Types available to be used withing RAP.

- Boolean
- Currency
- Double
- Extended
- Integer
- Single
- Char
- String
- Date
- DateTime
- Time
- Color
- Variant

To add a local variable declaration to a function, activate the Code Editor for the current item and place the cursor between the function's declaration and the begin. Type var and declare your variables.

```plaintext
procedure DEText1OnGetText(var Text: String);
var
  sMyString: String;
  sMyBool: Boolean;
  sMyInteger: Integer;
  sMyDouble: Double;
begin
  Text := '';
end;
```

3.3 Operators

Operators behave like predefined functions that are part of the internal RAP language. For example, the expression (a construction which returns a value) \((X + Y)\) is built from the variables \(X\) and \(Y\) --- which are called operands --- with the + operator; when \(X\) and \(Y\) represent integers or real numbers then \((X+Y)\) will return their sums.

Some operators behave differently depending on the type of data passed to them. For example the addition operator (+) when used with integers performs a sum but when used with two string performs concatenation.

**Example:**

\( X + Y = \text{sum of } X \text{ and } Y \)

\( 'One' + ' ' + 'One' = One One \)
Assignment ( := )
The **assignment** operator ( := ) assigns a variable to an expression.

\[
\text{variable} := \text{expression};
\]

**Examples:**

\[
\begin{align*}
1 &:= 3; \\
X &:= Y + Z; \\
mString &:= 'This is my string'; \\
bMyBoolean &:= \text{False};
\end{align*}
\]

**Example in Reportbuilder**

```pascal
procedure DetailBeforeGenerate;
var
  iCount: Integer;
  bStarted: Boolean;
  cMyColor: TColor;
begin
  ktr.First;
  iCount := 0;

  while not ktr.EOF do
  begin
    if ktr['DUNS4'] = RBVU_DISTINCT_KTR_DUNS4['DUNS'] then
    begin
      iCount := iCount+1;
      end;

    ktr.Next;
  end;

  case iCount of
    0..3: cMyColor := clBlack;
    4..6: cMyColor := clYellow;
    7..8: cMyColor := clBlue;
    else
      cMyColor := clRed;
    end;

  ktr.First;
  Label2.Text := IntToStr(iCount);
  Label2.Font.Color := cMyColor;
end;
```

As you can see we create a variable in the **var** section and then assign it in the code section which is
between the **begin** and **end** statements.

**Boolean (and, not, or, xor)**
The Boolean operators take operands of any Boolean type and return a value of type Boolean (True or False).
Used in if and While statements.

**Class (as, is)**

**Math (-, +, *, /, div, mod)**
Math operators, which take real or integer operands

**Relational (<, <=, <>, =, >, >=)**
Relational operators are used to compare two operands.

**String (+)**

**Unary (-, +)**

### 3.4 If Statement

**Using the If then statement.**
There are two forms of the **if** statement: **if...then** and the **if...then...else**. The syntax of an **if...then** statement is

```plaintext
if expression then statement
```

where expression returns a Boolean value. If expression is True, then statement is executed; otherwise it is not. For example,

```plaintext
if J<> 0 then Result := I/J;
```

The syntax of an **if...then...else** statement is

```plaintext
if expression then statement1 else statement2
```

where expression returns a Boolean value. If expression is True, then statement1 is executed; otherwise statement2 is executed. For example,

```plaintext
if J = 0 then
  Exit
else
  Result := I/J;
```

The then and else clauses contain one statement each, but it can be a structured statement. For example,

```plaintext
if J <> 0 then
  begin
    Result := I/J;
    Count := Count + 1;
  end
else if Count = Last then
Done := True
else
Exit;

if AValue = '1' then
  Result := 'FUTURE'
else if AValue = '2' then
  Result := 'ACTIVE'
else if AValue = '3' then
  Result := 'CONSTRUCTION COMPLETE'
else if AValue = '4' then
  Result := 'PHYSICAL COMPLETE'
else if AValue = '5' then
  Result := 'FINAL PAYMENT'
else if AValue = '6' then
  Result := 'FISCAL COMPLETE'
else
  Result := '';
procedure Memo1OnPrint;
var
  lsLine: String;
  lsState: String;
  lsZIP: String;
begin
  {clear memo}
  Memo1.Lines.Clear;

  {add contact}
  lsLine := pICustomer["Contact"]; {Check to see if lsLine has a value}
  if lsLine <> '' then
    Memo1.Lines.Add(lsLine);

  {Check to see if lsLine has a specific number of characters.}
  if Length(lsLine) = 1 then
    begin
      Memo1.Lines.Add('Has one character.')
    end
  else if Length(lsLine) >= 10 then
    Memo1.Lines.Add('Has multiple characters greater than 10')
  else if (Length(lsLine) > 0) AND (Length(lsLine) < 10) then
    Memo1.Lines.Add('Has multiple characters less than 10.')
  else
    Memo1.Lines.Add('Has no characters');
end;

3.5 Case Statement

Case Statements
The case statement may provide a readable alternative to deeply nested if conditionals. A case statement has the form

    case selectorExpression of
      caseList1: statement1;
      ...
      ...
    end;
caseList_n: statement_n;
end

A case statement can have a final else clause:

case selectorExpression of
  caseList1: statement1;
  .
  .
  caseList_n: statement_n;
else
  statements;
end

where selectorExpression is any expression of an ordinal type (string types are invalid) and each caseList is one of the following:
A numeral, declared constant, or other expression that the compiler can evaluate.

Examples of case statements

case I of
  1..5: Caption := 'Low';
  6..9: Caption := 'High';
  0, 10..99: Caption := 'Out of Range';
else
  Caption := '';
end;

is equivalent to the nested conditional

if I in [1..5] then
  Caption := 'Low'
else if I in [6..10] then
  Caption := 'High'
else if (I = 0) or (I in [10..99]) then
  Caption := 'Out of Range'
else
  Caption := '';
end;

Other examples of case statements:

case MyColor of
  Red: X := 1;
  Green: X := 1;
  Blue: X := 1;
  Yellow, Orange, Black: X := 0;
end;

case Selection of
  Done: Form1.Close;
  Compute: CalculateTotal();
else
  Beep;
end;
```plaintext
case AValue of
  1: Result := 'FUTURE';
  2: Result := 'ACTIVE';
  3: Result := 'CONSTRUCTION COMPLETE';
  4: Result := 'PHYSICAL COMPLETE';
  5: Result := 'FINAL PAYMENT';
  6: Result := 'FISCAL COMPLETE';
else
  Result := '';
end;
```

Example in Report Builder

```plaintext
procedure DetailBeforeGenerate;
var
  iCount: Integer;
  bStarted: Boolean;
  cMyColor: TColor;
begin
  ktr.First;
  iCount := 0;

  while not ktr.EOF do
    begin
      if ktr['DUNS4'] = RBVU_DISTINCT_KTR_DUNS4['DUNS'] then
        begin
          iCount := iCount+1;
        end;
      ktr.Next;
    end;
  case iCount of
    0..3: cMyColor := clBlack;
    4..6: cMyColor := clYellow;
    7..8: cMyColor := clBlue;
  else
    cMyColor := clRed;
  end;
  ktr.First;
  Label2.Text := IntToStr(iCount);
  Label2.Font.Color := cMyColor;
end;
```
3.6 While Statement

While Statements
The syntax of a while statement is

```
while expression do statement
```

where expression returns a Boolean value and statement can be a compound statement. The while statement executes its constituent statement repeatedly, testing expression before each iteration. As long as expression returns True, execution continues.

Examples of a while statements include

```
while Data[I] <> X do I := I + 1;

while I > 0 do
    begin
        if Odd(I) then Z := Z * X;
        I := I div 2;
        X := Sqr(X);
    end;
```

Example in Report Builder

```pascal
procedure DetailBeforeGenerate;
var
    S: String;
begin
    rap.First;
    Memo2.Lines.Clear;
    while not rap.EOF do
        begin
            S := rap['Name'];
            if Length(S) > 0 then
                Memo2.Lines.Add(S);
            rap.Next;
        end;
    rap.First;
end;
```
3.7 Procedures and Functions

Procedures and functions are self-contained statement blocks that can be called from different locations within RAP. A function is a routine that returns a value when it executes. A procedure is a routine that does not return a value.

Function calls, because they return a value, can be used as expressions in assignments and operations. For example,

\[ I := \text{SomeFunction}(X); \]

calls SomeFunction and assigns the result to I. Function calls cannot appear on the left side of an assignment statement.

\[ \text{SomeFunction}(X) := I; \]

Function calls can be used as complete statements though.

\[ \text{SomeFunction}(X); \]

The functions return value is then discarded.

Declaring Procedures and Functions in RAP

A procedure declaration in RAP has the form;

```
procedure procedureName(parametersList);
localDeclarations; //optional
begin
  statements
end;
```

A function declaration is like a procedure declaration except that it specifies a return type and a return value. Function declarations have the form;

```
function functionName(parametersList): returnType;
localDeclarations;
begin
  statements
end;
```

ParameterList
Most procedure and function headers include a parameter list. For example;

```
function TotalUp(X: Integer; Y: Integer): Integer;
```

the parameter list is (X: Integer; Y: Integer).

Other types of parameter list examples include;

\[(X, Y: \text{Integer});\]
(s1, s2: String);
(var S: string; X: Integer);

**Statements**

Any valid expression which include other functions or procedures.

Examples include:

\[ X := 1 + 2; \]
\[ 
\text{MyString} := 'Hello RMS';
\]
\[ 
\text{Result} := \text{TotalUp}(X,Y);
\]

### 3.8 Gotchas

When using the RAP code compiler there are many different things that could cause you to stumble when you are new to the language.

**Assignment** Operator ( := ) VS Logical Operator ( = )

**When** to use the semicolon in an *if statement* ( ; )

Example 2:

```pascal
if Project_Stage = 1 then
  s := 'Future'
else
  s := 'Not Future';
```

Example 2:

```pascal
if v = 1 then
  s := 'Future'
else if v = 2 then
  s := 'Active'
else if v = 3 then
  s := 'Construction Complete'
else
  s := 'Not 1, 2 or 3';
```

```pascal
if v = 1 then
  begin
    s := 'Future';
  end
else if v = 2 then
  begin
    s := 'Future';
  end
else if v = 3 then
```
s := 'Construction Complete'
else if v = 4 then
  s := 'Physical Complete'
else if v = 5 then
  s := 'Final Payment'
else
  s := 'Fiscal Complete';

When to use the begin and end statements.
Used for syntax requirements.
Used for multiline statements.